
Adaptive Sparsity Level during Training for Efficient Time Series Forecasting with Transformers

Zahra Atashgahi¹, Mykola Pechenizkiy², Raymond Veldhuis¹, Decebal Constantin Mocanu^{3,1,2}

¹Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente

²Department of Mathematics and Computer Science, Eindhoven University of Technology

³Department of Computer Science, University of Luxembourg

{z.atashgahi, r.n.j.veldhuis}@utwente.nl

m.pechenizkiy@tue.nl, decebal.mocanu@uni.lu

Abstract

Efficient time series forecasting has become critical for real-world applications, particularly with deep neural networks (DNNs). Efficiency in DNNs can be achieved through sparse connectivity and reducing the model size. However, finding the sparsity level automatically during training remains a challenging task due to the heterogeneity in the loss-sparsity tradeoffs across the datasets. In this paper, we propose “**Pruning with Adaptive Sparsity Level**” (**PALS**), to automatically seek an optimal balance between loss and sparsity, all without the need for a predefined sparsity level. PALS draws inspiration from both sparse training and during-training methods. It introduces the novel “expand” mechanism in training sparse neural networks, allowing the model to dynamically shrink, expand, or remain stable to find a proper sparsity level. In this paper, we focus on achieving efficiency in transformers known for their excellent time series forecasting performance but high computational cost. Nevertheless, PALS can be applied directly to any DNN. In the scope of these arguments, we demonstrate its effectiveness also on the DLinear model. Experimental results on six benchmark datasets and five state-of-the-art transformer variants show that PALS substantially reduces model size while maintaining comparable performance to the dense model. More interestingly, PALS even outperforms the dense model, in 12 and 14 cases out of 30 cases in terms of MSE and MAE loss, respectively, while reducing 65% parameter count and 63% FLOPs on average. Our code will be publicly available upon acceptance of the paper.

1 Introduction

The capabilities of transformers [52] for learning long-range dependencies [56, 9, 49] make them an ideal model for time series processing [54]. Several transformer variants have been proposed for the task of time series forecasting, which is crucial for real-world applications, e.g., weather forecasting, energy management, and financial analysis, and have proven to significantly increase the prediction capacity in long time series forecasting (LTSF) [35]. In addition, attention-based models are inherently an approach for increasing the interpretability for time series analysis in critical applications [29]. Moreover, recent transformer time series forecasting models (e.g., [58, 67, 35]) perform generally well in other time series analysis tasks, including, classification, anomaly detection, and imputation [59].

Despite the outstanding performance of transformers, these models are known to be computationally expensive due to their large model sizes [48]. The high training and inference costs of transformers can make their deployment cumbersome in real-world applications with limited memory and computational resources and impose adverse effects on the environment. Particularly, with the ever-increasing collection of large time series and the need to forecast millions of time series, the

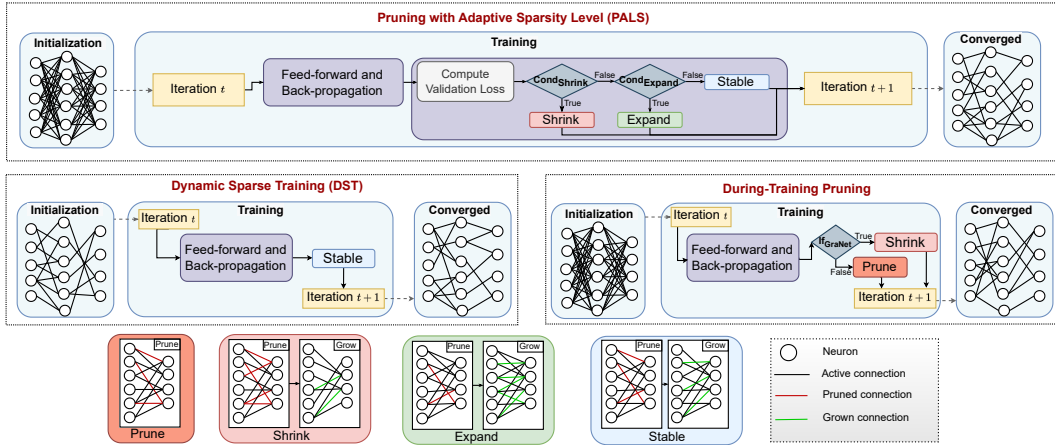


Figure 1: Schematic overview of the proposed method, **PALS** (Algorithm 1), Dynamic Sparse Training (DST) [39, 10], During-training pruning (Gradual Magnitude Pruning (GMP) [68], and GraNet [31]). While DST and during-training pruning use a fixed sparsity schedule to achieve a pre-determined sparsity level at the end of the training, PALS updates the sparse connectivity of the network at each Δt iterations during training, by deciding whether to "Shrink" (decrease density) or "Expand" (increase density) the network or remain "Stable" (same density), to automatically find a proper sparsity level.

requirement to develop computationally-efficient forecasting models is becoming significantly critical [50, 19, 44]. For industry-scale time series data, which are often high-dimensional and long-length, deploying transformers requires automatically discovering memory- and computationally-efficient architectures that are scalable and practical for real-world applications [54]. While there have been some efforts to reduce the computational complexity of transformers in time series forecasting [67, 66], these models have in order of millions of parameters. The over-parameterization of these networks causes high training and inference costs, and their deployment in low-resource environments (e.g., lack of GPUs) would be infeasible. To address these issues, we raise the research question: *How can we reduce the computational and memory overheads of training and deploying transformers for time series forecasting without compromising the model performance?*

Seeking sparsity through sparse connectivity is a widely-used technique to address the over-parameterization of deep learning models [17]. Early approaches for deriving a sparse sub-network prune a trained dense model [16], known as *post-training* pruning. While these methods can match the performance of the dense network as shown by the Lottery Ticket Hypothesis (LTH) [12], they are computationally expensive during training due to the training of the dense network. *During-training* pruning aims to maintain training efficiency by gradually pruning a dense network during training [31]. Sparse training [39] pushed the limits further by starting with a sparse network from scratch and optimizing the topology during training. However, as we study in Section 3, the main challenge when using any of these techniques for time series forecasting is to find the proper sparsity level automatically.

In this paper, we aim to move beyond optimizing a single objective (e.g. minimizing loss) and investigate sparsity in DNNs for time series prediction in order to find a good trade-off between computational efficiency and performance automatically. Our contributions are as follows:

- We analyze the effect of sparsity (using unstructured pruning) in the performance of state-of-the-art transformers for time series prediction [35, 67, 58, 66], and vanilla transformer [52]. We show they can be pruned up to 80% of their connections in most cases, without significant loss in performance.
- We propose an algorithm, called “**Pruning with Adaptive Sparsity Level**” (**PALS**) that finds a decent loss-sparsity trade-off by dynamically tuning the sparsity level during training using the loss heuristics and deciding at each connectivity update step whether to *Shrink* or *Expand* the network, or keep it *Stable*. PALS creates a bridge between during-training pruning and dynamic sparse training research areas by inheriting and enhancing some of their most successful mechanisms, while - up to our best knowledge - introducing for the first time into play also the *Expand* mechanism. Consequently, PALS does not require a desired pre-defined sparsity level which is necessary for most pruning or sparse training algorithms. Figure 1 presents the general concept behind PALS.

- We evaluate the performance of PALS in terms of the loss, the parameter count, and FLOPs on six widely-used benchmarks for time series prediction and show that PALS can substantially sparsify the models and reduce parameter count and FLOPs. Surprisingly, PALS can even outperform the dense model on average, in 12 and 14 cases out of 30 cases in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE) loss, respectively (Table 2).

2 Background

2.1 Sparse Neural Networks

Sparse neural networks (SNNs) use sparse connectivity among layers to reduce the computational complexity of DNNs while maintaining a close performance to the dense counterpart in terms of prediction accuracy. SNNs can be achieved using dense-to-sparse or sparse-to-sparse approaches [40].

Dense-to-sparse methods prune a dense network; based on the pruning phase, they are categorized into three classes: *post-training* [16, 12], *Before-training* [25], and *during-training* [68, 36, 31] pruning. Post-training pruning suffers from high computational costs during training and before-training approaches usually fall behind the performance of the dense counter-part network. In contrast, during-training approaches, maintain close or even better performance to the dense network while being efficient through the training process. A standard during-training pruning is Gradual Magnitude Pruning (GMP) [68] which gradually drops unimportant weights based on the magnitude during the training process. GraNet [31] is another during-training algorithm that gradually shrinks (decreasing density) a network to reach a pre-determined sparsity level. It prunes the weights (as performed in GMP) while allowing for connection regeneration (as seen in Dynamic Sparse Training (DST) which will be explained in the following). As the number of grown weights is less than the pruned ones, the network is shrunk and the density is decreased. For more details regarding GraNet, please refer to Appendix B.

Sparse-to-sparse methods start with a random sparse network from scratch and the number of parameters is usually fixed during training and can be determined based on the available computational budget. The sparse topology can remain fixed (static) [38] or dynamically optimized during training (a.k.a Dynamic Sparse Training (DST)) [39, 10, 33, 20, 1, 61, 47]. At each topology update iteration, a fraction of unimportant weights are dropped (usually based on magnitude), and the same number of weights are grown. The growth criteria can be random, as in the Sparse Evolutionary Training (SET) algorithm [39], or based on the gradient, as in the Rigged Lottery (RigL) algorithm [10].

In this work, we take advantage of the successful mechanism of “*Shrink*” from during-training pruning (such as GraNet [31]) and “*Stable*” from DST (such as RigL [10]) and propose for the first time the “*Expand*” mechanism, to design a method to automatically optimize the sparsity level during training without requiring to determine it beforehand. Each of these mechanisms is explained in Section 4. In Table 1, we present a summarized comparison with the closest related work in the literature. Figure 1 presents a comprehensive embedding of our proposed method in the literature. Unlike these methods, which update the network using fixed schedules to reach a pre-determined sparsity level, PALS proposes an adaptive approach. It automatically determines whether to shrink or expand the network or remain stable, in order to tune the sparsity level and find a good trade-off between loss and sparsity.

Table 1: Comparison of related work.

Method	Shrink	Stable	Expand	Adaptive Sparsity Schedule	Automatically tuning sparsity level
RigL [10]	×	✓	×	×	×
GMP [68]	✓	×	×	×	×
GraNet [31]	✓	✓	×	×	×
PALS (ours)	✓	✓	✓	✓	✓

Only a few works investigated SNNs for time series analysis [46]. [60] investigates sparsity in convolutional neural networks (CNNs) for the time series classification and shows their proposed method has superior prediction accuracy while reducing computational costs. [22] exploit sparse recurrent neural networks (RNNs) for outlier detection. [32] and [13] explore sparsity in RNNs for sequence learning.

Sparsity in Transformers. Several works have sought sparsity in transformers [15, 42]. These approaches can be categorized into structured (blocked) [37] or unstructured (fine-grained) pruning [5]. As discussed in [17], structured sparsity for transformers is able to only discover models with very low sparsity levels; therefore, we focus on unstructured pruning. [28] analyses pruning transformers for language modeling tasks and shows that large transformers are robust to compression. [6] dynamically extract and train sparse sub-networks from Vision Transformers (ViT) [9] while maintaining a fixed small parameter budget, and they could even improve the accuracy of the ViT in some cases. [8] investigates DST for BERT language modeling tasks and shows Pareto improvement

over the dense model in terms of FLOPs. However, these works mostly focus on vision and NLP tasks. To the best of our knowledge, there is no work that has investigated sparse connectivity in transformers for time series analysis that faces domain-specific challenges as we will elaborate in Section 3. Please note that there is a line of research focusing on *sparse attention* [51] aiming to develop an efficient self-attention mechanism that is orthogonal to our focus in this work (sparsity and pruning) [17].

2.2 Time Series Forecasting

Initial studies for time series forecasting [18] exploit classical tools such as ARIMA [3]. While traditional methods mostly rely on domain expertise or assume temporal dependencies follow specific patterns, machine learning techniques learn the temporal dependencies in a data-driven manner [29, 53, 27]. In recent years, various deep learning models, including RNNs [55, 43, 45], multi-layer perceptrons (MLP) [62, 63], CNNs [24], and Temporal convolution networks (TCN) [11] are utilized to perform time series forecasting [14, 41, 4, 21].

Transformers have been extensively used to perform time series forecasting due to their strong ability for sequence modeling. A class of models aims at improving the self-attention mechanism and addresses the computational complexity of vanilla transformers such as LogTrans [26], Informer [66], Reformer [23]. Another category of methods seeks to modify the model to capture the inherent properties of the time series: Autoformer [58] introduces a seasonal trend decomposition with an auto-correlation block as the attention module. NSTransformer [35] proposes to add two modules including series stationarization and de-stationary attention in the transformer architecture. FEDformer [67] proposes to combine transformers with a seasonal-trend decomposition method to capture global and detailed behaviour of the time series. The research into designing transformers for time series forecasting is ongoing, and many other transformer variants have been proposed, such as TDformer [64], Crossformer [65], ETSformer [57], Pyraformer [34].

2.3 Problem Formulation and Notations

Let $\mathbf{x}_t \in \mathbb{R}^m$ denote the observation of a multivariate time series \mathbf{X} with m variables at time step t . Given a look-back window $\mathbf{X}_{t-L:t} = [\mathbf{x}_{t-L}, \dots, \mathbf{x}_{t-1}]$ of size L , time series forecasting task aims to predict time series over a horizon H as $\tilde{\mathbf{X}}_{t:t+H} = [\tilde{\mathbf{x}}_t, \dots, \tilde{\mathbf{x}}_{t+H-1}]$ where $\tilde{\mathbf{x}}_t$ is the prediction at time step t . To achieve this, we need to train a forecasting function $f(\mathbf{X}_{t-L:t}, \theta)$ (e.g. a transformer network) that can predict future values over horizon H .

In this paper, we aim to reduce the model size by pruning the unimportant parameters from θ such that we find the sparse model $f(\mathbf{X}_{t-L:t}, \theta_s)$ where $\|\theta_s\|_0 \ll \|\theta\|_0$. $D = \frac{\|\theta_s\|_0}{\|\theta\|_0}$ is called the density level of the model f and $S = 1 - D$ is called as the sparsity level. The aim is to minimize the reconstruction loss between the prediction $\mathcal{L}(f(\mathbf{X}_{t-L:t}, \theta_s), \mathbf{X}_{t:t+H})$ while finding a proper sparsity level S automatically. We use Mean Squared Error (MSE) as the loss function such that:

$$\mathcal{L}(\tilde{\mathbf{X}}_{t:t+H}, \mathbf{X}_{t:t+H}) = \frac{1}{H} \sum_{i=0}^{H-1} (\tilde{\mathbf{x}}_{t+i} - \mathbf{x}_{t+i})^2. \quad (1)$$

3 Analyzing Sparsity Effect in Transformers for Time Series Forecasting

In this section, we explore sparsity in several time series forecasting transformers. In short, we apply GraNet [31] to prune each model and measure their performance over various sparsity levels.

Experimental Settings. We perform this experiment on six benchmark datasets, presented in Table 4. We have adapted GraNet [31], a well-performing during-training pruning algorithm developed for CNNs, to sparsify transformer models for time series forecasting. GraNet gradually shrinks a network (here, we start from a dense network) during the training to reach a pre-determined sparsity level, while allowing for connection regeneration inspired by DST. GraNet algorithm is described in Appendix B. For more details regarding the experimental settings, please refer to Section 5.1. For each sparsity level (%) in $\{25, 50, 65, 80, 90, 95\}$, we measure the prediction performance of each transformer model in terms of MSE loss. The results for prediction length = 96 (except 24 for the Illness dataset) are presented in Figure 2. The results for other prediction lengths are presented in Figure 3 in Appendix B.

Sparsity Effect. We present the results for pruning various transformers in Figure 2. It can be observed that most models can be pruned up to 80% or higher sparsity levels without significantly affecting performance. Moreover, a counter-intuitive observation is that in some cases, sparsity does

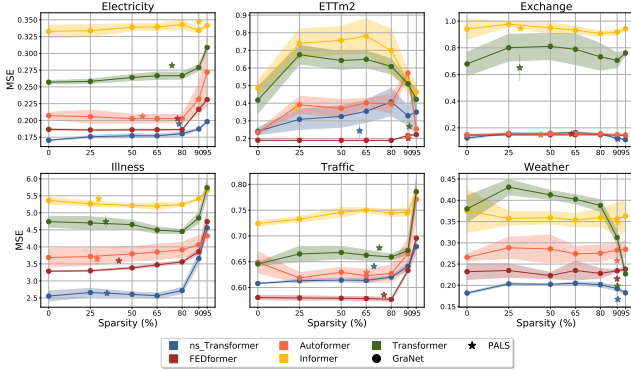


Figure 2: Sparsity effect on the performance of various transformer models for time series forecasting on benchmark datasets in terms of MSE loss (prediction length = 96, except 24 for the Illness dataset). Each model is sparsified using GraNet [31] to sparsity levels (%) $\in \{25, 50, 65, 80, 90, 95\}$ and PALS. $Sparsity = 0$ indicates the original dense model.

Algorithm 1 PALS

- 1: **Input:** Time series $X \in \mathbb{R}^T \times m$, number of training iterations t_{max} , Sequence length L , Prediction length H , model dimension d_{model} , pruning rate ζ , mask update frequency Δt , Initial density D_{init} , pruning rate factor $\gamma > 1$ and loss freedom factor $\lambda > 1$, sparsity bound S_{min} and S_{max} .
- 2: **Initialization:** Initialize the transformer model with density level D_{init} , $S = 1 - D_{init}$, $L_{best} = \text{inf}$.
- 3: **Training:**
- 4: **for** $t \in \{1, \dots, \#t_{max}\}$ **do**
- 5: **I. Standard feed-forward and back-propagation.** The network is trained on $batch_t$ of samples.
- 6: **II. Update sparsity mask**
- 7: **if** $(t \bmod \Delta t) = 0$ **then**
- 8: Compute Validation Loss L_{valid}^t
- 9: **if** $(S < S_{min})$ **or** $(L_{valid}^t \leq \lambda * L_{best}$ **and** $S < S_{max})$ **then**
- 10: **update_mask** ($\zeta_{prune} = \gamma * \zeta$, $\zeta_{grow} = \zeta$)
- 11: **else if** $L_{valid}^t > \lambda * L_{best}$ **and** $S > S_{best}$ **then**
- 12: **update_mask** ($\zeta_{prune} = \zeta$, $\zeta_{grow} = \gamma * \zeta$)
- 13: **else**
- 14: **update_mask** ($\zeta_{prune} = \zeta$, $\zeta_{grow} = \zeta$)
- 15: **end if**
- 16: **if** $L_{valid}^t < L_{best}$ **then**
- 17: $L_{best} = L_{valid}^t$, $S_{best} = S$
- 18: **end if**
- 19: Set S to the sparsity level of the network.
- 20: **end if**
- 21: **end for**

not necessarily lead to worse performance than the dense counterpart, and it can even improve the performance. For example, while on the Electricity, Illness, and Traffic datasets, the behavior is as usually expected (higher sparsity leads to lower performance), on the three other datasets, higher sparsity might even lead to better performance (lower loss) than the dense model. In addition, the sparsity effect is different among various models, particularly on the latter group of datasets, including the ETTm2, Exchange, and Weather datasets. We will discuss the potential reasons for the different behavior among datasets in Appendix H. Last but not least, by looking to Figure 3 in Appendix B, it can be observed that the prediction length can also be a contributing factor to the sparsity-loss trade-off.

Challenge. Based on the above observations, we can conclude that the sparsity effect is not homogeneous across various time series datasets, forecasting models, and prediction lengths for time series forecasting. Our findings in these experiments are not aligned with the statements in [17] for CNNs (vision) and Transformers (NLP), where for a given task and technique, increasing the sparsity level results in decreasing the prediction performance and they have a direct relationship with each other. However, we observe in Figures 2 and 3 that increasing the sparsity level does not necessarily lead to decreased performance and it might even significantly improve the performance (e.g. for the vanilla transformer on the Weather dataset). Therefore, it is challenging to decide how much we can push the sparsity level and what is the optimal sparsity level without having prior knowledge of the time series data, model, and experimental settings. While GraNet is the closest in spirit to our proposed method, it cannot automatically tune the sparsity level since it needs the initial and the final sparsity level as its hyperparameters. In this paper, we aim to address this challenge by proposing an algorithm that can automatically tune the sparsity level during training.

4 Proposed Methodology: PALS

This section presents our proposed method for automatically finding a proper sparsity level of a DNN for time series prediction, called “Pruning with Adaptive Sparsity Level” (PALS) (Algorithm 1). While our main focus in this paper is to sparsify transformer models, PALS is not specifically designed for transformers and can be applied directly to other artificial neural network architectures (See Appendix F for experiments on training with PALS the DLinear [62]) model.

Motivation and Broad Outline. As we discussed in Section 3, the main challenge when seeking sparsity for time series forecasting is to find the optimal sparsity level automatically. Therefore, PALS aims to tune the sparsity level during training without requiring prior information about models or datasets. PALS is in essence inspired by the DST framework [39] and gradual magnitude pruning (GMP) [68, 31]. While DST and GMP use fixed sparsification policies (fixed sparsity level (*Stable* in Figure 1) and constantly prune the network until the desired sparsity level is reached (*Shrink* in Figure 1), respectively) and require the final sparsity level before training, PALS exploits heuristic information from the network at each iteration to automatically determine whether to *increase*,

decrease, or *keep* the sparsity level at each connectivity update step. To the best of our knowledge, this is the first work that allows the network to *expand* by increasing the density during training. If the training starts from a dense neural network ($D_{init} = 1$) PALS can be seen as a dense-to-sparse method, while if $D_{init} < 1$ then PALS is a sparse-to-sparse method.

Training. The training of PALS (Algorithm 1) starts with initializing a network with density level $D_{init} = 1 - S_{init}$. Then, the training procedure of PALS consists of two steps:

1. Standard feed-forward and back-propagation. Network’s parameters are updated each training iteration t using a batch of samples.

2. Update Sparse Connectivity. The novelty of the method lies in updating the sparse connectivity. At every Δt iteration, the connectivity is updated in two steps. (2-1) The validation loss at step t is calculated as L_{valid}^t . (2-2) The sparsity mask is updated (*update_mask* in Algorithm 1) by first pruning ζ_{prune} of weights with the lowest magnitude:

$$\widetilde{\mathbf{W}}_l = \text{Update}(\mathbf{W}_l, \text{top}(|\mathbf{W}_l|, 1 - \zeta_{prune})), \quad (2)$$

where \mathbf{W}_l is the l^{th} weight matrix of the network, $\text{Update}(\mathbf{A}, idx)$ keeps only the indices idx of the matrix \mathbf{A} , $\text{top}(\mathbf{A}, \zeta)$ returns the indices of a fraction ζ of the largest elements of \mathbf{A} . Then, we grow ζ_{grow} of the weights with the highest gradients:

$$\mathbf{W}_l = \widetilde{\mathbf{W}}_l + \text{top}(|\mathbf{G}_{l, i \notin \widetilde{\mathbf{W}}_l}|, \zeta_{grow}) \quad (3)$$

where $\mathbf{G}_{l, i \notin \widetilde{\mathbf{W}}_l}$ is the gradient of zero weights in layer l . These new connections are initialized with zero values. This process is repeated for each layer in the model. Based on the values of ζ_{prune} and ζ_{grow} , PALS determines whether to decrease (*shrink*), increase (*expand*), or keep (*stable*) the network:

$\mathbf{S}_t > \mathbf{S}_{t-1}$ (**Shrink**). If the loss does not go beyond $\lambda * L_{best}$, we decrease the overall number of parameters such that $\zeta_{prune} = \gamma * \zeta$, $\zeta_{grow} = \zeta$. The loss freedom coefficient, $\lambda > 1$, is a hyperparameter of the network that determines how much the loss value can deviate from the best validation loss achieved so far L_{best} during training. The lower λ is, the more strict PALS becomes at allowing the network to go to the *shrink* phase, finally resulting in a lower sparsity network. $\gamma > 1$ is the pruning factor coefficient, which determines how much to prune or grow more in the shrink and expand phases, respectively. We analyze the sensitivity of PALS to λ and γ in Section 6.2. In addition, we define a boundary for sparsity determined by S_{min} and S_{max} which can be determined by the user based on the available resources. If the sparsity level does not meet the minimum sparsity level S_{min} , we prune the network more than we grow. If the network sparsity goes beyond S_{max} , we do not increase sparsity.

$\mathbf{S}_t < \mathbf{S}_{t-1}$ (**Expand**). If $S > S_{best}$ (S_{best} is the sparsity level corresponding to L_{best}) and the loss goes higher than $\lambda * L_{best}$, it means that the earlier pruning step(s) were not beneficial to decreasing the loss (improving forecasting quality in the time series forecasting) and the network requires a higher capacity to recover a good performance. Therefore, we expand the network and grow more connections than the pruned ones at this step: $\zeta_{prune} = \zeta$, $\zeta_{grow} = \gamma * \zeta$.

$\mathbf{S}_t = \mathbf{S}_{t-1}$ (**Stable**). If none of the above cases happened, we only update a fraction ζ of the network’s parameters without changing the sparsity level: $\zeta_{prune} = \zeta$, $\zeta_{grow} = \zeta$.

For a better understanding of how the sparsity level evolves during the training process of PALS, please refer to Appendix G.3.

5 Experiments and Results

In this section, we evaluate PALS on several transformers for time series forecasting.

5.1 Experimental Settings

Datasets. The experiments are performed on six widely-used benchmark datasets for time series forecasting. The datasets are summarized in Table 4 and described in Appendix A. These datasets have different characteristics including stationary and non-stationary with/without obvious periodicity.

Models. We have considered five state-of-the-art transformer models for time series forecasting, including Non-Stationary Transformer (NSTransformer) [35], FEDformer [67], Autoformer [58], Informer, and Transformer [52]. Please refer to Section 2.2 for more details.

Table 2: Summary of the results on the benchmark Datasets in Table 5. For each experiment on a transformer model and dataset, the average MSE, MAE, number of parameters ($\times 10^6$) and the inference FLOPs count ($\times 10^{12}$) for various prediction lengths are reported before and after applying PALS. The difference between these results is shown in % where the blue color means improvement of PALS compared to the corresponding dense model.

Model	Electricity				ETTM2				Exchange				Illness				Traffic				Weather			
	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs	MSE	MAE	#Params	#FLOPs
NSTransformer	0.19	0.30	12.0	9.25	0.49	0.43	10.6	19.82	0.54	0.49	10.6	1.89	2.14	0.92	10.5	0.05	0.63	0.34	14.2	6.61	0.29	0.31	10.7	18.10
+PALS	0.21	0.32	2.2	1.81	0.38	0.39	2.5	3.70	0.49	0.47	5.4	1.07	2.33	0.97	7.4	0.04	0.67	0.37	4.3	2.05	0.26	0.29	1.0	1.77
Difference	10.8% ↓	7.3% ↓	81.5% ↓	80.5% ↓	24.0% ↓	11.2% ↓	76.7% ↓	81.3% ↓	9.3% ↓	3.6% ↓	48.5% ↓	43.3% ↓	9.1% ↑	5.1% ↑	30.0% ↓	30.2% ↓	5.2% ↑	9.1% ↑	70.1% ↓	69.0% ↓	10.2% ↓	6.9% ↓	90.3% ↓	90.2% ↓
FEFormer	0.21	0.32	19.5	9.30	0.30	0.35	17.9	19.82	0.50	0.49	17.9	1.89	2.84	1.14	13.7	0.05	0.61	0.38	22.3	6.71	0.32	0.37	17.9	18.11
+PALS	0.23	0.34	3.0	1.35	0.30	0.25	1.8	1.96	0.51	0.50	10.5	1.15	3.05	1.19	8.3	0.03	0.52	0.38	5.6	1.83	0.31	0.36	1.8	1.81
Difference	9.0% ↑	4.9% ↑	84.7% ↓	85.5% ↓	1.5% ↓	0.5% ↓	90.2% ↓	90.1% ↓	2.1% ↑	1.1% ↑	41.2% ↓	38.9% ↓	7.2% ↑	5.0% ↑	39.5% ↓	39.6% ↓	1.0% ↑	1.1% ↑	74.6% ↓	72.7% ↓	2.8% ↓	3.2% ↓	90.0% ↓	90.0% ↓
Autoformer	0.24	0.34	12.1	9.30	0.33	0.37	10.5	19.82	0.58	0.53	10.5	1.89	3.08	1.18	10.5	0.05	0.64	0.40	14.9	6.71	0.34	0.38	10.6	18.11
+PALS	0.26	0.36	2.7	1.71	0.31	0.35	1.0	1.93	0.62	0.55	7.1	1.30	3.19	1.22	6.7	0.03	0.65	0.41	4.5	1.94	0.34	0.38	1.3	2.52
Difference	9.3% ↑	4.6% ↑	77.7% ↓	81.6% ↓	8.1% ↓	5.6% ↓	90.3% ↓	90.3% ↓	5.5% ↑	4.4% ↓	32.7% ↓	31.0% ↓	3.5% ↑	3.2% ↑	36.6% ↓	36.5% ↓	1.9% ↑	2.1% ↑	69.5% ↓	71.0% ↓	0.1% ↑	1.3% ↑	87.7% ↓	86.1% ↓
Informr	0.36	0.43	12.5	8.51	1.53	0.88	11.3	18.15	1.59	1.00	11.3	1.71	5.27	1.58	11.3	0.05	0.81	0.46	14.4	6.14	0.62	0.55	11.4	16.58
+PALS	0.42	0.48	1.4	0.94	1.39	0.83	5.3	8.45	1.53	0.98	8.6	1.33	5.23	1.57	7.4	0.03	0.94	0.53	2.3	1.19	0.69	0.56	4.3	8.35
Difference	18.9% ↑	11.3% ↑	88.6% ↓	88.9% ↓	9.0% ↓	5.8% ↓	53.4% ↓	53.4% ↓	3.9% ↓	1.0% ↓	24.2% ↓	22.4% ↓	0.8% ↓	1.0% ↓	34.9% ↓	34.8% ↓	15.5% ↑	15.3% ↑	83.9% ↓	80.6% ↓	12.1% ↑	2.0% ↑	61.9% ↓	49.7% ↓
Transformer	0.28	0.38	11.7	9.24	1.48	0.86	10.5	19.81	1.61	0.97	10.5	1.89	4.94	1.49	10.5	0.05	0.67	0.36	13.6	6.61	0.64	0.56	10.6	18.10
+PALS	0.31	0.40	2.5	2.24	1.08	0.75	3.2	8.17	1.41	0.91	6.6	1.16	4.91	1.48	7.7	0.04	0.69	0.38	3.8	1.86	0.32	0.38	1.0	1.76
Difference	10.5% ↑	5.7% ↑	78.3% ↓	75.8% ↓	26.7% ↓	13.3% ↓	69.9% ↓	58.8% ↓	12.5% ↓	6.1% ↓	37.5% ↓	38.4% ↓	0.7% ↓	0.9% ↓	27.2% ↓	27.3% ↓	3.3% ↑	5.4% ↑	71.7% ↓	71.8% ↓	49.6% ↓	32.5% ↓	90.2% ↓	90.3% ↓
Difference _{avg}	11.7% ↑	6.8% ↑	82.1% ↓	82.5% ↓	13.3% ↓	7.3% ↓	76.1% ↓	74.8% ↓	3.6% ↓	1.2% ↓	36.8% ↓	34.8% ↓	3.7% ↓	2.3% ↓	33.6% ↓	33.7% ↓	5.4% ↑	6.6% ↑	74.0% ↓	73.0% ↓	10.1% ↓	9.0% ↓	84.0% ↓	81.3% ↓

Evaluation metrics. We evaluate the methods in two aspects: 1) Quality of the prediction in terms of MSE and MAE, and 2) Computational complexity in terms of parameter count and FLOPs. We report the theoretical FLOPs to be independent of the used hardware, as it is done in the unstructured pruning literature [31, 10]. A lower value for these metrics indicates higher prediction quality and lower computational complexity, respectively. We measure the performance of each model for various prediction lengths $H \in \{96, 192, 336, 720\}$ (except $H \in \{24, 36, 48, 60\}$ for the Illness dataset).

Implementation. Experiments are implemented in PyTorch. The start of implementation is the NSTransformer¹ and GraNet². We repeat each experiment for three random seeds and report the average of the runs. In the experiments, D_{init} was set to 1, thus PALS can be used as a during-training pruning method. We have run the experiments on *Intel Xeon Platinum 8360Y CPU* and one *NVIDIA A100 GPU*. We will discuss the hyperparameters' settings in Appendix A.

5.2 Results

Multivariate Time Series Prediction. The results in terms of MSE and parameter count for the considered datasets and models are presented in Table 5 in Appendix C. As can be seen in this table, in most cases considered, PALS is able to decrease the model size by more than 50% without a significant increase in loss. More interestingly, in most cases on the ETTm2, Exchange, and Weather datasets PALS even achieves lower MSE than the dense counterpart model.

To summarize the results of Table 5 (Appendix C) and have a general overview of the performance of PALS on each model and dataset, we present the average MSE and MAE, and parameters count in addition to the difference between the dense and the sparse model using PALS (in percentage) in Table 2. Additionally, we include the inference FLOPs count (total FLOPs for all test samples). It can be observed that PALS even outperforms the dense model in 12 and 14 cases out of 30 cases in terms of MSE and MAE loss, respectively, while reducing 65% parameter count and 63% FLOPs on average.

Based on the experiments conducted in Section 3 and the description of datasets provided in Appendix A.1, we observed significant variations in the sparsity-loss trade-off across different datasets and models. The beauty of our proposed method consists in the fact that it does not have to consider any of these differences. We did not make any finetuning for PALS to account for these differences, and it does everything automatically. Of course, finetuning PALS per dataset and model specificity would improve its final performance, but it would reduce the generality of our proposed work and we prefer not to do it.

Univariate Time Series Prediction. The results of univariate prediction (using a single variable) on the ETTm2 and Exchange datasets are presented in Table 6 and summarized in Table 7 in Appendix D. In short, PALS outperforms the dense counterpart model on average, in 7 and 8 cases out of 12 cases in terms of MSE and MAE loss, respectively.

¹ https://github.com/thuml/Nonstationary_Transformers

² <https://github.com/VITA-Group/GraNet>

Table 3: Comparison with other during-training pruning methods (GMP, GraNet) and a sparse training method (RigL) when sparsifying NSTransformer. The results are average over four prediction lengths.

Dataset	PALS			GraNet ^[31]			RigL ^[10]			GMP ^[68]		
	loss	sparsity	epochs	loss	sparsity	epochs	loss	sparsity	epochs	loss	sparsity	epochs
Electricity	0.21	80.5%	8.83	0.20	31.2%	9.75	0.20	31.2%	9.12	0.20	47.5%	9.62
ETTm2	0.38	76.7%	4.58	0.60	95.0%	9.00	0.49	77.5%	4.33	0.60	56.2%	9.12
Exchange	0.49	48.5%	5.83	0.47	95.0%	9.42	0.44	90.0%	4.25	0.45	95.0%	9.50
Illness	2.33	30.0%	7.97	2.32	25.0%	9.58	2.37	25.0%	9.58	2.22	31.2%	9.92
Traffic	0.67	70.1%	8.83	0.64	41.2%	9.50	0.64	25.0%	8.17	0.64	50.0%	9.79
Weather	0.26	90.3%	7.00	0.28	95.0%	9.08	0.27	41.2%	4.08	0.29	95.0%	9.08
Average	0.72	66.0%	7.17	0.75	63.73%	9.38	0.79	48.3%	6.60	0.73	62.4%	9.47

* Optimized sparsity level (%) in {25, 65, 50, 80, 90, 95}. Therefore, GraNet, RigL, and GMP, each requires 6 runs to optimize the sparsity level while PALS needs only one run.

6 Discussion

In this section, we study the performance of PALS in comparison with other pruning and sparse training algorithms (6.1) and the hyperparameter sensitivity of PALS (6.2). Additionally in the Appendix, we analyze the performance of PALS in terms of model size effect (H), prediction quality by visualizing the predictions (I), pruning DLinear [62] (F), and finally we discuss the efficiency of PALS from various aspects (G).

6.1 Performance Comparison with Pruning and Sparse Training Algorithms

We compare PALS with a standard during-training pruning approach (GMP [68]), GraNet [31], and a well-known sparse training method (RigL [10]). These are the closest methods in the literature in terms of including gradual pruning and gradient-based weight regrowth.

While PALS derives a proper sparsity level automatically, other pruning approaches require the sparsity level as an input of the algorithm. Therefore, to compare PALS with existing pruning algorithms, the sparsity level should be optimized for them. We apply GraNet, RigL, and GMP to NSTransformer for prediction lengths of $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). For each of these methods (GraNet, RigL, and GMP), the sparsity level is optimized among values of $\{25, 50, 65, 80, 90, 95\}$. This means that for one run of PALS, we run the other methods 6 times. The model with the lowest validation loss is used to report the test loss. Table 3 summarizes the average loss, sparsity level, and training epochs (due to early stopping the algorithms might not require the full training) over different prediction lengths.

The closest competitor of PALS is GraNet. As shown in Table 3, for the Electricity dataset, PALS achieves a sparsity level of 80.5% with a loss of 0.21, while GraNet achieves a sparsity level of only 31.2% with a slightly lower loss of 0.20. Similarly, for the ETTm2 dataset, PALS achieves a sparsity level of 76.7% with a loss of 0.38, while GraNet achieves a higher sparsity level of 95.0% but with a much higher loss of 0.60. On the other datasets, they perform relatively close to each other.

By looking at the results of all methods in Table 3, PALS has the highest average sparsity value (66.0%) compared to GraNet (63.73%), RigL (48.3%), and GMP (62.4%). While RigL requires fewer training epochs (~ 6.6 epochs) compared to PALS (~ 7.2 epochs), it finds lower sparsity networks and has a higher average loss (RigL: 0.79 compared to PALS: 0.72). GraNet and GMP use fixed pruning schedules, and as a result, they need almost full training time (~ 9.5 epochs). We additionally compared the convergence speed of PALS with the dense model in Appendix G.2.

In short, PALS has the lowest average loss and highest sparsity values compared to the other algorithms, which suggests that PALS can be a good option for building efficient and accurate sparse neural networks for time series forecasting.

6.2 Hyperparameter Sensitivity

In this section, we discuss the sensitivity of PALS to its hyperparameters. These hyperparameters include pruning rate factor γ and loss freedom factor λ . We have changed their values in $\{1.05, 1.1, 1.2\}$ and measured the performance of PALS (with NSTransformer) in terms of MSE and parameter count on six benchmark datasets. The results are presented in Table 8 in Appendix E.

As shown in Table 8, PALS is not very sensitive to its hyperparameters and the results in each row are close in terms of loss in most cases considered. However, by increasing γ and λ PALS tends to find a sparser model. A small λ results in paying more attention to the loss value, while a large

value gives more freedom to PALS to explore a sparse sub-network that might sometimes result in a higher loss value. A small γ limits the amount of additional grow/prune in the expand/shrink phase, while a large γ gives more flexibility to the algorithm for exploring various sparsity levels. In short, a small value for each of these hyperparameters makes PALS more strict and allows for small changes in sparse connectivity, while a large value of them increases the exploration rate which potentially results in higher sparsity and/or reduced loss.

7 Conclusions

In this paper, we aim to decrease the computational and memory costs of training and deploying DNNs for time series forecasting by automatically finding a good trade-off between model size and prediction performance. Particularly, we focus on transformers while showing the generality of the model on an MLP-based model (Appendix F). We first showed that pruning networks for time series forecasting can be quite challenging in terms of determining the proper sparsity level for various datasets, prediction lengths, and models. Therefore, we proposed PALS, a novel method to obtain sparse neural networks, that exploits loss heuristics to automatically find the best trade-off between loss and sparsity in one round of training. PALS leverages the effective strategies of "Shrink" from during-training pruning (e.g., GraNet) and "Stable" from DST (e.g., SET, RigL). Additionally, we introduce a novel strategy called the "Expand" mechanism. The latter allows PALS to automatically optimize the sparsity level during training, eliminating the need for prior determination. Remarkably, PALS was able to outperform dense training in 12/14 cases out of 30 cases (5 transformer models, 6 datasets) in terms of MSE/MAE loss, while reducing 65% parameters count and 63% FLOPs on average. **Limitations and future work.** Due to the lack of proper hardware to support sparse matrices for on-GPU processing, PALS cannot currently take advantage of its theoretical training and inference speed-up and memory reduction in a real-world implementation. With the ever-increasing body of work on sparse neural networks, we hope that in the near future, the community paves the way to optimally train sparse neural networks on GPU (Please refer to Appendix G.4 for more details). An open direction to this research can be to start with a highly sparse neural network (as opposed to starting from a dense network used in PALS) and gradually expand the network to be even more efficient during training.

References

- [1] Zahra Atashgahi, Joost Pieterse, Shiwei Liu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. A brain-inspired algorithm for training highly sparse neural networks. *Machine Learning*, 111(12):4411–4452, 2022.
- [2] Zahra Atashgahi, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders. *Machine Learning*, pages 1–38, 2022.
- [3] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [4] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza, Max Mergenthaler, and Artur Dubrawski. N-hits: Neural hierarchical interpolation for time series forecasting. *arXiv preprint arXiv:2201.12886*, 2022.
- [5] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15834–15846. Curran Associates, Inc., 2020.
- [6] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. *Advances in Neural Information Processing Systems*, 34:19974–19988, 2021.
- [7] Selima Curci, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Truly sparse neural networks at scale. *arXiv preprint arXiv:2102.01732*, 2021.
- [8] Anastasia S. D. Dietrich, Frithjof Gressmann, Douglas Orr, Ivan Chelombiev, Daniel Justus, and Carlo Luschi. Towards structured dynamic sparse pre-training of BERT, 2022.
- [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [10] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.
- [11] Jean-Yves Franceschi, Aymeric Dieuleveut, and Martin Jaggi. Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems*, 32, 2019.
- [12] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.
- [13] Takashi Furuya, Kazuma Suetake, Koichi Taniguchi, Hiroyuki Kusumoto, Ryuji Saiin, and Tomohiro Daimon. Spectral pruning for recurrent neural networks. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3458–3482. PMLR, 28–30 Mar 2022.
- [14] John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*, 2017.
- [15] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Hassan Sajjad, Preslav Nakov, Deming Chen, and Marianne Winslett. Compressing large-scale transformer-based models: A case study on bert. *Transactions of the Association for Computational Linguistics*, 9: 1061–1080, 2021.
- [16] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

- [17] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.
- [18] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.
- [19] Rob J Hyndman, Alan J Lee, and Earo Wang. Fast computation of reconciled forecasts for hierarchical and grouped time series. *Computational statistics & data analysis*, 97:16–32, 2016.
- [20] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33: 20744–20754, 2020.
- [21] Xiaoyong Jin, Youngsuk Park, Danielle Maddix, Hao Wang, and Yuyang Wang. Domain adaptation for time series forecasting via attention sharing. In *International Conference on Machine Learning*, pages 10280–10297. PMLR, 2022.
- [22] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI*, pages 2725–2732, 2019.
- [23] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rkgNKkHtvB>.
- [24] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, 2018.
- [25] Namhoon Lee, Thalaisyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=B1VZqjAcYX>.
- [26] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyu Zhou, Wenhui Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.
- [27] Yan Li, Xinjiang Lu, Yaqing Wang, and Dejing Dou. Generative time series forecasting with diffusion, denoise, and disentanglement. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [28] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5958–5968. PMLR, 13–18 Jul 2020.
- [29] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.
- [30] Shiwei Liu and Zhangyang Wang. Ten lessons we have learned in the new "sparseland": A short handbook for sparse neural network researchers. *arXiv preprint arXiv:2302.02596*, 2023.
- [31] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. *Advances in Neural Information Processing Systems*, 34:9908–9922, 2021.
- [32] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In *International Conference on Machine Learning*, pages 6893–6904. PMLR, 2021.

- [33] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *International Conference on Machine Learning*, pages 6989–7000. PMLR, 2021.
- [34] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2021.
- [35] Yong Liu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Non-stationary transformers: Rethinking the stationarity in time series forecasting. *arXiv preprint arXiv:2205.14415*, 2022.
- [36] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l₀ regularization. In *International Conference on Learning Representations*, 2018.
- [37] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.
- [38] Decebal Constantin Mocanu, Elena Mocanu, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. A topological insight into restricted boltzmann machines. *Machine Learning*, 104(2):243–270, 2016.
- [39] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.
- [40] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H Nguyen, Madeleine Gibescu, Damien Ernst, and Zita Vale. Sparse training theory for scalable and efficient agents. In *20th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2021*, 2021.
- [41] Boris N Oreshkin, Dmitri Carпов, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2019.
- [42] Sai Prasanna, Anna Rogers, and Anna Rumshisky. When bert plays the lottery, all tickets are winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.
- [43] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2627–2633, 2017.
- [44] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270, 2012.
- [45] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3): 1181–1191, 2020.
- [46] Georg Stefan Schlake, Jan David Hüwel, Fabian Berns, and Christian Beecks. Evaluating the lottery ticket hypothesis to sparsify neural networks for time series classification. In *2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW)*, pages 70–73, 2022. doi: 10.1109/ICDEW55742.2022.00015.
- [47] Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. *arXiv preprint arXiv:2106.04217*, 2021.
- [48] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

- [49] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi, and Jianyuan Zhong. Attention is all you need in speech separation. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 21–25. IEEE, 2021.
- [50] Thiyanga S Talagala, Rob J Hyndman, George Athanasopoulos, et al. Meta-learning how to forecast time series. *Monash Econometrics and Business Statistics Working Papers*, 6(18):16, 2018.
- [51] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.
- [52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [53] Zhiyuan Wang, Xovee Xu, Weifeng Zhang, Goce Trajcevski, Ting Zhong, and Fan Zhou. Learning latent seasonal-trend representations for time series forecasting. In *Advances in Neural Information Processing Systems*, 2022.
- [54] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. *arXiv preprint arXiv:2202.07125*, 2022.
- [55] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.
- [56] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
- [57] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven C. H. Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting. 2022. URL <https://arxiv.org/abs/2202.01381>.
- [58] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.
- [59] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. *arXiv preprint arXiv:2210.02186*, 2022.
- [60] Qiao Xiao, Boqian Wu, Yu Zhang, Shiwei Liu, Mykola Pechenizkiy, Elena Mocanu, and Decebal Constantin Mocanu. Dynamic sparse network for time series classification: Learning what to see”. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [61] Geng Yuan, Xiaolong Ma, Wei Niu, Zhengang Li, Zhenglun Kong, Ning Liu, Yifan Gong, Zheng Zhan, Chaoyang He, Qing Jin, et al. Mest: Accurate and fast memory-economic sparse training framework on the edge. *Advances in Neural Information Processing Systems*, 34: 20838–20850, 2021.
- [62] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *arXiv preprint arXiv:2205.13504*, 2022.
- [63] Tianping Zhang, Yizhuo Zhang, Wei Cao, Jiang Bian, Xiaohan Yi, Shun Zheng, and Jian Li. Less is more: Fast multivariate time series forecasting with light sampling-oriented mlp structures. *arXiv preprint arXiv:2207.01186*, 2022.
- [64] Xiyuan Zhang, Xiaoyong Jin, Karthick Gopalswamy, Gaurav Gupta, Youngsuk Park, Xingjian Shi, Hao Wang, Danielle C Maddix, and Bernie Wang. First de-trend then attend: Rethinking attention for time-series forecasting. In *NeurIPS’22 Workshop on All Things Attention: Bridging Different Perspectives on Attention*, 2022.

- [65] Yunhao Zhang and Junchi Yan. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=vSVLM2j9eie>.
- [66] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.
- [67] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. *arXiv preprint arXiv:2201.12740*, 2022.
- [68] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

A Experimental Settings

A.1 Datasets

The datasets are summarized in Table 4. 1) *Electricity*³ dataset includes the hourly electricity consumption for 321 consumers between 2012 and 2014. 2) *ETT* [66] (Electricity Transformer Temperature) dataset contains load and oil temperature measurements from electricity transformers. 3) *Exchange* [24] dataset consists of daily exchange rates from 8 countries between 1990 and 2016. 4) *Illness*⁴ dataset includes weekly collected data from influenza-like illness patients between 2002 and 2021 reported by Centers for Disease Control and Prevention of the United States. 5) *Traffic*⁵ dataset contains road occupancy rates in San Francisco Bay area freeways. 6) *Weather*⁶ consists of measurements of 21 weather indicators collected every 10 minutes in 2020. All datasets are divided in chronological order into the train, validation, and test sets with a split ratio of 7:2:2 (except for the ETT dataset where we use 6:2:2 split ratio).

Table 4: Datasets.

Dataset	# Variables	Sampling Frequency	# Observations
Electricity	321	1 Hour	26304
ETTm2	7	15 Minutes	69680
Exchange	8	1 Day	7588
Illness	7	1 Week	966
Traffic	862	1 Hour	17544
Weather	21	10 Minutes	52695

Datasets in Table 4 have different characteristics. Such characteristics include: 1) *Sampling frequency*: while the sampling frequency for some time series datasets is very high (e.g., Electricity (1 hour) and Weather datasets (10 minutes)), it might be very low for some others (Exchange (1 day) and Illness (1 week)) 2) *Periodicity of the variables*: time series datasets can be periodic (ETTm2) or without obvious periodicity (Exchange) 3) *Number of variables*: the number of variables can vary significantly. Some datasets have below 10 variables (ETT, Illness) while others have in order of hundreds (Traffic, Electricity). This characteristic results in different levels of complexity. Therefore, automatically tuning the sparsity can help to tune the complexity of the task at hand that eventually help prevent overfitting in simple tasks (e.g., Weather) and maintaining overparameterization for complex tasks (e.g., Traffic, Electricity). The beauty of our proposed method consists in the fact that it does not have to consider any of these intrinsic differences. We did not make any finetuning for PALS to account for these differences, and it does everything automatically. Of course, finetuning PALS per dataset specificity would improve its final performance, but it would reduce the generality of our proposed work and we prefer not to do it.

A.2 Prediction Quality Evaluation Metrics

We use MSE and MAE as the evaluation metrics, which can be computed as below:

$$MSE(\widetilde{\mathbf{X}}_{t:t+H}, \mathbf{X}_{t:t+H}) = \frac{1}{H} \sum_{i=0}^{H-1} (\widetilde{\mathbf{x}}_{t+i} - \mathbf{x}_{t+i})^2. \quad (4)$$

$$MAE(\widetilde{\mathbf{X}}_{t:t+H}, \mathbf{X}_{t:t+H}) = \frac{1}{H} \sum_{i=0}^{H-1} |\widetilde{\mathbf{x}}_{t+i} - \mathbf{x}_{t+i}|. \quad (5)$$

A.3 Hyperparameters

The settings of the transformer models and the hyperparameter values are adopted from the NSTRansformer implementation⁷. Sequence length L was set to 36 for the Illness dataset and 96 for the other datasets. Several values for prediction length were tested in the experiments: $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). The models considered in the experiments are all trained with the ADAM optimizer with a learning rate of 10^{-4} . The batch size used in

³<https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014>

⁴<https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html>

⁵<https://pems.dot.ca.gov/>

⁶<https://www.bgc-jena.mpg.de/wetter/>

⁷https://github.com/thuml/Nonstationary_Transformers

the experiments was equal to 32. A maximum number of 10 epochs was used for each experiment. Each transformer model consisted of two encoder layers and one decoder layer. Model dimension d_{model} was set to 512 in the experiments unless stated otherwise. PALS starts from a dense model (Initial density $D_{init} = 1$). The pruning rate ζ is initialized to 0.5 and decreased during training with a cosine decay schedule. The values of pruning rate factor γ and loss freedom factor λ are optimized in $\{1.05, 1.1, 1.2\}$ on the validation set using a single random seed for each experiment. Minimum sparsity S_{min} and maximum sparsity S_{max} are set to 20 and 90, respectively. These values give the flexibility to the user to tune the range of the model sparsity based on the resource availability in their application. The mask update frequency Δt was equal to 5 for the Illness dataset and 20 for the other datasets. Each experiment was run on three different random seeds and the average measurements are reported for each metric.

B Analyzing Sparsity Effect in Transformers for Time Series Forecasting

This section presents the results for the sparsity effect in various time series forecasting transformers. Specifically, we employ GraNet [31] to prune each transformer model, and then evaluate their effectiveness at different sparsity levels.

GraNet gradually prunes a network (in this paper, we start from a dense network) during the training to reach a pre-determined sparsity level while allowing for connection regeneration. Therefore, it takes advantage of dense-to-sparse and sparse-to-sparse training by exploring faster the search space and dynamically optimizing the sparse connectivity during training, respectively, to find optimal connectivity patterns efficiently. During training, GraNet executes gradual pruning and zero-cost Neuroregeneration every δt iterations. Gradual pruning gradually reduces network density towards a specific sparsity level across multiple pruning iterations. The initial sparsity level can be zero (creating a dense network, resulting in a dense-to-sparse approach) or higher (starting from a random sparse topology, resulting in a sparse-to-sparse approach). At each pruning step, a portion of weights with the lowest magnitudes is pruned, based on a fixed schedule. This stage is similar to gradual magnitude pruning (GMP) [68]. Following each pruning step, a zero-cost Neuroregeneration is executed. This involves dropping a portion of the existing connections with low magnitudes, which are considered damaged, and adding an equal number of new connections back to the network. The new connections are chosen from non-existing connections with the highest gradient value.

The results for the sparsity effect on the performance of various transformer models are presented in Figure 2 in the paper and Figure 3. The findings are discussed in Section 3.

C Comparison Results

The detailed results of the experiments performed in Section 5.2 are presented in Table 5.

D Univariate results

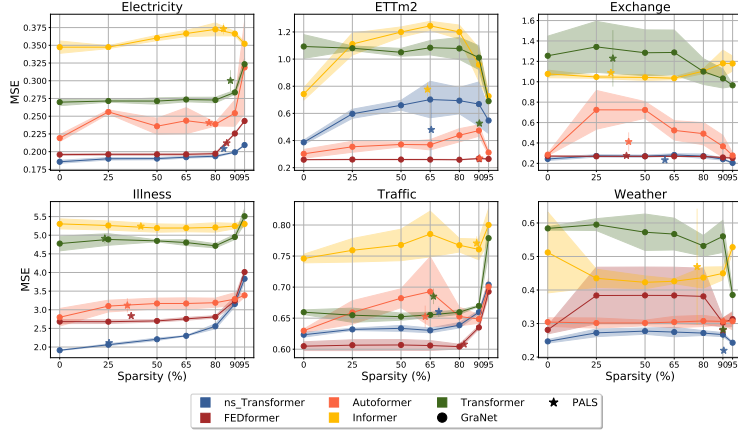
The results are presented in Figures 6 and 7 and discussed in Section 5.2 in the paper.

E Hyperparameter Sensitivity Analysis

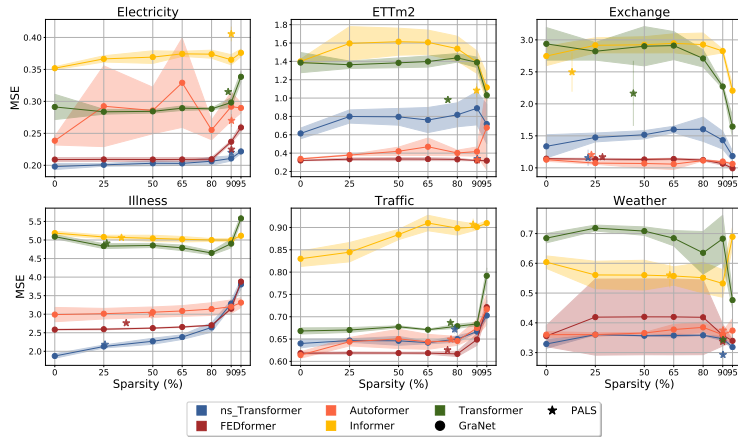
In this Appendix, we present the results for the hyperparameter sensitivity of PALS. We vary the values of γ and λ in $\{1.05, 1.1, 1.2\}$. The results are presented in Table 8. The results are discussed in Section 6.2.

F Pruning DLinear with PALS

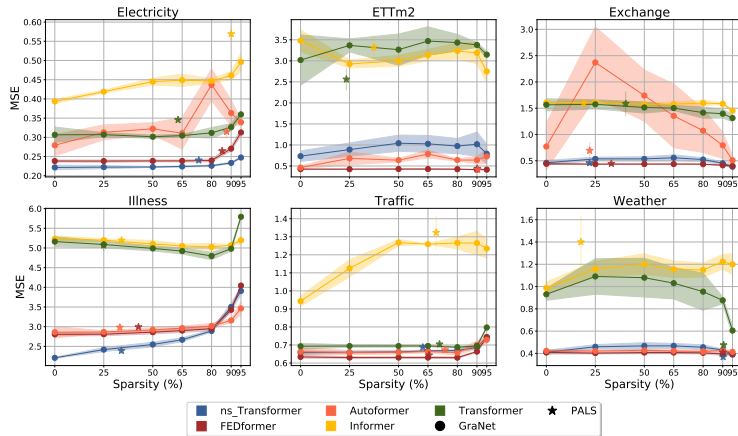
In this Appendix, we train PALS with the DLinear [62] model which is a MLP-based model for time series forecasting and has proven to be effective across various datasets. While our focus in this paper is to reduce the complexity of transformers for time series forecasting, we want to show the generality of our proposed approach to other models. We demonstrate that PALS can be also applied to these



(a) $H = 192$ ($H = 36$ for the Illness dataset)



(b) $H = 336$ ($H = 48$ for the Illness dataset)



(c) $H = 720$ ($H = 60$ for the Illness dataset)

Figure 3: Sparsity effect on the performance of various transformer models for time series forecasting on benchmark datasets in terms of MSE loss for various prediction lengths as indicated in each figure. Each model is sparsified using GraNet [31] to sparsity levels (%) $\in \{25, 50, 65, 80, 90, 95\}$. Sparsity=0 indicates the original dense model.

models (which are computationally cheaper than transformers) to decrease the model size even more. As an example, we apply PALS to DLinear.

Table 5: Comparison on the benchmark Datasets for prediction length $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$). For each model, the performance of the original (Dense) and the pruned model using PALS is presented in terms of MSE and parameter count ($\times 10^6$). The difference between the results compared to the dense counterpart is shown in parenthesis as % (blue indicates improvement in the results compared to the dense model). **Bold** entries are the best performer in each experiment among various models.

Datasets	H	NSTransformer				FEDformer				Autoformer				Informer				Transformer			
		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS	
		MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param
Electricity	96	0.170	12.2	0.195 (4.3% \uparrow)	2.6 (78.2% \downarrow)	0.187	19.5	0.203 (8.6% \uparrow)	4.4 (77.3% \downarrow)	0.207	12.1	0.206 (0.4% \downarrow)	5.3 (56.2% \downarrow)	0.333	12.5	0.347 (4.3% \uparrow)	1.2 (90.1% \downarrow)	0.257	11.7	0.282 (9.6% \uparrow)	3.0 (73.9% \downarrow)
	192	0.186	12.2	0.204 (10.0% \uparrow)	1.9 (84.3% \downarrow)	0.196	19.5	0.212 (8.3% \uparrow)	2.8 (85.9% \downarrow)	0.219	12.1	0.241 (9.9% \uparrow)	2.8 (76.7% \downarrow)	0.347	12.5	0.373 (7.4% \uparrow)	2.0 (84.2% \downarrow)	0.270	11.7	0.300 (11.3% \uparrow)	1.4 (87.8% \downarrow)
	336	0.198	11.9	0.220 (11.1% \uparrow)	1.2 (90.0% \downarrow)	0.209	19.5	0.225 (7.7% \uparrow)	1.9 (90.1% \downarrow)	0.239	12.1	0.270 (13.2% \uparrow)	1.2 (90.2% \downarrow)	0.352	12.5	0.405 (15.2% \uparrow)	1.2 (90.1% \downarrow)	0.291	11.7	0.315 (8.2% \uparrow)	1.3 (88.5% \downarrow)
	720	0.222	11.9	0.241 (8.6% \uparrow)	3.2 (73.5% \downarrow)	0.238	19.5	0.265 (11.0% \uparrow)	2.8 (85.4% \downarrow)	0.279	12.1	0.315 (12.7% \uparrow)	1.5 (87.7% \downarrow)	0.393	12.5	0.569 (44.8% \uparrow)	1.2 (90.0% \downarrow)	0.307	11.7	0.346 (12.7% \uparrow)	4.3 (62.9% \downarrow)
ETTm2	96	0.241	10.6	0.244 (1.0% \uparrow)	4.1 (61.0% \downarrow)	0.190	17.9	0.201 (5.7% \uparrow)	1.8 (90.2% \downarrow)	0.233	10.5	0.214 (7.9% \downarrow)	1.0 (90.1% \downarrow)	0.485	11.3	0.371 (23.3% \downarrow)	8.7 (22.9% \downarrow)	0.418	10.5	0.268 (35.8% \downarrow)	1.0 (90.8% \downarrow)
	192	0.387	10.7	0.480 (24.1% \uparrow)	3.7 (65.6% \downarrow)	0.259	17.9	0.263 (1.7% \uparrow)	1.8 (90.2% \downarrow)	0.302	10.5	0.269 (10.8% \downarrow)	1.0 (90.4% \downarrow)	0.742	11.3	0.777 (4.7% \uparrow)	4.1 (63.6% \downarrow)	1.093	10.5	0.527 (51.8% \downarrow)	1.0 (90.3% \downarrow)
	336	0.615	10.6	0.339 (44.9% \downarrow)	1.0 (90.2% \downarrow)	0.320	17.9	0.325 (1.4% \uparrow)	1.8 (90.2% \downarrow)	0.338	10.5	0.326 (3.4% \downarrow)	1.0 (90.2% \downarrow)	1.400	11.3	1.084 (22.5% \downarrow)	1.2 (89.8% \downarrow)	1.387	10.5	0.982 (29.2% \downarrow)	2.6 (75.2% \downarrow)
	720	0.734	10.6	0.439 (40.1% \downarrow)	1.0 (90.2% \downarrow)	0.425	17.9	0.423 (0.4% \downarrow)	1.8 (90.1% \downarrow)	0.463	10.5	0.417 (9.9% \downarrow)	1.0 (90.4% \downarrow)	3.479	11.3	3.320 (4.5% \downarrow)	7.1 (37.4% \downarrow)	3.018	10.5	2.561 (15.1% \downarrow)	8.1 (23.4% \downarrow)
Exchange	96	0.124	10.5	0.116 (6.8% \downarrow)	1.0 (90.4% \downarrow)	0.146	17.9	0.155 (5.7% \uparrow)	6.7 (62.7% \downarrow)	0.147	10.5	0.148 (0.4% \downarrow)	5.9 (44.2% \downarrow)	0.941	11.3	0.945 (0.5% \uparrow)	7.7 (32.1% \downarrow)	0.679	10.5	0.651 (4.2% \downarrow)	7.2 (31.6% \downarrow)
	192	0.243	10.5	0.233 (4.4% \downarrow)	4.2 (60.2% \downarrow)	0.271	17.9	0.276 (1.6% \uparrow)	10.6 (40.5% \downarrow)	0.285	10.5	0.412 (44.3% \uparrow)	6.2 (41.6% \downarrow)	1.077	11.3	1.087 (1.0% \uparrow)	7.7 (32.5% \downarrow)	1.255	10.5	1.229 (2.0% \downarrow)	7.0 (33.5% \downarrow)
	336	0.462	10.6	0.459 (0.5% \downarrow)	8.2 (22.0% \downarrow)	0.443	17.9	0.447 (0.9% \uparrow)	11.9 (33.2% \downarrow)	0.772	10.5	0.695 (9.9% \downarrow)	8.2 (22.1% \downarrow)	1.609	11.3	1.596 (0.8% \downarrow)	9.2 (19.1% \downarrow)	1.565	10.5	1.588 (1.3% \downarrow)	6.3 (40.5% \downarrow)
	720	1.336	10.6	1.156 (13.5% \downarrow)	8.5 (21.3% \downarrow)	1.143	17.9	1.169 (2.3% \uparrow)	12.8 (28.6% \downarrow)	1.128	10.5	1.206 (6.9% \uparrow)	8.1 (23.0% \downarrow)	2.747	11.3	2.497 (9.1% \downarrow)	9.8 (13.1% \downarrow)	2.938	10.5	2.164 (26.3% \downarrow)	5.9 (44.4% \downarrow)
Illness	24	2.555	10.5	2.641 (3.3% \uparrow)	6.9 (35.0% \downarrow)	3.285	13.1	3.590 (9.3% \uparrow)	7.6 (42.1% \downarrow)	3.686	10.5	3.648 (1.0% \downarrow)	7.5 (29.2% \downarrow)	5.361	11.3	5.412 (1.0% \uparrow)	7.9 (29.9% \downarrow)	4.740	10.5	4.739 (0.0% \downarrow)	6.9 (34.3% \downarrow)
	36	1.913	10.5	2.111 (10.4% \uparrow)	7.9 (25.4% \downarrow)	2.682	13.5	2.835 (5.7% \uparrow)	8.5 (36.7% \downarrow)	2.799	10.5	3.118 (11.4% \uparrow)	6.9 (34.8% \downarrow)	5.304	11.3	5.239 (1.2% \downarrow)	6.6 (41.7% \downarrow)	4.776	10.5	4.911 (2.8% \uparrow)	8.1 (23.2% \downarrow)
	48	1.873	10.5	2.186 (16.7% \uparrow)	7.8 (25.6% \downarrow)	2.585	13.9	2.762 (6.9% \uparrow)	8.8 (36.4% \downarrow)	2.990	10.5	3.017 (0.9% \downarrow)	5.3 (49.3% \downarrow)	5.187	11.3	5.064 (2.4% \downarrow)	7.5 (34.0% \downarrow)	5.090	10.5	4.914 (3.5% \downarrow)	7.7 (26.5% \downarrow)
	60	2.209	10.5	2.391 (8.2% \uparrow)	7.0 (34.0% \downarrow)	2.807	14.3	2.994 (6.7% \uparrow)	8.2 (42.7% \downarrow)	2.860	10.5	2.984 (4.3% \uparrow)	7.0 (33.2% \downarrow)	5.234	11.3	5.197 (0.7% \downarrow)	7.5 (34.1% \downarrow)	5.162	10.5	5.070 (1.8% \downarrow)	7.9 (25.0% \downarrow)
Traffic	96	0.608	14.1	0.641 (5.4% \uparrow)	4.3 (69.6% \downarrow)	0.581	22.3	0.585 (0.8% \uparrow)	5.4 (75.6% \downarrow)	0.649	14.9	0.629 (3.1% \downarrow)	5.2 (65.2% \downarrow)	0.724	14.4	0.744 (2.7% \uparrow)	1.5 (89.3% \downarrow)	0.646	13.6	0.677 (4.8% \uparrow)	3.7 (72.8% \downarrow)
	192	0.623	14.1	0.660 (5.9% \uparrow)	4.3 (69.4% \downarrow)	0.605	22.3	0.608 (0.5% \uparrow)	3.9 (82.4% \downarrow)	0.630	14.9	0.653 (3.6% \uparrow)	5.6 (62.4% \downarrow)	0.746	14.4	0.771 (3.3% \uparrow)	1.6 (88.9% \downarrow)	0.660	13.6	0.685 (3.9% \uparrow)	4.5 (66.7% \downarrow)
	336	0.640	14.6	0.672 (5.0% \uparrow)	3.1 (78.7% \downarrow)	0.618	22.3	0.625 (1.1% \uparrow)	5.6 (75.0% \downarrow)	0.614	14.9	0.649 (5.7% \uparrow)	3.5 (76.8% \downarrow)	0.830	14.4	0.907 (9.3% \uparrow)	1.7 (88.1% \downarrow)	0.668	13.6	0.687 (2.8% \uparrow)	3.2 (76.5% \downarrow)
	720	0.658	14.1	0.687 (4.5% \uparrow)	5.3 (62.4% \downarrow)	0.634	22.3	0.643 (1.5% \uparrow)	7.7 (65.5% \downarrow)	0.663	14.9	0.673 (1.5% \uparrow)	3.9 (73.7% \downarrow)	0.943	14.4	1.324 (40.3% \uparrow)	4.4 (69.2% \downarrow)	0.693	13.6	0.705 (1.7% \downarrow)	4.0 (70.9% \downarrow)
Weather	96	0.182	10.8	0.167 (8.0% \downarrow)	1.0 (90.3% \downarrow)	0.232	17.9	0.215 (7.2% \downarrow)	1.8 (90.0% \downarrow)	0.266	10.6	0.258 (2.9% \downarrow)	1.1 (90.0% \downarrow)	0.375	11.4	0.349 (7.0% \downarrow)	1.1 (90.2% \downarrow)	0.380	10.6	0.198 (47.8% \downarrow)	1.0 (90.1% \downarrow)
	192	0.247	10.6	0.220 (11.0% \downarrow)	1.0 (90.5% \downarrow)	0.281	17.9	0.280 (0.4% \downarrow)	1.8 (90.0% \downarrow)	0.304	10.6	0.306 (0.7% \uparrow)	1.0 (90.4% \downarrow)	0.512	11.4	0.469 (8.4% \downarrow)	2.6 (76.8% \downarrow)	0.584	10.6	0.282 (51.6% \downarrow)	1.0 (90.1% \downarrow)
	336	0.329	10.6	0.293 (10.9% \downarrow)	1.0 (90.1% \downarrow)	0.355	17.9	0.337 (5.1% \downarrow)	1.8 (90.0% \downarrow)	0.360	10.6	0.375 (4.1% \uparrow)	1.0 (90.1% \downarrow)	0.604	11.4	0.560 (7.3% \downarrow)	4.2 (63.0% \downarrow)	0.684	10.6	0.343 (49.9% \downarrow)	1.0 (90.1% \downarrow)
	720	0.410	10.6	0.368 (10.1% \downarrow)	1.0 (90.2% \downarrow)	0.410	17.9	0.410 (0.0% \downarrow)	1.8 (90.1% \downarrow)	0.423	10.6	0.416 (1.7% \downarrow)	2.1 (80.3% \downarrow)	0.987	11.4	1.400 (41.8% \uparrow)	9.4 (17.6% \downarrow)	0.930	10.6	0.476 (48.9% \downarrow)	1.0 (90.4% \downarrow)

Table 6: Univariate prediction comparison on the ETTm2 and Exchange datasets for prediction length $H \in \{96, 192, 336, 720\}$. For each model, the performance of the original (Dense) and the pruned model using PALS is presented in terms of MSE and parameter count. The difference between the results compared to the dense counterpart is shown in parenthesis as % (blue indicates improvement in the results compared to the dense model). **Bold** entries are the best performer in each experiment among various models.

Datasets	H	NSTransformer				FEDformer				Autoformer				Informer				Transformer			
		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS		Dense		+PALS	
		MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param	MSE	#Param
ETTm2	96	0.074	10.6	0.068 (7.6% \downarrow)	1.1 (90.0% \downarrow)	0.069	17.8	0.065 (5.9% \downarrow)	1.7 (90.2% \downarrow)	0.125	10.5	0.127 (1.7% \uparrow)	2.5 (76.6% \downarrow)	0.092	11.3	0.092 (0.2% \downarrow)	2.7 (76.5% \downarrow)	0.079	10.5	0.070 (12.0% \downarrow)	1.1 (89.3% \downarrow)
	192	0.128	10.7	0.107 (16.3% \downarrow)	1.1 (90.2% \downarrow)	0.100	17.8	0.101 (0.6% \uparrow)	1.7 (90.2% \downarrow)	0.141	10.5	0.144 (2.2% \uparrow)	1.0 (90.4% \downarrow)	0.137	11.3	0.129 (6.0% \downarrow)	1.1 (90.1% \downarrow)	0.119	10.5	0.117 (1.3% \downarrow)	7.0 (33.8% \downarrow)
	336	0.146	10.5	0.153 (5.4% \uparrow)	1.0 (90.1% \downarrow)	0.133	17.8	0.131 (1.6% \downarrow)	2.6 (85.7% \downarrow)	0.146	10.5	0.143 (2.4% \downarrow)	1.0 (90.4% \downarrow)	0.174	11.3	0.170 (2.5% \downarrow)	5.2 (54.3% \downarrow)	0.171	10.5	0.137 (19.5% \downarrow)	1.0 (90.1% \downarrow)
	720	0.225	10.5	0.230 (2.0% \uparrow)	3.7 (65.3% \downarrow)	0.185	17.8	0.186 (0.2% \uparrow)	2.6 (85.4% \downarrow)	0.195	10.5	0.181 (7.3% \downarrow)	1.0 (90.3% \downarrow)	0.211	11.3	0.213 (1.0% \uparrow)	6.3 (44.6% \downarrow)	0.192	10.5	0.189 (1.7% \downarrow)	6.0 (42.6% \downarrow)
Exchange	96	0.161	10.5	0.150 (6.8% \downarrow)	8.4 (20.0% \downarrow)	0.122	17.8	0.131 (7.4% \downarrow)	12.2 (31.4% \downarrow)	0.161	10.5	0.159 (1.2% \downarrow)	8.2 (21.8% \downarrow)	0.374	11.3	0.316 (15.6% \downarrow)	6.9 (39.2% \downarrow)	0.329	10.5	0.268 (18.4% \downarrow)	8.6 (18.5% \downarrow)
	192	0.252	10.5	0.252 (13.7% \uparrow)	8.5 (19.1% \downarrow)	0.257	17.8	0.276 (7.3% \uparrow)	9.2 (48.4% \downarrow)	0.304	10.5	0.345 (13.5% \uparrow)	8.1 (23.2% \downarrow)	1.180	11.3	1.064 (9.8% \downarrow)	8.4 (25.3% \downarrow)	1.549	10.5	1.339 (13.5% \downarrow)	8.0 (23.8% \downarrow)
	336	0.395	10.5	0.327 (17.3% \downarrow)	8.6 (18.3% \downarrow)	0.499	17.8	0.525 (5.2% \uparrow)	13.8 (22.9% \downarrow)	0.669	10.5	0.616 (8.0% \downarrow)	7.4 (29.8% \downarrow)	1.771	11.3	1.747 (1.4% \downarrow)	9.2 (18.3%				

Table 8: Hyperparameter Sensitivity of PALS. The prediction MSE and model’s parameter count ($\times 10^6$) when applying PALS on NSTRansformers is measured when changing the values of γ and λ in $\{1.05, 1.1, 1.2\}$. **Blue** indicates improvement in the results compared to the dense model. **Bold** entries are the best performer in each row.

	H	Dense	$\gamma = 1.05$			$\gamma = 1.1$			$\gamma = 1.2$		
			$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$	$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$	$\lambda = 1.05$	$\lambda = 1.1$	$\lambda = 1.2$
Electricity	96	0.170	0.195 (78.2%)	0.193 (74.8%)	0.193 (74.8%)	0.200 (90.3%)	0.198 (90.1%)	0.199 (90.1%)	0.206 (90.6%)	0.206 (90.5%)	0.206 (90.2%)
	192	0.186	0.204 (84.3%)	0.211 (78.0%)	0.209 (78.6%)	0.215 (90.1%)	0.214 (90.1%)	0.214 (90.1%)	0.258 (90.3%)	0.260 (90.6%)	0.260 (90.3%)
	336	0.198	0.215 (90.0%)	0.218 (90.0%)	0.219 (90.0%)	0.243 (90.1%)	0.246 (90.1%)	0.244 (90.1%)	0.271 (90.6%)	0.271 (90.4%)	0.270 (90.2%)
	720	0.222	0.240 (78.1%)	0.242 (73.5%)	0.242 (73.5%)	0.250 (90.1%)	0.251 (90.1%)	0.252 (90.0%)	0.297 (90.4%)	0.295 (90.3%)	0.300 (90.1%)
ETTm2	96	0.241	0.272 (25.7%)	0.263 (30.3%)	0.268 (64.6%)	0.244 (40.0%)	0.248 (61.0%)	0.202 (90.4%)	0.193 (90.3%)	0.193 (90.3%)	0.195 (90.1%)
	192	0.387	0.644 (21.4%)	0.626 (21.5%)	0.650 (25.9%)	0.574 (27.4%)	0.547 (41.1%)	0.498 (54.5%)	0.541 (40.4%)	0.393 (65.4%)	0.484 (65.3%)
	336	0.615	0.529 (20.5%)	0.529 (21.2%)	0.480 (27.9%)	0.753 (21.9%)	0.725 (35.7%)	0.563 (67.5%)	0.473 (73.7%)	0.340 (90.2%)	0.340 (90.1%)
	720	0.734	0.751 (19.7%)	0.780 (27.7%)	0.793 (42.1%)	0.777 (30.9%)	0.804 (34.9%)	0.641 (78.5%)	0.678 (45.5%)	0.600 (79.2%)	0.441 (90.2%)
Exchange	96	0.124	0.139 (21.1%)	0.138 (22.3%)	0.135 (32.5%)	0.139 (22.3%)	0.143 (22.5%)	0.139 (64.2%)	0.137 (28.9%)	0.113 (79.5%)	0.125 (87.0%)
	192	0.243	0.269 (16.3%)	0.269 (16.3%)	0.269 (16.3%)	0.266 (22.9%)	0.264 (27.7%)	0.263 (30.1%)	0.249 (34.6%)	0.241 (38.1%)	0.239 (60.9%)
	336	0.462	0.486 (17.7%)	0.486 (17.7%)	0.486 (17.7%)	0.459 (22.0%)	0.459 (22.0%)	0.459 (22.0%)	0.429 (13.1%)	0.478 (18.3%)	0.460 (28.7%)
	720	1.336	1.263 (17.0%)	1.263 (17.0%)	1.263 (17.0%)	1.134 (22.9%)	1.134 (22.9%)	1.134 (22.9%)	1.236 (15.6%)	1.236 (15.6%)	1.236 (15.6%)
Illness	24	2.555	2.593 (27.5%)	2.638 (34.3%)	2.603 (31.6%)	2.668 (44.5%)	2.620 (47.9%)	2.613 (53.0%)	2.849 (65.6%)	2.784 (70.9%)	2.884 (75.7%)
	36	1.913	2.112 (24.8%)	2.118 (27.5%)	2.125 (30.8%)	2.193 (34.3%)	2.220 (41.3%)	2.246 (51.6%)	2.433 (53.0%)	2.496 (59.7%)	2.644 (70.9%)
	48	1.873	2.186 (25.6%)	2.207 (31.3%)	2.214 (34.0%)	2.246 (39.6%)	2.277 (48.8%)	2.332 (53.1%)	2.492 (60.4%)	2.650 (65.4%)	2.632 (63.5%)
	60	2.209	2.413 (30.7%)	2.425 (34.5%)	2.425 (34.5%)	2.523 (50.9%)	2.549 (56.4%)	2.577 (48.0%)	2.772 (57.8%)	2.895 (64.7%)	3.019 (58.6%)
Traffic	96	0.608	0.641 (69.6%)	0.647 (68.9%)	0.647 (68.9%)	0.676 (70.6%)	0.666 (90.1%)	0.681 (80.3%)	0.671 (90.4%)	0.669 (90.2%)	0.674 (90.6%)
	192	0.623	0.658 (75.7%)	0.661 (69.6%)	0.661 (59.9%)	0.688 (76.4%)	0.694 (80.2%)	0.720 (60.2%)	0.711 (90.1%)	0.697 (90.1%)	0.693 (90.4%)
	336	0.640	0.672 (78.7%)	0.671 (76.7%)	0.675 (72.9%)	0.712 (76.9%)	0.688 (90.3%)	0.690 (90.3%)	0.715 (90.4%)	0.720 (90.1%)	0.718 (90.6%)
	720	0.658	0.687 (62.4%)	0.687 (62.4%)	0.687 (62.4%)	0.730 (76.4%)	0.735 (68.8%)	0.735 (68.8%)	0.728 (90.3%)	0.721 (90.3%)	0.724 (90.9%)
Weather	96	0.182	0.166 (78.2%)	0.167 (80.8%)	0.167 (80.8%)	0.167 (90.3%)	0.167 (90.5%)	0.167 (90.5%)	0.172 (90.2%)	0.172 (90.1%)	0.172 (90.1%)
	192	0.247	0.225 (83.7%)	0.224 (85.4%)	0.224 (85.4%)	0.222 (90.5%)	0.222 (90.5%)	0.222 (90.5%)	0.220 (90.1%)	0.220 (90.1%)	0.220 (90.1%)
	336	0.329	0.303 (80.0%)	0.301 (80.4%)	0.301 (80.4%)	0.294 (90.3%)	0.295 (90.5%)	0.295 (90.5%)	0.297 (90.2%)	0.298 (90.1%)	0.298 (90.1%)
	720	0.410	0.413 (68.9%)	0.392 (85.2%)	0.392 (85.2%)	0.372 (90.3%)	0.382 (90.5%)	0.382 (90.5%)	0.364 (90.3%)	0.375 (90.1%)	0.375 (90.1%)

Finally, we want to highlight that our goal in this paper is not to propose a new forecasting model and beat the state-of-the-art for time series forecasting. Instead, we aim to decrease the high computational costs of models for time series forecasting while finding automatically their optimal sparsity level and potentially improving their generalization. The reason that we focus on transformers is that they are considered to be computationally expensive while performing well in time series forecasting, and as shown in [59], transformer-based models perform generally well in other time series analysis tasks, including, classification, anomaly detection, and imputation compared to the MLP-based models [62]. Therefore, they can be a promising direction for future time series analysis research. However, PALS is orthogonal to forecasting models and can be applied to any deep learning-based model to reduce its computational costs.

Table 9: Effectiveness of PALS for pruning DLinear model [62]. Each row presents the results of DLinear before and after applying PALS in terms of MSE, for each prediction length. The achieved sparsity level is shown in parenthesis as %.

	Model	Electricity	ETTm2	Exchange	Illness	Traffic	Weather
96/24	DLinear	0.140	0.172	0.094	1.997	0.413	0.175
	+PALS	0.141 (90.2%)	0.181 (90.0%)	0.086 (25.5%)	1.984 (34.1%)	0.412 (90.3%)	0.177 (90.2%)
192/36	DLinear	0.153	0.235	0.168	2.090	0.424	0.217
	+PALS	0.154 (90.2%)	0.249 (90.1%)	0.173 (20.8%)	2.130 (29.8%)	0.424 (90.2%)	0.218 (90.2%)
336/48	DLinear	0.169	0.307	0.322	2.058	0.437	0.263
	+PALS	0.170 (90.2%)	0.301 (82.7%)	0.324 (24.7%)	2.124 (32.6%)	0.436 (90.1%)	0.262 (90.2%)
720/60	DLinear	0.204	0.390	0.959	2.375	0.467	0.328
	+PALS	0.204 (90.1%)	0.433 (90.3%)	0.963 (38.0%)	2.358 (44.3%)	0.467 (90.2%)	0.324 (90.2%)

G Efficiency of PALS

In this appendix, we discuss the efficiency of PALS in terms of computational costs from various perspectives.

G.1 Pruning Capabilities

Based on the observations in Section 6.1, PALS achieves the highest average sparsity level among the considered pruning and sparse training methods. More importantly, it finds the optimal sparsity level automatically without requiring any prior information, while most pruning and sparse training algorithms need to receive the sparsity level as an input of the algorithm. In short, PALS can find a

network with higher sparsity than the competitors (GMP, GraNet, and RigL), where the sparsity level is found automatically.

G.2 Convergence Speed

Another factor that we consider regarding the efficiency of the methods is the convergence speed. If a method converges faster than the others, it can be considered to be more efficient in terms of resource usage.

To compare the convergence speed of each model, we compare the training epochs. As we use early stopping during training, each training round might not need the full training epochs (which is set to 10). In Table 10, we report the average number of training epochs for various datasets. The results are an average for different prediction lengths and three random seeds. While GraNet needs almost full training time due to using a fixed pruning schedule which is determined based on the number of epochs, PALS can automate the pruning speed based on the loss. On four out of six datasets, PALS converges faster than GraNet and the dense model. On the Weather dataset, PALS requires longer training time than the dense model. As will be explained in Section G.3, PALS performs most part of this training at a very high sparsity level ($\sim 90\%$), thus being resource-efficient. Overall, PALS is able to achieve a higher or comparable speed to the dense model in most cases considered.

Table 10: Comparison of convergence speed in terms of the number of training epochs on the NSTRansformer model. The results are average over various prediction lengths of $H \in \{96, 192, 336, 720\}$ (except for the Illness dataset for which $H \in \{24, 36, 48, 60\}$).

Dataset	PALS	GraNet	Dense
Electricity	8.83	9.75	8.92
ETTM2	4.58	9.00	4.92
Exchange	5.83	9.42	4.33
Illness	7.97	9.58	8.92
Traffic	8.83	9.50	9.5
Weather	7.00	9.08	4.08

G.3 Sparsity During Training

Another factor affecting the efficiency of PALS is the sparsity during training. As in our experiment, PALS starts from a dense network, we analyze when it reaches the final sparsity and how the sparsity changes during training. Then, would be able to analyze whether the forward pass is mostly performed sparsely or not.

To achieve this, we plot the sparsity level during the training of PALS for each transformer model and dataset for various prediction lengths. The sparsity levels are measured after each pruning iteration which is repeated every 5 batches on the Illness dataset and every 20 batches on the other datasets. These values are measured till the last pruning iteration before saving the model. The results for all models are presented in Figure 4.

In Figure 4, due to different convergence speeds on each dataset, each model requires a different number of training epochs. As the pruning is tuned based on the loss value, the pruning speed varies for each dataset and model. For example, on the Weather, Electricity, and ETTM2 datasets, for most of the considered transformer models and prediction lengths, the models reach the final sparsity level within a few training epochs. However, for the Exchange, Illness, and Traffic datasets, the convergence speed can be different among different models and prediction lengths. Also, in these datasets, the final sparsity is reached slower than the earlier category (Weather, Electricity, and ETTM2 datasets). However, in most cases considered, the final sparsity is reached only within half of the training period.

Overall, it can be observed that in most cases PALS reaches the final sparsity level in a few epochs. Therefore, the forward pass during training is performed sparsely for a large fraction of the training process.

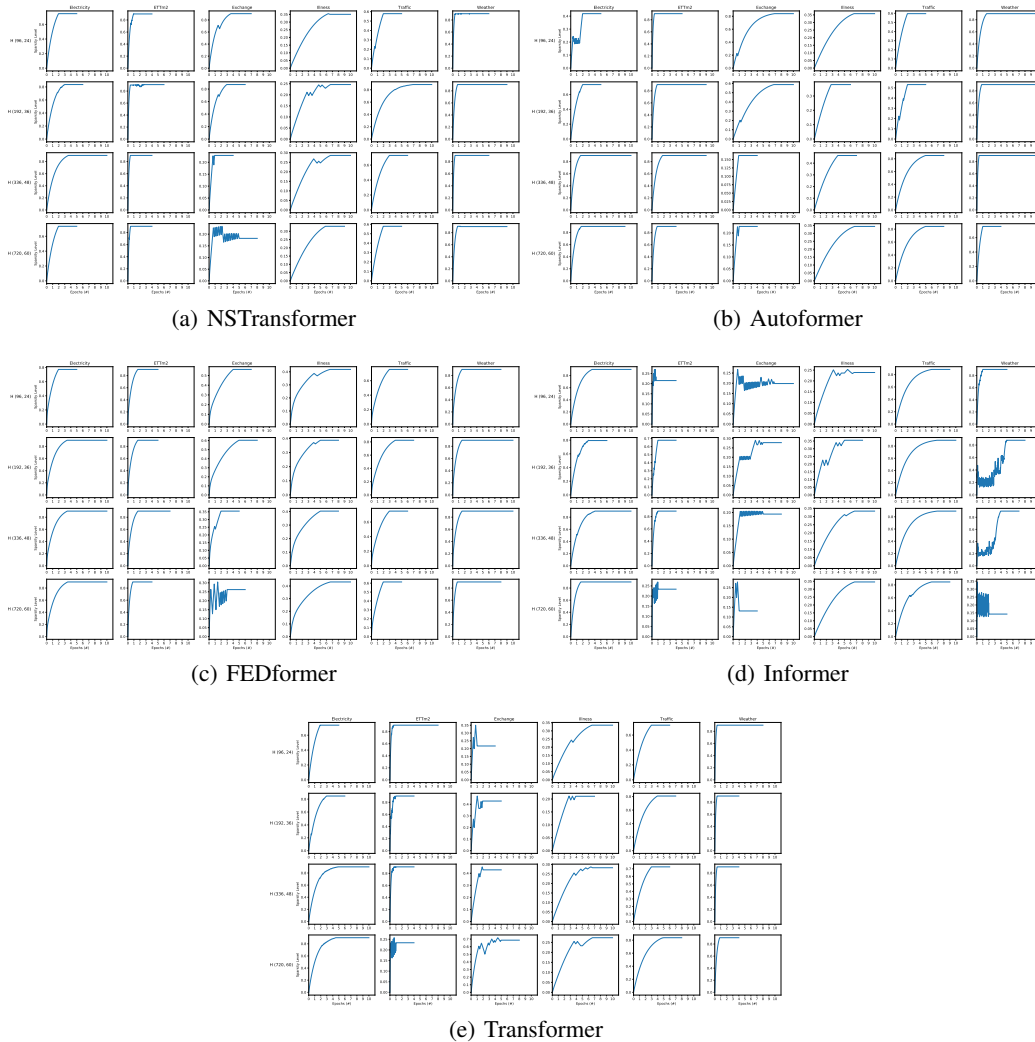


Figure 4: Sparsity level of each network during training of PALS. In most cases, the final sparsity is achieved within a few epochs after the training starts. Therefore, the forward pass during training is performed sparsely for a large fraction of the training process.

G.4 Sparse Implementation.

Another aspect regarding efficiency is concerned with the true sparse implementation. Further steps are required to achieve the full advantage of GPU speedup. However, this is a challenging research question that is out of the scope of the current work. Unstructured sparsity (e.g., pruning connections of a network as we use in this paper) still suffers from hardware support [30], while proven to be more effective than structured sparsity (e.g., pruning neurons or other units) [17]. Although there have been some works that have implemented truly sparse neural networks efficiently on CPUs [31, 7, 2] and just for MLPs and Autoencoders, proper and efficient GPU support for sparse matrix computation is yet to be developed. In this paper, we focus on the algorithmic aspect of developing resource-efficient algorithms, while getting full advantage of sparsity is a large amount of research that we cannot address with our current human resources. We hope that works such as ours can light an awareness signal, bringing more researchers from the community together to start addressing seriously the “elephant in the room”.

H Model Size Effect

In this Appendix, we study the effect of model size on the performance of PALS by changing the hidden dimension (d_{model}) in $\{256, 512(\text{default}), 768\}$ to analyze the trade-off between loss and parameters count. We have also considered the results of the original dense model ($d_{model} = 512$). The results for all models in terms of MSE and parameters count (which are the average results over different prediction lengths) are presented in Figure 5.

In Figure 5, we can observe that in most cases considered, the model can be pruned significantly with little or no increase in loss. Similar to our discussion in Section 3, we see consistent behavior among the Electricity, Illness, and traffic datasets, where a higher number of parameters is mostly in favor of loss reduction. This might come from the inherent complexity of predicting these datasets due to either a high number of variables (such as Electricity and Traffic datasets) or the relatively non-stationary nature of data (such as the illness datasets as shown in [35]). While being a non-stationary dataset, on the Exchange dataset, a small ($d_{model} = 256$) sparse model can perform very closely or better than the original ($d_{model} = 512$) dense model on average among various transformer architectures. This might be caused by the abilities of the transformer variants to learn this behavior such that even a pruned small model can substitute this model. On the other datasets (ETTm2 and Weather which are relatively stationary with a low number of variables), sparsity results mostly in a better performance than the dense model.

In short, it can be concluded that various time-series datasets do not demonstrate homogenous behavior due to their intrinsic differences. Therefore, we need to take these differences into consideration when choosing the model size or the right sparsity level (Section 3).

I Prediction Quality

In this Appendix, we evaluate the prediction of different models and discuss how PALS affects this prediction when pruning them.

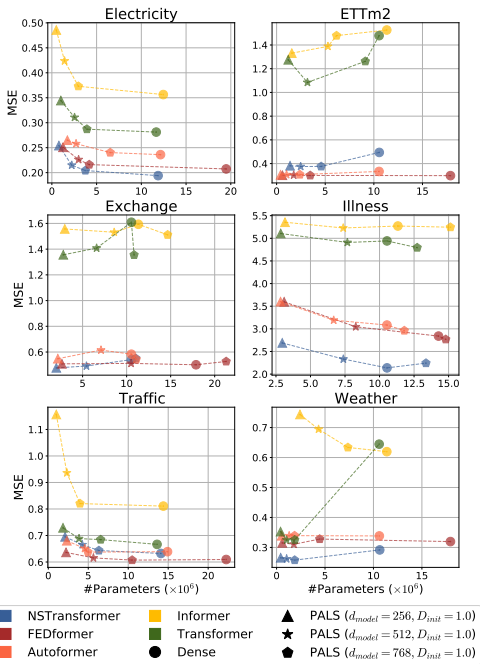


Figure 5: Model size effect by varying $d_{model} \in \{256, 512(\text{default}), 768\}$ on the prediction performance of PALS compared to the original dense model.

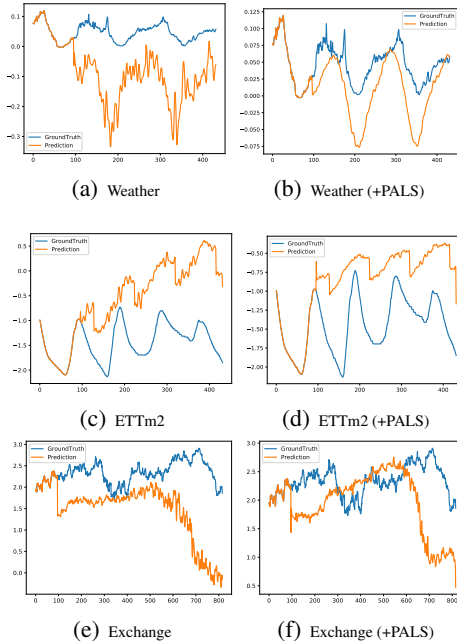


Figure 6: Forecasting Visualization on Transformer with and without PALS on three datasets.

The predictions for the transformer on the Weather, ETTm2, and Exchange datasets, where the most significant changes in loss and parameters count were seen, are visualized in Figure 6. It can be observed that PALS significantly improves the prediction of this model on these three datasets. As summarized in Table 5 (Appendix C), PALS is able to gain very close prediction performance on the transformer to the best performer on the Weather dataset (NStransformer) with only pruning 90% of unimportant connections.

Next, we visualize the predictions for each model with and without PALS on the Weather, Illness, ETTm2 (periodic dataset), and Exchange (without obvious periodicity) datasets for all transformer variants in Figures 7, 8, 9, and 10, respectively. In most cases considered in these figures, PALS is able to gain very similar or better performance than the dense counterpart model. By removing unimportant connections, the prediction using PALS is mostly smoother than the prediction of the dense model. Therefore, it can be concluded that it is possible to significantly prune transformers for time series forecasting using PALS without compromising performance.

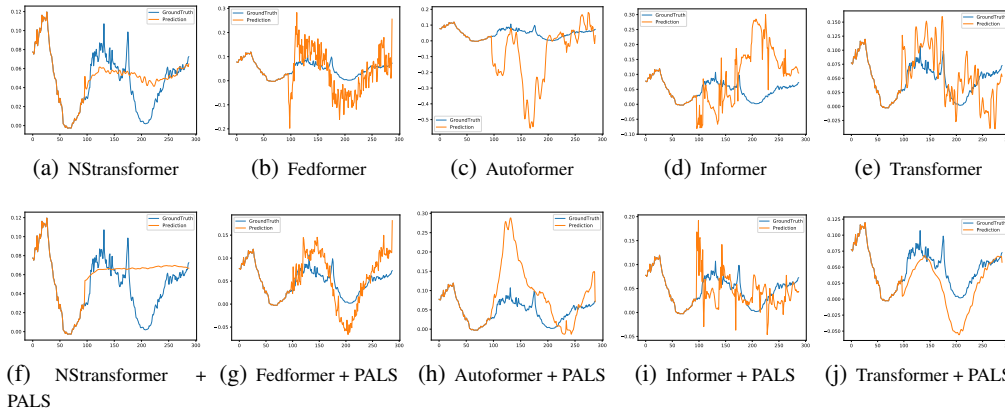


Figure 7: Forecasting Visualization on transformers with and without PALS on the Weather dataset ($H = 192$).

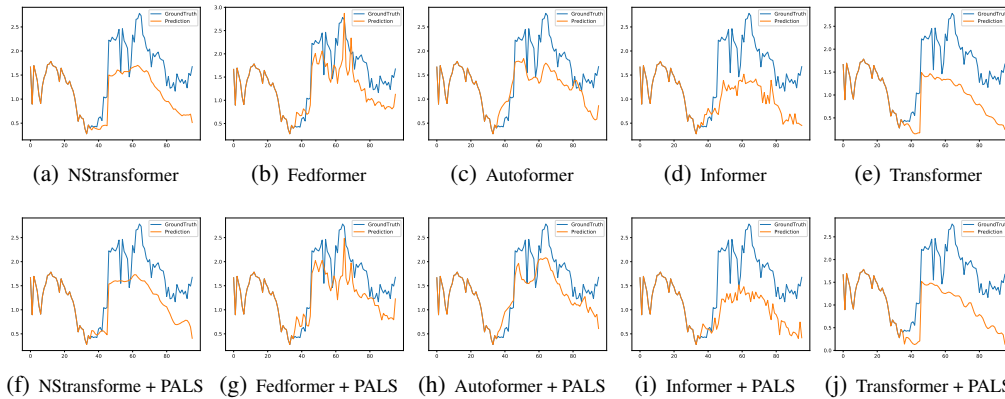


Figure 8: Forecasting Visualization on transformers with and without PALS on the illness dataset ($H = 60$).

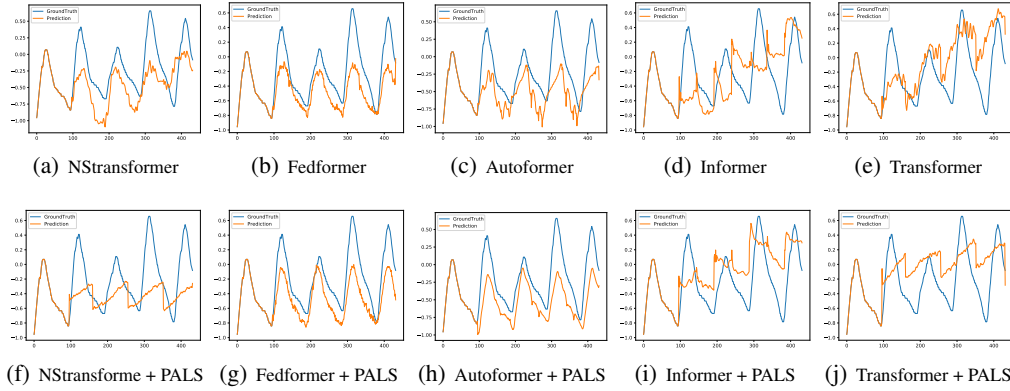


Figure 9: Forecasting Visualization on transformers with and without PALS on the ETTm2 dataset ($H = 336$).

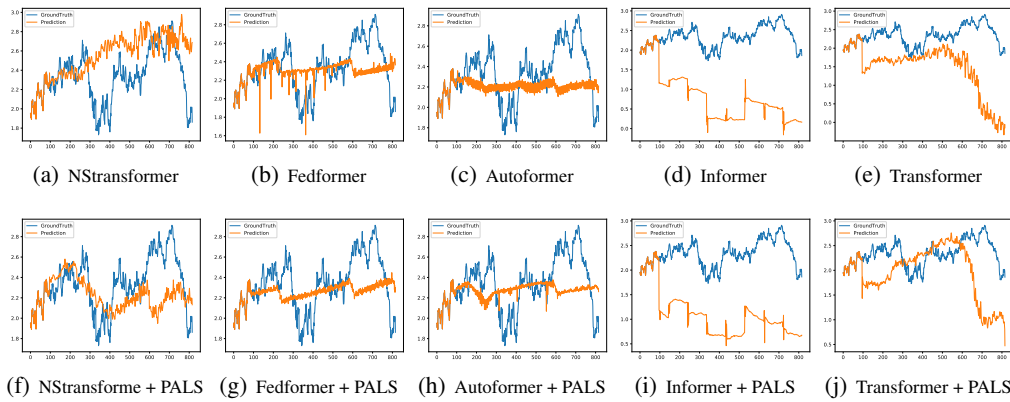


Figure 10: Forecasting Visualization on transformers with and without PALS on the Exchange dataset ($H = 720$).