Efficient Time Series Prediction Model for Embedded or Mobile System

Dongjun Lee 2023-22406*
elite1717@snu.ac.kr
Interdisciplinary Program in Artificial Intelligence
Seoul National University

Minjun Park 2023-20349*
minjunpark@snu.ac.kr
Interdisciplinary Program in Artificial Intelligence
Seoul National Univiersity

ACM Reference Format:
Dongjun Lee 2023-22406 and Minjun Park 2023-20349. . Efficient Time Series Prediction Model for Embedded or Mobile System. In . , 7 pages.

1 INTRODUCTION

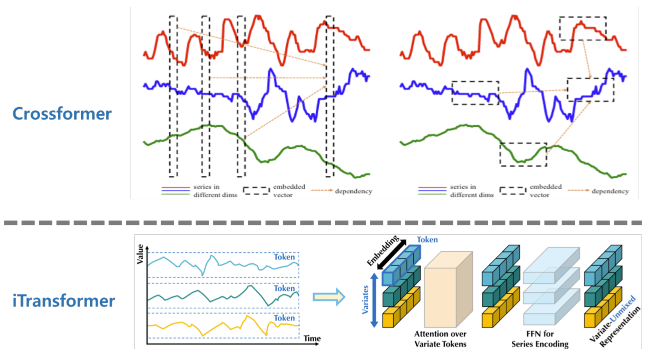

Figure 1: Two major channel-dependent time series models.

Embedded systems should be a primary concern in the field of personalized AI. As AI technology advances, the demand for personalized AI is surging. Large language models (LLMs) and generative models are among the most actively researched areas within personalized AI. However, most recent real-world applications of AI often involve embedded systems, such as those found in home appliances. Therefore, it is valuable to consider personalization in conjunction with embedded systems.

Embedded systems have two key features distinguishing them from LLMs and generative models. First, they heavily rely on multi-channel sensor inputs. Since most sensors passively collect data, the model is consistently streamed with continuous multivariate time series data. Second, embedded systems have limited power resources, which necessitates the use of small and efficient models. Considering these two attributes, we must develop power-efficient models capable of handling multivariate time series data.

In the domain of multivariate time series analysis, there are two competing approaches: explicit and implicit temporal dependency models. Crossformer[6] is a representative of models that explicitly consider temporal dependency as well as inter-channel dependency. While it is an intuitive approach for handling time series data, it is computationally intensive, making it unsuitable for power-limited environments. On the other hand, iTransformer[3] represents implicit ones. It embeds a short time sequence into a token using a multi-layer perceptron (MLP). This model assumes that a simple linear model is sufficient to capture temporal dependencies and focuses solely on inter-channel dependencies. While this guarantees efficient inference, linear temporal modeling may be insufficient for real-world applications beyond standard benchmarks.

Long-term information handling is another critical aspect of temporal dependency modeling. While theoretically, increasing the size of the look-back window can provide long-term information to the forecasting model, recent studies indicate that doing so without proper processing can result in significantly adverse effects, such as decreasing accuracy with the self-evident increased model size and longer training and inference times [5]. Adding to this, conventional time series models often make inferences independently for each time step, neglecting the high correlations between sequential inputs.

Efficient long-term information processing is even more crucial in mobile and embedded systems. These systems constantly stream data from various sensors, necessitating short inference times. To achieve this with the limited resources of embedded systems, the model must reduce the size of the look-back window, which limits the model's ability to handle long-term information and severely undermines its performance. Moreover, if the service demands high resolution, sensors must adopt a higher sampling rate, which exacerbates the problem by further decreasing the effective time slice.

In this project, we will explore two methods for developing a power-efficient model for multivariate time series. The first approach aims to improve the performance of the iTransformer model by enhancing the temporal token encoder. This improvement is beneficial for embedded systems as it allows for a lighter version of the model without sacrificing performance. We will use Neural Architecture Search (NAS) to find the optimal network structure from a combination of given building blocks. By constructing the component set with efficient operations, we can achieve an encoder that is both efficient and performant for mobile systems. For the second part, we propose a novel module that enables us to learn long-term information that exceeds the look-back window(input) and efficiently utilizes constantly streaming inputs. This allows us to reduce the input size while keeping the performance, leading to a smaller model size and inference time to fulfill the constraints on mobile and embedded devices.

*Both authors contributed equally to this research.

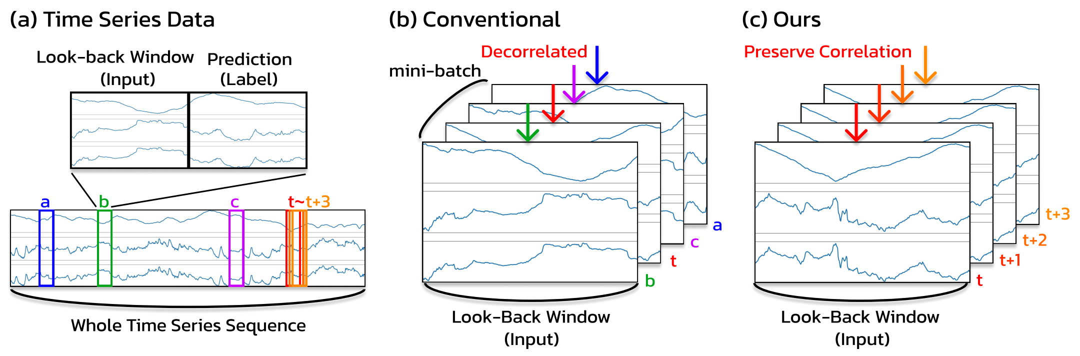

Figure 2: (a) Training data are sampled for each time step from continuous sequences, exhibiting high temporal correlations. (b) Conventional approaches simply ignore this temporal information with a shuffled batch. (c) We address the temporal correlation between the samples, enabling the model to consider long-range dependencies that surpass the look-back window.

2 METHOD

We adopted iTransformer as our base model. Crossformer has a high computational cost due to its multi-stage and hierarchical structure. Considering embedded systems' stricter hardware constraints, iTransformer, which is more efficient, is more suitable. Furthermore, we shrank the model size of the iTansformer assuming the harsh condition of the embedded systems.

In this project, we focus on the temporal token encoder. We conjecture that the iTransformer's major drawback compared to Crossformer is its strong assumption of linear temporal dependency. Additionally, we aimed to lighten the base model for embedded systems, which necessitates a more effective feature extractor. To address this challenge, we employed two approaches. First, we utilized Neural Architecture Search (NAS) for the temporal encoder to enhance performance while maintaining efficiency. Second, we implemented a Spectral Attention module, allowing the model to consider longer-range correlations while preserving a limited short look-back window size.

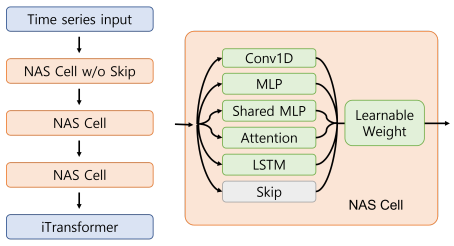

Figure 3: Modified temporal token encoder diagram. Unlike the original encoder, the input sequence is divided into several chunks. The encoder consists of three NAS searchable cells. Each cell has five possible modules or a skip connection and the weighted sum of each module's output is used. In this diagram, *MLP* denotes channel-dependent MLP and *Shared MLP* denotes channel-independent MLP.

2.1 Time Series Forecasting Problem Definition

In multivariate time series forecasting, time series data is given $\mathbb{D}_T : \{x_1, ..., x_T\} \in \mathbb{R}^{T\times N}$ at time T with N variates. Our goal is, at arbitrary future time $t > T$, to predict future S time steps $Y_t = \{x_{t+1}, ..., x_{t+S}\} \in \mathbb{R}^{S\times N}$. To achieve this goal, TSF model f utilizes length L look-back window as input $X_t = \{x_{t-L+1}, ..., x_t\} \in \mathbb{R}^{L\times N}$ making prediction $P_t = f(X_t)$, $P \in \mathbb{R}^{S\times N}$. Model is trained with the training dataset $D_T = \{(X_t, Y_t)|L \le t \le T - S\}$.

2.2 NAS for Temporal Encoder

NAS is a technique for finding an optimal architecture within a predefined search space. The optimality metric can encompass not only model accuracy but also computational complexity. Therefore, it can help in finding a suitable architecture under hardware constraints. Moreover, we can construct the search space using only lightweight operations, ensuring that the resultant model's efficiency after NAS does not deviate significantly from that of the original iTransformer.

2.2.1 NAS: Search Space. The most important aspect to consider in NAS is defining the search space. We defined a cell-level search space for our encoder. First, we decided to use three layers for the encoder. As candidates for these three layers, we defined five possible modules: 1D convolution, LSTM, channel-independent MLP, channel-dependent MLP, and patched temporal attention. For the second and third layers, we added an identity block as an option to simulate a dynamic number of layers.

2.2.2 NAS: Search Strategy and Performance Estimation. Search and performance estimation strategies are also required for NAS. For the search strategy, we used gradient-based search. In gradient-based search, we define multiple paths, each passing through different modules. The weighted average of each path is used as the output of the layer, with the weights for the modules being learnable. This allows the training process through backpropagation to dynamically select which path to use. Given that we are training a light iTransformer model, we can load the entire NAS components onto a single GPU. Therefore, the gradient-based search with a large network is the most efficient method, as it requires training only once to consider all combinations of the modules.

To estimate performance, we count MACs (Multiply–Accumulate Operations) as a proxy for the computational efficiency metric. Although MACs are not a perfect estimate of the model's latency, we decided to use them for two main reasons. First, MACs are weight-independent, allowing us to compute them before training. Once we precompute the MACs, we can integrate them into the NAS optimization without any overhead. Second, we are assuming an embedded device with limited resources. Since such devices may lack sophisticated compute cores and may not support massive parallelism, we conjecture that the number of computations is linearly correlated with the latency. The final objective of the optimization is the sum of the model's prediction accuracy and the expected MACs:

$$\mathcal{L} = \mathcal{L}_{acc} + \sum_{l=1}^{3}\sum_{i} w_{l,i} MACS_{l,i}$$

S where $\mathcal{L}_{acc}$ is the model's accuracy, $MACS_{l,i}$ represents MACs of the i^{th} module in layer l, and $w_{l,i}$ is the corresponding weight of that module.

2.3 Spectral Attention for learning long-range information

In mid-March, while selecting a new research topic in the time series forecasting domain, the author specifically chose to align the research with the efficient model discussed in this class. This indicates that the research was genuinely initiated in connection with the class subject, rather than being a pre-existing project submitted as a class assignment.

2.3.1 Spectral Attention. We introduce Spectral Attention, a mechanism that allows the model to leverage long-term temporal dependencies within sequential data. Spectral Attention (SA) is adaptable to any TSF model. Let the base TSF model be represented as $P = f(X)$, and SA can be integrated with any activation vector F within this model. Thus, the base model can be redefined as $P = f_2(F, E)$ with $F, E = f_1(X)$. The Spectral Attention module takes the feature F as input and transforms it into F' of the same dimensions, $F' = SA(F)$, thereby modifying the base model to: $f_{SA} = f_2 \cdot SA \cdot f_1$.

Given X_t as the input to the base model, SA processes a D-dimensional feature vector $F_t \in \mathbb{R}^D$. SA updates the exponential moving average (EMA) $M_t \in \mathbb{R}^{K\times D}$ of F_t within its internal memory, utilizing K *smoothing factors* $\{\alpha_1, ..., \alpha_K\} \in \mathbb{R}^K$ ($\alpha_1 < \cdots < \alpha_K$).

$$M_{t+1}^{k,i} = \alpha_k \times M_t^{k,i} + (1-\alpha_k) \times F_t^i \tag{1}$$

The EMA captures the trend of features across extended time frames based on the smoothing factor, acting as a low-pass filter. The -3db (half) cut-off frequency is given by $f_{cut} = \frac{1}{2\pi}\cos^{-1}\left[1 - \frac{(1-\alpha)^2}{2\alpha}\right]$, effectively maintaining the trend over a span of 6,000 periods with $\alpha = 0.999$.

To highlight high-frequency patterns in contrast to the low-pass filtered long-term trend M, we generate $H_t \in \mathbb{R}^{K\times D}$ by subtracting M_t from F_t.

$$H_t^{k,i} = F_t^i - M_t^{K-k-1,i} \tag{2}$$

SA comprises learnable parameters: the *sa-matrix* $\in \mathbb{R}^{(2K+1)\times D}$, which is responsible for learning the specific frequencies the model should focus on for each feature. The vectors $2 \times H_t$, F_t, and $2 \times M_t$ are concatenated along dimension 0, forming a tensor of shape $\mathbb{R}^{(2K+1)\times D}$. This tensor is then subjected to a weighted summation with the *sa-matrix* along dimension 0, producing the output F'_t.

$$\begin{aligned} F'_t = \sum(&\text{softmax}(\textit{sa-matrix}, \text{dim } 0)\cdot \\ &\text{concat}((2\times H_t, F_t, 2\times M_t), \text{dim } 0), \text{dim } 0) \end{aligned} \tag{3}$$

The *sa-matrix* is initialized in such a way that $softmax(\textit{sa-matrix})$ approximates a Gaussian distribution along axis 0. This initialization results in symmetric values along axis 0 ($\textit{sa-matrix}^{K+1-i} = \textit{sa-matrix}^{K+1+i}$) and ensures that SA functions as an identity at initialization ($\because H^k + M^{K-k-1} = F$).

$$F = SA_{init}(F) \tag{4}$$

SA enables the model to focus on various frequencies of its feature signal, thereby allowing it to attend to long-range dependencies or high-frequency patterns as required, and to shift the feature F distribution within the frequency domain. By initializing SA as an identity function, the model can be fine-tuned using the pre-trained base model, facilitating efficient implementation.

2.3.2 Batched Spectral Attention. Batched Spectral Attention (BSA) enables the model to perform batch training over multiple time steps. The primary idea involves unfolding the EMA, which allows gradients to flow through successive samples within a mini-batch, similar to Backpropagation Through Time (BPTT). This approach encourages the model to extract long-range information beneficial for future predictions, effectively extending the look-back window. The overall process of BSA is illustrated in Figure 4.

For a mini-batch of size B, consecutive samples $X_{[t,t+B-1]} = \{X_t, ...X_{t+B-1}\} \in \mathbb{R}^{B\times S\times N}$ are provided as input. Following the previously described SA configuration, BSA takes $F_{[t,t+B-1]} = \{F_t, ...F_{t+B-1}\} \in \mathbb{R}^{B\times D}$ as input. In the subsequent step, BSA uses $F_{[t,t+B-1]}$ along with the stored $M_t \in \mathbb{R}^{K\times D}$ to generate $M_{t+b} (0 \leq b \leq B)$ by unfolding Equation 1.

$$M_{t+b}^{k,i} = \alpha_k^b \times M_t^{k,i} + (1-\alpha_k)\alpha_k^{b-1} \times F_t^i + \cdots + (1-\alpha_k) \times F_{t+b-1}^i \tag{5}$$

This equation can be transformed to calculate $M_{[t,t+B]} \in \mathbb{R}^{(B+1)\times K\times D}$ in parallel as follows.

$$M_{[t,t+B]}^{:,k,i} = \textit{lower-triangle}(A^k) \times \textit{concat}((M_t^{k,i}, F_{[t,t+B-1]}^{:,i}), \textit{dim } 0) \tag{6}$$

$$A \in \mathbb{R}^{K\times(B+1)\times(B+1)},\ A^{k,p,q} = (1-\alpha_k)^{I\{q>0\}}\alpha_k^{p-q} \tag{7}$$

Let A denote the unfolding matrix and I the indicator function. M_{t+B} is retained in BSA for the next mini-batch input. The computation of $F'[t, t+B-1]$ is performed in parallel, akin to Equations 2 and 3, using $F[t, t+B-1]$, $B_{[t,t+B-1]}$, and the *sa-matrix* $\in \mathbb{R}^{(2K+1)\times D}$. The *lower-triangle* function ensures that gradients from previous timesteps do not flow into future models, preserving the time-series data characteristics.

At the start of each training epoch, M_0 is initialized to F_0 for all α, enhancing stability for the subsequent EMA accumulation. Since SA aims to capture long-range dependencies during training,

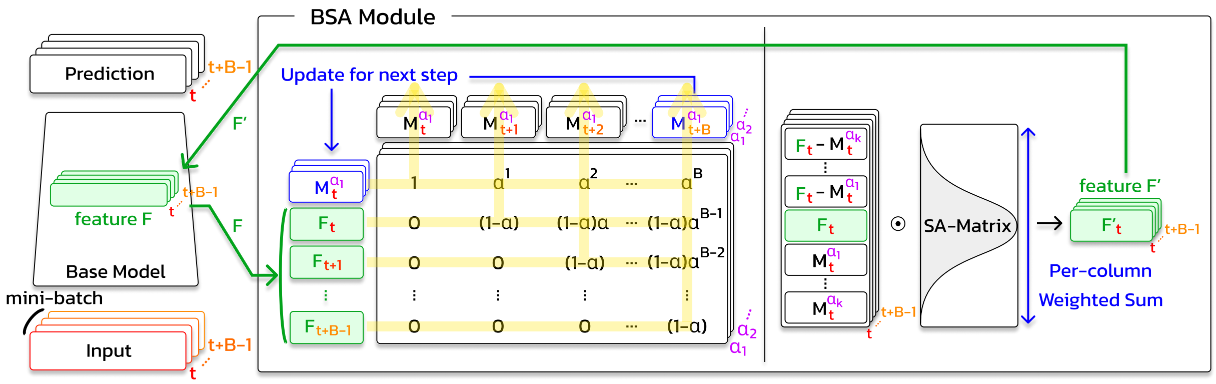

Figure 4: Overall architecture of *BSA*. The base model on the left can be any TSF model. *BSA* can take arbitrary feature F as input and replace it with a richer feature F'. The *BSA* module empowers the base model to learn the long-term dependency way over the look-back window, allowing the model to take a smaller input size and, thus, a smaller model size.

it initially lacks sufficient information when not enough steps have been observed. Therefore, to stabilize training, we linearly warm up the learning rate for the first $1/(1 - \max(\textit{smoothing factors}))$ timesteps at the beginning of each training epoch.

In *SA*, the *smoothing factors* $\{\alpha_1, ..., \alpha_K\} \in \mathbb{R}^K$ were scalar values. In contrast, in *BSA*, they are learnable parameters. This change allows *BSA* to leverage additional past information during training, beyond the look-back window, by incorporating a batch-sized time window to determine the extent of long-range dependency required for training. To constrain the smoothing factors between 0 and 1, we initialize the learnable parameters by applying an inverse sigmoid to the initial smoothing factors and then apply a sigmoid function during training.

Thus far, we have considered the feature F from the base model as a vector. However, intermediate layers of the model often output tensors with two or more dimensions. In practice, we use additional channel dimensions in *BSA* to handle the activation tensor, effectively applying multiple *BSA* modules simultaneously.

3 EXPERIMENT SETTINGS

We decided to compare our methods with the original iTransformer using the public Appliances Energy dataset[1]. This dataset consists of 28 variables, 19,735 timesteps, and includes measurements from temperature and humidity sensors in a wireless network, weather data from a nearby airport station, and recorded energy use of lighting fixtures.

The second real-world public dataset used is the Weather dataset available from TimesNet[4] GitHub page. The Weather dataset includes 21 meteorological factors acquired every 10 minutes in 2020 from the Weather Station of the Max Planck Institute for Biogeochemistry.

Evaluation metrics for model performance are mean squared error(MSE) and mean absolute error(MAE).

4 EXPERIMENT RESULTS

4.1 NAS for Temporal Encoder

The iTransformer [3]'s hyperparameters settings are (encoder_layers = 2, d_model = 64, d_ff = 64, dropout = 0.1, num_heads = 8, activation = GELU [2]). The look-back window length(input size) and the prediction length(output size) are both set to 96, which is the standard setting in time series forecasting. We used an AdamW optimizer with a 0.0003 learning rate, 0.01 weight decay, and 0.9 lr decay for 40 epochs to prevent overfitting while fully saturating the whole model.

4.1.1 Weighted Average: Softmax vs. LASSO. We had two options when computing a weighted sum of each NAS cell's outputs. One option was to apply softmax, allowing us to interpret the weighted sum as an expected value. The other option was to use the weights directly without any probabilistic interpretation. While using softmax is a more common approach in machine learning, it lacks any sparsity constraint. We decided to test the LASSO regularizer, which induces sparsity, with the second approach. Figure 5 shows how the weights change during NAS training. Similar results were observed for the weather dataset, so we only plotted the energy data for this section.

Notably, both approaches selected the same top-2 modules for the energy usage task. Therefore, the NAS-selected model architecture might be independent of the weighting strategy. As intended, LASSO assigned lower weights to the less important modules. However, it assigned negative weights to the conv1D modules starting from the middle of the training, indicating a sudden architectural change during NAS. This might cause instability in training, but the resulting model showed similar or slightly better performance. An interesting observation is that the residual module, which is merely an identity operation, received high weights, suggesting that adopting a single embedding layer might be a valid option under harsh constraints.

4.1.2 Finetune Stage after NAS. Once the NAS search network is trained, we can select the top one or two modules for each layer based on the weight size. The NAS-trained weights depend on the other modules, which will be dropped out. Therefore, using the top

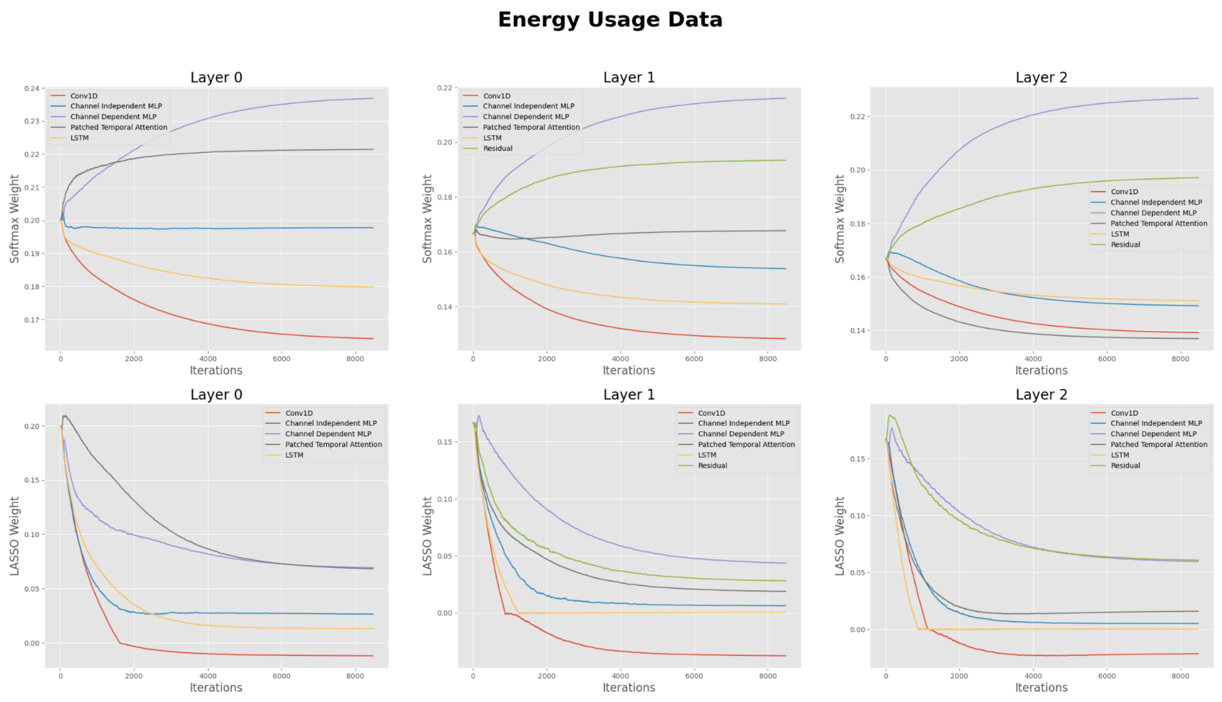

Figure 5: Trained weights of the modules via NAS. The upper row applied softmax to the learned weights. The lower row used the weights directly and employed a LASSO regularizer to enforce sparsity.

one or two modules directly results in poor performance. Consequently, we fine-tuned the chosen network for a fair comparison. Table 1 shows the result after fine-tune. As mentioned in the previous section, using LASSO or softmax does not make a significant difference. Although they had different weights after NAS, they converged to similar performance after fine-tuning. Using only the top module gives a similar performance to using the top two modules.

Table 1: Performance of NAS searched model after finetune.

		Energy Data		Weather	
		MAE	MSE	MAE	MSE
Original iTransformer		0.4736	0.4825	0.2163	0.1761
LASSO	NAS Search	0.4499	0.4465	0.2047	0.1552
	top-1 module only	0.4481	0.4437	0.2073	0.161
	top-2 modules	0.4487	0.446	0.2037	0.1554
Softmax	NAS Search	0.4521	0.4527	0.2054	0.1558
	top-1 module only	0.4631	0.468	0.2082	0.161
	top-2 modules	0.4564	0.4611	0.2032	0.1546

4.2 Spectral Attention for learning long-range information

4.2.1 Method Performance Evaluation. We demonstrate that BSA improves forecasting performance by providing the ability to address long-range dependencies beyond the look-back window. This enables us to make the input and the model size smaller while keeping the same or even showing higher performance.

The baseline iTransformer model configuration is directly copied from the iTransformer [3] GitHub page, where we can find the standard iTransformer settings for the Weather dataset. The setting is (encoder_layers = 3, d_model = 512, d_ff = 512, dropout = 0.1, num_heads = 8, activation = GELU).

We implemented the BSA module to a lighter iTransformer model. The module is placed right before the temporal embedding linear layer of the iTransformer. We did a greedy hyperparameter search on the initialization of the smoothing factors, learning rate, and weight decay to fully saturate both the baseline iTransformer and the light iTransformer with the BSA module. We only changed the e_layer, d_model, and d_ff to make the model smaller. More importantly, the input look-back window is halved from 96 to 48 while the forecasting length is still 96.

As can be seen in Table 2 and 3, our Spectral Attention method allows the light model with significantly fewer parameters, and MACs achieve similar or even better performance than the baseline model. With the torchinfo module, we couldn't get the exact MACs for the light model with the BSA method since the BSA includes user-defined non-typical operations. The number of parameters added by the BSA module largely depends on the input variable size and the d_model since we placed the BSA in the embedding layer.

4.2.2 Method Inspection. We analyzed whether the performance improvement was due to the BSA module learning long-term information longer than the input window. In Figure 6, we display the

Table 2: Experiment results on Weather dataset. All experiments are conducted with three random seeds and the average value of the seeds is reported.

	BSA	input	output	e_layer	d_model	d_ff	# Param.	MACs	MSE	MAE
Baseline	X	96	96	3	512	512	4,833,888	42,656,352	0.17064	0.20957
Light	X	48	96	2	64	64	59,936	459,296	0.20470	0.23127
	O	**48**	**96**	**2**	**64**	**64**	**66,992**		**0.17042**	**0.21255**
	X	48	96	1	16	16	4,144	17,200	0.20951	0.23958
	O	**48**	**96**	**1**	**16**	**16**	**11,200**		**0.17685**	**0.22035**

Table 3: Experiment results on EnergyData dataset. All experiments are conducted with three random seeds and the average value of the seeds is reported.

	BSA	input	output	e_layer	d_model	d_ff	# Param.	MACs	MSE	MAE
Baseline	X	96	96	3	512	512	4,833,888	53,687,904	0.48430	0.46995
Light	X	48	96	1	16	16	4,144	21,008	0.60844	0.52195
	O	**48**	**96**	**1**	**16**	**16**	**13,552**		**0.48109**	**0.46742**

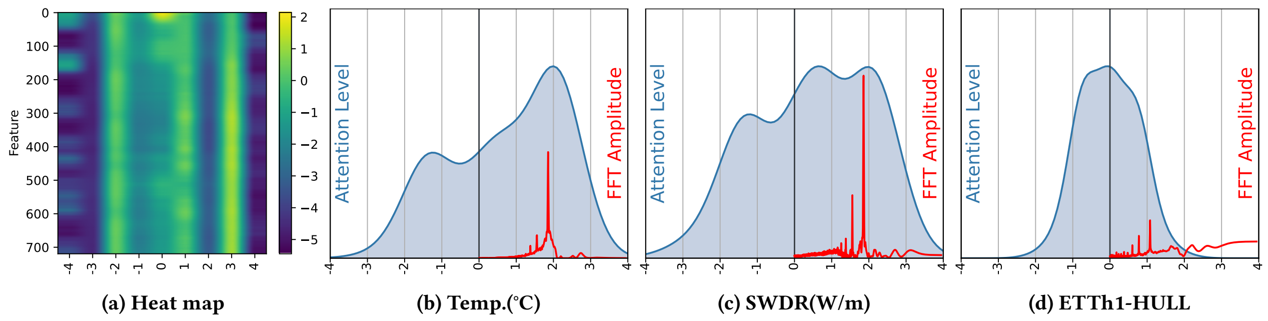

Figure 6: This figure illustrates the analysis of the SA-matrix trained on the prediction task for the Weather dataset. Panel (a) shows the heatmap of the SA-matrix, and (b)-(d) show the attention and FFT graphs.

heat map of the trained SA-Matrix for the Temperature (°C) channel in the Weather dataset. The positive x-axis indicates the log_{10} values of the periods retained by the low-pass filter, derived from the smoothing factor. Negative values represent high-frequency components. The blue line in Figure 6b applies a Gaussian kernel to the SA-Matrix, averaging over the feature dimension, showing the frequencies to which BSA attends overall. The red line depicts the Fast Fourier Transform (FFT) and denoising applied to the raw signal. The blue curve's skew towards the low-frequency side indicates that BSA effectively captures the long-range trend of the data. Figure 6c shows the graph for the SWDR (Short Wave Downward Radiation per unit area, W/m) channel in the same SA-Matrix. Although not identical to Figure 6b, it also emphasizes attention to low-frequency patterns. In contrast, Figure 6d, the graph for the HULL (High UseLess Load) channel from the ETTh1 dataset, illustrates that the signal itself lacks long-range trends, resulting in an SA-Matrix resembling the Identity. This outcome demonstrates that BSA functions as intended, learning the low-frequency components of the signal for future prediction.

5 CONCLUSION

In this project, we developed an efficient yet high-performing time series model for embedded systems. Building on the efficient iTransformer model, we further reduced its size and improved its performance using two approaches. First, we searched for a better temporal encoder with three layers using NAS. This study was based on the assumption that the iTransformer lacks temporal dependency modeling compared to Crossformer. Our experiments showed that enhancing the temporal encoder boosts the iTransformer's prediction performance with minimal computational overhead.

The second part of the study focused on efficient long-range dependency modeling for the iTransformer embedding. We focused on a method that leverages the characteristics of time series data by utilizing the high correlation information between sequential inputs, which traditional forecasting models have ignored. Experiments have shown that Spectral Attention can handle long-term information that exceeds the look-back window, enabling multivariate time series forecasting models to achieve the same performance efficiently with shorter input lengths and smaller model sizes.

The code for this project can be found in GitHub page. https://github.com/DJLee1208/MUC.git

REFERENCES

[1] Luis Candanedo. 2017. Appliances Energy Prediction. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5VC8G.

[2] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[3] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. 2023. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625* (2023).

[4] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. 2023. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. *ICLR* (2023).

[5] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting?. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 11121–11128.

[6] Yunhao Zhang and Junchi Yan. 2022. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The eleventh international conference on learning representations*.