Sparseformer: Improving Long-Range Dependency Modeling for Time-Series Forecasting

Haiyang Gao
Wuhan University

Working Draft - December 27, 2025

Abstract

Transformer-based models have recently advanced long sequence time-series forecasting (LSTF) in finance. However, they remain constrained by the quadratic self-attention complexity, memory bottlenecks, slow generation speed, and limited capacity for modeling long sequence dependencies. Existing solutions alleviate these issues only partially and often rely on restrictive assumptions. To address these limitations, we propose Sparseformer, an encoder-decoder based tranformer architecture that preserves $O(L\log L)$ computational efficiency of similar models (such as Informer, etc.) while introducing a sparsity-enhanced attention mechanism to expand the effective receptive field (ERF). This design enables more accurate long-range dependency modeling without large patch sizes. The model provides a scalable and robust framework for long sequence time-series forecasting.

1 Introduction

Time series forecasting is widely used in real-world applications, such as transportation management, economic planning, energy planning, and weather forecasting, etc. Because of the immense practical value, time series forecasting has received great attention and has grown tremendously in recent years [Wen et al., 2023, Lim and Zohren, 2021a, Miller et al., 2024, Benidis et al., 2022, Mahmoud and Mohammed, 2020, Masini et al., 2023]. In particular, earlier machine learning models like SVM can make much better predictions with less effort [Tay and Cao, 2001].

However, things underwent a profound transformation after the Transformer [Vaswani et al., 2023] architecture was introduced. Benefiting from its *attention* mechanism [Ji et al., 2019], the Transformer architecture is quite suitable for modeling long-range dependency in financial markets and capture the relationship between multiple variables. Thus, based on the increasing data availability and computing power in recent times, machine learning, specifically Transformer-based models, has become a vital part of the next generation of time series forecasting models [Lim and Zohren, 2021b]. Yongchareon [Yongchareon, 2025] has demonstrated that Transformer variants—such as Informer [Zhou et al., 2021], Autoformer [Wu et al., 2022], and PatchTST [Nie et al., 2023]—consistently outperform classical models not only on conventional statistical metrics such as mean squared error (MSE) and mean absolute error (MAE), but also in simulation-based evaluations that better reflect real-world financial performance. When integrated into trading strategies, these models perform higher returns, improved Sharpe ratios, and reduced maximum draw-downs, indicating superior robustness and adaptability under volatile market conditions. Such evidence suggests that attention-based architectures provide an excellent framework for predictive modeling and portfolio optimization, among other financial applications.

Preprint.

However, since the the Transformer [Vaswani et al., 2023] architecture is designed for NLP and although there are many similarities between NLP and LSTF, significant limitations still exists when using this architecture on LSTF:

1. The quadratic computation of self-attention. When calculating the output for the attention layer, the time complexity for step $\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$ is $O(L^2)$ when $L \gg d$. When solving LSTF problems, the model always faces a relatively long input time series compared to NLP problems. Thus, it requires significantly more computation when switching to LSTF problems, which slows down the decoding speed and increases memory usage. Former models focused on decreasing the time complexity. For instance, Sparse Transformer [Child et al., 2019], LogSparse Transformer [Li et al., 2020], and Longformer [Beltagy et al., 2020] all achieved $O(L\log L)$ self-attention complexity. However, their efficiency gain, measured by memory savings and speedup of training and inference, is limited due to the lack of an efficient algorithm for sparse matrix [Qiu et al., 2020]. Reformer [Kitaev et al., 2020] also achieves the same time complexity with locally sensitive hashing self-attention, while only works on extremely long sequences. Linformer [Wang et al., 2020] reduces the self-attention time complexity from the quadratic $O(L^2)$ of the vanilla Transformer to a linear $O(L)$ by projecting the key and value matrices into a lower-dimensional space through learnable linear transformations. This design relies on the empirical observation that the attention matrix is approximately low-rank, allowing the model to compute attention in a compressed representation without significant loss of information. However, in real-world long-sequence scenarios where attention patterns are complex and the true attention matrix may not be low-rank, this assumption can break down. To maintain accuracy, the projection dimension k must increase , causing the computational and memory costs to grow quadratically again, leading to a potential degradation of efficiency back to $O(L^2)$. Informer [Zhou et al., 2021] further improves upon former works by introducing a ProbSparse self-attention mechanism, which selects the most informative query-key pairs rather than computing attention over all positions. This mechanism preserves only the dominant dependencies that contribute most to the prediction. As a result, Informer not only maintains the $O(L\log L)$ time and memory complexity which is independent of the input, but also achieves better forecasting accuracy on LSTF problems.

2. The memory bottleneck in stacking layers for long inputs. Vanilla Transformer architecture always stacks several encoder and decoder layers to enhance the model's representation capability. This procedure leads to a linear increase of total memory consumption with respect to the number of layers J, leading to an overall complexity of $O(J \cdot L^2)$.Such quadratic dependence on sequence length L imposes a severe memory bottleneck when processing long time series inputs. Informer [Zhou et al., 2021] provided a solution to this issue by self-attention distilling. The "distilling" procedure forwards from j-th layer to $(j+1)$-th layer is: $X_{j+1}^t = \text{MaxPool}(\text{ELU}(\text{Con1d}([X_j^t]_{AB})))$, where $[\cdot]_{AB}$ represents the attention block. This operation applies a convolution followed by a non-linear activation function and a pooling step that halves the sequence length at each layer, effectively reduce the whole memory usage to $O((2-\epsilon)L\log L)$, enabling deep architectures to handle much longer input sequences efficiently.

3. The speed plunge in predicting long outputs. When facing LSTF problems, vanilla Transformer makes a step-by-step autoregressive decoding, which slows down the decoding speed since each future point depends on the previously generated ones. This sequential dependency makes the inference speed drop dramatically as the output length increases. Informer [Zhou et al., 2021] proposed a generative inference to alleviate the speed plunge in long prediction. Technically, they feed the decoder with the following vector as $X_{de}^t = \text{Concat}(X_{token}^t, X_0^t) \in \mathbb{R}^{(L_{token}+L_y)\times d_{model}}$, using a placeholder $X_0^t \in \mathbb{R}^{L_{token}\times d_{model}}$ to represent the target sequence. After two stacked attention layers and a fully connected projection layer, the decoder simultaneously generates the entire output sequence $\hat{Y}_t \in \mathbb{R}^{L_y \times d_y}$. This design removes the autoregressive dependency and transforms the decoding process from $O(L_y)$ sequential steps to a single forward pass ($O(1)$), achieving substantial acceleration for long-horizon forecasting.

4. The limited capacity of conventional Transformer architectures to model long-range dependencies. According to [Kong et al., 2025], Transformer architecture relies on embedding mechanisms to obtain positional relationships within sequences, but when processing long sequence inputs, it often fails to adequately encode them and tends to lose long-term dependency items. Latest research like PatchTST [Nie et al., 2023] focusing on using patching to capture long-term dependency. However, the large size patching leads to a new problem: patched-based Transformers have to work with a very long input length and a very large patch size to achieve ideal performance [Luo and Wang, 2024,

Zhou et al., 2021, 2022, Wu et al., 2022]. According to [Luo and Wang, 2024], the core problem in LSTF lays in the limited Effective Receptive Field (ERF) of vanilla Transformer and patching is a method to help the instruct attention to focus on core time steps. Based on this, they introduced sparsity into the attention layer and introduced deformable attention [Luo and Wang, 2024] to help model focusing on important time points, which does not rely on patching.

In this note, we intend to leverage the sparsity in the self-attention mechanism while retaining the computational advantages of similar models (such as Informer), thereby enhancing the model's ability to capture long-range temporal dependencies and improving overall forecasting performance. **To accomplish this goal, we propose Sparseformer. Our contributions are as follows:**

- Sparseformer is designed upon the classical encoder-decoder Transformer backbone, ensuring that the overall computational complexity remains at $O(L\log L)$ (at par with state of the art transfomer models such as Informer, etc.) and the memory usage retains the efficient $O((2-\epsilon)L\log L)$ scaling, with no sacrifice in inference speed.
- We incorporate a tailored sparsity mechanism into the attention module, which effectively enlarges the model's ERF and enables it to better identify informative time steps within long sequences. This sparsity-enhanced attention improves the model's capacity to model long-term dependencies without relying on excessively large patch sizes or heavy architectural modifications.
- We also introduce a novel Seasonal–Trend Decomposition scheme to enrich the input representation of the time series.

2 Preliminary

Consider a dynamical system observed from time 0 through T at discrete time intervals. The periodically *recorded* observations, will generate a sequence $\hat{\mathbf{X}}_0, \hat{\mathbf{X}}_1, \ldots, \hat{\mathbf{X}}_T$. The time series forecasting problem is to predict the most likely length-K sequence in the future given the previous J observations (including the current one):

$$\tilde{\mathbf{X}}_{t+1}, \ldots, \tilde{\mathbf{X}}_{t+K} = \text{argmax}\, \mathbb{P}(\mathbf{X}_{t+1}, \ldots, \mathbf{X}_{t+K} \mid \hat{\mathbf{X}}_{t-J+1}, \hat{\mathbf{X}}_{t-J+2}, \ldots \hat{\mathbf{X}}_t).$$

2.1 Sequence to Sequence Forecasting Framework

In the rolling forecasting setting with a fixed size window, a t-th sequence input is a series of vectors $\mathcal{X}^t = \{\mathbf{x}_1^t, \ldots, \mathbf{x}_{L_x}^t : \mathbf{x}_i^t \in \mathbb{R}^{d_x}\}$ of L_x elements and its target is the corresponding vectors $\mathcal{Y}^t = \{\mathbf{y}_1^t, \ldots, \mathbf{y}_{L_y}^t : \mathbf{y}_i^t \in \mathbb{R}^{d_y}\}$. In particular, Long Sequence Time-series Forecasting (LSTF) problem aims to predict the target sequence $\mathcal{Y}^t$ from the input sequence $\mathcal{X}^t$ and encourages a longer output's length L_y than previous works.

Our proposed model Sparseformer follows the encoder-decoder mechanism and the overall architecture is given in Figure 1. Mostly, encoder-decoder models are devised to "encode" the input representations $\mathcal{X}^t$ into hidden state representations $\mathcal{H}^t = \{\mathbf{h}_1^t, \ldots, \mathbf{h}_{L_h}^t\}$ and "decode" the output representations (predictions) $\hat{\mathcal{Y}}^t$ from $\mathcal{H}^t$.

2.2 Input Representation

Since the observations are recorded at fixed time intervals, the temporal order of points carries essential structural information. To preserve this local context information, we incorporate a positional encoding:

$$\text{PE}_{(pos,2j)} = \sin(\frac{pos}{(2L_x)^{\frac{2j}{d_{model}}}}), \quad \text{PE}_{(pos,2j+1)} = \cos(\frac{pos}{(2L_x)^{\frac{2j}{d_{model}}}}),$$

where $j \in 1, \cdots, [d_{model}/2]$. Each global timestamp is labeled by a learnable stamp embedding $\text{SE}_{(pos)}$ with a limited vocabulary size, enabling the computation of self-attention regarding similarity to have access to global context in an affordable manner. To align the dimensions, we project the scalar context x_i^t into $u_i^t \in \mathbb{R}^{d_{model}}$ with 1-D convolution filters. Thus, we have the feeding vector

$$\mathcal{X}_{\text{feed}[i]}^t = \alpha \text{u}_i^t + \text{PE}_{L_x \times (t-1)+i} + \sum_p [\text{SE}_{\text{L}_\text{x} \times (\text{t}-1)+\text{i}}]_p,$$

where $i \in 1, \cdots, L_x$, and α (always set to 1) is the factor balancing the magnitude between the scalar projection and local/global embeddings. We have also introduced a seasonal–trend decomposition to enrich the input representation. The detailed formulation and preprocessing procedure are provided in Appendix A.

2.3 Forecasting Objective

The model is trained to minimize a discrepancy between predicted values and ground truth, typically using mean squared error:

$$\mathcal{L} = \frac{1}{K} \sum_{k=1}^{K} \left\| \tilde{\mathbf{X}}_{t+k} - \mathbf{X}_{t+k} \right\|^2.$$

3 The Model and Methodology

3.1 Sparseformer: Structure Overview

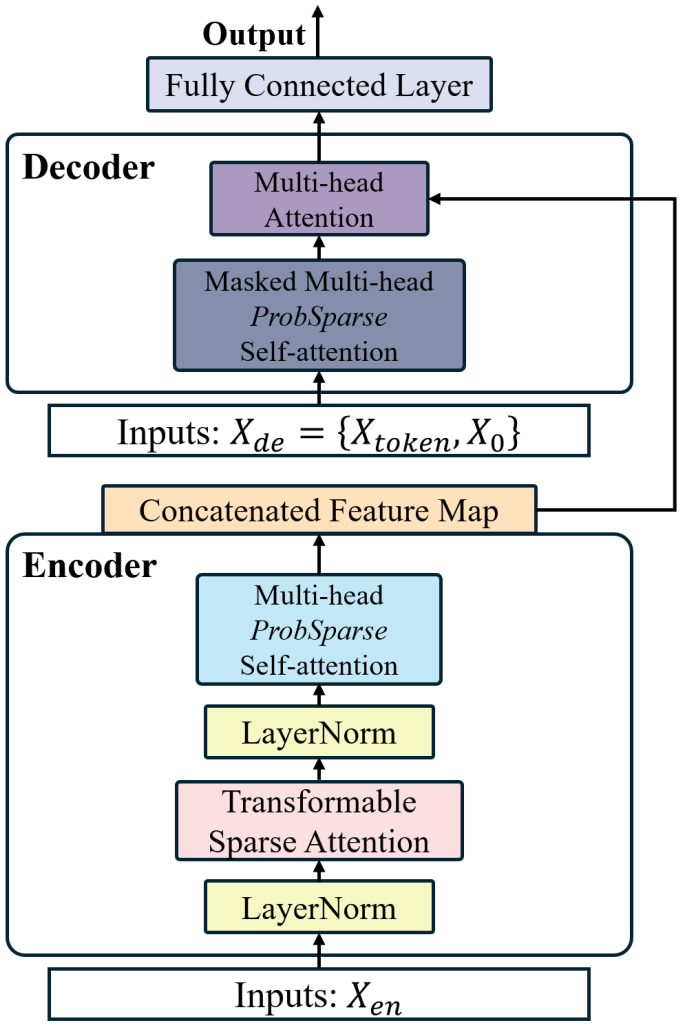

Figure 1: Structure Overview of Sparseformer

The raw time series and their associated timestamp features are first fed into the model. The numerical sequence itself is treated as a multi-dimensional signal and mapped into a unified high-dimensional

representation space through a convolution layer. At the same time, each time-step's position is assigned a fixed sinusoidal positional encoding to provide relative positional information. In addition, timestamp information is independently embedded into high-dimensional vectors. The numerical, positional, and temporal embeddings are then summed to form a per-step representation, which serves as the encoder input.

The encoder is composed of multiple layers, each containing two attention mechanisms and a feed-forward network. The first mechanism is deformable attention: based on the input representation, the model dynamically predicts a set of continuous sampling locations to construct the keys and values for attention, while the original sequence positions act as queries. Each encoder layer also performs a multi-head ProbSparce self-attention operation, which uses the entire sequence to calculate querys, keys and values to capture global dependencies. Every attention operation is followed by residual connections and layer normalization, ensuring stable information flow across layers. After stacking multiple layers, the encoder output yields a multi-scale time-series representation that integrates both local and global information.

The decoder follows a similar structure but is designed to generate future predictions. It first applies the same three types of embeddings to the "known historical segment and placeholder positions for future steps." In each decoder layer, the model performs causally masked self-attention to ensure that each future position can only access past information. This is followed by cross-attention, which allows decoder representations to extract relevant information from the encoder output for conditional generation, and then a feed-forward transformation. After passing through multiple decoder layers, the output remains in a high-dimensional latent space. Finally, a linear projection maps these latent vectors to real-valued predictions, and the model extracts the required number of forecast steps from the end of the decoder sequence as the final output.

3.1.1 Input and Embedding

Given an input time series

$$X \in \mathbb{R}^{B \times L_x \times C},$$

where B stands for Batch size, L_x stands for the length of X, C stands for the number of variables (Batch size is the number of time series we put into a model in one training step, and $L_x$is the length of the input. i.e. , L_x is the "look back window", and B is how many look back window we have at once.), and its associated timestamp features

$$M_x \in \mathbb{R}^{B \times L_x \times d_t},$$

where d_t stands for the dimension of the input timestamps.

The model first constructs three types of embeddings:

Value embedding (Conv1D):

$$E_x^{\text{val}} = \text{Conv1D}(X) \in \mathbb{R}^{B \times L_x \times D},$$

where D stands for the model's dimension.

Positional embedding (sinusoidal):

$$E_x^{\text{pos}} = \text{PositionalEmbedding}(L_x) \in \mathbb{R}^{1 \times L_x \times D}.$$

Temporal embedding:

$$E_x^{\text{temp}} = \text{TemporalEmbedding}(M_x) \in \mathbb{R}^{B \times L_x \times D}.$$

The encoder input is the sum of the three embeddings:

$$H^{(0)} = E_x^{\text{val}} + E_x^{\text{pos}} + E_x^{\text{temp}} \ \in \ \mathbb{R}^{B \times L_x \times D}.$$

3.1.2 Encoder

The encoder consists of L_e layers, each containing a transformable sparse attention module, a ProbSparse self-attention module, and a feed-forward network.

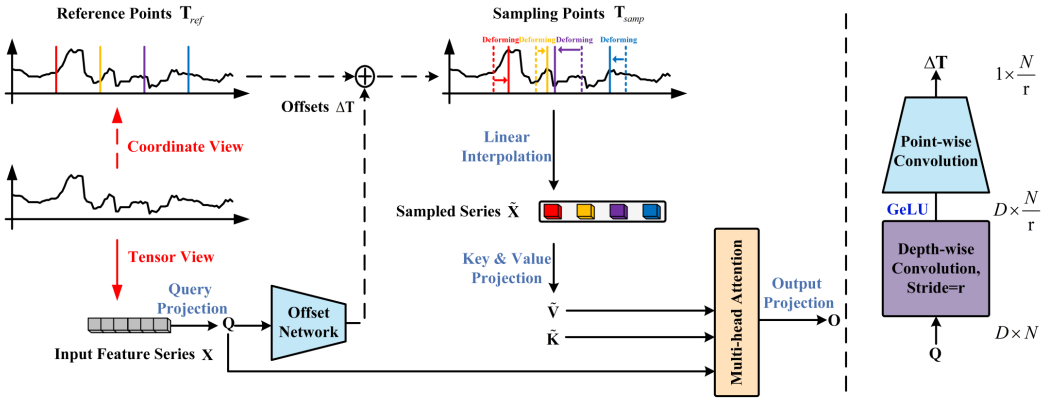

Figure 2: Transformable Sparse Attention

Transformable Sparse Attention: Figure 2 introduces the detailed process of our Transformable Sparse attention. For the (l)-th encoder layer, sampling locations are predicted by

$$P^{(l)} = f_\theta\big(H^{(l-1)}\big) \in \mathbb{R}^{B\times L_x\times K},$$

where f_θ stands for the offset network, $H^{(l-1)}$ stands for the hidden representation of $l-1$-th layer and K stands for the number of sampling points.

The sequence is sampled at continuous locations through interpolation:

$$\hat{X}^{(l)} = \text{Interp}\Big(H^{(l-1)}, P^{(l)}\Big).$$

Queries, keys, and values are obtained as

$$Q = H^{(l-1)}W_Q, \quad K = \hat{X}^{(l)}W_K, \quad V = \hat{X}^{(l)}W_V.$$

Transformable Sparse attention produces

$$Z^{(l)}_{\text{TSA}} = \text{Attention}(Q, K, V).$$

Residual connection and layer normalization yield

$$\tilde{H}^{(l)} = \text{LayerNorm}\Big(H^{(l-1)} + Z^{(l)}_{\text{TSA}}\Big).$$

ProbSparse Self-Attention:

$$Q' = \tilde{H}^{(l)}W'_Q, \quad K' = \tilde{H}^{(l)}W'_K, \quad V' = \tilde{H}^{(l)}W'_V.$$

Using the ProbSparse top-u selection:

$$Z^{(l)}_{\text{prob}} = \text{ProbSparseAttention}(Q', K', V').$$

Residual and normalization:

$$\bar{H}^{(l)} = \text{LayerNorm}\Big(\tilde{H}^{(l)} + Z^{(l)}_{\text{prob}}\Big).$$

Feed-forward network:

$$F^{(l)} = \sigma\Big(\bar{H}^{(l)}W_1 + b_1\Big)W_2 + b_2.$$

$$H^{(l)} = \text{LayerNorm}\Big(\bar{H}^{(l)} + F^{(l)}\Big).$$

The final encoder output is

$$H^{\text{enc}} = H^{(L_e)} \in \mathbb{R}^{B\times L'_x\times D}.$$

3.1.3 Decoder

The decoder is composed of L_d stacked layers. Each layer consists of (1) a masked self-attention module, (2) a cross-attention module, and (3) a position-wise feed-forward network (FFN). Given the decoder input sequence

$$Y^{(0)} \in \mathbb{R}^{B\times L_y\times D},$$

and the encoder output

$$H^{\text{enc}} \in \mathbb{R}^{B\times L'_x\times D},$$

the computations of the (l)-th decoder layer are described below.

Masked Self-Attention: The decoder first applies masked self-attention:

$$Z^{(l)}_{\text{self}} = \text{softmax}\left(\frac{(Y^{(l-1)}W_Q)(Y^{(l-1)}W_K)^\top}{\sqrt{D}} + M_y\right)(Y^{(l-1)}W_V),$$

where $M_y[i,j] = \begin{cases} 0, & j \le i, \\ -\infty, & j > i, \end{cases}$ is a causal mask preventing positions from attending to the future.
A residual connection and layer normalization are then applied:

$$\tilde{Y}^{(l)} = \text{LayerNorm}\Big(Y^{(l-1)} + Z^{(l)}_{\text{self}}\Big).$$

Cross-Attention: For cross-attention, the query is projected from the decoder representation $\tilde{Y}^{(l)}$, while the keys and values are projected from the encoder output H^{enc}:

$$Q_c = \tilde{Y}^{(l)}W^c_Q, \quad K_c = H^{\text{enc}}W^c_K, \quad V_c = H^{\text{enc}}W^c_V.$$

Then the cross-attention computation is given by:

$$Z^{(l)}_{\text{cross}} = \text{softmax}\left(\frac{Q_cK_c^\top}{\sqrt{D}} + M_{\text{cross}}\right)V_c,$$

where M_{cross} is the attention mask applied between the decoder queries and encoder memory. Again, residual connection and normalization are applied:

$$\bar{Y}^{(l)} = \text{LayerNorm}\Big(\tilde{Y}^{(l)} + Z^{(l)}_{\text{cross}}\Big).$$

Feed-Forward Network: A feed-forward network further transforms the representation position-wise:

$$F^{(l)} = \text{Dropout}\Big(\sigma\Big(\bar{Y}^{(l)}W_1 + b_1\Big)\Big)W_2 + b_2,$$

where the activation $\sigma(\cdot)$ can be ReLU or GELU, and Dropout randomly masks a portion of activations during training to reduce overfitting:

$$\text{Dropout}(h) = h \odot m, \quad m_i \sim \text{Bernoulli}(1-p).$$

The final output of the layer is computed as:

$$Y^{(l)} = \text{LayerNorm}\Big(\bar{Y}^{(l)} + F^{(l)}\Big).$$

Decoder Output: After all L_d layers, the final decoder representation is:

$$H^{\text{dec}} = Y^{(L_d)} \in \mathbb{R}^{B\times L_y\times D}.$$

4 Experiments

4.1 Datasets

We perform experiments on 3 datasets, including:

ETTh1 (Electricity Transformer Temperature): The dataset contains two years of measurements collected from a county-level power grid in China. To examine the performance of the LSTF task, we adopt the ETTh1 subset, which provides 1-hour–resolution data. Each time step includes the target variable "oil temperature" together with six associated power-load features. The train/val/test is 12/4/4 months.

WTH: This dataset contains local climatological data for nearly 1,600 U.S. locations, 4 years from 2010 to 2013, where data points are collected every 1 hour. Each data point consists of the target value "wet bulb" and 11 climate features. The train/val/test is 28/10/10 months.

ECL (Electricity Consuming Load): The dataset records the electricity consumption (kWh) of 321 clients. The raw records were converted into an hourly consumption series spanning two years and use "MT 320" as the prediction target. The train/val/test is 15/3/4 months.

4.2 Experimental Details

Baselines: We have selected 3 time-series forecasting methods as comparison, including Informer [Zhou et al., 2021], Reformer [Kitaev et al., 2020], and DeepAR [Salinas et al., 2020].

Hyper-parameter Tuning: Following the configurations suggested in the Informer [Zhou et al., 2021], we directly adopt their recommended hyperparameter configuration in our experiments. The Informer model is implemented with a 3-layer encoder stack and an additional 1-layer distilling stack (1/4 input), together with a 2-layer decoder. All proposed methods are trained using the Adam optimizer, with the learning rate initialized at $1e^{-4}$ and decaying two times smaller every epoch. The training process runs for 8 epochs with early stopping, and all baseline methods are configured according to the original recommendations. The batch size is fixed at 32.

Metrics: We use 2 evaluation metrics, including $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2$ and $\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y-\hat{y}|$ on each prediction window, and roll the whole set with stride = 1.

Platform: All the models were trained/tested on a single Nvidia RTX5070Ti 16GB GPU.

4.3 Results and Analysis

Table 1 summarizes the univariate evaluation results of all the methods on 3 datasets. The best results are highlighted in boldface.

Table 1: Univariate LSTF results on 3 datasets (prediction length = 48)

Methods	Sparseformer		Informer		Reformer		DeepAR	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	**0.59**	**0.56**	0.73	0.65	2.76	2.72	5.24	5.19
WTH	**0.32**	**0.37**	**0.39**	0.43	0.48	0.47	1.13	1.13
ECL	**0.24**	**0.35**	0.55	0.58	1.52	1.54	3.92	3.83

From Table 1, it is evident that Sparseformer consistently outperforms the three baseline models across all datasets. This demonstrates that the proposed Transformable Sparse Attention effectively enlarges the ERF and consequently enhances predictive performance. Nevertheless, this conclusion is preliminary, as it is derived from a limited set of experiments constrained by time. To strengthen the reliability of our findings, we plan to evaluate the model under a broader range of hyperparameter configurations and conduct comprehensive ablation studies in the future.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys*, 55(6):1–36, 2022.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers, 2019. URL https://arxiv.org/abs/1904.10509.

Shuiwang Ji, Yaochen Xie, and Hongyang Gao. A mathematical view of attention models in deep learning. *Texas A&M University: College Station, TX, USA*, 2019.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020. URL https://arxiv.org/abs/2001.04451.

Xiangjie Kong, Zhenghao Chen, Weiyao Liu, Kaili Ning, Lechao Zhang, Syauqie Muhammad Marier, Yichen Liu, Yuhao Chen, and Feng Xia. Deep learning for time series forecasting: a survey. *International Journal of Machine Learning and Cybernetics*, 16(7–8):5079–5112, February 2025. ISSN 1868-808X. doi: 10.1007/s13042-025-02560-w. URL http://dx.doi.org/10.1007/s13042-025-02560-w.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, 2020. URL https://arxiv.org/abs/1907.00235.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379 (2194):20200209, February 2021a. ISSN 1471-2962. doi: 10.1098/rsta.2020.0209. URL http://dx.doi.org/10.1098/rsta.2020.0209.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021b.

Donghao Luo and Xue Wang. DeformableTST: Transformer for time series forecasting without over-reliance on patching. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=B1Iq1EOiVU.

Amal Mahmoud and Ammar Mohammed. A survey on deep learning for time-series forecasting. In *Machine learning and big data analytics paradigms: analysis, applications and challenges*, pages 365–392. Springer, 2020.

Ricardo P Masini, Marcelo C Medeiros, and Eduardo F Mendes. Machine learning advances for time series forecasting. *Journal of economic surveys*, 37(1):76–111, 2023.

John A Miller, Mohammed Aldosari, Farah Saeed, Nasid Habib Barna, Subas Rana, I Budak Arpinar, and Ninghao Liu. A survey of deep learning and foundation models for time series forecasting. *arXiv preprint arXiv:2401.13912*, 2024.

Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023. URL https://arxiv.org/abs/2211.14730.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding, 2020. URL https://arxiv.org/abs/1911.02972.

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.

Francis E.H Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001. ISSN 0305-0483. doi: https://doi.org/10.1016/S0305-0483(01)00026-3. URL https://www.sciencedirect.com/science/article/pii/S0305048301000263.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. URL https://arxiv.org/abs/2006.04768.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2023. URL https://arxiv.org/abs/2202.07125.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2022. URL https://arxiv.org/abs/2106.13008.

Sira Yongchareon. Ai-driven intelligent financial forecasting: A comparative study of advanced deep learning models for long-term stock market prediction. *Machine Learning and Knowledge Extraction*, 7(3), 2025. ISSN 2504-4990. doi: 10.3390/make7030061. URL https://www.mdpi.com/2504-4990/7/3/61.

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, 2021. URL https://arxiv.org/abs/2012.07436.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting, 2022. URL https://arxiv.org/abs/2201.12740.

A Seasonal–Trend Decomposition

In this appendix we describe the seasonal–trend decomposition used in our data preprocessing pipeline.

A.1 STL-based decomposition

Let $\{x_t\}_{t=1}^{T}$ denote a univariate time series obtained from the raw observations. For each dataset we specify a fundamental seasonal period P according to the sampling frequency . We apply the STL (Seasonal–Trend decomposition using Loess) procedure to decompose x_t into three additive components:

$$x_t = T_t + S_t + R_t, \qquad t = 1, \ldots, T,$$

where T_t stands for trend, S_t stands for Seasonal, R_t stands for residual.

A.2 Augmented input construction

Let $x_t \in \mathbb{R}$ denote the original scalar context at time t. After STL decomposition, we construct an augmented feature vector

$$\tilde{x}_t = \left[x_t,\ T_t,\ S_t,\ R_t\right]^\top \in \mathbb{R}^4,$$

which explicitly exposes the trend, seasonal, and residual dynamics to the model. For the multivariate setting, let $v_t \in \mathbb{R}^m$ be the original m-dimensional observation and let y_t be the designated target channel on which STL is applied. We then form

$$\tilde{v}_t = \left[v_t^\top,\ T_t,\ S_t,\ R_t\right]^\top \in \mathbb{R}^{m+3}.$$

To ensure numerical stability, feature-wise standardization is performed on the training split. Let μ_x, σ_x denote the empirical mean and standard deviation of the augmented inputs over the training interval, and let μ_y, σ_y be the corresponding statistics for the prediction target. We apply the transformations

$$\hat{\tilde{v}}_t = \frac{\tilde{v}_t - \mu_x}{\sigma_x}, \qquad \hat{y}_t = \frac{y_t - \mu_y}{\sigma_y},$$

for all t in the training, validation, and test sets. The normalized inputs $\hat{\tilde{v}}_t$ are then projected into the model dimension and combined with the positional and time-stamp embeddings as described in the main text.