

Full Terms & Conditions of access and use can be found at
<https://www.tandfonline.com/action/journalInformation?journalCode=uiie21>



Smartformer: An intelligent transformer compression framework for time-series modeling

Xiaojian Wang^a, Yinan Wang^b , Jin Yang^a , and Ying Chen^a 

^aSchool of Management, Harbin Institute of Technology, Heilongjiang, China; ^bDepartment of Industrial and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA

ABSTRACT

Transformer, as one of the cutting-edge deep neural networks (DNNs), has achieved outstanding performance in time-series data analysis. However, this model usually requires large numbers of parameters to fit. Over-parameterization not only brings storage challenges in a resource-limited setting, but also inevitably results in the model over-fitting. Even though literature works introduced several ways to reduce the parameter size of Transformers, none of them addressed this over-parameterized issue by concurrently achieving the following three objectives: preserving the model architecture, maintaining the model performance, and reducing the model complexity (number of parameters). In this study, we propose an intelligent model compression framework, Smartformer, by incorporating reinforcement learning and CP-decomposition techniques to satisfy the aforementioned three objectives. In the experiment, we apply Smartformer and five baseline methods to two existing time-series Transformer models for model compression. The results demonstrate that our proposed Smartformer is the only method that consistently generates the compressed model on various scenarios by satisfying the three objectives. In particular, the Smartformer can mitigate the overfitting issue and thus improve the accuracy of the existing time-series models in all scenarios.

ARTICLE HISTORY

Received 15 November 2023
Accepted 28 June 2024

KEYWORDS

Intelligent model
compression; transformer;
CP-decomposition;
reinforcement learning;
deep learning

1. Introduction

1.1. Overview

Recent years have witnessed a rapid development in Transformer, which is a variant of Deep Neural Networks (DNNs) (Han *et al.*, 2022). Different from Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), Transformer has an encoder-decoder architecture that is solely based on the attention mechanism. Such a flexible structure helps Transformer achieve success in time-series modeling, such as influenza forecasting (Wu *et al.*, 2020), wind power forecasting (Qu *et al.*, 2022), EEG classification (Xie *et al.*, 2022), solar irradiation prediction (Liu *et al.*, 2023), and energy consumption forecasting (Zheng *et al.*, 2023). Despite its effective modeling architecture, similar to other popular variants of DNNs, Transformer usually requires a massive number of parameters (Vaswani *et al.*, 2017). Although over-parameterization is one key factor in the success of DNNs (Neyshabur *et al.*, 2017), deploying these parameters is a challenge in a resource-limited setting (Ma *et al.*, 2019). Furthermore, over-parameterization inevitably introduces overfitting and local optima (Ding *et al.*, 2022) that hinders convergence in training. Therefore, an immediate question is how to intelligently select a subset of parameters in Transformer to simultaneously meet three objectives: (i) preserve the model architecture, (ii) maintain

the model performance, and (iii) reduce the model complexity (number of parameters).

Literature works have shown that weight sharing, knowledge distillation, pruning, tensor decomposition and quantization are likely to make the Transformer models smaller. Particularly, Xu and McAuley (2022) provided a decision tree to determine which of these five techniques to use in practice. For example, unstructured pruning supports the sparsity and quantization only considers the model compression by ignoring the model accuracy. As a comparison, knowledge distillation and structured pruning have a requirement for the inference speed, whereas weight sharing and tensor decomposition take into account the model accuracy. In terms of the three objectives mentioned above, we focus on the tensor decomposition since weight sharing alters the way the Transformer model computes in sequence, restricting the flexibility of the model.

Tensor decomposition tries to both reduce the model complexity and preserve its architecture. The idea is to decompose the weight matrices/tensors of each self-attention layer in Transformer and propose low-rank alternatives for existing layers. For example, Ma *et al.* (2019) proposed a multi-linear attention module with Block-term Tensor Decomposition (BTD), which is a combination of CANDECOMP/PARAFAC (CP)-decomposition (Kiers, 2000) and Tucker decomposition (Tucker, 1966). Cordonnier *et al.*

(2021) utilized the Tucker decomposition to convert the standard attention layer to a collaborative attention layer, which enables the heads of attention to learn the shared projections so as to decrease the parameters of Transformer. Wang *et al.* (2020) decomposed the self-attention layer into multiple smaller attentions via linear projections, which then forms a low-rank factorization of the original attention. In addition, some other studies not only exploited the tensor decomposition methods to decompose the self-attention layer, but also decomposed the embedding layer or fully-connected layer in Transformer (see Chen *et al.* (2021) and Li *et al.* (2021)). The main advantage of these methods is to approximately retain the original operation in the Transformer and break down the structured weight matrices/tensors for flexible selection. Despite the success, not all of them are suitable for time-series modeling. For instance, BTD (Ma *et al.*, 2019) will generate a tensor of $Seq.len^3$ ($Seq.len$ means sequence length), which indicates that for a long sequence time-series forecasting problem, it requires quite large internal storage if the sequence length is large. Another example is found in Gu *et al.* (2022), which proposes a hardware-efficient automatic tensor decomposition for Transformer compression, aiming to reduce the computational cost in the hardware. However, this method requires the hardware cost in the optimization, while the general time-series modeling problems do not have hardware cost in hand. In addition to this, all the aforementioned studies have the following two limitations:

1. All existing studies fail to address the trade-off between reducing the parameters versus maintaining the model performance.
2. None of these studies completely derived the expressions of forward and backward propagations for the decomposed layers.

As to the first issue, literature works usually decomposed the model layers in a brute-force way and did not achieve a good balance between model compression and performance. Consequently, we observe a performance drop in numerous studies (see Ma *et al.* (2019) and Chen *et al.* (2021)). As to the second issue, even though AutoGrad (Maclaurin *et al.*, 2015) can easily output the weight derivative for the decomposed layers, it is less flexible to generate derivatives for each decomposed weight kernel so as to facilitate fine-tuning on partial weights in each decomposed layer.

Considering these, we aim to develop a new tensor decomposition method to adaptively compress Transformer for time-series modeling so that we can simultaneously satisfy the objectives of preserving the model architecture, maintaining the model performance, and reducing the model complexity.

1.2. Contribution

In this study, we propose an intelligent model compression framework, Smartformer, to concurrently fulfill the three objectives mentioned above for time-series modeling. Smartformer has two main sub-tasks: the first one is to replace

the Multi-head Attention layer with a low-rank alternative; the second one is to re-design the training process to both fit the model parameters and optimize the model decomposition and compression process. It should be noted that the first sub-task can make the Transformer model smaller, whereas the second sub-task can maintain the model performance. Our technical contributions mainly lie in two aspects.

Our first technical contribution is to propose using the CP-decomposition method to approximately compress the Multi-head Attention layer in Transformer. Note that this application is an extension of CP-decomposition in CNNs (Wang *et al.*, 2022). The main difference between the convolutional layer and the attention layer is that the convolutional kernel has the sliding property, whereas the attention layer does not. To the best of our knowledge, CP-decomposition has not been explored in Transformer. We only account for the attention layer decomposition with two reasons: (i) Ma *et al.* (2019) pointed out that the Multi-head Attention layer, as the key component in Transformer for the contextual learning, required a large number of parameters; (ii) Xu and McAuley (2022) stated that decomposing weight matrices in the token embedding layers were not efficient. We name this new layer after the CPAC-Attention layer. As a result of incorporating the CPAC-Attention layer, we are able to derive the expressions for forward and backward propagation in Transformer accordingly. Such derivations provide the theoretical investigation for independently tuning each decomposed weight matrix/tensor for Transformer to the literature.

Our second technical contribution is to model the optimization of rank R for each layer as a Markov Decision Process (MDP). In CP-decomposition, rank R is a key hyperparameter that determines the model complexity. However, there are two issues that have been ignored. The first one is that these studies have not considered optimizing the rank R with an efficient algorithm, but rather just show the results from several different rank values (see Lebedev *et al.* (2015) and Wang *et al.* (2022)). The second one is that these studies have not considered assigning different values of R for different layers while just using a uniform R for all layers. Layers of a DNN generally have different contributions to the model performance (referred to as the layer importance in Molchanov *et al.* (2019)). Hence, each layer should have its own rank value. There are unique challenges for determining different rank values of different layers. First, the optimization of different rank values of all layers is computationally expensive. For example, if there is a Transformer with three attention layers and each layer can be decomposed with 10 candidate rank values, there will be 10^3 combinations of ranks that need to be evaluated. It is almost practically infeasible to repeat the training for every single combination and evaluate the performance, which makes memoryless optimization methods (i.e., heuristic optimization, mathematical programming, etc.) invalid for our problem. In addition, the design space is gigantic and exponentially grows with the increase of layers, as well as the candidate rank values for every single layer. Therefore, the optimization method is required to be capable of effectively exploring the high-dimensional design space. These two

challenges make the MDP formulation, along with the Deep-Reinforced Learning (DRL)-based online optimization method, a unique fit to solve this problem. The basic idea is that as the model training, the DRL agent can sequentially determine the rank value for each layer. More specifically, this method can first decompose the least important layer to learn the impact of layer decomposition on model performance and improve its policy on finding the best rank (exploration). Starting the decomposition from the least important layer can minimize the model performance loss in the initial stage caused by the under-developed policy. After that, this method can decompose the important layers by exploiting the well-developed policy (exploitation). It should be noted that we use a DRL method to solve this MDP problem with a single sample trajectory. Literature works (Castronovo *et al.*, 2013; Wang and Zou, 2021) have reported that in spite of effectiveness, we may not obtain the optimal solution with reinforcement learning under the setting of a single sample trajectory. Therefore, we target to provide a quality rank R value for each layer.

In the experiments, we conduct the practical experiments by applying the Smartformer to two existing time-series Transformer models. To effectively evaluate the strength of the Smartformer, we exploit five existing model compression methods from the literature as benchmarks. To ensure fair comparisons, we utilize the datasets that were used for the two backbone models. Each case contains at least one scenario. After conducting a comprehensive analysis, we find out that (i) some models can only satisfy one or two objectives we mentioned above; (ii) some models can satisfy all three objectives only in specific scenarios; (iii) the proposed Smartformer is the only one model that can satisfy the three objectives simultaneously in all scenarios.

All in all, the contributions of this study are listed below:

1. We propose a Smartformer framework to intelligently compress the Transformer for time-series modeling.
2. We use the CP-decomposition method to decompose the Multi-head Attention layer and conduct the theoretical investigations for the newly proposed CPAC-Attention layer.
3. We model the rank R determination problem for each attention layer as an MDP, and exploit a DRL-based online training algorithm to simultaneously optimize the model parameters and decompose the model structures.
4. We conduct extensive experiments to showcase the performance of Smartformer.

1.3. Organization of this article

The remainder of this article is organized as follows. Section 2 presents related works to this study. Section 3 develops the CPAC-Attention layer in forward and backward propagations and analyzes the computational properties of the CPAC-Attention layer. Section 4 proposes the Smartformer framework. Section 5 mainly shows the results of the experiments and comprehensively analyzes the

performance of Smartformer. Section 6 gives the concluding remarks.

2. Related works

In this study, we consider the use of a CP-decomposition method to decompose the attention layer in Transformer and optimize the rank values of CP-decomposition with a DRL algorithm. In the following, we present the related works in applying CP-decomposition for model compression in deep learning and the selection of hyperparameters in model compression.

2.1. CP-decomposition

The CP-decomposition method has been widely used in deep learning. For example, Lebedev *et al.* (2015) and Wang *et al.* (2022) made use of CP-decomposition to compress CNNs. Ma *et al.* (2021) exploited CP-decomposition to compress the Long-Short Term Memory (LSTM) networks, Wang *et al.* (2021) utilized CP-decomposition to compress the RNNs, Ma *et al.* (2019) developed a BTM method with CP-decomposition and Tucker decomposition methods to compress the attention layer for Transformer. Moreover, the CP-decomposition method has some advantages over the others. Compared with Tensor Train Decomposition (TTD) and Tucker decomposition, the CP-decomposition method decomposes the tensor into multiple groups of vectors, which are unique and can uncover the actual latent components (Wang *et al.*, 2022). Compared with BTM (Ma *et al.*, 2019) and TTD (Li *et al.*, 2021), the CPAC-Attention layer can achieve a single-step training, which means it can be obtained directly from the original attention layer without additional steps to process the tensors. In addition, Li *et al.* (2020) provided the theoretical guarantees in applying the CP-decomposition to DNNs regarding compressibility and generalizability. For Transformer, to the best of our knowledge, work that solely uses the CP-decomposition method to reduce model parameters has not been explored.

2.2. Selection of hyperparameters

The selection of hyperparameters in model compression (e.g., rank R in CP-decomposition) directly affects the model performance and model complexity. The existing works only intuitively set R with different levels for investigation (Wang *et al.*, 2022). As demonstrated, if the value of R is small, the model complexity will be significantly reduced, but the model performance will also be largely decayed. However, how to efficiently optimize this hyperparameter in CP-decomposition is not clear. Such a phenomenon also occurs in other tensor decomposition works (e.g., Wang *et al.* (2020) and Cordonnier *et al.* (2021)) because the decomposition methods in them, such as Tucker decomposition, also have a similar hyperparameter that needs to be optimized (i.e., the dimension of the core tensor in Tucker decomposition). In general, optimizing the value of rank R in decomposing the weight tensor (e.g., Tucker decomposition,

CP-decomposition, and Singular Vector Decomposition (SVD)) during the deep learning model's training process is an extremely challenging task. The model weights (tensor to be decomposed) will be continuously updated according to the current batch of training data, which makes the optimal decomposition solution also dynamically changed and thus makes the optimization of the decomposition rank a highly non-linear and non-convex optimization problem. Some recent works have considered this issue. For example, Chen *et al.* (2021) utilized a grid search to select a proper R when using SVD, whereas Ma *et al.* (2019) assumed that the rank R values of Tucker decomposition for query (Q), key (K) and value (V) matrices in the self-attention layer of Transformer are the same as the dimension of Q , K , and V . However, such a grid search method is not intelligent since it has to train the model with each rank R from scratch, and the assumption about rank R in Ma *et al.* (2019) does not provide any theoretical explanations. Therefore, we innovatively incorporate a DRL algorithm into the model training process to iteratively train a DRL agent, which aims to select an optimized R value for each decomposed attention layer.

3. Development of the CPAC-Attention layer

In this section, we adapt CP-decomposition to approximately compress the attention layer in Transformer and propose the CPAC-Attention layer. To provide necessary backgrounds for the derivations, we first review the CP-decomposition and attention mechanism in Section 3.1. After that, we follow Wang *et al.* (2022) and conduct the forward and backward propagation derivation and analysis of the CPAC-Attention layer. Note that the main difference between the convolutional layer and attention layer is that the convolutional layer has a sliding property, whereas the attention layer does not have that property. To decompose the convolutional layer with the CP-decomposition method, the sliding characteristic of convolutional layers necessitates a high-dimensional unfolding of the input tensor according to the shape of the convolutional kernel. In contrast, the attention layer just need to tensorize the input. In Section 3.2, we first derive the complete expressions of forward and back propagations of the proposed CPAC-Attention layer; subsequently, we discuss the computational properties of the CPAC-Attention layer.

3.1. Preliminaries

3.1.1. Formulation of CP-decomposition

The CP-decomposition method decomposes a high-order tensor into multiple groups of rank-one tensors (vectors). The expression of CP-decomposition on a three-way tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is given in (1), where $a_i \in \mathbb{R}^I, b_i \in \mathbb{R}^J, c_i \in \mathbb{R}^K, i = 1, \dots, R$, are decomposed rank-one tensors, and operator \circ represents the outer product:

$$\mathcal{X} \approx \sum_{i=1}^R a_i \circ b_i \circ c_i. \quad (1)$$

3.1.2. Self-attention mechanism

The main idea of the self-attention mechanism is to guide the model to focus on the most relevant information in the input when generating a certain output (Vaswani *et al.*, 2017). It is formulated as mapping the query and a set of key-value pairs to the output, in which the similarity between the query and key will determine which part of the value will be focused. In practice, we calculate a set of attention functions at the same time and formulate all sets of query, key, and value into matrices Q , K , and V , respectively:

$$Q = IW^Q, \quad (2)$$

$$K = IW^K, \quad (3)$$

$$V = IW^V, \quad (4)$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \quad (5)$$

where $W^Q \in \mathbb{R}^{d_{\text{model}} \times hd_k}$, $W^K \in \mathbb{R}^{d_{\text{model}} \times hd_k}$, $W^V \in \mathbb{R}^{d_{\text{model}} \times hd_v}$, d_k is the dimension of the matrix Q and K in each head, d_v is the dimension of the matrix V in each head, I is the input, d_{model} is the dimension of input, and W^Q, W^K, W^V are weight matrices for Q, K, V , respectively. In the Multi-head Attention procedure, the attention is calculated on h separated heads. For each head, the query, key, and value are firstly projected into lower-dimensional feature spaces with dimensions d_k, d_k , and d_v using different weight matrices. Then, the attention is calculated separately on each head. Finally, all the outputs are concatenated and fed through a linear layer.

3.2. Derivation and analysis of the CPAC-Attention layer

In the Multi-head Attention layer, $d_k = d_v = d_{\text{model}}/h$. Therefore, the input can be denoted as $I \in \mathbb{R}^{S \times hd}$, where S is the sequence length of the input, and d is the dimension of the head. Then, the Q, K , and V are generated by mapping the input via weight matrices $W^Q \in \mathbb{R}^{hd \times hd}$, $W^K \in \mathbb{R}^{hd \times hd}$, and $W^V \in \mathbb{R}^{hd \times hd}$, respectively. We can find out that most of the parameters in the Multi-head Attention layer exist in these weight matrices. Without loss of generality, the input and output of the mapping are assumed to have the same dimension d for simplicity of notation. Hence, we use the matrix calculus (Magnus and Neudecker, 1985) presented at Appendix A in the Supplemental Online Materials to derive the complete expressions of forward propagations of the proposed CPAC-Attention layer in terms of Q, K, V , which are given below:

$$Q = \sum_{r=1}^R \left(\left(\mathcal{J} \times_3 q_r^d \right) \times_2 q_r^h \right) \circ q_r^{hd}, \quad (6)$$

$$K = \sum_{r=1}^R \left(\left(\mathcal{J} \times_3 k_r^d \right) \times_2 k_r^h \right) \circ k_r^{hd}, \quad (7)$$

$$V = \sum_{r=1}^R \left((\mathcal{J} \times_3 v_r^d) \times_2 v_r^h \right) \circ v_r^{hd}, \quad (8)$$

where $\mathcal{J} \in \mathbb{R}^{S \times h \times d}$ is a three-way tensor and operator \times_n represents the n -mode vector product. Moreover, q_r^d, q_r^h and q_r^{hd} ($r = 1, \dots, R$) denote the decomposed vectors along each dimension of \mathcal{W}^Q ($\mathcal{W}^Q \in \mathbb{R}^{h \times d \times hd}$), which is a reformulated W^Q . Similarly, k_r^d, k_r^h and k_r^{hd} denote the decomposed vectors along each dimension of \mathcal{W}^K , and v_r^d, v_r^h and v_r^{hd} denote the decomposed vectors along each dimension of \mathcal{W}^V . \mathcal{W}^K and \mathcal{W}^V ($\mathcal{W}^K \in \mathbb{R}^{h \times d \times hd}$, $\mathcal{W}^V \in \mathbb{R}^{h \times d \times hd}$) are the reformulated W^K and W^V , respectively. The detailed information of these derivations is presented in [Appendix B.1](#) of Supplemental Online Materials.

In addition, we derive the backward propagation of our proposed CPAC-Attention layer. Since the CPAC-Attention layer only modifies the mapping from the input I to Q , K , and V , we can denote the operations afterward as a function $g(\circ)$. Thus, the output of the CPAC-Attention layer is:

$$Y = g(Q, K, V). \quad (9)$$

Gradient-based optimization methods are commonly used in backward propagation to train the model parameters. Taking Q as an example, we need to derive the gradients of the layer output Y with respect to (w.r.t.) the newly introduced weight vectors $q_r^d, q_r^h, q_r^{hd}, r = 1, \dots, R$, which are $\left(\frac{\partial Y}{\partial q_r^d}, \frac{\partial Y}{\partial q_r^h}, \frac{\partial Y}{\partial q_r^{hd}} \right)$. Specifically, we have these three gradients as follows:

$$\frac{\partial Q}{\partial q_r^{hd}} = I_M \otimes A_1^\top, \quad (10)$$

$$\frac{\partial Q}{\partial q_r^h} = B_1^\top \otimes A_2^\top, \quad (11)$$

$$\frac{\partial Q}{\partial q_r^d} = P(\mathcal{J}_{(3)}^\top) (B_2 \otimes I_{Sh}). \quad (12)$$

where I_M and I_{Sh} are the identity matrices with M and Sh dimensions, respectively, \otimes represents the Kronecker product, $A_1 \in \mathbb{R}^{S \times 1}$ denotes the constant part $((\mathcal{J} \times_3 q_r^d) \times_2 q_r^h)$, $A_2 \in \mathbb{R}^{S \times h}$ denotes the constant part $(\mathcal{J} \times_3 q_r^d)$, $B_1 \in \mathbb{R}^{hd \times 1}$ denotes the constant part q_r^{hd} , $B_2 \in \mathbb{R}^{h \times hd}$ denotes $q_r^h (q_r^{hd})^\top$, $\mathcal{J}_{(3)} \in \mathbb{R}^{d \times Sh}$ denotes the mode-3 unfolding of \mathcal{J} , and $P(\circ)$ represents a permutation operator indicating the elements in the matrix are arranged differently. The detailed derivation of $\frac{\partial Q}{\partial q_r^{hd}}$, $\frac{\partial Q}{\partial q_r^h}$ and $\frac{\partial Q}{\partial q_r^d}$ can be found at [Appendix B.2](#) in the Supplemental Online Materials. Combining (10), (11), and (12), we generate the detailed expressions of $\left(\frac{\partial Q}{\partial q_r^{hd}}, \frac{\partial Q}{\partial q_r^h}, \frac{\partial Q}{\partial q_r^d} \right)$, which will be used to update the decomposed tensors of the CPAC-Attention layer in the backward propagation. Similarly, we can derive the gradients of the layer output Y w.r.t. the decomposed attention layers K and V . We put the forward and backward-propagations for Transformer with CPAC-Attention layers in [Appendix C.1](#) at the Online Supplemental Materials.

Table 1. Comparison of properties between CPAC-Attention and Multi-head Attention layers.

	Multi-head Attention	CPAC-Attention
Number of Parameters	$3(hd)^2$	$3R(hd + h + d)$
FLOPs	$3Shd(2hd - 1)$	$3RS[h(3d + 1) - 1]$
Required Memory (MB)	$12(hd)^2 \times 10^{-6}$	$12R(hd + h + d) \times 10^{-6}$

Lastly, we analyze the CPAC-Attention layer considering the number of parameters, the number of Floating-point Operations (FLOPs), and the required memory. A comparison of the properties of the proposed CPAC-Attention layer and the original Multi-head Attention layer is presented in [Table 1](#). It is worth noting that the objectives of fewer parameters and smaller FLOPs are not monotonic. This means that reducing the number of parameters does not necessarily lead to a reduction in FLOPs. The goals of this work are to reduce the number of parameters, maintain the model performance, and preserve the model architecture. Thus, we explicitly include reducing the number of parameters as an objective in our proposed framework, which may not necessarily lead to a reduction in FLOPs. The detailed analysis of the CPAC-Attention layer related to the number of parameters, the number of FLOPs, and the required memory are presented in [Appendix B.3](#).

4. Development of Smartformer

In this section, we propose a Smartformer framework that intelligently optimizes the model decomposition and compression in the training process. The basic ideas include:

1. When building the Transformer model using the CPAC-Attention layer, the performance is inevitably influenced by the hyperparameter rank R introduced in the CP-decomposition (see (6)–(8)). To address this, the Smartformer leverages DRL to iteratively optimize the hyperparameter R for the selected Multi-head Attention layer.
2. To maintain the model performance when reducing the parameters, the layer importance is specifically defined to select the least important layer for approximation. In this manner, the negative influence of the decomposition on the model performance will be mitigated.

This DRL-based online optimization method is included in the training process of the transformer to reduce the number of parameters while preserving the model performance. It only has a fixed number of decompositions applied to each layer (the length of each episode is fixed), and the decomposition also terminates at the point of training stops (the total number of episodes for RL is capped). Therefore, it is hard to guarantee that the DRL agent will converge to receive enough training and then learn the optimal policy. However, it is a valid justification that the DRL agent can select the best action (value of rank R) given its current experience. An example could be referred to Cai *et al.* (2018). In the rest of this section, we first model the

decision-making process in Smartformer as a MDP. After that, we present the overall architecture of Smartformer.

4.1. MDP in Smartformer

Smartformer focuses on efficiently optimizing the rank R in CP-decomposition during the training process. To implement one-time training for saving computational time, we consider online learning to dynamically adjust the rank R . Hence, we model such a decision-making problem as an MDP and solve it with a DRL algorithm (Sutton and Barto, 2018). A DRL agent (modeled by a neural network) is trained to maximize the cumulative reward by learning from repeated episodes. Each episode is a sequence of states, actions, and rewards, which will be introduced in the following sections. Stage indicates the interval between two consecutive actions, which can be defined according to the training settings. For example, if a model is trained with 1000 epochs, every 50 epochs could be a stage; if a problem is trained with limited epochs, several batches within each epoch could be a stage. In the following, we will describe the reward, state, and action as well as the DRL algorithm, respectively.

4.1.1. Reward

The reward function used in the DRL can be defined in multiple ways. We present an example (13):

$$\mathcal{R}(a_t, s_t) = \begin{cases} acc_{t+1}/acc_t, & \text{if } acc_{t+1} \geq acc_t - \epsilon \\ -acc_t/acc_{t+1}, & \text{otherwise} \end{cases} \quad (13)$$

where t represents the t th stage ($t \geq 1$) in the decomposition process of a selected layer, s_t represents the state at stage t , a_t represents the selected action given s_t , acc_t represents the average accuracy of training given the model structure at stage t (i.e., s_t), acc_{t+1} represents the average accuracy of training given the model structure at stage $t+1$ (i.e., s_{t+1}), and ϵ is the loss tolerance threshold that can be adjusted manually based on the preference of practitioners. If one prefers to focus more on compressing the model, ϵ should be set larger; otherwise, ϵ should be set smaller to maintain the model performance. The objective of the DRL algorithm is to maximize the cumulative rewards. Hence, when a decomposition can maintain the model performance within a given threshold, it will be encouraged by giving a positive reward intuitively. Otherwise, the reward is negative.

Alternatively, the reward can also be defined using the value of training loss, which is given in (14):

$$\mathcal{R}(a_t, s_t) = \begin{cases} loss_t/loss_{t+1}, & \text{if } loss_{t+1} \leq loss_t \epsilon \\ -loss_{t+1}/loss_t, & \text{else} \end{cases} \quad (14)$$

where $loss_t$ represents the training loss given the model structure at stage t (i.e., s_t), $loss_{t+1}$ represents the training loss given the model structure at stage $t+1$ (i.e., s_{t+1}), and ϵ here is used as a scaling factor to control the trade-off between model performance and model complexity. The choice of reward function to use depends on the task that the network needs to solve. For classification tasks, accuracy is often used to measure the performance of the model.

In this case, (13) can be used. For prediction tasks, Mean Squared Error (MSE), Mean Absolute Error (MAE), or more direct loss can be used as a metric to measure the model performance, in which (14) can be used. Note that, in (14), due to the unbounded range of loss values, which can span from zero to positive infinity, it may not be feasible to determine a specific threshold for calculating the difference between the previous loss and the current loss. To address this, we adopt a threshold represented as the product of $loss_t$ and ϵ , where ϵ reflects the acceptable level of performance degradation. Similar to the ϵ in (13), a larger value of ϵ in (14) indicates a preference for compressing the model over maintaining its performance. Otherwise, maintaining the model performance is encouraged. The practitioners can set their own reward functions only if the reward functions are well aligned with the optimization objectives. We test the performance of Smartformer using these two types of rewards in the experiments.

4.1.2. State and action

State in MDP describes the current observation. In our problem, the state represents the structure of the selected Multi-head Attention layer. Given the state at a stage, the policy network will generate an action to determine the value of rank R for the selected layer of interest. Following Cai *et al.* (2018), we exploit an encoder net to encode the structure of the selected Multi-head Attention layer into vectors with the following steps:

1. A dictionary containing all architectures of layers is established.
2. The given network structure is then represented as a string, which is fed into the dictionary to look up the corresponding sequence of indexes.
3. An embedding layer with size of 512 is followed to map each index into a vector.
4. The vector corresponding to the layer of interest is selected through positional encoding.
5. A one-layer bidirectional LSTM network with 64 hidden units is further used to process the vector.

The output of the encoder net is the state representation corresponding to the input layer, which will be further entered into the policy network and value network to update the action and estimate the value, respectively. For example, we represent the CPAC-Attention layer using a set of strings “MultiheadAttention-0-5-2-6”, where 0, 5, 2, and 6 respectively denote the layer’s position, the number of attention heads, the dimension of each head, and the current rank R . We use the encoder to transform the string representation into a feature vector, which thus serves as the state representation. After an action R (e.g., $R = 2$) is chosen, the parameters within the CPAC-Attention layer are reshaped back into a tensor format through the inverse operation of CP-decomposition and then decomposed according to the action ($R = 2$). Consequently, the string representation of CPAC-Attention would change to “MultiheadAttention-0-5-2-2”,

and the corresponding state representation would also be changed.

4.1.3. Advantage actor-critic

Advantage Actor-Critic (A2C) (Konda and Tsitsiklis, 1999) algorithm consists of an Actor (policy network) and a Critic (value network). The Actor determines the action (value of rank R) given the current state (the current structure of the selected Multi-head Attention layer). Specifically, the Actor outputs the probability of selecting each action based on the current state. During the action selection process, an ϵ -greedy strategy is implemented to maintain a balance between exploration and exploitation. With a probability of ϵ , the Actor selects an action randomly from the action space, thereby fostering exploration of the action space. With a probability of $1 - \epsilon$, the Actor selects the action that has the highest probability, which is the greedy choice aiming to maximize the expected reward. The ϵ will gradually decrease as the training progresses. The Critic gives an estimation of the cumulative reward for the current state. Furthermore, the Critic calculates the Temporal Difference (TD) error, which quantifies the difference between the actual reward obtained following an action and the previously predicted value of the next state. This TD error serves as a feedback signal that reflects the performance of the current policy. If the TD error is positive, it indicates the reward received following an action is actually better (greater) than the predicted value. Therefore, such an action should be encouraged by updating the policy. In the training process, the Actor refines its policy by continuously exploiting the signal from the TD error to optimize long-term outcomes. The Actor and Critic are approximated by two separate neural networks, which are both made up of two fully connected layers with 40 hidden units and with the rectified linear unit (ReLU) activation function. The loss function of the Actor is the negative log-likelihood function, and the loss function of the Critic is a linear function. In this study, we use the policy gradient and TD to update the parameters of the policy network and value network, respectively. The Actor is updated through policy gradients, whereas the Critic is updated through value gradients, as shown in (15) and (16):

$$L_{actor}(\theta) = -\log \pi_{\theta}(a_t|s_t)A(s_t, a_t), \quad (15)$$

where advantage function $A(s_t, a_t) = R_t + \gamma v_{\phi}(s_{t+1}) - v_{\phi}(s_t)$;

$$L_{critic}(\phi) = ||R_t + \gamma v_{\phi}(s_{t+1}) - v_{\phi}(s_t)||_2^2, \quad (16)$$

where advantage function $A(s_t, a_t) = R_t + \gamma v_{\phi}(s_{t+1}) - v_{\phi}(s_t)$, π_{θ} is the policy network, v_{ϕ} is the value network, θ and ϕ are the parameters of policy network and value network, separately, and γ is the discount factor (we set it to 0.95). We train the actor and critic with the ADAM optimizer and the learning rate of 10^{-3} . The decision-making process and update process of the A2C algorithm are shown in Algorithms C.2 and C.3 at the Online Supplemental Materials, respectively.

4.2. The architecture of Smartformer

In this section, we introduce the architecture of Smartformer. Specifically, we use a Transformer model with three Multi-head Attention layers as an example and present the sequential decomposition process of each layer from left to right in Figure 1. As seen, Layer 2 is first selected with the minimum layer importance score after training the model with several initial iterations/epochs. An encoder net is then utilized to map the structure of Layer 2 into the current state in the MDP. After that, the policy network takes the encoded state as the input to generate the action (rank R), which is then exploited to replace Layer 2 with the corresponding CPAC-Attention layer. The reward of this action will be generated according to either (13) or (14) after training another several iterations/epochs. Such a process will be repeated until the decomposition of Layer 2 is converged, which is evaluated by the pre-defined stopping criteria on consecutive actions. During this process, a reverse operation on the decomposed matrices/tensors is conducted before each decomposition so that the decomposition is implemented on the high-order matrices/tensors as usual. After Layer 2, a similar process will be repeated on the next unselected layer with the minimum importance score. Please refer to the Supplemental Online Materials for the calculation of the layer importance score (Molchanov *et al.*, 2019).

We summarize the proposed training framework, Smartformer, in Algorithm 1. Generally, for each layer decomposition, there are five key steps: (i) identifying the layer with minimum importance score; (ii) generating the state representation; (iii) generating the action (rank R in decomposition) and estimating the value of the current state; (iv) checking the convergence of consecutive actions; (v) updating parameters in the policy network and the value network.

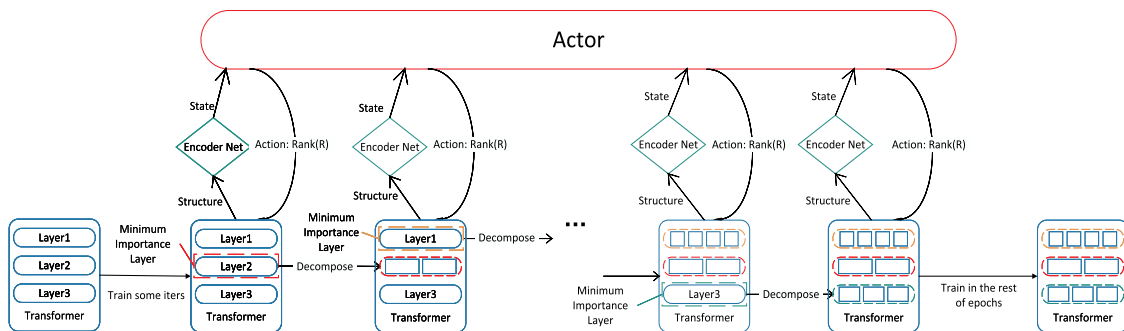


Figure 1. Overview of the architecture of Smartformer.

Algorithm 1 Intelligent model compression process of Smartformer**1: Inputs:**

Policy network $\pi_\theta(a|s)$; Value network $v_\phi(s)$
 Encoder Net $g_{enc}(\cdot)$

▷ Encode the selected layer to generate the state representation

2: Initialize:

θ, ϕ
 $\alpha^\theta, \alpha^\phi$

▷ Parameters in policy network and value network

▷ Step size in fitting parameters

$\mathcal{T}_{CPAC-Attention}$

▷ A Transformer using CPAC-Attention layers with CR = 1

γ

▷ discount factor

\mathcal{M}

▷ List of strings storing the structure of each layer in current model

\mathcal{D}_s

▷ The dictionary of layer structures

T

▷ The total epochs

\mathcal{E}

▷ Interval between two actions, either the number of batches or epochs

3: while T are not reached **do****4: while** not all layers are decomposed **do**

5: $t \leftarrow 1$

6: $s_t = g_{enc}(\mathcal{D}_s(\mathcal{M}(l^*)))$

▷ Generate the state representation using the selected layer

7: $acc_t, loss_t, \mathcal{I} \leftarrow \text{train } \mathcal{T}_{CPAC-Attention} \text{ using Algorithm C.1 for } \mathcal{E}$

8: $l_* = \text{argmin}_l \mathcal{I}$

▷ Get the index of the layer with minimum importance score

9: while the action does not converge **do**

10: $a_t \leftarrow \text{chosen action using Algorithm C.2}$

11: Update the l_*^{th} CPAC-Attention layer in $\mathcal{T}_{CPAC-Attention}$ using a_t

12: Update \mathcal{M} with the latest layer structure

13: $s_{t+1} = g_{enc}(\mathcal{D}_s(\mathcal{M}(l^*)))$

▷ Update state

14: $acc_{t+1}, loss_{t+1}, \mathcal{I} \leftarrow \text{train } \mathcal{T}_{CPAC-Attention} \text{ using Algorithm C.1 for } \mathcal{E}$

15: Calculate reward \mathcal{R}

▷ (13,14)

16: Update $\pi_\theta(a|s)$ and $v_\phi(s)$ using Algorithm C.3

17: $t \leftarrow t + 1$

18: **if** the consecutive actions satisfy the pre-defined stopping criterion **then**

19: the action converges

20: this layer will not be selected in the future training process.

21: **end if**

22: **end while**

23: **end while**

24: Continue to train the $\mathcal{T}_{CPAC-Attention}$ using Algorithm C.1.

25: **end while**

5. Experiment results

In this section, we conduct two practical experiments by applying the Smartformer to two backbone models and using their datasets for a fair investigation. For the baseline methods, we use three existing decomposition methods from Ma *et al.* (2019), Wang *et al.* (2020) and Cordonnier *et al.* (2021), which target to decompose the attention layers like the Smartformer. In addition, we replace CP-decomposition in Smartformer with SVD as another baseline method. SVD can be treated as the simpler version of CP-decomposition for matrices. Such a comparison is designed to demonstrate the advantage of using CP-decomposition in this study. Lastly, we utilize a weight-sharing method from Takase and Kiyono (2022) and apply it to the attention layers in Transformer as another benchmark. In summary, these five groups of baseline methods are denoted as Lin-X (Wang *et al.*, 2020), Collab-X (Cordonnier *et al.*, 2021), BT-D-X (Ma *et al.*, 2019), SVD-Smartformer, and Pshared-X (Takase and Kiyono, 2022), respectively, which are also applied to the

two selected backbone models to complete the comparison. For the Lin-X model, we follow Wang *et al.* (2020) and set k as 64, 128, and 256, respectively, where k quantifies the level of approximation used for the self-attention mechanism as a low-rank matrix. For the BT-D-X model, we follow the setting from Ma *et al.* (2019) and assume that the number of cores (# cores) is the same as the number of attention heads, which is one or two. We also assume the order of the core tensor to be the same as the dimension of attention heads, i.e., $n_1 = n_2 = n_3 = d$, where $d = 2$ in case 1 and $d = 64$ in cases 2. For Collab-X and Pshared-X models, we directly follow their settings in Cordonnier *et al.* (2021) and Takase and Kiyono (2022) for model compression.

The idea of Smartformer is to optimize rank R in the training process of Transformer by introducing DRL. Literature works (Lebedev *et al.*, 2015; Wang *et al.*, 2022) usually determine the rank R evenly to observe the effects of different rank values. Following this logic, we establish the action space of DRL with evenly distributed rank R values.

Specifically, if the number of rank values are less than 10, we will use all the rank values as the action candidates. Otherwise, we will use the 10 compression ratios (ranging from 0.1 to 1) to determine the rank values, which construct the action space. After that, we show the results of two practical experiments as well as the comparison analysis of computational properties among our Smartformer and the selected baseline methods. Lastly, we conduct a further validation experiment to support our motivation that uses DRL to select the rank R in the practical applications. For all the practical experiments, if the DRL agent generates the same action (rank value for decomposition) in several consecutive steps, we regard this as a sign of action stabilization and we will go to the next layer for decomposition. In this way, the learning experience from previously decomposed layers will be well accumulated and benefit the decomposition of the following layers. Combined with the fact that the more important layers have a lower priority in decomposition, our proposed method can make sure a better decision will be made for decomposing more important layers. Note that we use ϵ -greedy policy that follows the “exploration and exploitation” paradigm, to train the DRL agent. The experiments are implemented in PyTorch (Paszke *et al.*, 2019) on a single NVIDIA A40 GPU.

5.1. Practical experiments

5.1.1. Case study 1

In the first case, we work on the S3T model (Song *et al.*, 2021) using the 2a dataset of BCI competition IV (Brunner *et al.* 2008). As introduced in Song *et al.* (2021), this model contains four parts: preprocessing and spatial filter part, spatial transforming part, temporal transforming part, and classifier part. The middle two parts use the self-attention mechanism to extract the spatial and temporal features within the electroencephalogram (EEG) data. In the S3T model, there are three encoders, and each has one Multi-head Attention layer. To train S3T, Song *et al.* (2021) employed the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.0002, the cross-entropy loss function, and the GeLU activation function (Hendrycks and Gimpel, 2016). For other specific parameters, please refer to more details in Song *et al.* (2021). The 2a dataset of BCI competition IV contains the EEG data of nine subjects with four different tasks, namely, the imagination of moving the left hand, right hand, both feet, and tongue. For each subject,

two sessions of data are collected, and each of them contains 288 trials (72 trials for each task).

In the experiment, we apply the Smartformer to intelligently decompose and train the S3T model and repeat the process on nine subjects, separately. We follow the rank determination rule introduced above and set the action space (rank R) of the DRL agent as $\{1, 2, 3, 4, 5, 6\}$. In addition, we set the loss tolerance threshold ϵ as 0.1 (see (13) about this hyperparameter). The DRL agent takes an action every 50 epochs, and there are 2000 epochs in total for the experiment of each subject. Table 2 summarizes the results from both the proposed framework and other benchmark models. As observed, Smartformer achieves the best average accuracy among all the models with 86.67, which, to the best of our knowledge, has been the best accuracy on this dataset so far. Moreover, for the specific examination on the nine subjects, Smartformer outperforms the original model (S3T) in eight out of nine. If we keep the framework of Smartformer and change the CP-decomposition into SVD, the average accuracy is slightly lower but still outperforms all the other baseline methods. The improvement in model performance with Smartformer compared with the original S3T, indicates that the original model might be overfitted, and the reduction in model parameters mitigates this issue. In addition, when comparing our proposed Smartformer with other model compression methods (e.g., Lin-S3T and Pshared-S3T models), we can clearly see the advantage of including maintaining the performance as an objective. Particularly, the existing benchmark models hardly achieve better performance than the original models in each subject.

Table 3. The experiment results of Case 2 (the ones in each column that are better than the accuracy of the original model, are highlighted with boldface).

Model	96		192		336	
	MSE	MAE	MSE	MAE	MSE	MAE
Autoformer	0.266	0.336	0.307	0.367	0.359	0.395
Lin-Autoformer($k = 64$)	0.246	0.324	0.337	0.381	0.409	0.424
Lin-Autoformer($k = 128$)	0.237	0.318	0.329	0.397	0.466	0.468
Lin-Autoformer($k = 256$)	0.270	0.350	0.390	0.446	0.594	0.563
Collab-Autoformer	0.338	0.416	0.429	0.475	0.481	0.501
BTD-Autoformer(# cores = 1)	0.312	0.390	0.280	0.334	0.325	0.359
BTD-Autoformer(# cores = 2)	0.225	0.295	0.281	0.336	0.333	0.369
Pshared-Autoformer	0.265	0.332	0.318	0.369	0.365	0.397
SVD-Smartformer	0.285	0.332	0.311	0.370	0.344	0.382
Smartformer	0.236	0.307	0.286	0.345	0.346	0.381

Note: MSE is short for mean squared error, and MAE is short for mean absolute error.

Table 2. The experiment results of Case 1 (the ones in each column that are better than the accuracy of the original model, are highlighted with boldface).

Model	S01	S02	S03	S04	S05	S06	S07	S08	S09	Ave Acc	STD
S3T	91.67	71.67	95.00	78.33	61.67	66.67	96.67	93.33	88.33	82.59	12.52
Lin-S3T($k = 64$)	71.67	50.00	81.67	55.00	41.67	40.00	76.67	75.00	71.67	62.59	15.07
Lin-S3T($k = 128$)	75.00	36.67	73.33	43.33	36.67	41.67	63.33	73.33	85.00	58.70	17.96
Lin-S3T($k = 256$)	73.33	46.67	71.67	45.00	51.67	40.00	61.67	68.33	78.33	59.63	13.30
Collab-S3T	90.00	68.33	88.33	68.33	63.33	65.00	88.33	93.33	81.67	78.52	11.42
Pshared S3T	83.33	65.00	95.00	60.00	45.00	58.33	93.33	86.67	85.00	74.63	16.81
BTD-S3T(# cores = 1)	88.33	71.67	96.67	75.00	56.67	75.00	90.00	93.33	86.67	81.48	12.08
BTD-S3T(# cores = 2)	83.33	68.33	93.33	68.33	56.67	61.67	96.67	93.33	85.00	78.52	14.15
SVD-Smartformer	96.67	75.00	96.67	76.67	65.00	73.33	100.00	98.33	96.67	86.48	12.90
Smartformer	96.67	80.00	96.67	80.00	66.67	70.00	100.00	93.33	96.67	86.67	11.97

Note: Ave Acc is short for average accuracy, and STD is short for standard deviation.

5.1.2. Case study 2

In the second case, we work on the Autoformer. Autoformer renovates Transformer into a decomposition architecture, including the inner series decomposition block, auto-correlation mechanism, and corresponding encoder and decoder for long-term time-series forecasting. Wu *et al.* (2021) used five datasets to showcase the performance of the proposed model. For this experiment, we randomly select the Weather dataset, which records meteorological indicators (e.g., air temperature, humidity, etc.) every 10 minutes for the entire year 2020. For this case, we exploit three different prediction horizons, $O \in \{96, 192, 336\}$, and split the Weather dataset into the training, validation, and testing set in chronological order by the ratio of 7:1:2. For other parameters settings such as learning rate and loss function, please refer to Wu *et al.* (2021) for detailed information. It should be noted that in this case, Wu *et al.* (2021) set the training process to be early stopped within 10 epochs. Therefore, the DRL agent takes an action every 100 batches within each epoch. The action space contains 10 actions with rank values equal to 45, 90, 135, ..., 450. The architecture of Autoformer consists of two encoders and one decoder. Each of them contains one Multi-head Attention layer, and the parameter size is with the shape of 512×512 . We set the loss tolerance threshold ϵ to 1.125 (see (14) about this hyperparameter). As demonstrated in Wu *et al.* (2021), Autoformer has achieved the state-of-the-art performance w.r.t. MSE and MAE. Hence, we also use MSE and MAE as evaluation metrics and apply Smartformer as well as the selected five baseline models to compressing the Autoformer. The experiment results are demonstrated in Table 3.

As seen, when the forecasting horizon is 96, BTDAutoformer (# cores = 2) achieves the lowest MSE (0.225) and MAE (0.295); when the forecasting horizon is 192, BTDAutoformer (# cores = 1) achieves the lowest MSE (0.280) and MAE (0.334); when the forecasting horizon is 336, BTDAutoformer (# cores = 1) also achieves the lowest MSE (0.325) and MAE (0.359). Such a superior performance of the BTDAutoformer is because this method re-constructs the attention mechanism, and the number of parameters is related to the sequence length of the input tensor. Particularly, the $Attention(Q, K, V)$ in BTDAutoformer is a tensor of Seq_Len^3 (Seq_Len means the sequence length of Q , K and V). This results in the output layer of the multi-head

attention layer containing a large number of parameters that map the obtained tensor. As a result, the BTDAutoformer has a large number of parameters, which enhances to the model performance. It should be noted that both BTDAutoformer (#cores = 2) and Smartformer achieves the better performance than Autoformer in all three scenarios. As to the baseline models, some of them can also improve the forecasting accuracies (e.g., Lin-Autoformer ($k = 64, 128$) in the “96” scenario, BTDAutoformer (# cores = 2) in the “96, 192, 336” scenario). However, some of them have much worse performance than Autoformer and are not actually appropriate for this dataset (e.g., Collab-Autoformer in the “96, 192” scenarios and Lin-Autoformer in the 336 scenario). In terms of SVD-Smartformer, it does not have a competitive performance since it results in a worse MSE value than Autoformer in “96” scenario and worse MSE and MAE values in “192” scenario.

On the other hand, we also visualize the decision-making process of Smartformer in Appendix E at the Online Supplemental Materials.

5.2. Computational property analysis

In this section, we mainly investigate the complexity of all models in the above two cases since one of our three objectives in this study is to reduce the number of parameters. As seen in Table 4, when comparing with the backbone model, the benchmark models have a varied performance from the perspective of the number of parameters. Pshared-S3T model has the least amount of parameters in all scenarios. Some models have significantly more parameters than their backbone models. For example, Lin-X models in EEG scenario have a much larger amount of parameters than the others. The reason is that for the Lin-X method, the main idea is to add two linear projection matrices, $E_i, F_i \in \mathbb{R}^{N \times k}$, for the calculation of key and value components in Transformer, which introduces extra parameters. The number of parameters in E_i and F_i is jointly determined by N (the length of sequence) and k (the level of approximation). Considering three self-attention layers ($i = 1, 2, 3$), E_i and F_i introduce $190 \times 64 \times 2 \times 3 = 72,960$ parameters ($N = 190$, $k = 64$), which contributes most of the parameters out of 81.65k (see Lin-S3T ($k = 64$) model in EEG(S3T) of Table 4). As to the BTDAutoformer, the output layer has the parameter matrix of $W^O \in \mathbb{R}^{N \times N \times d_{model}}$, which will contain a large

Table 4. The number of parameters of Cases 1 and 2 (the ones in each column that are better than the accuracy of the original model, are highlighted with boldface).

Model	Scenarios (Model)			
	EEG (S3T)	Weather 96 (Autoformer)	Weather 192 (Autoformer)	Weather 336 (Autoformer)
Backbone	8.69 k	10.61 M	10.61 M	10.61 M
Lin-Backbone($k = 64$)	81.65 k	10.66 M	10.68 M	10.69 M
Lin-Backbone($k = 128$)	154.61 k	10.72 M	10.74 M	10.78 M
Lin-Backbone($k = 256$)	300.53 k	10.83 M	10.88 M	10.95 M
Collab-Backbone	8.93 k	10.64 M	10.64 M	10.64 M
Pshared-Backbone	7.36 k	7.46 M	7.46 M	7.46 M
BTDAutoformer(# cores = 1)	1.09 M	31.57 M	50.45 M	96.45 M
BTDAutoformer(# cores = 2)	1.09 M	31.97 M	50.84 M	96.85 M
SVD-Smartformer	8.62 k	10.6 M	10.6 M	10.6 M
Smartformer	8.24 k	10.37 M	10.37 M	10.53 M

Table 5. The FLOPs of Cases 1 and 2.

Model	Scenarios (Model)			
	EEG (S3T)	Weather 96 (Autoformer)	Weather 192 (Autoformer)	Weather 336 (Autoformer)
Backbone	6.17 M	1.20 G	1.56 G	2.10 G
Lin-Backbone($k = 64$)	5.47 M	1.23 G	1.59 G	2.14 G
Lin-Backbone($k = 128$)	6.93 M	1.25 G	1.63 G	2.19 G
Lin-Backbone($k = 256$)	9.85 M	1.31 G	1.70 G	2.27 G
Collab-Backbone	10.54 M	1.44 G	2.02 G	3.03 G
Pshared-Backbone	6.17 M	1.20 G	1.56 G	2.10 G
BTD-Backbone(# cores = 1)	250.85 M	4.27 G	11.24 G	36.99 G
BTD-Backbone(# cores = 2)	292.15 M	4.70 G	12.44 G	41.04 G
SVD-Smartformer	6.13 M	1.20 G	1.56 G	2.10 G
Smartformer	6.10 M	1.43 G	1.86 G	2.54 G

Table 6. The running time (s)(epochs) of Cases 1 and 2.

Model	Scenarios (Model)			
	EEG (S3T)	Weather 96 (Autoformer)	Weather 192 (Autoformer)	Weather 336 (Autoformer)
Backbone	6046.13	113.22(2)	368.01(5)	496.94(5)
Lin-Backbone($k = 64$)	5665.67	117.79(2)	304.50(4)	399.93(4)
Lin-Backbone($k = 128$)	5862.02	121.14(2)	378.25(5)	608.46(6)
Lin-Backbone($k = 256$)	6266.37	119.35(2)	309.65(4)	408.56(4)
Collab-Backbone	6123.22	79.90(2)	450.15(9)	272.76(4)
Pshared-Backbone	5899.19	116.46(2)	513.40(7)	593.05(6)
BTD-Backbone(# cores = 1)	22173.39	126.38(2)	1255.49(10)	1782.33(4)
BTD-Backbone(# cores = 2)	33091.08	160.41(2)	1483.23(9)	5752.61(10)
SVD-Smartformer	8256.21	221.51(2)	670.23(7)	809.81(6)
Smartformer	8114.09	1323.36(2)	2716.13(5)	3734.89(7)

number of parameters when the length of the sequence is large. Even though in Table 3, BTD models show outstanding performance, the number of parameters in three Weather scenarios determines that this model is not appropriate for time-series modeling. Some other benchmark models have slightly more parameters than their backbone models (e.g., Collab-S3T and Lin-CCT). Nonetheless, both the SVD-Smartformer and Smartformer always have a smaller number of parameters than the corresponding backbones in all scenarios. Such results indicate that some of the existing model compression methods fail to reduce the number of parameters in all scenarios consistently. In fact, a few of them might even significantly increase the model parameters in Transformer. Among all the methods, we can summarize that the P-shared, SVD-Smartformer, and proposed Smartformer are widely applicable to compress the model parameters in all scenarios.

In addition, we also investigate the FLOPs and running time, which are summarized in Tables 5 and 6, respectively. We have the following findings. For FLOPs comparison, a few models (e.g., Lin-S3T ($k = 64$), SVD-Smartformer, and Smartformer) have smaller FLOPs than the backbones in the EEG case. Note that most of the models have larger FLOPs than the backbone models since the number of parameters are not linear with the number of FLOPs (see Table 1 for CP-decomposition). Regarding running time, Autoformer has the early stop setting, so we include the number of epochs in Table 6. As seen, Smartformer usually has a longer running time than the others (e.g., Backbone, Lin-X, Collab-X, and Pshared-X). The reason is that the training process of Smartformer includes the optimization process on rank R , while the hyperparameters in other methods have to be pre-determined. If the benchmark methods also need to optimize the selection of hyperparameters, the grid search

needs to be conducted, which will be much more computationally expensive compared with the proposed Smartformer. For example, in the EEG scenario, we have six ranks for the S3T model, and the running time for a predefined rank ($R = 2$) is 6142.40 seconds, which will take around $6^3 \times 6142.40$ seconds to go through all the combinations of rank R for three layers. On the contrary, Smartformer only takes 8509.01 seconds for the entire running. As to the weather case, there are 10 action candidates; therefore, the computational time will increase 1000 times if going through all the combinations of rank values. As to SVD-Smartformer, in the EEG scenario, the computational time is a little higher than that of Smartformer, while in the three weather scenarios, they are lower. The reason could be that SVD-Smartformer can more easily identify the appropriate rank R for each layer based on the action stabilization criteria in the three weather scenarios. Even though both SVD-Smartformer and Smartformer utilize the DRL to determine the rank R for each layer, the computational time comparison between SVD-Smartformer and Smartformer is scenario-dependent and such a result cannot show any advantages of each of them.

To better show the results from the two practical experiments, we conduct a comprehensive analysis regarding Tables 2 to 4 since we boldface the values that are better than those of the original models in these three tables. The results demonstrate that our proposed Smartformer is the only model that can satisfy the three objectives simultaneously in all scenarios. Specifically, our Smartformer is the only one that can improve the model performance and simultaneously reduce the model parameters in all scenarios, which indicates the better generalizability and robustness of the Smartformer than the other baseline models. On the other hand, the results from the BTD method also show

Table 7. The results of using different ranks on the EEG case (we highlight the best value in each column with boldface).

Model	S01	S02	S03	S04	S05	S06	S07	S08	S09	Ave Acc	STD	Params
S3T	91.67	71.67	95.00	78.33	61.67	66.67	96.67	93.33	88.33	82.59	12.52	8.69 k
CP-S3T($R = 1$)	90.00	61.67	95.00	71.67	60.00	68.33	93.33	93.33	88.33	80.19	13.71	7.86 k
CP-S3T($R = 2$)	93.33	61.67	91.67	71.67	55.00	68.33	91.67	93.33	88.33	79.44	14.38	8.01 k
CP-S3T($R = 3$)	90.00	65.00	91.67	63.33	56.67	68.33	93.33	95.00	90.00	79.26	14.60	8.16 k
CP-S3T($R = 4$)	90.00	61.67	91.67	66.67	55.00	71.67	91.67	95.00	88.33	79.07	14.40	8.32 k
CP-S3T($R = 5$)	91.67	63.33	91.67	68.33	55.00	68.33	91.67	90.00	86.67	78.52	13.78	8.47 k
CP-S3T($R = 6$)	88.33	63.33	91.67	66.67	58.33	68.33	90.00	95.00	88.33	78.89	13.54	8.62 k
Smartformer	96.67	80.00	96.67	80.00	66.67	70.00	100.00	93.33	96.67	86.67	11.97	8.24 k

Table 8. The results of using different ranks on the weather case (we highlight the best value in each column with boldface).

Model	96			192			336		
	MSE	MAE	Params	MSE	MAE	Params	MSE	MAE	Params
Autoformer	0.266	0.336	10.61 M	0.307	0.367	10.61 M	0.359	0.395	10.61 M
CP-Autoformer($R = 45$)	0.247	0.319	7.77 M	0.312	0.369	7.77 M	0.361	0.393	7.77 M
CP-Autoformer($R = 90$)	0.235	0.307	8.09 M	0.317	0.371	8.09 M	0.363	0.394	8.09 M
CP-Autoformer($R = 135$)	0.239	0.309	8.40 M	0.305	0.363	8.40 M	0.359	0.390	8.40 M
CP-Autoformer($R = 180$)	0.252	0.321	8.72 M	0.312	0.369	8.72 M	0.354	0.389	8.72 M
CP-Autoformer($R = 225$)	0.267	0.332	9.03 M	0.301	0.356	9.03 M	0.343	0.382	9.03 M
CP-Autoformer($R = 270$)	0.251	0.319	9.35 M	0.294	0.351	9.35 M	0.348	0.385	9.35 M
CP-Autoformer($R = 315$)	0.258	0.328	9.66 M	0.301	0.360	9.66 M	0.350	0.384	9.66 M
CP-Autoformer($R = 360$)	0.237	0.308	9.98 M	0.297	0.355	9.98 M	0.350	0.384	9.98 M
CP-Autoformer($R = 405$)	0.258	0.330	10.29 M	0.302	0.362	10.29 M	0.361	0.392	10.29 M
CP-Autoformer($R = 450$)	0.244	0.312	10.61 M	0.296	0.355	10.61 M	0.353	0.385	10.61 M
Smartformer	0.236	0.307	10.37 M	0.286	0.345	10.37 M	0.346	0.381	10.53 M

Table 9. Sensitivity analysis of the number of epochs in each stage (action interval) in the EEG case.

Model	S01	S02	S03	S04	S05	S06	S07	S08	S09	Ave Acc	STD
Smartformer(action interval = 50)	96.67	80.00	96.67	80.00	66.67	70.00	100.00	93.33	96.67	86.67	11.97
Smartformer(action interval = 100)	88.33	70.00	93.33	71.67	61.67	60.00	96.67	93.33	95.00	81.11	14.23
Smartformer(action interval = 200)	90.00	76.67	93.33	76.67	58.33	70.00	91.67	96.67	95.00	83.15	11.91
Smartformer(action interval = 300)	91.67	73.33	98.33	78.33	50.00	71.67	95.00	93.33	88.33	82.22	14.59

some advantages in the three weather scenarios, while it does not show some competitive performance in the EEG case. Therefore, in real applications, if practitioners have a high requirement for the model accuracy and do not consider the computational cost, the BTD method could be considered as an option to examine whether it can further improve the model performance.

5.3. Support of motivation

In this section, we further conduct an experiment to demonstrate the necessity of proposing Smartformer by comparing the performance of using DRL in Smartformer versus using the grid search in the regular training process. Specifically, we implement the CP-decomposition on the backbone models for the above two cases with predefined values of ranks. According to the action space setting, for Case 1, there are six rank values in total, whereas for Case 2, there are 10 rank values.

We present the results in Tables 7 and 8. As seen in Table 7, all the six CP-decomposition models with $R = 1, 2, \dots, 6$ have the worse average accuracy than the original S3T model. This indicates that assigning the rank values with grid search cannot guarantee the performance of the resulting model. As to our Smartformer, it generates much better results than the original S3T model. Observing Table 8, Smartformer achieves the best MAE in “96” scenario, best MSE and MAE values in the “192” and “336” scenarios. Note

Table 10. Sensitivity analysis of the number of epochs in each stage (action interval) in the weather case.

Model	96		192		336	
	MSE	MAE	MSE	MAE	MSE	MAE
Smartformer(action interval = 50)	0.248	0.316	0.301	0.361	0.353	0.386
Smartformer(action interval = 100)	0.236	0.307	0.286	0.345	0.346	0.381
Smartformer(action interval = 200)	0.246	0.313	0.291	0.340	0.353	0.387
Smartformer(action interval = 300)	0.260	0.326	0.294	0.353	0.355	0.388

that the MSE value of Smartformer in the “96” scenario is just slightly worse than the best one. Such results clearly demonstrate the superior performance of our Smartformer in decomposing the attention layers within a dynamic way. Moreover, these results also provide strong support to the motivation of proposing Smartformer for the practical applications since the grid-search-based method is not only time-consuming but also cannot always perform well.

5.4. Sensitivity analysis

In this section, we conduct a sensitivity analysis of the number of epochs in each stage (action interval) to investigate how this factor influences the results. For these two cases, we use the four action intervals, namely, 50, 100, 200, 300. We present the results in Tables 9 and 10. As seen, when the action interval is equal to 50, it leads to a better overall performance than the other three in the EEG case. When the action interval is equal to 100, it generates a consistently

better performance than the other three intervals in the three weather scenarios.

6. Conclusion

Although Transformer has achieved success in time-series modeling, it is also limited by the over-parameterization issue. Existing works have focused on addressing this issue in several ways, but none of them have taken the three objectives, i.e., preserving the model architecture, maintaining the model performance, and reducing the model parameters, simultaneously into account. Considering this, we propose the Smartformer, an intelligent model compression framework for Transformer by incorporating DRL and CP-decomposition.

To demonstrate the effectiveness of the proposed Smartformer, we conduct two practical experiments to validate the effectiveness of the Smartformer framework. More specifically, we re-train two existing variants of Transformer, i.e., S3T and Autoformer, with Smartformer to find the best compression solution that can satisfy the three objectives, respectively. On the other hand, we select five existing model compression methods for Transformer as benchmarks. Based on the experiment results, we can conclude that our proposed Smartformer has a better generalized and robust performance in satisfying the three objectives on various scenarios. More importantly, our Smartformer is also the only one that can improve the model performance and simultaneously reduce the model parameters in all scenarios. Moreover, the Smartformer is friendly and convenient for applications, such that researchers can design their Transformer models as usual and train the model with the original hyperparameter settings. Lastly, this intelligent decomposition idea also has a broader impact on compressing other types of DNN.

In the future, we will consider the proposed online-DRL optimization algorithm to other methods, such as weight sharing, knowledge distillation, quantization, and pruning after modeling the key hyperparameter optimization problems of them as MDPs. In addition, some studies could take into account the drawbacks of the existing methods and compare them with the same datasets for their performance exploration.

Funding

Xiaojuan Wang, Jin Yang, and Dr. Ying Chen were supported by the National Natural Science Foundation of China (Grant No. 72121001, 72101066, 72131005). Dr. Ying Chen was also supported by Heilongjiang Natural Science Excellent Youth Fund (YQ2022G004).

Notes on contributors




Xiaojuan Wang received a BS degree in management from Harbin Institute of Technology, Harbin, China, in 2022. Currently, he is pursuing a MS degree with the School of Management at Xi'an Jiaotong University, Xi'an, China. His research interests include deep learning and reinforcement learning for combinatorial optimization problems.

Dr. Yinan Wang is an assistant professor in the Department of Industrial and Systems Engineering at Rensselaer Polytechnic Institute. He received a BS degree in electrical engineering and automation from Xi'an Jiaotong University, a MS in electrical engineering from Columbia University, and a PhD in industrial and systems engineering from Virginia Tech. His research interests include data analytics and machine learning techniques in quality control of advanced manufacturing systems. He is the recipient of FTC Early Career Award, 10 Best Paper/Poster/Featured Article Awards, and two Best PhD Dissertation Awards. He was selected as the Mary and Joseph Natrella Scholar from the American Statistical Association (ASA), Scialog Fellow from the Research Corporation for Science Advancement (RCSA), and the Panel Fellow in the NSF CMMI Game Changer Academy. He was co-organizer for the Symposium and Workshop in Manufacturing Science and Engineering Conference (MSEC) 2023 and SIAM International Conference on Data Mining (SDM) 2023 and 2024. He serves as an active Board Director in the IIE Data Analytics and Information Systems (DAIS) division and program director in the Automatic Controls and Robotics Division (ACARD) of the International Society of Automation (ISA).

Jin Yang received BS and MS degrees from Harbin Institute of Technology in China. Currently, he is a PhD candidate at Harbin Institute of Technology. His research interests include deep learning, simulation, and decision-making under uncertainties. He has published papers in *INFORMS Journal on Computing and Fundamental Research*.

Dr. Ying Chen is an associate professor in the School of Management at Harbin Institute of Technology, Harbin, China. He received his PhD degree in the department of industrial engineering from the University of Texas at Arlington. His research interests include deep learning, reinforcement learning, and optimization. His works have been published in journals, including *INFORMS Journal on Computing*, *IIE Transactions*, *European Journal of Operational Research*, and *Annals of Operations Research*.

ORCID

Yinan Wang  <http://orcid.org/0000-0002-4079-1658>
Jin Yang  <http://orcid.org/0000-0003-2741-138X>
Ying Chen  <http://orcid.org/0000-0002-0366-131X>

Data availability statement

The authors confirm that the data supporting the findings of this study are available in the [supplementary materials](#).

References

- Brunner, C., Leeb, R., Müller-Putz, G. and Schlögl, A. (2008) BCI competition 2008-Graz data set A, page p.6.
- Cai, H., Chen, T., Zhang, W., Yu, Y. and Wang, J. (2018) Efficient architecture search by network transformation, in *The Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI Press, New Orleans, LA, pp. 2787–2794.
- Castronovo, M., Maes, F., Fonteneau, R. and Ernst, D. (2013) Learning exploration/exploitation strategies for single trajectory reinforcement learning, in *European Workshop on Reinforcement Learning*, PMLR Press, Edinburgh, Scotland, pp. 1–10.
- Chen, P., Yu, H.-F., Dhillon, I. and Hsieh, C.-J. (2021) Drone: Data-aware low-rank compression for large NLP models. *Advances in Neural Information Processing Systems*, **34**, 29321–29334.
- Cordonnier, J.-B., Loukas, A. and Jaggi, M. (2021) Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*.
- Ding, T., Li, D. and Sun, R. (2022) Suboptimal local minima exist for wide neural networks with smooth activations. *Mathematics of Operations Research*, **47**(4), 2784–2814.

- Gu, J., Keller, B., Kossaifi, J., Anandkumar, A., Khailany, B. and Pan, D.Z. (2022) Heat: Hardware-efficient automatic tensor decomposition for transformer compression. *arXiv preprint arXiv:2211.16749*.
- Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., et al. (2022) A survey on vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1), 87–110.
- Hendrycks, D. and Gimpel, K. (2016) Gaussian error linear units (GELUs). *Preprint arXiv:1606.08415*.
- Kiers, H.A. (2000) Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics*, 14(3), 105–122.
- Kingma, D.P. and Ba, J. (2014) Adam: A method for stochastic optimization. *Preprint arXiv:1412.6980*.
- Konda, V.R. and Tsitsiklis, J.N. (1999) Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12, 1008–1014.
- Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I. and Lempitsky, V. (2015) Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *International Conference on Learning Representation*, Conference Track Proceedings, San Diego, CA.
- Li, J., Sun, Y., Su, J., Suzuki, T. and Huang, F. (2020) Understanding generalization in deep learning via tensor methods, in *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 108, PMLR Press, Palermo, Italy, pp. 504–515.
- Li, X., Shi, Q., Hu, G., Chen, L., Mao, H., Yang, Y., Yuan, M., Zeng, J. and Cheng, Z. (2021) Block access pattern discovery via compressed full tensor Transformer, in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, Association for Computing Machinery, New York, NY, pp. 957–966.
- Liu, J., Zang, H., Cheng, L., Ding, T., Wei, Z. and Sun, G. (2023) A transformer-based multimodal-learning framework using sky images for ultra-short-term solar irradiance forecasting. *Applied Energy*, 342, 121160.
- Ma, R., Wang, C. and Li, X. (2021) CP decomposition for fast training of Bi-LSTM, in *2021 IEEE International Conference on Dependable, Automatic and Secure Computing*, IEEE Press, Piscataway, NJ, pp. 25–28.
- Ma, X., Zhang, P., Zhang, S., Duan, N., Hou, Y., Zhou, M. and Song, D. (2019) A tensorized Transformer for language modeling. *Advances in Neural Information Processing Systems*, 32, 2232–2242.
- Maclaurin, D., Duvenaud, D., Johnson, M. and Adams, R.P. (2015) *Autograd: Reverse-mode differentiation of native Python*. ICML workshop on Automatic Machine Learning.
- Magnus, J.R. and Neudecker, H. (1985) Matrix differential calculus with applications to simple, Hadamard, and Kronecker products. *Journal of Mathematical Psychology*, 29(4), 474–492.
- Molchanov, P., Mallya, A., Tyree, S., Frosio, I. and Kautz, J. (2019) Importance estimation for neural network pruning, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, CVF, IEEE Xplore, Long Beach, CA, pp. 11256–11264.
- Neyshabur, B., Bhojanapalli, S., McAllester, D. and Srebro, N. (2017) Exploring generalization in deep learning. *Advances in Neural Information Processing Systems*, 30, 5949–5958.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019) Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 8026–8037.
- Qu, K., Si, G., Shan, Z., Kong, X. and Yang, X. (2022) Short-term forecasting for multiple wind farms based on transformer model. *Energy Reports*, 8, 483–490.
- Song, Y., Jia, X., Yang, L. and Xie, L. (2021) Transformer-based spatial-temporal feature learning for eeg decoding. *arXiv preprint arXiv:2106.11170*.
- Sutton, R.S. and Barto, A.G. (2018) *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- Takase, S. and Kiyono, S. (2022) Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.
- Tucker, L.R. (1966) Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3), 279–311.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. (2017) Attention is all you need. *Advances in Neural Information Processing Systems*, 30.
- Wang, D., Wu, B., Zhao, G., Yao, M., Chen, H., Deng, L., Yan, T. and Li, G. (2021) Kronecker CP decomposition with fast multiplication for compressing RNNs. *IEEE Transactions on Neural Networks and Learning Systems*, 34(5), 2205–2219.
- Wang, S., Li, B.Z., Khabsa, M., Fang, H. and Ma, H. (2020) Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.
- Wang, Y., Guo, W.G. and Yue, X. (2022) Tensor decomposition to compress convolutional layers in deep learning. *IJSE Transactions*, 54(5), 481–495.
- Wang, Y. and Zou, S. (2021) Online robust reinforcement learning with model uncertainty. *Advances in Neural Information Processing Systems*, 34, 7193–7206.
- Wu, H., Xu, J., Wang, J. and Long, M. (2021) Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34, 22419–22430.
- Wu, N., Green, B., Ben, X. and O’Banion, S. (2020) Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*.
- Xie, J., Zhang, J., Sun, J., Ma, Z., Qin, L., Li, G., Zhou, H. and Zhan, Y. (2022) A transformer-based approach combining deep learning network and spatial-temporal information for raw EEG classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 30, 2126–2136.
- Xu, C. and McAuley, J. (2022) A survey on model compression for natural language processing. *arXiv preprint arXiv:2202.07105*.
- Zheng, P., Zhou, H., Liu, J. and Nakanishi, Y. (2023) Interpretable building energy consumption forecasting using spectral clustering algorithm and temporal fusion transformers architecture. *Applied Energy*, 349, 121607.