Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting

Haoyi Zhou, [1] **Shanghang Zhang,** [2] **Jieqi Peng,** [1] **Shuai Zhang,** [1] **Jianxin Li,** [1]
Hui Xiong, [3] **Wancai Zhang** [4]

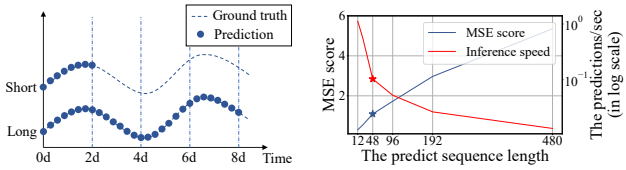
[1] Beihang University [2] UC Berkeley [3] Rutgers University [4] SEDD Company
{zhouhy, pengjq, zhangs, lijx}@act.buaa.edu.cn, shz@eecs.berkeley.edu, {xionghui,zhangwancaibuaa}@gmail.com

Abstract

Many real-world applications require the prediction of long sequence time-series, such as electricity consumption planning. Long sequence time-series forecasting (LSTF) demands a high prediction capacity of the model, which is the ability to capture precise long-range dependency coupling between output and input efficiently. Recent studies have shown the potential of Transformer to increase the prediction capacity. However, there are several severe issues with Transformer that prevent it from being directly applicable to LSTF, including quadratic time complexity, high memory usage, and inherent limitation of the encoder-decoder architecture. To address these issues, we design an efficient transformer-based model for LSTF, named Informer, with three distinctive characteristics: (i) a *ProbSparse* self-attention mechanism, which achieves $\mathcal{O}(L \log L)$ in time complexity and memory usage, and has comparable performance on sequences' dependency alignment. (ii) the self-attention distilling highlights dominating attention by halving cascading layer input, and efficiently handles extreme long input sequences. (iii) the generative style decoder, while conceptually simple, predicts the long time-series sequences at one forward operation rather than a step-by-step way, which drastically improves the inference speed of long-sequence predictions. Extensive experiments on four large-scale datasets demonstrate that Informer significantly outperforms existing methods and provides a new solution to the LSTF problem.

1 Introduction

Time-series forecasting is a critical ingredient across many domains, such as sensor network monitoring (Papadimitriou and Yu 2006), energy and smart grid management, economics and finance (Zhu and Shasha 2002), and disease propagation analysis (Matsubara et al. 2014). In these scenarios, we can leverage a substantial amount of time-series data on past behavior to make a forecast in the long run, namely long sequence time-series forecasting (LSTF). However, existing methods are mostly designed under short-term problem setting, like predicting 48 points or less (Hochreiter and Schmidhuber 1997; Li et al. 2018; Yu et al. 2017; Liu et al. 2019; Qin et al. 2017; Wen et al. 2017). The increasingly long sequences strain the models' prediction capacity to the point where this trend is holding the research on LSTF. As an empirical example, Fig.(1) shows the forecasting results on a real dataset, where the LSTM network predicts the hourly temperature of an electrical transformer station from the short-term period (12 points, 0.5 days) to the long-term period (480 points, 20 days). The overall performance gap is substantial when the prediction length is greater than 48 points (the solid star in Fig.(1b)), where the MSE rises to unsatisfactory performance, the inference speed gets sharp drop, and the LSTM model starts to fail.



(a) Sequence Forecasting. (b) Run LSTM on sequences.

Figure 1: (a) LSTF can cover an extended period than the short sequence predictions, making vital distinction in policy-planning and investment-protecting. (b) The prediction capacity of existing methods limits LSTF's performance. E.g., starting from length=48, MSE rises unacceptably high, and the inference speed drops rapidly.

The major challenge for LSTF is to enhance the prediction capacity to meet the increasingly long sequence demand, which requires (a) extraordinary long-range alignment ability and (b) efficient operations on long sequence inputs and outputs. Recently, Transformer models have shown superior performance in capturing long-range dependency than RNN models. The self-attention mechanism can reduce the maximum length of network signals traveling paths into the theoretical shortest $\mathcal{O}(1)$ and avoid the recurrent structure, whereby Transformer shows great potential for the LSTF problem. Nevertheless, the self-attention mechanism violates requirement (b) due to its L-quadratic computation and memory consumption on L-length inputs/outputs. Some large-scale Transformer models pour resources and yield impressive results on NLP tasks (Brown et al. 2020), but the training on dozens of GPUs and expensive deploying cost make theses models unaffordable on real-world LSTF problem. The efficiency of the self-attention mechanism and Transformer architecture becomes the bottleneck of applying them to LSTF problems. Thus, in this paper, we seek to answer the question: *can we improve Transformer models to*

be computation, memory, and architecture efficient, as well as maintaining higher prediction capacity?

Vanilla Transformer (Vaswani et al. 2017) has three significant limitations when solving the LSTF problem:

1. **The quadratic computation of self-attention.** The atom operation of self-attention mechanism, namely canonical dot-product, causes the time complexity and memory usage per layer to be $\mathcal{O}(L^2)$.
2. **The memory bottleneck in stacking layers for long inputs.** The stack of J encoder/decoder layers makes total memory usage to be $\mathcal{O}(J \cdot L^2)$, which limits the model scalability in receiving long sequence inputs.
3. **The speed plunge in predicting long outputs.** Dynamic decoding of vanilla Transformer makes the step-by-step inference as slow as RNN-based model (Fig.(1b)).

There are some prior works on improving the efficiency of self-attention. The Sparse Transformer (Child et al. 2019), LogSparse Transformer (Li et al. 2019), and Longformer (Beltagy, Peters, and Cohan 2020) all use a heuristic method to tackle limitation 1 and reduce the complexity of self-attention mechanism to $\mathcal{O}(L \log L)$, where their efficiency gain is limited (Qiu et al. 2019). Reformer (Kitaev, Kaiser, and Levskaya 2019) also achieves $\mathcal{O}(L \log L)$ with locally-sensitive hashing self-attention, but it only works on extremely long sequences. More recently, Linformer (Wang et al. 2020) claims a linear complexity $\mathcal{O}(L)$, but the project matrix can not be fixed for real-world long sequence input, which may have the risk of degradation to $\mathcal{O}(L^2)$. Transformer-XL (Dai et al. 2019) and Compressive Transformer (Rae et al. 2019) use auxiliary hidden states to capture long-range dependency, which could amplify limitation 1 and be adverse to break the efficiency bottleneck. All these works mainly focus on limitation 1, and the limitation 2&3 remains unsolved in the LSTF problem. To enhance the prediction capacity, we tackle all these limitations and achieve improvement beyond efficiency in the proposed Informer.

To this end, our work delves explicitly into these three issues. We investigate the sparsity in the self-attention mechanism, make improvements of network components, and conduct extensive experiments. The contributions of this paper are summarized as follows:

- We propose Informer to successfully enhance the prediction capacity in the LSTF problem, which validates the Transformer-like model's potential value to capture individual long-range dependency between long sequence time-series outputs and inputs.
- We propose *ProbSparse* self-attention mechanism to efficiently replace the canonical self-attention. It achieves the $\mathcal{O}(L \log L)$ time complexity and $\mathcal{O}(L \log L)$ memory usage on dependency alignments.
- We propose self-attention distilling operation to privilege dominating attention scores in J-stacking layers and sharply reduce the total space complexity to be $\mathcal{O}((2-\epsilon)L \log L)$, which helps receiving long sequence input.
- We propose generative style decoder to acquire long sequence output with only one forward step needed, simultaneously avoiding cumulative error spreading during the inference phase.

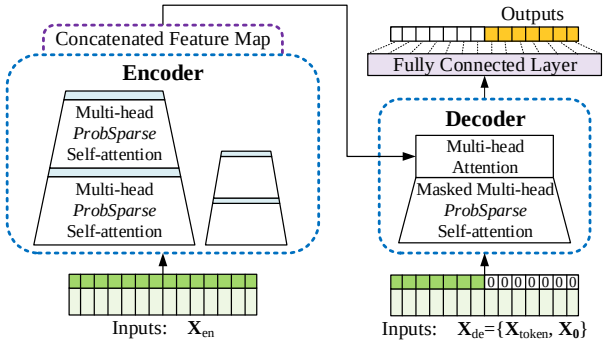


Figure 2: Informer model overview. Left: The encoder receives massive long sequence inputs (green series). We replace canonical self-attention with the proposed *ProbSparse* self-attention. The blue trapezoid is the self-attention distilling operation to extract dominating attention, reducing the network size sharply. The layer stacking replicas increase robustness. Right: The decoder receives long sequence inputs, pads the target elements into zero, measures the weighted attention composition of the feature map, and instantly predicts output elements (orange series) in a generative style.

2 Preliminary

We first provide the LSTF problem definition. Under the rolling forecasting setting with a fixed size window, we have the input $\mathcal{X}^t = \{\mathbf{x}_1^t, \dots, \mathbf{x}_{L_x}^t \mid \mathbf{x}_i^t \in \mathbb{R}^{d_x}\}$ at time t, and the output is to predict corresponding sequence $\mathcal{Y}^t = \{\mathbf{y}_1^t, \dots, \mathbf{y}_{L_y}^t \mid \mathbf{y}_i^t \in \mathbb{R}^{d_y}\}$. The LSTF problem encourages a longer output's length L_y than previous works (Cho et al. 2014; Sutskever, Vinyals, and Le 2014) and the feature dimension is not limited to univariate case ($d_y \geq 1$).

Encoder-decoder architecture Many popular models are devised to "encode" the input representations $\mathcal{X}^t$ into a hidden state representations $\mathcal{H}^t$ and "decode" an output representations $\mathcal{Y}^t$ from $\mathcal{H}^t = \{\mathbf{h}_1^t, \dots, \mathbf{h}_{L_h}^t\}$. The inference involves a step-by-step process named "dynamic decoding", where the decoder computes a new hidden state $\mathbf{h}_{k+1}^t$ from the previous state $\mathbf{h}_k^t$ and other necessary outputs from k-th step then predict the $(k+1)$-th sequence $\mathbf{y}_{k+1}^t$.

Input Representation A uniform input representation is given to enhance the global positional context and local temporal context of the time-series inputs. To avoid trivializing description, we put the details in Appendix B.

3 Methodology

Existing methods for time-series forecasting can be roughly grouped into two categories[1]. Classical time-series models serve as a reliable workhorse for time-series forecasting (Box et al. 2015; Ray 1990; Seeger et al. 2017; Seeger, Salinas, and Flunkert 2016), and deep learning techniques mainly develop an encoder-decoder prediction paradigm by using RNN and their variants (Hochreiter and Schmidhuber 1997; Li et al. 2018; Yu et al. 2017). Our proposed Informer holds the encoder-decoder architecture while targeting the

[1]Related work is in Appendix A due to space limitation.

LSTF problem. Please refer to Fig.(2) for an overview and the following sections for details.

Efficient Self-attention Mechanism

The canonical self-attention in (Vaswani et al. 2017) is defined based on the tuple inputs, i.e, query, key and value, which performs the scaled dot-product as $\mathcal{A}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\mathbf{V}$, where $\mathbf{Q}\in\mathbb{R}^{L_Q\times d}$, $\mathbf{K}\in\mathbb{R}^{L_K\times d}$, $\mathbf{V}\in\mathbb{R}^{L_V\times d}$ and d is the input dimension. To further discuss the self-attention mechanism, let $\mathbf{q}_i$, $\mathbf{k}_i$, $\mathbf{v}_i$ stand for the i-th row in $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ respectively. Following the formulation in (Tsai et al. 2019), the i-th query's attention is defined as a kernel smoother in a probability form:

$$\mathcal{A}(\mathbf{q}_i,\mathbf{K},\mathbf{V}) = \sum_j \frac{k(\mathbf{q}_i,\mathbf{k}_j)}{\sum_l k(\mathbf{q}_i,\mathbf{k}_l)}\mathbf{v}_j = \mathbb{E}_{p(\mathbf{k}_j|\mathbf{q}_i)}[\mathbf{v}_j]\ , \quad (1)$$

where $p(\mathbf{k}_j|\mathbf{q}_i) = k(\mathbf{q}_i,\mathbf{k}_j)/\sum_l k(\mathbf{q}_i,\mathbf{k}_l)$ and $k(\mathbf{q}_i,\mathbf{k}_j)$ selects the asymmetric exponential kernel $\exp(\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d})$. The self-attention combines the values and acquires outputs based on computing the probability $p(\mathbf{k}_j|\mathbf{q}_i)$. It requires the quadratic times dot-product computation and $\mathcal{O}(L_QL_K)$ memory usage, which is the major drawback when enhancing prediction capacity.

Some previous attempts have revealed that the distribution of self-attention probability has potential sparsity, and they have designed "selective" counting strategies on all $p(\mathbf{k}_j|\mathbf{q}_i)$ without significantly affecting the performance. The Sparse Transformer (Child et al. 2019) incorporates both the row outputs and column inputs, in which the sparsity arises from the separated spatial correlation. The LogSparse Transformer (Li et al. 2019) notices the cyclical pattern in self-attention and forces each cell to attend to its previous one by an exponential step size. The Longformer (Beltagy, Peters, and Cohan 2020) extends previous two works to more complicated sparse configuration. However, they are limited to theoretical analysis from following heuristic methods and tackle each multi-head self-attention with the same strategy, which narrows their further improvement.

To motivate our approach, we first perform a qualitative assessment on the learned attention patterns of the canonical self-attention. The "sparsity" self-attention score forms a long tail distribution (see Appendix C for details), i.e., a few dot-product pairs contribute to the major attention, and others generate trivial attention. Then, the next question is how to distinguish them?

Query Sparsity Measurement From Eq.(1), the i-th query's attention on all the keys are defined as a probability $p(\mathbf{k}_j|\mathbf{q}_i)$ and the output is its composition with values $\mathbf{v}$. The dominant dot-product pairs encourage the corresponding query's attention probability distribution away from the uniform distribution. If $p(\mathbf{k}_j|\mathbf{q}_i)$ is close to a uniform distribution $q(\mathbf{k}_j|\mathbf{q}_i) = 1/L_K$, the self-attention becomes a trivial sum of values $\mathbf{V}$ and is redundant to the residential input. Naturally, the "likeness" between distribution p and q can be used to distinguish the "important" queries. We measure the "likeness" through Kullback-Leibler divergence $KL(q||p) = \ln\sum_{l=1}^{L_K} e^{\mathbf{q}_i\mathbf{k}_l^\top/\sqrt{d}} - \frac{1}{L_K}\sum_{j=1}^{L_K}\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d} - \ln L_K$. Dropping the constant, we define the i-th query's sparsity measurement as

$$M(\mathbf{q}_i,\mathbf{K}) = \ln\sum_{j=1}^{L_K} e^{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}} - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}} \quad , \quad (2)$$

where the first term is the Log-Sum-Exp (LSE) of $\mathbf{q}_i$ on all the keys, and the second term is the arithmetic mean on them. If the i-th query gains a larger $M(\mathbf{q}_i,\mathbf{K})$, its attention probability p is more "diverse" and has a high chance to contain the dominate dot-product pairs in the header field of the long tail self-attention distribution.

***ProbSparse* Self-attention** Based on the proposed measurement, we have the *ProbSparse* self-attention by allowing each key to only attend to the u dominant queries:

$$\mathcal{A}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Softmax}(\frac{\overline{\mathbf{Q}}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V} \quad , \quad (3)$$

where $\overline{\mathbf{Q}}$ is a sparse matrix of the same size of $\mathbf{q}$ and it only contains the Top-u queries under the sparsity measurement $M(\mathbf{q},\mathbf{K})$. Controlled by a constant sampling factor c, we set $u = c\cdot\ln L_Q$, which makes the *ProbSparse* self-attention only need to calculate $\mathcal{O}(\ln L_Q)$ dot-product for each query-key lookup and the layer memory usage maintains $\mathcal{O}(L_K\ln L_Q)$. Under the multi-head perspective, this attention generates different sparse query-key pairs for each head, which avoids severe information loss in return.

However, the traversing of all the queries for the measurement $M(\mathbf{q}_i,\mathbf{K})$ requires calculating each dot-product pairs, i.e., quadratically $\mathcal{O}(L_QL_K)$, besides the LSE operation has the potential numerical stability issue. Motivated by this, we propose an empirical approximation for the efficient acquisition of the query sparsity measurement.

Lemma 1. *For each query $\mathbf{q}_i\in\mathbb{R}^d$ and $\mathbf{k}_j\in\mathbb{R}^d$ in the keys set $\mathbf{K}$, we have the bound as $\ln L_K \le M(\mathbf{q}_i,\mathbf{K}) \le \max_j\{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\{\mathbf{q}_i\mathbf{k}_j^\top/\sqrt{d}\} + \ln L_K$. When $\mathbf{q}_i\in\mathbf{K}$, it also holds.*

From the Lemma 1 (proof is given in Appendix D.1), we propose the max-mean measurement as

$$\overline{M}(\mathbf{q}_i,\mathbf{K}) = \max_j\{\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}}\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{\mathbf{q}_i\mathbf{k}_j^\top}{\sqrt{d}} \quad . \quad (4)$$

The range of Top-u approximately holds in the boundary relaxation with Proposition 1 (refers in Appendix D.2). Under the long tail distribution, we only need to randomly sample $U = L_K\ln L_Q$ dot-product pairs to calculate the $\overline{M}(\mathbf{q}_i,\mathbf{K})$, i.e., filling other pairs with zero. Then, we select sparse Top-u from them as $\overline{\mathbf{Q}}$. The max-operator in $\overline{M}(\mathbf{q}_i,\mathbf{K})$ is less sensitive to zero values and is numerical stable. In practice, the input length of queries and keys are typically equivalent in the self-attention computation, i.e $L_Q = L_K = L$ such that the total *ProbSparse* self-attention time complexity and space complexity are $\mathcal{O}(L\ln L)$.

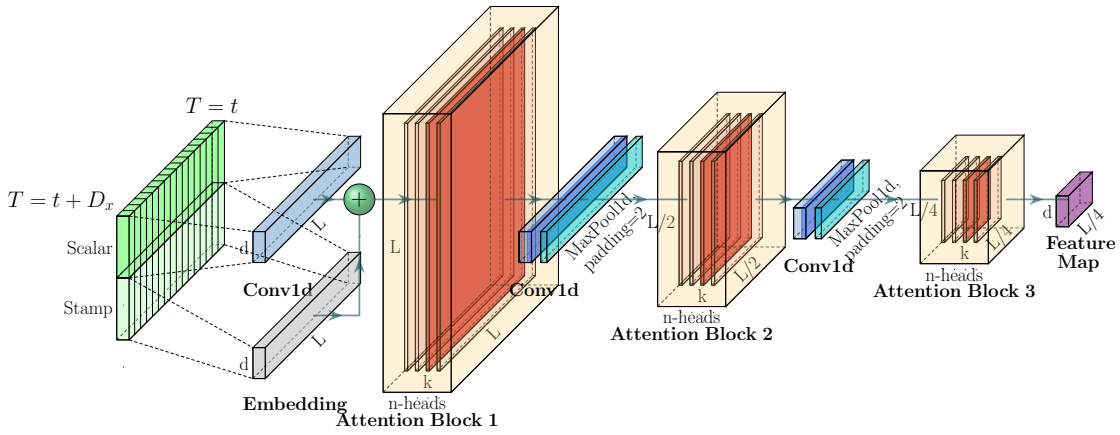


Figure 3: The single stack in Informer's encoder. (1) The horizontal stack stands for an individual one of the encoder replicas in Fig.(2). (2) The presented one is the main stack receiving the whole input sequence. Then the second stack takes half slices of the input, and the subsequent stacks repeat. (3) The red layers are dot-product matrixes, and they get cascade decrease by applying self-attention distilling on each layer. (4) Concatenate all stacks' feature maps as the encoder's output.

Encoder: Allowing for Processing Longer Sequential Inputs under the Memory Usage Limitation

The encoder is designed to extract the robust long-range dependency of the long sequential inputs. After the input representation, the t-th sequence input $\mathcal{X}^t$ has been shaped into a matrix $\mathbf{X}^t_{\text{en}} \in \mathbb{R}^{L_x \times d_{\text{model}}}$. We give a sketch of the encoder in Fig.(3) for clarity.

Self-attention Distilling As the natural consequence of the *ProbSparse* self-attention mechanism, the encoder's feature map has redundant combinations of value $\mathbf{V}$. We use the distilling operation to privilege the superior ones with dominating features and make a focused self-attention feature map in the next layer. It trims the input's time dimension sharply, seeing the n-heads weights matrix (overlapping red squares) of Attention blocks in Fig.(3). Inspired by the dilated convolution (Yu, Koltun, and Funkhouser 2017; Gupta and Rush 2017), our "distilling" procedure forwards from j-th layer into $(j+1)$-th layer as:

$$\mathbf{X}^t_{j+1} = \text{MaxPool}\left(\text{ELU}(\text{Conv1d}([\mathbf{X}^t_j]_{\text{AB}}))\right) \quad , \quad (5)$$

where $[\cdot]_{\text{AB}}$ represents the attention block. It contains the Multi-head *ProbSparse* self-attention and the essential operations, where Conv1d$(\cdot)$ performs an 1-D convolutional filters (kernel width=3) on time dimension with the ELU$(\cdot)$ activation function (Clevert, Unterthiner, and Hochreiter 2016). We add a max-pooling layer with stride 2 and downsample $\mathbf{X}^t$ into its half slice after stacking a layer, which reduces the whole memory usage to be $\mathcal{O}((2-\epsilon)L\log L)$, where ϵ is a small number. To enhance the robustness of the distilling operation, we build replicas of the main stack with halving inputs, and progressively decrease the number of self-attention distilling layers by dropping one layer at a time, like a pyramid in Fig.(2), such that their output dimension is aligned. Thus, we concatenate all the stacks' outputs and have the final hidden representation of encoder.

Decoder: Generating Long Sequential Outputs Through One Forward Procedure

We use a standard decoder structure (Vaswani et al. 2017) in Fig.(2), and it is composed of a stack of two identical multi-head attention layers. However, the generative inference is employed to alleviate the speed plunge in long prediction. We feed the decoder with the following vectors as

$$\mathbf{X}^t_{\text{de}} = \text{Concat}(\mathbf{X}^t_{\text{token}}, \mathbf{X}^t_{\mathbf{0}}) \in \mathbb{R}^{(L_{\text{token}}+L_y)\times d_{\text{model}}} \quad , \quad (6)$$

where $\mathbf{X}^t_{\text{token}} \in \mathbb{R}^{L_{\text{token}}\times d_{\text{model}}}$ is the start token, $\mathbf{X}^t_{\mathbf{0}} \in \mathbb{R}^{L_y \times d_{\text{model}}}$ is a placeholder for the target sequence (set scalar as 0). Masked multi-head attention is applied in the *ProbSparse* self-attention computing by setting masked dot-products to $-\infty$. It prevents each position from attending to coming positions, which avoids auto-regressive. A fully connected layer acquires the final output, and its outsize d_y depends on whether we are performing a univariate forecasting or a multivariate one.

Generative Inference Start token is efficiently applied in NLP's "dynamic decoding" (Devlin et al. 2018), and we extend it into a generative way. Instead of choosing specific flags as the token, we sample a L_{token} long sequence in the input sequence, such as an earlier slice before the output sequence. Take predicting 168 points as an example (7-day temperature prediction in the experiment section), we will take the known 5 days before the target sequence as "start-token", and feed the generative-style inference decoder with $\mathbf{X}_{\text{de}} = \{\mathbf{X}_{5d}, \mathbf{X_0}\}$. The $\mathbf{X_0}$ contains target sequence's time stamp, i.e., the context at the target week. Then our proposed decoder predicts outputs by one forward procedure rather than the time consuming "dynamic decoding" in the conventional encoder-decoder architecture. A detailed performance comparison is given in the computation efficiency section.

Loss function We choose the MSE loss function on prediction w.r.t the target sequences, and the loss is propagated back from the decoder's outputs across the entire model.

Methods		Informer		Informer†		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	24	0.098	0.247	**0.092**	**0.246**	0.103	0.259	0.222	0.389	0.114	0.272	0.107	0.280	0.108	0.284	0.115	0.275
	48	**0.158**	**0.319**	0.161	0.322	0.167	0.328	0.284	0.445	0.193	0.358	0.162	0.327	0.175	0.424	0.168	0.330
	168	**0.183**	**0.346**	0.187	0.355	0.207	0.375	1.522	1.191	0.236	0.392	0.239	0.422	0.396	0.504	1.224	0.763
	336	0.222	0.387	**0.215**	**0.369**	0.230	0.398	1.860	1.124	0.590	0.698	0.445	0.552	0.468	0.593	1.549	1.820
	720	0.269	0.435	**0.257**	**0.421**	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253
$ETTh_2$	24	**0.093**	**0.240**	0.099	0.241	0.102	0.255	0.263	0.437	0.155	0.307	0.098	0.263	3.554	0.445	0.199	0.381
	48	**0.155**	**0.314**	0.159	0.317	0.169	0.348	0.458	0.545	0.190	0.348	0.163	0.341	3.190	0.474	0.304	0.462
	168	**0.232**	**0.389**	0.235	0.390	0.246	0.422	1.029	0.879	0.385	0.514	0.255	0.414	2.800	0.595	2.145	1.068
	336	0.263	**0.417**	**0.258**	0.423	0.267	0.437	1.668	1.228	0.558	0.606	0.604	0.607	2.753	0.738	2.096	2.543
	720	**0.277**	**0.431**	0.285	0.442	0.303	0.493	2.030	1.721	0.640	0.681	0.429	0.580	2.878	1.044	3.355	4.664
$ETTm_1$	24	**0.030**	**0.137**	0.034	0.160	0.065	0.202	0.095	0.228	0.121	0.233	0.091	0.243	0.090	0.206	0.120	0.290
	48	0.069	0.203	**0.066**	**0.194**	0.078	0.220	0.249	0.390	0.305	0.411	0.219	0.362	0.179	0.306	0.133	0.305
	96	0.194	**0.372**	**0.187**	0.384	0.199	0.386	0.920	0.767	0.287	0.420	0.364	0.496	0.272	0.399	0.194	0.396
	288	**0.401**	0.554	0.409	**0.548**	0.411	0.572	1.108	1.245	0.524	0.584	0.948	0.795	0.462	0.558	0.452	0.574
	672	**0.512**	**0.644**	0.519	0.665	0.598	0.702	1.793	1.528	1.064	0.873	2.437	1.352	0.639	0.697	2.747	1.174
Weather	24	**0.117**	**0.251**	0.119	0.256	0.136	0.279	0.231	0.401	0.131	0.254	0.128	0.274	0.219	0.355	0.302	0.433
	48	**0.178**	0.318	0.185	**0.316**	0.206	0.356	0.328	0.423	0.190	0.334	0.203	0.353	0.273	0.409	0.445	0.536
	168	**0.266**	**0.398**	0.269	0.404	0.309	0.439	0.654	0.634	0.341	0.448	0.293	0.451	0.503	0.599	2.441	1.142
	336	**0.297**	**0.416**	0.310	0.422	0.359	0.484	1.792	1.093	0.456	0.554	0.585	0.644	0.728	0.730	1.987	2.468
	720	**0.359**	**0.466**	0.361	0.471	0.388	0.499	2.087	1.534	0.866	0.809	0.499	0.596	1.062	0.943	3.859	1.144
ECL	48	0.239	0.359	0.238	0.368	0.280	0.429	0.971	0.884	0.493	0.539	**0.204**	**0.357**	0.879	0.764	0.524	0.595
	168	0.447	0.503	0.442	0.514	0.454	0.529	1.671	1.587	0.723	0.655	**0.315**	**0.436**	1.032	0.833	2.725	1.273
	336	0.489	0.528	0.501	0.552	0.514	0.563	3.528	2.196	1.212	0.898	**0.414**	**0.519**	1.136	0.876	2.246	3.077
	720	**0.540**	**0.571**	0.543	0.578	0.558	0.609	4.891	4.047	1.511	0.966	0.563	0.595	1.251	0.933	4.243	1.415
	960	**0.582**	**0.608**	0.594	0.638	0.624	0.645	7.019	5.105	1.545	1.006	0.657	0.683	1.370	0.982	6.901	4.264
Count		32		12		0		0		0		6		0		0	

Table 1: Univariate long sequence time-series forecasting results on four datasets (five cases).

4 Experiment

Datasets

We extensively perform experiments on four datasets, including 2 collected real-world datasets for LSTF and 2 public benchmark datasets.

ETT (Electricity Transformer Temperature)[2]: The ETT is a crucial indicator in the electric power long-term deployment. We collected 2-year data from two separated counties in China. To explore the granularity on the LSTF problem, we create separate datasets as $\{ETTh_1, ETTh_2\}$ for 1-hour-level and $ETTm_1$ for 15-minute-level. Each data point consists of the target value "oil temperature" and 6 power load features. The train/val/test is 12/4/4 months.

ECL (Electricity Consuming Load)[3]: It collects the electricity consumption (Kwh) of 321 clients. Due to the missing data (Li et al. 2019), we convert the dataset into hourly consumption of 2 years and set 'MT_320' as the target value. The train/val/test is 15/3/4 months.

Weather [4]: 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.

[2]We collected the ETT dataset and published it at https://github.com/zhouhaoyi/ETDataset.

[3]ECL dataset was acquired at https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.

[4]Weather dataset was acquired at https://www.ncei.noaa.gov/data/local-climatological-data/.

Experimental Details

We briefly summarize basics, and more information on network components and setups are given in Appendix E.

Baselines: We have selected five time-series forecasting methods as comparison, including ARIMA (Ariyo, Adewumi, and Ayo 2014), Prophet (Taylor and Letham 2018), LSTMa (Bahdanau, Cho, and Bengio 2015), LSTnet (Lai et al. 2018) and DeepAR (Flunkert, Salinas, and Gasthaus 2017). To better explore the *ProbSparse* self-attention's performance in our proposed Informer, we incorporate the canonical self-attention variant (Informer†), the efficient variant Reformer (Kitaev, Kaiser, and Levskaya 2019) and the most related work LogSparse self-attention (Li et al. 2019) in the experiments. The details of network components are given in Appendix E.1.

Hyper-parameter tuning: We conduct grid search over the hyper-parameters, and detailed ranges are given in Appendix E.3. Informer contains a 3-layer stack and a 1-layer stack (1/4 input) in the encoder, and a 2-layer decoder. Our proposed methods are optimized with Adam optimizer, and its learning rate starts from $1e^{-4}$, decaying two times smaller every epoch. The total number of epochs is 8 with proper early stopping. We set the comparison methods as recommended, and the batch size is 32.
Setup: The input of each dataset is zero-mean normalized.

Methods		Informer		Informer†		LogTrans		Reformer		LSTMa		LSTnet	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTh_1$	24	**0.577**	**0.549**	0.620	0.577	0.686	0.604	0.991	0.754	0.650	0.624	1.293	0.901
	48	**0.685**	**0.625**	0.692	0.671	0.766	0.757	1.313	0.906	0.702	0.675	1.456	0.960
	168	**0.931**	**0.752**	0.947	0.797	1.002	0.846	1.824	1.138	1.212	0.867	1.997	1.214
	336	1.128	0.873	**1.094**	**0.813**	1.362	0.952	2.117	1.280	1.424	0.994	2.655	1.369
	720	**1.215**	**0.896**	1.241	0.917	1.397	1.291	2.415	1.520	1.960	1.322	2.143	1.380
$ETTh_2$	24	**0.720**	**0.665**	0.753	0.727	0.828	0.750	1.531	1.613	1.143	0.813	2.742	1.457
	48	**1.457**	**1.001**	1.461	1.077	1.806	1.034	1.871	1.735	1.671	1.221	3.567	1.687
	168	3.489	**1.515**	3.485	1.612	4.070	1.681	4.660	1.846	4.117	1.674	**3.242**	2.513
	336	2.723	1.340	2.626	**1.285**	3.875	1.763	4.028	1.688	3.434	1.549	**2.544**	2.591
	720	**3.467**	**1.473**	3.548	1.495	3.913	1.552	5.381	2.015	3.963	1.788	4.625	3.709
$ETTm_1$	24	0.323	**0.369**	**0.306**	0.371	0.419	0.412	0.724	0.607	0.621	0.629	1.968	1.170
	48	0.494	0.503	**0.465**	**0.470**	0.507	0.583	1.098	0.777	1.392	0.939	1.999	1.215
	96	**0.678**	0.614	0.681	**0.612**	0.768	0.792	1.433	0.945	1.339	0.913	2.762	1.542
	288	**1.056**	**0.786**	1.162	0.879	1.462	1.320	1.820	1.094	1.740	1.124	1.257	2.076
	672	**1.192**	**0.926**	1.231	1.103	1.669	1.461	2.187	1.232	2.736	1.555	1.917	2.941
Weather	24	**0.335**	**0.381**	0.349	0.397	0.435	0.477	0.655	0.583	0.546	0.570	0.615	0.545
	48	0.395	0.459	**0.386**	**0.433**	0.426	0.495	0.729	0.666	0.829	0.677	0.660	0.589
	168	**0.608**	**0.567**	0.613	0.582	0.727	0.671	1.318	0.855	1.038	0.835	0.748	0.647
	336	**0.702**	**0.620**	0.707	0.634	0.754	0.670	1.930	1.167	1.657	1.059	0.782	0.683
	720	**0.831**	**0.731**	0.834	0.741	0.885	0.773	2.726	1.575	1.536	1.109	0.851	0.757
ECL	48	0.344	**0.393**	**0.334**	0.399	0.355	0.418	1.404	0.999	0.486	0.572	0.369	0.445
	168	0.368	0.424	**0.353**	**0.420**	0.368	0.432	1.515	1.069	0.574	0.602	0.394	0.476
	336	0.381	**0.431**	0.381	0.439	**0.373**	0.439	1.601	1.104	0.886	0.795	0.419	0.477
	720	0.406	0.443	**0.391**	**0.438**	0.409	0.454	2.009	1.170	1.676	1.095	0.556	0.565
	960	**0.460**	**0.548**	0.492	0.550	0.477	0.589	2.141	1.387	1.591	1.128	0.605	0.599
Count		33		14		1		0		0		2	

Table 2: Multivariate long sequence time-series forecasting results on four datasets (five cases).

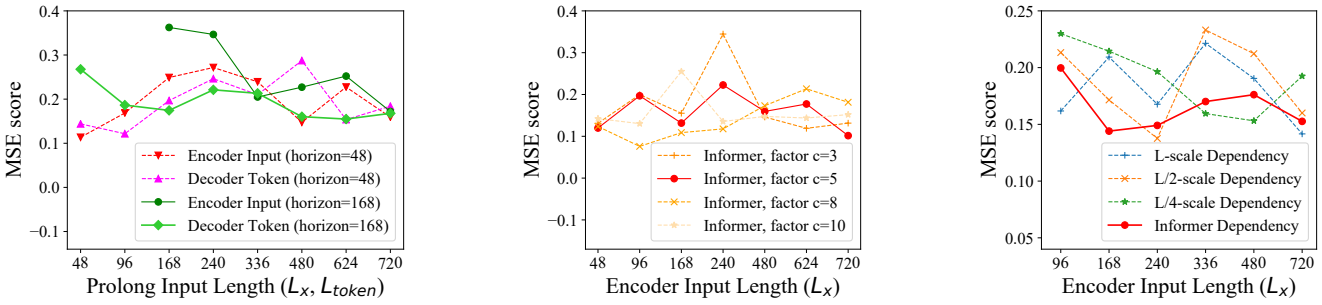
Under the LSTF settings, we prolong the prediction windows size L_y progressively, i.e., {1d, 2d, 7d, 14d, 30d, 40d} in {ETTh, ECL, Weather}, {6h, 12h, 24h, 72h, 168h} in ETTm. **Metrics:** We use two evaluation metrics, including MSE $= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}-\hat{\mathbf{y}})^2$ and MAE $= \frac{1}{n}\sum_{i=1}^{n}|\mathbf{y}-\hat{\mathbf{y}}|$ on each prediction window (averaging for multivariate prediction), and roll the whole set with stride $= 1$. **Platform:** All the models were trained/tested on a single Nvidia V100 32GB GPU. The source code is available at https://github.com/zhouhaoyi/Informer2020.

Results and Analysis

Table 1 and Table 2 summarize the univariate/multivariate evaluation results of all the methods on 4 datasets. We gradually prolong the prediction horizon as a higher requirement of prediction capacity, where the LSTF problem setting is precisely controlled to be tractable on one single GPU for each method. The best results are highlighted in boldface.

Univariate Time-series Forecasting Under this setting, each method attains predictions as a single variable over time series. From Table 1, we can observe that: **(1)** The proposed model Informer significantly improves the inference performance (wining-counts in the last column) across all datasets, and their predict error rises smoothly and slowly within the growing prediction horizon, which demonstrates the success of Informer in enhancing the prediction capacity in the LSTF problem. **(2)** The Informer beats its canonical degradation Informer† mostly in wining-counts, i.e., 32>12, which supports the query sparsity assumption in providing a comparable attention feature map. Our proposed method also out-performs the most related work LogTrans and Reformer. We note that the Reformer keeps dynamic decoding and performs poorly in LSTF, while other methods benefit from the generative style decoder as nonautoregressive predictors. **(3)** The Informer model shows significantly better results than recurrent neural networks LSTMa. Our method has a MSE decrease of 26.8% (at 168), 52.4% (at 336) and 60.1% (at 720). This reveals a shorter network path in the self-attention mechanism acquires better prediction capacity than the RNN-based models. **(4)** The proposed method outperforms DeepAR, ARIMA and Prophet on MSE by decreasing 49.3% (at 168), 61.1% (at 336), and 65.1% (at 720) in average. On the ECL dataset, DeepAR performs better on shorter horizons ($\leq$ 336), and our method surpasses on longer horizons. We attribute this to a specific example, in which the effectiveness of prediction capacity is reflected with the problem scalability.

Multivariate Time-series Forecasting Within this setting, some univariate methods are inappropriate, and LSTnet is the state-of-art baseline. On the contrary, our proposed Informer is easy to change from univariate prediction to multivariate one by adjusting the final FCN layer. From Table 2, we observe that: **(1)** The proposed model Informer greatly outperforms other methods and the findings 1 & 2 in the univariate settings still hold for the multivariate time-series. **(2)** The Informer model shows better results than RNN-based



(a) Input length. (b) Sampling Factor. (c) Stacking Combination.

Figure 4: The parameter sensitivity of three components in Informer.

Prediction length		336			720		
Encoder's input		336	720	1440	720	1440	2880
Informer	MSE	0.249	0.225	0.216	0.271	0.261	0.257
	MAE	0.393	0.384	0.376	0.435	0.431	0.422
Informer†	MSE	0.241	0.214	-	0.259	-	-
	MAE	0.383	0.371	-	0.423	-	-
LogTrans	MSE	0.263	0.231	-	0.273	-	-
	MAE	0.418	0.398	-	0.463	-	-
Reformer	MSE	1.875	1.865	1.861	2.243	2.174	2.113
	MAE	1.144	1.129	1.125	1.536	1.497	1.434

[1] Informer† uses the canonical self-attention mechanism.
[2] The '-' indicates failure for the out-of-memory.

Table 3: Ablation study of the *ProbSparse* self-attention mechanism.

Methods	Training		Testing
	Time	Memory	Steps
Informer	$\mathcal{O}(L \log L)$	$\mathcal{O}(L \log L)$	1
Transformer	$\mathcal{O}(L^2)$	$\mathcal{O}(L^2)$	L
LogTrans	$\mathcal{O}(L \log L)$	$\mathcal{O}(L^2)$	$1^\star$
Reformer	$\mathcal{O}(L \log L)$	$\mathcal{O}(L \log L)$	L
LSTM	$\mathcal{O}(L)$	$\mathcal{O}(L)$	L

[1] The LSTnet is hard to present in a closed form.
[2] The $\star$ denotes applying our proposed decoder.

Table 4: L-related computation statics of each layer.

LSTMa and CNN-based LSTnet, and the MSE decreases 26.6% (at 168), 28.2% (at 336), 34.3% (at 720) in average. Compared with the univariate results, the overwhelming performance is reduced, and such phenomena can be caused by the anisotropy of feature dimensions' prediction capacity. It is beyond the scope of this paper, and we will explore it in the future work.

LSTF with Granularity Consideration We perform an additional comparison to explore the performance with various granularities. The sequences {96, 288, 672} of ETTm_1 (minutes-level) are aligned with {24, 48, 168} of ETTh_1 (hour-level). The Informer outperforms other baselines even if the sequences are at different granularity levels.

Parameter Sensitivity

We perform the sensitivity analysis of the proposed Informer model on ETTh1 under the univariate setting. **Input Length:** In Fig.(4a), when predicting short sequences (like 48), initially increasing input length of encoder/decoder degrades performance, but further increasing causes the MSE to drop because it brings repeat short-term patterns. However, the MSE gets lower with longer inputs in predicting long sequences (like 168). Because the longer encoder input may contain more dependencies, and the longer decoder token has rich local information. **Sampling Factor:** The sampling factor controls the information bandwidth of *ProbSparse* self-attention in Eq.(3). We start from the small factor (=3) to large ones, and the general performance increases a little and stabilizes at last in Fig.(4b). It verifies our query sparsity assumption that there are redundant dot-product pairs in the self-attention mechanism. We set the sample factor $c = 5$ (the red line) in practice. **The Combination of Layer Stacking:** The replica of Layers is complementary for the self-attention distilling, and we investigate each stack {L, L/2, L/4}'s behavior in Fig.(4c). The longer stack is more sensitive to the inputs, partly due to receiving more long-term information. Our method's selection (the red line), i.e., joining L and L/4, is the most robust strategy.

Ablation Study: How well Informer works?

We also conducted additional experiments on ETTh_1 with ablation consideration.

The performance of *ProbSparse* self-attention mechanism In the overall results Table 1 & 2, we limited the problem setting to make the memory usage feasible for the canonical self-attention. In this study, we compare our methods with LogTrans and Reformer, and thoroughly explore their extreme performance. To isolate the memory efficient problem, we first reduce settings as {batch size=8, heads=8, dim=64}, and maintain other setups in the univariate case. In Table 3, the *ProbSparse* self-attention shows better performance than the counterparts. The LogTrans gets OOM

Prediction length		336					480				
Encoder's input		336	480	720	960	1200	336	480	720	960	1200
Informer†	MSE	0.249	0.208	0.225	0.199	0.186	0.197	0.243	0.213	0.192	0.174
	MAE	0.393	0.385	0.384	0.371	0.365	0.388	0.392	0.383	0.377	0.362
Informer‡	MSE	0.229	0.215	0.204	-	-	0.224	0.208	0.197	-	-
	MAE	0.391	0.387	0.377	-	-	0.381	0.376	0.370	-	-

[1] Informer‡ removes the self-attention distilling from Informer†.
[2] The '-' indicates failure for the out-of-memory.

Table 5: Ablation study of the self-attention distilling.

Prediction length		336					480				
Prediction offset		+0	+12	+24	+48	+72	+0	+48	+96	+144	+168
Informer‡	MSE	0.207	0.209	0.211	0.211	0.216	0.198	0.203	0.203	0.208	0.208
	MAE	0.385	0.387	0.391	0.393	0.397	0.390	0.392	0.393	0.401	0.403
Informer§	MSE	0.201	-	-	-	-	0.392	-	-	-	-
	MAE	0.393	-	-	-	-	0.484	-	-	-	-

[1] Informer§ replaces our decoder with dynamic decoding one in Informer‡.
[2] The '-' indicates failure for the unacceptable metric results.

Table 6: Ablation study of the generative style decoder.

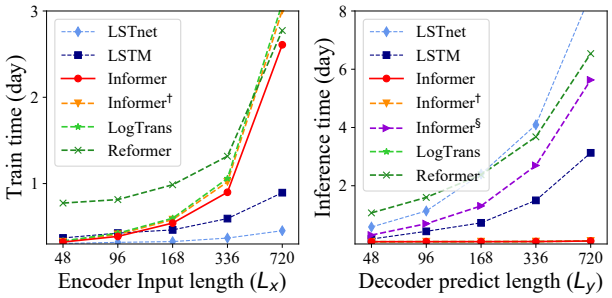


Figure 5: The total runtime of training/testing phase.

in extreme cases because its public implementation is the mask of the full-attention, which still has $\mathcal{O}(L^2)$ memory usage. Our proposed *ProbSparse* self-attention avoids this from the simplicity brought by the query sparsity assumption in Eq.(4), referring to the pseudo-code in Appendix E.2, and reaches smaller memory usage.

The performance of self-attention distilling In this study, we use Informer† as the benchmark to eliminate additional effects of *ProbSparse* self-attention. The other experimental setup is aligned with the settings of univariate Time-series. From Table 5, Informer† has fulfilled all the experiments and achieves better performance after taking advantage of long sequence inputs. The comparison method Informer‡ removes the distilling operation and reaches OOM with longer inputs (> 720). Regarding the benefits of long sequence inputs in the LSTF problem, we conclude that the self-attention distilling is worth adopting, especially when a longer prediction is required.

The performance of generative style decoder In this study, we testify the potential value of our decoder in acquiring a "generative" results. Unlike the existing methods, the labels and outputs are forced to be aligned in the training and inference, our proposed decoder's predicting relies solely on the time stamp, which can predict with offsets. From Table 6, we can see that the general prediction performance of Informer‡ resists with the offset increasing, while the counterpart fails for the dynamic decoding. It proves the decoder's ability to capture individual long-range dependency between arbitrary outputs and avoid error accumulation.

Computation Efficiency

With the multivariate setting and all the methods' current finest implement, we perform a rigorous runtime comparison in Fig.(5). During the training phase, the Informer (red line) achieves the best training efficiency among Transformer-based methods. During the testing phase, our methods are much faster than others with the generative style decoding. The comparisons of theoretical time complexity and memory usage are summarized in Table 4. The performance of Informer is aligned with the runtime experiments. Note that the LogTrans focus on improving the self-attention mechanism, and we apply our proposed decoder in LogTrans for a fair comparison (the $\star$ in Table 4).

5 Conclusion

In this paper, we studied the long-sequence time-series forecasting problem and proposed Informer to predict long sequences. Specifically, we designed the *ProbSparse* self-attention mechanism and distilling operation to handle the challenges of quadratic time complexity and quadratic memory usage in vanilla Transformer. Also, the carefully designed generative decoder alleviates the limitation of traditional encoder-decoder architecture. The experiments on real-world data demonstrated the effectiveness of Informer for enhancing the prediction capacity in LSTF problem.

Appendices

Appendix A Related Work

We provide a literature review of the long sequence time-series forecasting (LSTF) problem below.

Time-series Forecasting Existing methods for time-series forecasting can be roughly grouped into two categories: classical models and deep learning based methods. Classical time-series models serve as a reliable workhorse for time-series forecasting, with appealing properties such as interpretability and theoretical guarantees (Box et al. 2015; Ray 1990). Modern extensions include the support for missing data (Seeger et al. 2017) and multiple data types (Seeger, Salinas, and Flunkert 2016). Deep learning based methods mainly develop sequence to sequence prediction paradigm by using RNN and their variants, achieving ground-breaking performance (Hochreiter and Schmidhuber 1997; Li et al. 2018; Yu et al. 2017). Despite the substantial progress, existing algorithms still fail to predict long sequence time series with satisfying accuracy. Typical state-of-the-art approaches (Seeger et al. 2017; Seeger, Salinas, and Flunkert 2016), especially deep-learning methods (Yu et al. 2017; Qin et al. 2017; Flunkert, Salinas, and Gasthaus 2017; Mukherjee et al. 2018; Wen et al. 2017), remain as a sequence to sequence prediction paradigm with step-by-step process, which have the following limitations: (i) Even though they may achieve accurate prediction for one step forward, they often suffer from accumulated error from the dynamic decoding, resulting in the large errors for LSTF problem (Liu et al. 2019; Qin et al. 2017). The prediction accuracy decays along with the increase of the predicted sequence length. (ii) Due to the problem of vanishing gradient and memory constraint (Sutskever, Vinyals, and Le 2014), most existing methods cannot learn from the past behavior of the whole history of the time-series. In our work, the Informer is designed to address this two limitations.

Long sequence input problem From the above discussion, we refer to the second limitation as to the long sequence time-series input (LSTI) problem. We will explore related works and draw a comparison between our LSTF problem. The researchers truncate / summarize / sample the input sequence to handle a very long sequence in practice, but valuable data may be lost in making accurate predictions. Instead of modifying inputs, Truncated BPTT (Aicher, Foti, and Fox 2019) only uses last time steps to estimate the gradients in weight updates, and Auxiliary Losses (Trinh et al. 2018) enhance the gradients flow by adding auxiliary gradients. Other attempts includes Recurrent Highway Networks (Zilly et al. 2017) and Bootstrapping Regularizer (Cao and Xu 2019). Theses methods try to improve the gradient flows in the recurrent network's long path, but the performance is limited with the sequence length growing in the LSTI problem. CNN-based methods (Stoller et al. 2019; Bai, Kolter, and Koltun 2018) use the convolutional filter to capture the long term dependency, and their receptive fields grow exponentially with the stacking of layers, which hurts the sequence alignment. In the LSTI problem, the main task is to enhance the model's capacity of receiving long sequence inputs and extract the long-range dependency from these inputs. But the LSTF problem seeks to enhance the model's prediction capacity of forecasting long sequence outputs, which requires establishing the long-range dependency between outputs and inputs. Thus, the above methods are not feasible for LSTF directly.

Attention model Bahdanau et al. firstly proposed the addictive attention (Bahdanau, Cho, and Bengio 2015) to improve the word alignment of the encoder-decoder architecture in the translation task. Then, its variant (Luong, Pham, and Manning 2015) has proposed the widely used location, general, and dot-product attention. The popular self-attention based Transformer (Vaswani et al. 2017) has recently been proposed as new thinking of sequence modeling and has achieved great success, especially in the NLP field. The ability of better sequence alignment has been validated by applying it to translation, speech, music, and image generation. In our work, the Informer takes advantage of its sequence alignment ability and makes it amenable to the LSTF problem.

Transformer-based time-series model The most related works (Song et al. 2018; Ma et al. 2019; Li et al. 2019) all start from a trail on applying Transformer in time-series data and fail in LSTF forecasting as they use the vanilla Transformer. And some other works (Child et al. 2019; Li et al. 2019) noticed the sparsity in self-attention mechanism and we have discussed them in the main context.

Appendix B The Uniform Input Representation

The RNN models (Schuster and Paliwal 1997; Hochreiter and Schmidhuber 1997; Chung et al. 2014; Sutskever, Vinyals, and Le 2014; Qin et al. 2017; Chang et al. 2018) capture the time-series pattern by the recurrent structure itself and barely relies on time stamps. The vanilla transformer (Vaswani et al. 2017; Devlin et al. 2018) uses pointwise self-attention mechanism and the time stamps serve as local positional context. However, in the LSTF problem, the ability to capture long-range independence requires global information like hierarchical time stamps (week, month and year) and agnostic time stamps (holidays, events). These are hardly leveraged in canonical self-attention and consequent query-key mismatches between the encoder and decoder bring underlying degradation on the forecasting performance. We propose a uniform input representation to mitigate the issue, the Fig.(6) gives an intuitive overview.

Assuming we have t-th sequence input $\mathcal{X}^t$ and p types of global time stamps and the feature dimension after input representation is d_{model}. We firstly preserve the local context by using a fixed position embedding:

$$\begin{aligned} \text{PE}_{(pos,2j)} &= \sin(pos/(2L_x)^{2j/d_{\text{model}}}) \\ \text{PE}_{(pos,2j+1)} &= \cos(pos/(2L_x)^{2j/d_{\text{model}}}) \end{aligned}, \tag{7}$$

where $j \in \{1, \ldots, \lfloor d_{\text{model}}/2 \rfloor\}$. Each global time stamp is employed by a learnable stamp embeddings $\text{SE}_{(pos)}$ with limited vocab size (up to 60, namely taking minutes as the finest granularity). That is, the self-attention's similarity

computation can have access to global context and the computation consuming is affordable on long inputs. To align the dimension, we project the scalar context $\mathbf{x}_i^t$ into d_{model}-dim vector $\mathbf{u}_i^t$ with 1-D convolutional filters (kernel width=3, stride=1). Thus, we have the feeding vector

$$\mathcal{X}_{\text{feed}[i]}^t = \alpha \mathbf{u}_i^t + \text{PE}_{(L_x \times (t-1)+i,\ )} + \sum_p [\text{SE}_{(L_x \times (t-1)+i)}]_p \quad , \quad (8)$$

where $i \in \{1, \ldots, L_x\}$, and α is the factor balancing the magnitude between the scalar projection and local/global embeddings. We recommend $\alpha = 1$ if the sequence input has been normalized.

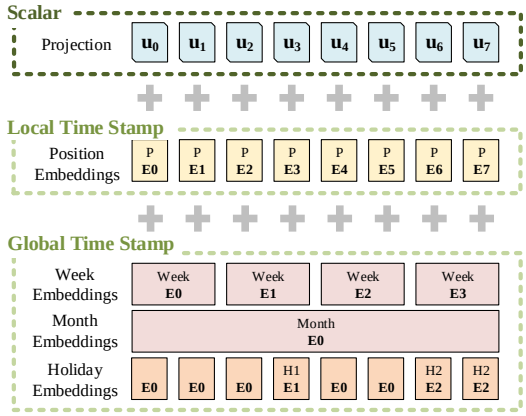


Figure 6: The input representation of Informer. The inputs's embedding consists of three separate parts, a scalar projection, the local time stamp (Position) and global time stamp embeddings (Minutes, Hours, Week, Month, Holiday etc.).

Appendix C The long tail distribution in self-attention feature map

We have performed the vanilla Transformer on the **ETTh**$_1$ dataset to investigate the distribution of self-attention feature map. We select the attention score of {Head1,Head7} @ Layer1. The blue line in Fig.(7) forms a long tail distribution, i.e. a few dot-product pairs contribute to the major attention and others can be ignored.

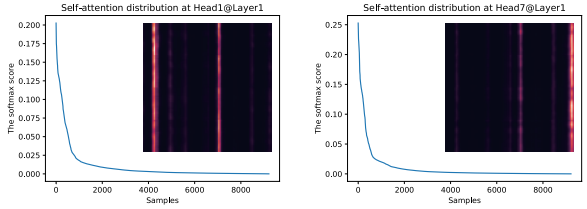


Figure 7: The Softmax scores in the self-attention from a 4-layer canonical Transformer trained on **ETTh**$_1$ dataset.

Appendix D Details of the proof

Proof of Lemma 1

Proof. For the individual $\mathbf{q}_i$, we can relax the discrete keys into the continuous d-dimensional variable, i.e. vector $\mathbf{k}_j$. The query sparsity measurement is defined as the $M(\mathbf{q}_i, \mathbf{K}) = \ln \sum_{j=1}^{L_K} e^{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}} - \frac{1}{L_K} \sum_{j=1}^{L_K} (\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d})$.

Firstly, we look into the left part of the inequality. For each query $\mathbf{q}_i$, the first term of the $M(\mathbf{q}_i, \mathbf{K})$ becomes the log-sum-exp of the inner-product of a fixed query $\mathbf{q}_i$ and all the keys , and we can define $f_i(\mathbf{K}) = \ln \sum_{j=1}^{L_K} e^{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}}$. From the Eq.(2) in the Log-sum-exp network(Calafiore, Gaubert, and Possieri 2018) and the further analysis, the function $f_i(\mathbf{K})$ is convex. Moreover, $f_i(\mathbf{K})$ add a linear combination of $\mathbf{k}_j$ makes the $M(\mathbf{q}_i, \mathbf{K})$ to be the convex function for a fixed query. Then we can take the derivation of the measurement with respect to the individual vector $\mathbf{k}_j$ as $\frac{\partial M(\mathbf{q}_i, \mathbf{K})}{\partial \mathbf{k}_j} = \frac{e^{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}}}{\sum_{j=1}^{L_K} e^{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}}} \cdot \frac{\mathbf{q}_i}{\sqrt{d}} - \frac{1}{L_K} \cdot \frac{\mathbf{q}_i}{\sqrt{d}}$. To reach the minimum value, we let $\vec{\nabla} M(\mathbf{q}_i) = \vec{0}$ and the following condition is acquired as $\mathbf{q}_i \mathbf{k}_1^\top + \ln L_K = \cdots = \mathbf{q}_i \mathbf{k}_j^\top + \ln L_K = \cdots = \ln \sum_{j=1}^{L_K} e^{\mathbf{q}_i \mathbf{k}_j^\top}$. Naturally, it requires $\mathbf{k}_1 = \mathbf{k}_2 = \cdots = \mathbf{k}_{L_K}$, and we have the measurement's minimum as $\ln L_K$, i.e.

$$M(\mathbf{q}_i, \mathbf{K}) \geq \ln L_K \quad . \quad (9)$$

Secondly, we look into the right part of the inequality. If we select the largest inner-product $\max_j \{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}\}$, it is easy that

$$\begin{aligned} M(\mathbf{q}_i, \mathbf{K}) &= \ln \sum_{j=1}^{L_K} e^{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}} - \frac{1}{L_K} \sum_{j=1}^{L_K} (\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}) \\ &\leq \ln(L_K \cdot \max_j \{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}\}) - \frac{1}{L_K} \sum_{j=1}^{L_K} (\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}). \quad (10) \\ &= \ln L_K + \max_j \{\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}\} - \frac{1}{L_K} \sum_{j=1}^{L_K} (\frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}}) \end{aligned}$$

Combine the Eq.(14) and Eq.(15), we have the results of Lemma 1. When the key set is the same with the query set, the above discussion also holds. □

Proposition 1. *Assuming* $\mathbf{k}_j \sim \mathcal{N}(\mu, \Sigma)$ *and we let* $\mathbf{q}\mathbf{k}_i$ *denote set* $\{(\mathbf{q}_i \mathbf{k}_j^\top)/\sqrt{d} \mid j = 1, \ldots, L_K\}$, *then* $\forall M_m = \max_i M(\mathbf{q}_i, \mathbf{K})$ *there exist* $\kappa > 0$ *such that: in the interval* $\forall \mathbf{q}_1, \mathbf{q}_2 \in \{\mathbf{q} | M(\mathbf{q}, \mathbf{K}) \in [M_m, M_m - \kappa)\}$, *if* $\overline{M}(\mathbf{q}_1, \mathbf{K}) > \overline{M}(\mathbf{q}_2, \mathbf{K})$ *and* $\text{Var}(\mathbf{q}\mathbf{k}_1) > \text{Var}(\mathbf{q}\mathbf{k}_2)$, *we have high probability that* $M(\mathbf{q}_1, \mathbf{K}) > M(\mathbf{q}_2, \mathbf{K})$.

Proof of Proposition 1

Proof. To make the further discussion simplify, we can note $a_{i,j} = q_i k_j^T / \sqrt{d}$, thus define the array $A_i = [a_{i,1}, \cdots, a_{i,L_k}]$. Moreover, we denote $\frac{1}{L_K} \sum_{j=1}^{L_K} (\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}) = \text{mean}(A_i)$, then we can denote $M(\mathbf{q}_i, \mathbf{K}) = \max(A_i) - \text{mean}(A_i)$, $i = 1, 2$.

As for $M(\mathbf{q}_i, \mathbf{K})$, we denote each component $a_{i,j} = \text{mean}(A_i) + \Delta a_{i,j}, j = 1, \cdots, L_k$, then we have the following:

$$\begin{aligned} M(\mathbf{q}_i, \mathbf{K}) &= \ln \sum_{j=1}^{L_K} e^{\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}} - \frac{1}{L_K} \sum_{j=1}^{L_K} (\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}) \\ &= \ln(\Sigma_{j=1}^{L_k} e^{\text{mean}(A_i)} e^{\Delta a_{i,j}}) - \text{mean}(A_i) \ , \\ &= \ln(e^{\text{mean}(A_i)} \Sigma_{j=1}^{L_k} e^{\Delta a_{i,j}}) - \text{mean}(A_i) \\ &= \ln(\Sigma_{j=1}^{L_k} e^{\Delta a_{i,j}}) \end{aligned}$$

and it is easy to find $\Sigma_{j=1}^{L_k} \Delta a_{i,j} = 0$.

We define the function $ES(A_i) = \Sigma_{j=1}^{L_k} \exp(\Delta a_{i,j})$, equivalently defines $A_i = [\Delta a_{i,1}, \cdots, \Delta a_{i,L_k}]$, and immediately our proposition can be written as the equivalent form:

For $\forall A_1, A_2$, if

1. $\max(A_1) - \text{mean}(A_1) \geq \max(A_2) - \text{mean}(A_2)$
2. $\text{Var}(A_1) > \text{Var}(A_2)$

Then we rephrase the original conclusion into more general form that $ES(A_1) > ES(A_2)$ with high probability, and the probability have positive correlation with $\text{Var}(A_1) - \text{Var}(A_2)$.

Furthermore, we consider a fine case, $\forall M_m = \max_i M(\mathbf{q}_i, \mathbf{K})$ there exist $\kappa > 0$ such that in that interval $\forall \mathbf{q}_i, \mathbf{q}_j \in \{\mathbf{q} | M(\mathbf{q}, \mathbf{K}) \in [M_m, M_m - \kappa)\}$ if $\max(A_1) - \text{mean}(A_1) \geq \max(A_2) - \text{mean}(A_2)$ and $\text{Var}(A_1) > \text{Var}(A_2)$, we have high probability that $M(\mathbf{q}_1, \mathbf{K}) > M(\mathbf{q}_2, \mathbf{K})$,which is equivalent to $ES(A_1) > ES(A_2)$.

In the original proposition, $\mathbf{k_j} \sim \mathcal{N}(\mu, \Sigma)$ follows multivariate Gaussian distribution, which means that $k_1, \cdots, k_n$ are I.I.D Gaussian distribution, thus defined by the Wiener-khinchin law of large Numbers, $a_{i,j} = q_i k_j^T / \sqrt{d}$ is one-dimension Gaussian distribution with the expectation of 0 if $n \to \infty$. So back to our definition, $\Delta a_{1,m} \sim N(0, \sigma_1^2), \Delta a_{2,m} \sim N(0, \sigma_2^2), \forall m \in 1, \cdots, L_k$, and our proposition is equivalent to a lognormal-distribution sum problem.

A lognormal-distribution sum problem is equivalent to approximating the distribution of $ES(A_1)$ accurately, whose history is well-introduced in the articles (Dufresne 2008),(Vargasguzman 2005). Approximating lognormality of sums of lognormals is a well-known rule of thumb, and no general PDF function can be given for the sums of lognormals. However, (Romeo, Da Costa, and Bardou 2003) and (Hcine and Bouallegue 2015) pointed out that in most cases, sums of lognormals is still a lognormal distribution, and by applying central limits theorem in (Beaulieu 2011), we can have a good approximation that $ES(A_1)$ is a lognormal distribution, and we have $E(ES(A_1)) = ne^{\frac{\sigma_1^2}{2}}$, $\text{Var}(ES(A_1)) = ne^{\sigma_1^2}(e^{\sigma_1^2} - 1)$. Equally, $E(ES(A_2)) = ne^{\frac{\sigma_2^2}{2}}$, $\text{Var}(ES(A_2)) = ne^{\sigma_2^2}(e^{\sigma_2^2} - 1)$.

We denote $B_1 = ES(A_1), B_2 = ES(A_2)$, and the probability $Pr(B_1 - B_2 > 0)$ is the final result of our proposition in general conditions, with $\sigma_1^2 > \sigma_2^2$ WLOG. The difference of lognormals is still a hard problem to solve.

By using the theorem given in(Lo 2012), which gives a general approximation of the probability distribution on the sums and difference for the lognormal distribution. Namely S_1 and S_2 are two lognormal stochastic variables obeying the stochastic differential equations $\frac{dS_i}{S_i} = \sigma_i dZ_i$, $i = 1, 2$, in which $dZ_{1,2}$ presents a standard Weiner process associated with $S_{1,2}$ respectively, and $\sigma_i^2 = \text{Var}(\ln S_i)$, $S^\pm \equiv S_1 \pm S_2$,$S_0^\pm \equiv S_{10} \pm S_{20}$. As for the joint probability distribution function $P(S_1, S_2, t; S_{10}, S_{20}, t_0)$, the value of S_1 and S_2 at time $t > t_0$ are provided by their initial value S_{10} and S_{20} at initial time t_0. The Weiner process above is equivalent to the lognormal distribution(Weiner and Solbrig 1984), and the conclusion below is written in general form containing both the sum and difference of lognormal distribution approximation denoting $\pm$ for sum $+$ and difference $-$ respectively.

In boundary condition

$$\bar{P}_\pm \left(S^\pm, t; S_{10}, S_{20}, t_0 \longrightarrow t\right) = \delta \left(S_{10} \pm S_{20} - S^\pm\right) \quad ,$$

their closed-form probability distribution functions are given by

$$\begin{aligned} & f^{\text{LN}} \left(\tilde{S}^\pm, t; \tilde{S}_0^\pm, t_0\right) \\ =& \frac{1}{\tilde{S}^\pm \sqrt{2\pi \tilde{\sigma}_\pm^2 (t - t_0)}} \\ & \cdot \exp\left\{ - \frac{\left[\ln\left(\tilde{S}^+ / \tilde{S}_0^+\right) + (1/2)\tilde{\sigma}_\pm^2 (t - t_0)\right]^2}{2\tilde{\sigma}_\pm^2 (t - t_0)} \right\} \quad . \end{aligned}$$

It is an approximately normal distribution, and $\tilde{S}^+$, $\tilde{S}^-$ are lognormal random variables, $\tilde{S}_0^\pm$ are initial condition in t_0 defined by Weiner process above. (Noticed that $\tilde{\sigma}_\pm^2 (t - t_0)$ should be small to make this approximation valid.In our simulation experiment, we set $t - t_0 = 1$ WLOG.) Since

$$\widetilde{S}_0^- = (S_{10} - S_{20}) + \left(\frac{\sigma_-^2}{\sigma_1^2 - \sigma_2^2}\right)(S_{10} + S_{20}),$$

and

$$\begin{aligned} \tilde{\sigma_-} &= \left(\sigma_1^2 - \sigma_2^2\right) / (2\sigma_-) \\ \sigma_- &= \sqrt{\sigma_1^2 + \sigma_2^2} \end{aligned}$$

Noticed that $E(B_1) > E(B_2)$, $\text{Var}(B_1) > \text{Var}(B_2)$, the mean value and the variance of the approximate normal distribution shows positive correlation with $\sigma_1^2 - \sigma_2^2$.Besides, the closed-form PDF $f^{\text{LN}}\left(\tilde{S}^\pm, t; \tilde{S}_0^\pm, t_0\right)$ also show positive correlation with $\sigma_1^2 - \sigma_2^2$. Due to the limitation of $\tilde{\sigma}_\pm^2 (t - t_0)$ should be small enough, such positive correlation is not significant in our illustrative numerical experiment.

By using Lie-Trotter Operator Splitting Method in (Lo 2012), we can give illustrative numeral examples for the distribution of $B_1 - B_2$,in which the parameters are well chosen to fit for our top-u approximation in actual LLLT experiments. Figure shows that it is of high probability

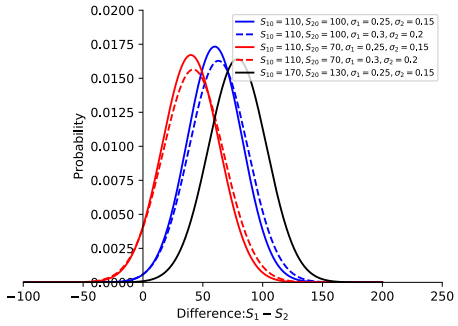


Figure 8: Probability Density verses $S_1 - S_2$ for the approximation of shifted lognormal distribution.

that when $\sigma_1^2 > \sigma_2^2$, the inequality holds that $B_1 > B_2$, $ES(A_1) > ES(A_2)$.

Finishing prooving our proposition in general conditions, we can consider a more specific condition that if $\mathbf{q}_1, \mathbf{q}_2 \in \{\mathbf{q}|M(\mathbf{q}, \mathbf{K}) \in [M_m, M_m - \kappa)\}$, the proposition still holds with high probability.

First, we have $M(q_1, \mathbf{k}) = ln(B_1) > (M_m - \kappa)$ holds for $\forall q_1, q_2$ in this interval. Since we have proved that $E(B_1)) = ne^{\frac{\sigma_1^2}{2}}$, we can conclude that $\forall q_i$ in the given interval,$\exists \alpha, \sigma_i^2 > \alpha, i = 1, 2$. Since we have $\widetilde{S_0^-} = (S_{10} - S_{20}) + \left(\frac{\sigma_-^2}{\sigma_1^2 - \sigma_2^2}\right)(S_{10} + S_{20})$, which also shows positive correlation with $\sigma_1^2 + \sigma_2^2 > 2\alpha$, and positive correlation with $\sigma_1^2 - \sigma_2^2$. So due to the nature of the approximate normal distribution PDF, if $\sigma_1^2 > \sigma_2^2$ WLOG, $Pr(M(q_1, \mathbf{k}) > M(q_2, \mathbf{k})) \approx \Phi(\frac{\widetilde{S_0^-}}{\sigma_-})$ also shows positive correlation with $\sigma_1^2 + \sigma_2^2 > 2\alpha$.

We give an illustrative numerical examples of the approximation above in Fig.(8). In our actual LTTnet experiment, we choose Top-k of A_1, A_2, not the whole set.Actually, we can make a naive assumption that in choosing $top - \lfloor \frac{1}{4} L_k \rfloor$ variables of A_1, A_2 denoted as $A_1^{'}, A_2^{'}$,the variation σ_1, σ_2 don't change significantly, but the expectation $E(A_1^{'}), E(A_2^{'})$ ascends obviously, which leads to initial condition S_{10}, S_{20} ascends significantly, since the initial condition will be sampled from $top - \lfloor \frac{1}{4} L_k \rfloor$ variables, not the whole set.

In our actual LTTnet experiment, we set U, namely choosing around $top - \lfloor \frac{1}{4} L_k \rfloor$ of A_1 and A_2, it is guaranteed that with over 99% probability that in the $[M_m, M_m - \kappa)$ interval, as shown in the black curve of Fig.(8). Typically the condition 2 can be relaxed, and we can believe that if q_1, q_2 fits the condition 1 in our proposition, we have $M(\mathbf{q}_1, \mathbf{K}) > M(\mathbf{q}_2, \mathbf{K})$. □

Appendix E Reproducibility

Details of the experiments

The details of proposed Informer model is summarized in Table 7. For the *ProbSparse* self-attention mechanism, we let d=32, n=16 and add residual connections, a position-wise feed-forward network layer (inner-layer dimension is 2048) and a dropout layer ($p = 0.1$) likewise. Note that we preserves 10% validation data for each dataset, so all the experiments are conducted over 5 random train/val shifting selection along time and the results are averaged over the 5 runs. All the datasets are performed standardization such that the mean of variable is 0 and the standard deviation is 1.

Table 7: The Informer network components in details

Encoder:			N
Inputs	1x3 Conv1d	Embedding ($d = 512$)	4
ProbSparse Self-attention Block	Multi-head ProbSparse Attention ($h = 16$, $d = 32$)		
	Add, LayerNorm, Dropout ($p = 0.1$)		
	Pos-wise FFN ($d_{\text{inner}} = 2048$), GELU		
	Add, LayerNorm, Dropout ($p = 0.1$)		
Distilling	1x3 conv1d, ELU		
	Max pooling (stride = 2)		
Decoder:			N
Inputs	1x3 Conv1d	Embedding ($d = 512$)	2
Masked PSB	add Mask on Attention Block		
Self-attention Block	Multi-head Attention ($h = 8$, $d = 64$)		
	Add, LayerNorm, Dropout ($p = 0.1$)		
	Pos-wise FFN ($d_{\text{inner}} = 2048$), GELU		
	Add, LayerNorm, Dropout ($p = 0.1$)		
Final:			
Outputs	FCN ($d = d_{\text{out}}$)		

Implement of the *ProbSparse* self-attention

We have implemented the *ProbSparse* self-attention in Python 3.6 with Pytorch 1.0. The pseudo-code is given in Algo.(1). The source code is available at https://github.com/zhouhaoyi/Informer2020. All the procedure can be highly efficient vector operation and maintains logarithmic total memory usage. The masked version can be achieved by applying positional mask on step 6 and using cmusum$(\cdot)$ in mean$(\cdot)$ of step 7. In the practice, we can use sum$(\cdot)$ as the simpler implement of mean$(\cdot)$.

Algorithm 1 ProbSparse self-attention

Require: Tensor $\mathbf{Q} \in \mathbb{R}^{m \times d}$, $\mathbf{K} \in \mathbb{R}^{n \times d}$, $\mathbf{V} \in \mathbb{R}^{n \times d}$
1: **print** set hyperparameter c, $u = c \ln m$ and $U = m \ln n$
2: randomly select U dot-product pairs from $\mathbf{K}$ as $\bar{\mathbf{K}}$
3: set the sample score $\bar{\mathbf{S}} = \mathbf{Q}\bar{\mathbf{K}}^\top$
4: compute the measurement $M = \max(\bar{\mathbf{S}}) - \text{mean}(\bar{\mathbf{S}})$ by row
5: set Top-u queries under M as $\bar{\mathbf{Q}}$
6: set $\mathbf{S}_1 = \text{softmax}(\bar{\mathbf{Q}}\mathbf{K}^\top/\sqrt{d}) \cdot \mathbf{V}$
7: set $\mathbf{S}_0 = \text{mean}(\mathbf{V})$
8: set $\mathbf{S} = \{\mathbf{S}_1, \mathbf{S}_0\}$ by their original rows accordingly
Ensure: self-attention feature map $\mathbf{S}$.

The hyperparameter tuning range

For all methods, the input length of recurrent component is chosen from {24, 48, 96, 168, 336, 720} for the ETTh1, ETTh2, Weather and Electricity dataset, and chosen from {24, 48, 96, 192, 288, 672} for the ETTm dataset. For

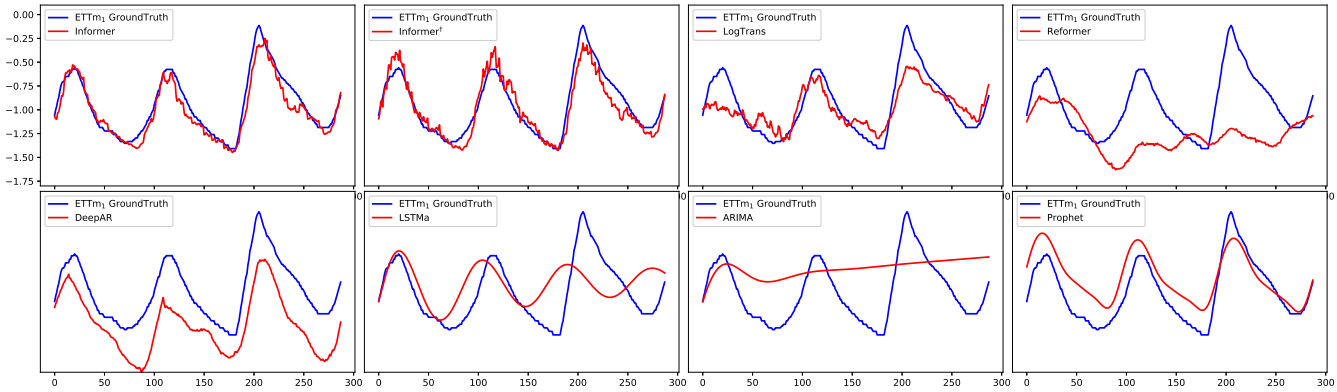


Figure 9: The predicts (len=336) of Informer, Informer†, LogTrans, Reformer, DeepAR, LSTMa, ARIMA and Prophet on the ETTm dataset. The red / blue curves stand for slices of the prediction / ground truth.

LSTMa and DeepAR, the size of hidden states is chosen from {32, 64, 128, 256}. For LSTnet, the hidden dimension of the Recurrent layer and Convolutional layer is chosen from {64, 128, 256} and {32, 64, 128} for Recurrent-skip layer, and the skip-length of Recurrent-skip layer is set as 24 for the ETTh1, ETTh2, Weather and ECL dataset, and set as 96 for the ETTm dataset. For Informer, the layer of encoder is chosen from {6, 4, 3, 2} and the layer of decoder is set as 2. The head number of multi-head attention is chosen from {8, 16}, and the dimension of multi-head attention's output is set as 512. The length of encoder's input sequence and decoder's start token is chosen from {24, 48, 96, 168, 336, 480, 720} for the ETTh1, ETTh2, Weather and ECL dataset, and {24, 48, 96, 192, 288, 480, 672} for the ETTm dataset. In the experiment, the decoder's start token is a segment truncated from the encoder's input sequence, so the length of decoder's start token must be less than the length of encoder's input.

The RNN-based methods perform a dynamic decoding with left shifting on the prediction windows. Our proposed methods Informer-series and LogTrans (our decoder) perform non-dynamic decoding.

Appendix F Extra experimental results

Fig.(9) presents a slice of the predicts of 8 models. The most realted work LogTrans and Reformer shows acceptable results. The LSTMa model is not amenable for the long sequence prediction task. The ARIMA and DeepAR can capture the long trend of the long sequences. And the Prophet detects the changing point and fits it with a smooth curve better than the ARIMA and DeepAR. Our proposed model Informer and Informer† show significantly better results than above methods.

Appendix G Computing Infrastructure

All the experiments are conducted on Nvidia Tesla V100 SXM2 GPUs (32GB memory). Other configuration includes 2 * Intel Xeon Gold 6148 CPU, 384GB DDR4 RAM and 2 * 240GB M.2 SSD, which is sufficient for all the baselines.

References

Aicher, C.; Foti, N. J.; and Fox, E. B. 2019. Adaptively Truncating Backpropagation Through Time to Control Gradient Bias. *arXiv:1905.07473* .

Ariyo, A. A.; Adewumi, A. O.; and Ayo, C. K. 2014. Stock price prediction using the ARIMA model. In *The 16th International Conference on Computer Modelling and Simulation*, 106–112. IEEE.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR 2015*.

Bai, S.; Kolter, J. Z.; and Koltun, V. 2018. Convolutional sequence modeling revisited. *ICLR* .

Beaulieu, N. C. 2011. An extended limit theorem for correlated lognormal sums. *IEEE transactions on communications* 60(1): 23–26.

Beltagy, I.; Peters, M. E.; and Cohan, A. 2020. Longformer: The Long-Document Transformer. *CoRR* abs/2004.05150.

Box, G. E.; Jenkins, G. M.; Reinsel, G. C.; and Ljung, G. M. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165.

Calafiore, G. C.; Gaubert, S.; and Possieri, C. 2018. Log-sum-exp neural networks and posynomial models for convex and log-log-convex data. *CoRR* abs/1806.07850.

Cao, Y.; and Xu, P. 2019. Better Long-Range Dependency By Bootstrapping A Mutual Information Regularizer. *arXiv:1905.11978* .

Chang, Y.-Y.; Sun, F.-Y.; Wu, Y.-H.; and Lin, S.-D. 2018. A Memory-Network Based Solution for Multivariate Time-Series Forecasting. *arXiv:1809.02105* .

Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509* .

Cho, K.; van Merrienboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of SSST@EMNLP 2014*, 103–111.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555* .

Clevert, D.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *ICLR 2016*.

Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.; Le, Q. V.; and Salakhutdinov, R. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv:1901.02860* .

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805* .

Dufresne, D. 2008. Sums of lognormals. In *Actuarial Research Conference*, 1–6.

Flunkert, V.; Salinas, D.; and Gasthaus, J. 2017. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *arXiv:1704.04110* .

Gupta, A.; and Rush, A. M. 2017. Dilated convolutions for modeling long-distance genomic dependencies. *arXiv:1710.01278* .

Hcine, M. B.; and Bouallegue, R. 2015. On the approximation of the sum of lognormals by a log skew normal distribution. *arXiv preprint arXiv:1502.03619* .

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2019. Reformer: The Efficient Transformer. In *ICLR*.

Lai, G.; Chang, W.-C.; Yang, Y.; and Liu, H. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *ACM SIGIR 2018*, 95–104. ACM.

Li, S.; Jin, X.; Xuan, Y.; Zhou, X.; Chen, W.; Wang, Y.-X.; and Yan, X. 2019. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. *arXiv:1907.00235* .

Li, Y.; Yu, R.; Shahabi, C.; and Liu, Y. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR 2018*.

Liu, Y.; Gong, C.; Yang, L.; and Chen, Y. 2019. DSTP-RNN: a dual-stage two-phase attention-based recurrent neural networks for long-term and multivariate time series prediction. *CoRR* abs/1904.07464.

Lo, C.-F. 2012. The sum and difference of two lognormal random variables. *Journal of Applied Mathematics* 2012.

Luong, T.; Pham, H.; and Manning, C. D. 2015. Effective Approaches to Attention-based Neural Machine Translation. In Màrquez, L.; Callison-Burch, C.; Su, J.; Pighin, D.; and Marton, Y., eds., *EMNLP*, 1412–1421. The Association for Computational Linguistics. doi:10.18653/v1/d15-1166. URL https://doi.org/10.18653/v1/d15-1166.

Ma, J.; Shou, Z.; Zareian, A.; Mansour, H.; Vetro, A.; and Chang, S.-F. 2019. CDSA: Cross-Dimensional Self-Attention for Multivariate, Geo-tagged Time Series Imputation. *arXiv:1905.09904* .

Matsubara, Y.; Sakurai, Y.; van Panhuis, W. G.; and Faloutsos, C. 2014. FUNNEL: automatic mining of spatially coevolving epidemics. In *ACM SIGKDD 2014*, 105–114.

Mukherjee, S.; Shankar, D.; Ghosh, A.; Tathawadekar, N.; Kompalli, P.; Sarawagi, S.; and Chaudhury, K. 2018. Armdn: Associative and recurrent mixture density networks for eretail demand forecasting. *arXiv:1803.03800* .

Papadimitriou, S.; and Yu, P. 2006. Optimal multi-scale patterns in time series streams. In *ACM SIGMOD 2006*, 647–658. ACM.

Qin, Y.; Song, D.; Chen, H.; Cheng, W.; Jiang, G.; and Cottrell, G. W. 2017. A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. In *IJCAI 2017*, 2627–2633.

Qiu, J.; Ma, H.; Levy, O.; Yih, S. W.-t.; Wang, S.; and Tang, J. 2019. Blockwise Self-Attention for Long Document Understanding. *arXiv:1911.02972* .

Rae, J. W.; Potapenko, A.; Jayakumar, S. M.; and Lillicrap, T. P. 2019. Compressive transformers for long-range sequence modelling. *arXiv:1911.05507* .

Ray, W. 1990. Time series: theory and methods. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 153(3): 400–400.

Romeo, M.; Da Costa, V.; and Bardou, F. 2003. Broad distribution effects in sums of lognormal random variables. *The European Physical Journal B-Condensed Matter and Complex Systems* 32(4): 513–525.

Schuster, M.; and Paliwal, K. K. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11): 2673–2681.

Seeger, M.; Rangapuram, S.; Wang, Y.; Salinas, D.; Gasthaus, J.; Januschowski, T.; and Flunkert, V. 2017. Approximate bayesian inference in linear state space models for intermittent demand forecasting at scale. *arXiv:1709.07638* .

Seeger, M. W.; Salinas, D.; and Flunkert, V. 2016. Bayesian intermittent demand forecasting for large inventories. In *NIPS*, 4646–4654.

Song, H.; Rajan, D.; Thiagarajan, J. J.; and Spanias, A. 2018. Attend and diagnose: Clinical time series analysis using attention models. In *AAAI 2018*.

Stoller, D.; Tian, M.; Ewert, S.; and Dixon, S. 2019. Seq-U-Net: A One-Dimensional Causal U-Net for Efficient Sequence Modelling. *arXiv:1911.06393* .

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 3104–3112.

Taylor, S. J.; and Letham, B. 2018. Forecasting at scale. *The American Statistician* 72(1): 37–45.

Trinh, T. H.; Dai, A. M.; Luong, M.-T.; and Le, Q. V. 2018. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144* .

Tsai, Y.-H. H.; Bai, S.; Yamada, M.; Morency, L.-P.; and Salakhutdinov, R. 2019. Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel. In *ACL 2019*, 4335–4344.

Vargasguzman, J. A. 2005. Change of Support of Transformations: Conservation of Lognormality Revisited. *Mathematical Geosciences* 37(6): 551–567.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*, 5998–6008.

Wang, S.; Li, B.; Khabsa, M.; Fang, H.; and Ma, H. 2020. Linformer: Self-Attention with Linear Complexity. *arXiv:2006.04768* .

Weiner, J.; and Solbrig, O. T. 1984. The meaning and measurement of size hierarchies in plant populations. *Oecologia* 61(3): 334–336.

Wen, R.; Torkkola, K.; Narayanaswamy, B.; and Madeka, D. 2017. A multi-horizon quantile recurrent forecaster. *arXiv:1711.11053* .

Yu, F.; Koltun, V.; and Funkhouser, T. 2017. Dilated residual networks. In *CVPR*, 472–480.

Yu, R.; Zheng, S.; Anandkumar, A.; and Yue, Y. 2017. Long-term forecasting using tensor-train rnns. *arXiv:1711.00073* .

Zhu, Y.; and Shasha, D. E. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB 2002*, 358–369.

Zilly, J. G.; Srivastava, R. K.; Koutník, J.; and Schmidhuber, J. 2017. Recurrent highway networks. In *ICML*, 4189–4198.