Block Recurrence for Time-Series Forecasting

Berkay Dur
220834005
Dr. Fredrik Dahlqvist
Artificial Intelligence MSc
Queen Mary University of London

***Abstract*—In this paper, we explore the claim that recurrence in Transformers is necessary to effectively model sequential dependencies. Specifically, we explore it from the viewpoint of Long-Term Time-Series Forecasting (LTSF). To investigate this claim, we introduce the Block-Recurrent Time-Series Transformer (BRTST), which fuses the Recurrent state of the RNN to the Transformer Encoder to effectively model sequential dependencies while addressing the constraints of both architectures. Through experimentation, we show that incorporating recurrence in Transformers provides an enhanced sequence modelling ability. Notably, we also question the necessity of this approach in LTSF in the presence of non-recurrent models such as TiDE and PatchTST which achieve a similar performance to the BRTST.**

***Index Terms*—Long-Term Time-Series Forecasting, Recurrent Transformers**

I. Introduction

Long-term time-series forecasting (LTSF), a subset of Time-series forecasting (TSF), is concerned with predicting future time-steps with a large prediction length given the context of historical data. It is a crucial task in many fields with applications in finance, energy, medicine, transport and logistics. TSF has seen rapid development alongside the advancements in artificial intelligence (AI), as evidenced by the M-competitions for time-series forecasting (Toumbas 2017), which was largely dominated by statistical methods from its start in 1982 until the M4 competition (Makridakis et al. 2018) in 2018. The M4 competition highlighted the capabilities of machine learning (ML) methods in this domain, with a few of the top-performing models being combinations of ML and statistical approaches. This trend has only continued with all of the State-of-the-Art (SOTA) models in the M5 competition (Makridakis et al. 2022) in 2020 being ML-based.

Despite these advancements in TSF, LTSF still remains a challenging task as it requires the model to capture temporal dependencies such as seasonality, trend, long- and short-term dependencies while filtering noise (Lai et al. 2018, Wen et al. 2023), which is particularly difficult in the context of non-stationary data (Liu et al. 2022) and for time-series that suffer from a distribution shift (Kim et al. 2022). Deep Neural Networks (DNNs) have shown a lot of promise when dealing with these problems. DNNs like the Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) perform well for TSF with a short prediction length, but they are largely unable to capture these dependencies when working with longer prediction lengths (Shi et al. 2022). As such, most literature in LTSF chooses to focus on Multi-Layer Perceptron (MLP)-based models and Transformer-based models. Despite the prevalence of Transformer-based models in LTSF, recent research has cast doubt onto them due to most existing Transformer-based models being outperformed by a simple MLP (Zeng et al. 2022). Nevertheless, a recent Transformer-based model, PatchTST (Nie et al. 2023), achieves SOTA performance alongside TiDE (Das et al. 2023), an MLP-based model, indicating that both architectures are viable.

However, despite the impressive performance the Transformer has shown in LTSF, it lack an inherent sequence modelling mechanism, like recurrence. Instead, Transformers rely upon position encodings to capture sequential dependencies (Vaswani et al. 2023). Researchers have suggested the Transformers lack of recurrence hinders its performance in sequence modelling tasks where sequence information is crucial (Hao et al. 2019, Katrompas et al. 2022), like LTSF. Despite this promising research, Recurrent Transformers are yet to be explored in LTSF, which could greatly benefit from the enhanced sequence modelling these studies have shown.

This paper seeks to investigate this claim in LTSF by introducing the Block-Recurrent Time-Series Transformer (BRTST), which introduces recurrence to the Transformer Encoder in the form of a Recurrent state, which enables it to directly encode sequence information. It is inspired by the Block-Recurrent Transformer (Hutchins et al. 2022), which is motivated by the way humans think when processing sequential data - summarizing past information while directly referring to the local context. The main contributions of this paper are as follows:

- We introduce the BRTST, which is the first model in LTSF literature that combines recurrence with the Transformer.
- We use the BRTST to highlight the capabilities of Recurrent Transformers by outperforming solely Attention-based and Reccurence-based models such as the Crossformer, Canonical Transformer, LSTMa, and LSTnet.
- We combine the BRTST with the channel-independent patching method to achieve the SOTA in 16 test cases.
- We use the BRTST to address the deficiencies of both Recurrent models and Attention-based models.

II. Related Work

LTSF with Transformer-based models. As stated earlier, there is no existing literature on Recurrent Transformers in

LTSF. Instead, we highlight some of the Transformer-based models in LTSF. LogTrans (Li et al. 2020) and Informer (Zhou et al. 2021) both use a sparsity mask to prevent the dilution of Attention and achieve $\mathrm{O}(N \log N)$ space- and time-complexity. LogTrans does this via a log sparse mask. The Informer uses a probabilistic sparse mask to extract the most relevant keys. AutoFormer (Wu et al. 2022) and FEDFormer (Zhou et al. 2022) both use a trend-seasonality decomposition architecture and replace the Transformer Attention. AutoFormer uses an auto-correlation mechanism to learn sub-series similarities based on series periodicity - it achieves $\mathrm{O}(N \log N)$ space- and time-complexity. FEDFormer uses the Fourier enhanced block to achieve $\mathrm{O}(N)$ space- and time-complexity. PatchTST (Nie et al. 2023) uses channel independent patching to effectively extract series and cross-channel information while reducing the space- and time-complexity to $\mathrm{O}((\frac{N}{S})^2)$, where S is a hyper-parameter.

Recurrent Transformers. Here we give a brief overview of the existing literature on Recurrent Transformers. Katrompas et al. (2022) uses an LSTM to encode sequence information while using a self-Attention mechanism to extract the most relevant information from this. Hao et al. (2019) introduces an additional recurrence encoder to the Canonical Transformer which learns a recurrence representation that it cross-Attends to in the Transformer Encoder. The Feedback Transformer (Fan et al. 2021) uses a Feedback memory mechanism to pass hidden representations through the Transformer layers and through time. The R-Transformer (Wang et al. 2019) introduces a local Parallelizable RNN to the Transformer to learn local dependencies. The Block-Recurrent Transformer Hutchins et al. (2022) processes the input sequence in blocks to effectively combine the recurrent state of the RNN to the Transformer.

III. Background

A. RNN

RNNs are auto-regressive models that propagate a hidden state, summarizing past information, through time. Their sequential nature makes them an ideal candidate for modelling temporal dependencies but it also makes them costly to train (Vaswani et al. 2023). They also suffer from vanishing or exploding gradients which causes instability and prevents them from learning long-term dependencies (Bengio et al. 1994). The use of gates, a cell state, gradient clipping, and attention partially alleviate this problem. However, as these problems becoming increasingly pronounced with longer contexts, they are no longer the dominant architecture in sequence modelling.

As illustrated in Figure 1, RNNs work by sequentially stacking an RNN cell to generate the required outputs. In this setup, the t^{th} cell receives as input the previous hidden state, h_{t-1} and the current time-series input x_t. Using the non-linear function f_h, it computes the next hidden state, h_t:

$$h_t = f_h(h_{t-1}, x_t) \tag{1}$$

This updated hidden state is then used to forecast the next time-step, O_{t+1}, using f_o, a non-linear function,

$$O_{t+1} = f_o(h_t) \tag{2}$$

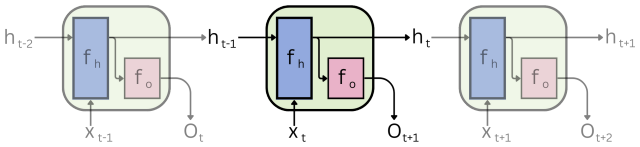

Fig. 1. Illustration of RNN Cells.

B. Transformer

The Transformer Architecture, introduced by Vaswani et al. (2023), operates as both an encoder and a decoder, with the main mechanism being Attention. Unlike RNNs, the Transformer's Attention enables it to access all sequence information concurrently, without relying upon a sequence summary. This enables the Transformer to effectively capture both short- and long-range dependencies. Due to the non-sequential nature of Attention, the Transformer is able to utilize modern optimization techniques such as parallelization, which makes training much faster when compared to the RNN. For these reasons, the Transformer has been the dominant architecture in sequence modelling since its introduction in 2017 and it is effective at TSF (Wu et al. 2020). Nevertheless, it is not without challenges. Similar to the RNN, the Transformer encounters difficulties at long context lengths. Training the Transformer is costly due to the quadratic cost of Attention with respect to the context length. Moreover, long contexts lead to the dilution of relevant attention scores, impacting its learning potential (Li et al. 2020). Additionally, the Transformer lacks an inherent mechanism for modelling sequence information which requires the introduction of a position encoding.

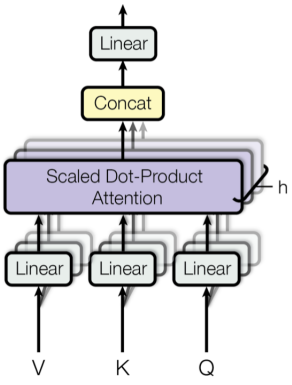

Fig. 2. Illustration of scaled dot-product multi-head Attention. Taken from Vaswani et al. (2023).

The Transformer architecture employs a scaled dot-product multi-head Attention mechanism, as illustrated in figure 2, to

perform its feature extraction. It is used in the context of self-Attention and cross-Attention. It is designed to capture the relationships between different positions in a set (sequence if using a position encoding) of data. In the following section, we break down the scaled dot-product Attention and we refer the reader to Appendix A for an explanation of Multi-Head Attention.

At its core, scaled dot-product Attention is simply a weighted look up of rows of the Value matrix (V), where the relevancy weights are computed between the rows of 2 matrices, the Query matrix (Q) and the Key matrix (K), using a scaled dot-product. In the context of time-series data, the rows of these matrices represent information relating to a time-step, and they are obtained by linear projections of some input time-series.

The scaled dot-product Attention relevancy weights (scores) for time-series data is computed as follows:

$$\text{Score}(Q, K) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}} + M) \tag{3}$$

where $Q \in \mathbb{R}^{n \times d_k}, K \in \mathbb{R}^{m \times d_k}$. Here, n, m represents the number of time-steps in the Query and Key, respectively. And d_k is the embedding dimension. The optional mask, $M \in \mathbb{R}^{n \times m}$, can indicate which positions should or should not be attended by setting its entries to 0 or $-\infty$, respectively. The Softmax function is applied row-wise to ensure that the scores assigned to each Query time-step form a probability distribution. In this distribution, each component represents the Attention weight of a Key time-step towards the corresponding Query time-step. These scores are then used to calculate the Attention output, which is the weighted sum of the rows of the Value matrix (V):

$$\text{Attention}(Q, K, V) = \text{Score}(Q, K)V \tag{4}$$

where $V \in \mathbb{R}^{m \times d_v}$, with d_v being the value dimension. In the context of self-attention, the Query, Key, and Value projections are derived from the same input. In cross-attention, the Query projection is derived from a different input than the Key and Value projections.

The scaled dot-product Attention above is generally coupled with multiple heads. Vaswani et al. (2023) states that multi-head attention enhances the model's learning capabilities by allowing the Attention to focus on different aspects of the input at the same time without much computational overhead.

IV. Block-Recurrent Time-Series Transformer

This section introduces the Block-Recurrent Time-Series Transformer (BRTST), which is illustrated in Figure 3 (Left). It is an adaptation of the Block-Recurrent Transformer (Hutchins et al. 2022) to the domain of LTSF. The BRTST introduces the Recurrent state of the RNN to the Transformer Encoder to provide the Transformer with an inherent recurrence mechanism to model temporal information while addressing the deficiencies of both architectures.

A. Overall Design

The BRTST uses a generative decoding design inspired by the Informer (Zhou et al. 2021), which decodes the entire output sequence through one forward process. This is preferred over the auto-regressive decoding used in the Canonical Transformer and RNN as it greatly speeds up testing and prevents the introduction of an accumulating decoding error (Zhou et al. 2021).

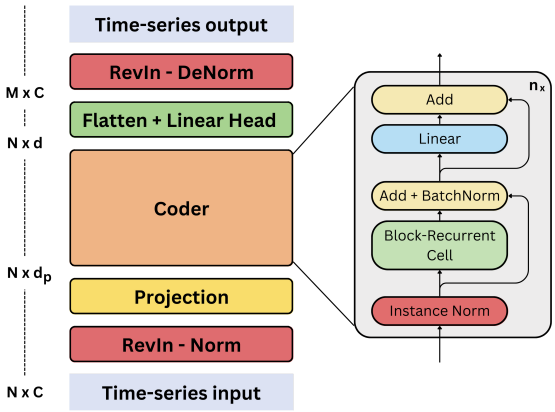

Fig. 3. Illustration of the Block Recurrent Time-Series Transformer (Left), and the Block-Recurrent Cell (Right).

B. Input and Output

The input to the BRTST is of shape $N \times C$, where N is the context length and C is the number of time-series.

The output is of shape $M \times C$, where M is the prediction length and C is the number of time-series.

C. RevIN

The first and last layer of the BRTST is RevIN (Reversible Instance Normalization) (Kim et al. 2022). The first RevIN layer normalizes the time-series input using a standard instance normalization (Ulyanov et al. 2017). The feature-extraction is then performed on this instance normalized time-series. The final layer is then a RevIN denormalization layer that reverse this instance normalization. Kim et al. (2022) shows that RevIN is a cheap and effective method for dealing with time-series that suffer from a distribution shift. We refer the reader to Appendix B for a more detailed explanation of RevIN.

D. Projection

The next layer of the BRTST is the projection layer. This is simply a linear layer that expands the input channels (number of time-series) to a latent dimension. It takes the input from $\mathbb{R}^{N \times C}$ to $\mathbb{R}^{N \times d_p}$, where d_p is the latent dimension. Projecting is a necessary step as it prevents the model from under-fitting by increasing its representation capacity and over-fitting by decreasing its representation capacity.

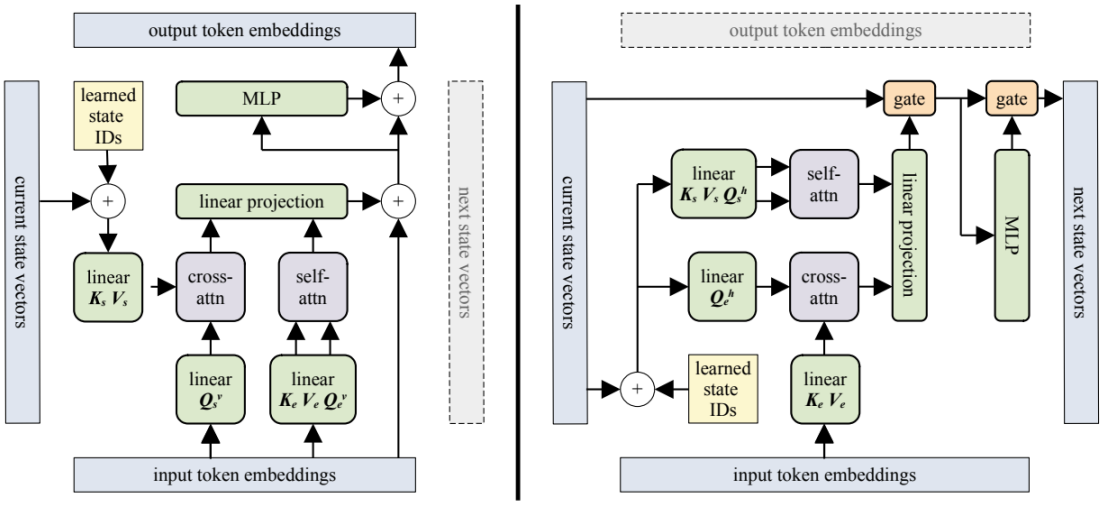

Fig. 4. Block-Recurrent Cell: The vertical direction, Left. The horizontal direction, Right. Taken from Hutchins et al. (2022).

E. Block-Recurrent Coder

The next layer is the Block-Recurrent Coder, which is illustrated in Figure 3 (Right). Its role is synonymous with the Transformer Encoder as it performs a feature extraction by encoding the contextual information to a latent dimension. Like the Encoder it can be stacked as needed to increase the model complexity. We have elected to call it a coder as it uses functionality from both the Transformer Encoder and the Transformer Decoder, like the cross-Attention and the causal mask (Vaswani et al. 2023).

The first layer of the Block-Recurrent Coder instance normalizes (Ulyanov et al. 2017) the time-series input, which is then passed to the Block-Recurrent Cell.

1) Block-Recurrent Cell: The Block-Recurrent Cell is the main mechanism of the BRTST. It operates by looping through blocks of the masked input sequence, performing local Attention while using a Recurrent state to sequentially pass information across the blocks, eliminating the need for a positional encoding. This is all achieved by the cell concurrently computing the feature extraction and the new state via the vertical direction and the horizontal direction, respectively (See Figure 4).

Masking. The Block-Recurrent Cell sparsifies the Attention by applying a causal sliding-window mask of size W. This localization of Attention enables each time-step to only attend to itself and the W previous time-steps, which in-turn prevents the dilution of relevant Attention scores as local dependencies are generally more relevant than global dependencies (Li et al. 2020).

Blocking. Masking in this manner enables the cell to operate on blocks, of size $W \times W$ along the diagonal, where each block only needs to attend to itself and the previous block. This in-turn linearizes the Attention with respect to the context length (Hutchins et al. 2022). Figure 5 highlights the effects of masking and blocking on the Attention score.

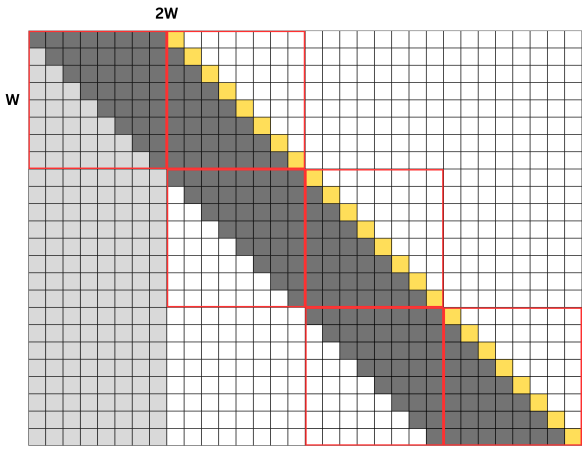

Fig. 5. The effect of masking and blocking on the Attention score. Attention blocks (red), Current time-step (yellow), Attend cells for that time-step (dark gray), 0-initialized column block (light gray). Own diagram inspired by Hutchins et al. (2022).

Recurrence. Local Attention prevents the model from directly encoding global context. But the blocking mechanism introduces recurrence which enables the addition of recurrent states that are able encode global information by passing information across the blocks. The recurrent states introduce a sequential bias to the model that enables it to directly encode temporal information.

Recurrent Hidden States. The Block-Recurrent Cell operates with multiple hidden states, alleviating the issues the RNN experiences due to its recurrent nature. The Block-Recurrent Cell processes the input in blocks of size W, using W hidden states, for $\frac{N}{W}$ iterations. In contrast, the RNN processes the input individually, using a single hidden state,

for N iterations. As a consequence, the Block-Recurrent Cell operates for a shorter number of iterations, speeding up the training. Also, the propagation of gradients is limited to a shorter distance, reducing the impact of vanishing or exploding gradients. Additionally, it has a much larger memory than the RNN as it has a hidden state for each position in the Block which only needs to encode temporal information for $\frac{N}{W}$ time-steps.

Caching. The Block-Recurrent Cell uses the scaled dot-product multi-head Attention four times. Each Attention has a non-differentiable cache that it uses to store the Keys and Queries of the current block (this includes the Recurrent states). This enables each block to attend to itself and the previous block through pre-concatenating the previous block's cached Key and Query to its own and then performing a scaled dot-product multi-head Attention as outlined in Equations (4) and (5).

The Vertical and Horizontal Directions. The Block-Recurrent Cell serves a dual purpose: extracting features from the time-series and updating the hidden states. To simplify explanation, Hutchins et al. (2022) divided the cell into two distinct directions that are computed concurrently. The Vertical direction focuses on the feature extraction, while the Horizontal direction is responsible for updating the states. As depicted in Figure 4, the Vertical and Horizontal directions are nearly identical, with the key distinction being the replacement of residual connections with gates when transitioning from the Vertical direction to the Horizontal direction.

The Vertical Direction performs a feature extraction on the input block (See Figure 4 (Left)). It takes in the current time-series block with a shape of $W \times d_p$ and the current state vectors with a shape of $W \times d_s$. Both the time-series and states undergo a linear projection to dimensions $W \times d_e$, generating the Queries, Keys, and Values, where d_e represents the embedding dimension. Subsequently, self-Attention is performed on the time-series and a cross-Attention is performed on the state. As outlined earlier, these Attentions incorporate cached Keys and Values, which are pre-concatenated to the Keys and Values of the current block, taking them both to dimension $2W \times d_e$. The resulting Attention outputs are then combined using concatenation, yielding a dimension of $W \times 2d_e$. This combined output is further linearly projected to dimension $W \times d_p$. To ensure a more direct path for gradients to flow, a residual connection is added to the time-series input. The processed data then passes through an MLP-layer (see Appendix D), introducing another residual connection. The cell is able to utilize global information by cross-Attending to the state while utilizing local information by self-Attending.

The Horizontal Direction updates the recurrent states. It is the same as the vertical direction with four notable differences: the self-Attention is performed on the states; the cross-Attention is reversed (Q is derived from the states and K, V are derived from the time-series input); the concatenated linear-projection is to dimension $W \times d_s$; the residual connections are replaced with gates to control the updating of the states. The states learns global information by self-Attending and local information by cross-Attending to the current time-series block.

It is important to emphasize that the Attention, linear projection, and MLP layers are specific to each direction and are not shared between the Vertical and Horizontal directions. Additionally, the feature extraction of the Vertical direction contributes to the overall output of the cell. However, the Horizontal direction's output updates the hidden states, remaining localized to the cell.

Positional encoding. The inherent recurrent nature of the cell enables it to model positional information across the blocks. However, the cell's usage of an Attention mechanism to capture relationships within the block introduces a limitation in modelling positional information. To address this, Hutchins et al. (2022) proposes the integration of a relative positional encoding exclusively in the self-Attention of the Vertical direction. Specifically, we employ the rotary position encoding (Su et al. 2022).

Learned Stated IDs. Before any operations on the states, we add learned State IDs. Hutchins et al. (2022) states this is necessary to differentiate and update the state vectors individually, contributing to the effective utilization of multiple states.

Gating. The gating mechanism plays a pivotal role in updating the states in the horizontal direction. As input, it takes the operated hidden states and the unoperated hidden states and decides how to update the unoperated states based on the operated states. To simplify comparison, we a fixed LSTM-style gate. We refer the reader to Appendix C for a detailed explanation of the gating mechanism.

2) Linear Layer: The feature extracted blocks output from the Block-Recurrent Cell are concatenated. Then a residual connection is introduced to bypasses the Block-Recurrent Cell. Subsequently, this output undergoes Batch Normalization and is fed through a linear layer, which also features a residual connection. In contrast to the MLP layer (Appendix D) within the Cell, the linear layer operates across the entire time-series, enabling it to capture interactions spanning the entire sequence.

F. Flatten Head

The output of the final coder is passed to the Linear Head which projects the time-series to the correct dimensions. It operates as follows:

1) Take the coder output of shape $N \times d_p$ and flatten it to shape Nd_p.
2) Project the flattened input to the desired size of MC using a trainable linear layer, where M is the number

of predicted time-steps and C is the number of time-series.

3) Reshape the output to shape $M \times C$.

The output of the linear layer is the normalized time-series prediction. As stated earlier, RevIN denormalization is performed to obtain the final time-series prediction.

V. Experiments

We evaluate the BRTST in two ways: first, without patching, as illustrated in figure 3; and second, with the channel-independent patching used by PatchTST (Nie et al. 2023). We deemed this necessary as channel-independent patching has been shown to enhance model performance. However, integrating it requires some architectural changes that would divert focus away from evaluating the impact of introducing Recurrence to the Transformer in LTSF. Consequently, we assess the BRTST with patching against SOTA models in LTSF, and the BRTST without patching against models using a Recurrent state or Attention to evaluate the effect of introducing Recurrence to the Transformer.

A. Datasets.

The BRTST is evaluated across seven real-world benchmarks for LTSF, as summarized in table I. Our evaluation exclusively focuses on the multivariate setting, which is the most common in the literature. To ensure consistency, we adopt the experimental framework introduced by Zhou et al. (2021), and Wu et al. (2022). Specifically, we use a rolling window to partition the dataset into training, validation, and test sets using the ratios specified in Table I. Subsequently, a z-score normalization is applied, using the mean and variance of the training set. The models are then assessed for specified prediction lengths using varying context lengths. The best performing context length for that prediction length is then reported.

TABLE I
Datasets Summary

Datasets	***Features***	***Timesteps***	**Split Ratio**		
			Train	Val	Test
Weather	*21*	*52696*	0.7	0.1	0.2
ILI	*7*	*966*	0.7	0.1	0.2
ETTh1	*7*	*17420*	0.6	0.2	0.2
ETTh2	*7*	*17420*	0.6	0.2	0.2
ETTm1	*7*	*69680*	0.6	0.2	0.2
ETTm2	*7*	*69680*	0.6	0.2	0.2
Exchange rate	*8*	*7588*	0.7	0.1	0.2

B. Baselines and Data setup

BRTST with patching is evaluated on all seven benchmarks against six models: TiDE, PatchTST, DLinear, FEDFormer, Autoformer, and Informer. TiDE, an MLP-based model, and PatchTST, a Transformer-based model, are chosen as they are the current SOTA models. DLinear is included as it is the simple MLP-model that outperformed the previous SOTA Transformer-based models in Zeng et al. (2022). Additionally, FEDFormer, Autoformer, and Informer are included as they are recognized Transformer-based models in LTSF that were previously SOTA.

For the BRTST with patching comparison, all models are evaluated using the context and prediction lengths (96, 192, 336, 720), except for ILI. For ILI, the baseline models are evaluated on the context and prediction lengths of (24, 36, 48, 60). The BRTST with patching is evaluated on the same predictions lengths except the context is limited to (48, 60).

BRTST without patching is assessed on four benchmarks (Weather, ILI, ETTh1, ETTm1) against four different models that incorporate Recurrence or Attention mechanisms. The evaluated models include LSTMa, which employs an LSTM with attention; LSTNet, utilizes an LSTM with a CNN; the Canonical Transformer with generative decoding, which employs the standard Transformer Attention as outlined in figure 2; and the Crossformer, which uses a variant of Attention known as the Two-Stage Attention (TSA).

For the BRTST without patching comparison, the baseline models are evaluated across specified context and prediction lengths: Weather (24, 48, 168, 336, 720); ILI (24, 36, 48, 60); ETTh1 (24, 48, 168, 336, 720); and ETTm1 (24, 48, 96, 288, 672). In comparison, our model, BRTST without patching is evaluated on the same prediction lengths except the contexts differ as follows: Weather (168, 336, 720); ILI (48, 60); ETTh1 (168, 336, 720); and ETTm1 (96, 288, 672).

C. Experimental Setup

All experiments were conducted [1] on a Google Colab instance using the T4 GPU (15GB VRAM) with 12.7GB RAM. The models are trained using a Mean Square Error (MSE) loss function and evaluated using both the MSE and Mean Absolute Error (MAE) metrics. The training process used the ADAM optimizer. The BRTST is trained for a maximum of 50 epochs with an early stopping patience of 5 epochs. Its initial learning rate was $1e-4$, which was updated using the 1cycle learning rate policy. Notably, the BRTST was trained three times (with seeds 2021, 2022, and 2023), except on the Weather dataset where it was trained once (with seed 2021), and the reported results are the mean of those experiments. The results of the comparison models for the BRTST with patching is taken from Zeng et al. (2022) for Exchange, Nie et al. (2023) for ILI, and Das et al. (2023) for the others. For the BRTST without patching, the results of the comparison models are taken from Zhang & Yan (2023).

VI. Results and Analysis

We present the results of our experiments in Table II for the BRTST with patching, and Table III for the BRTST without patching. In the following we analyze our results.

BRTST with patching. BRTST with patching is the second-best performing model, behind TiDE, demonstrating SOTA

[1] Refer to Appendix G for hyper-parameters.

TABLE II
MULTIVARIATE LTSF RESULTS OF BRTST WITH PATCHING

		Percentage change[a]		***BRTST w/ p***		***TiDE***		***PatchTST/64***		***DLinear***		***FEDFormer***		***Autoformer***		***Informer***	
Metrics		***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***
Weather	*96*	*+1.34*	*-0.51*	***0.147***	*0.199*	*0.166*	*0.222*	*0.149*	***0.198***	*0.176*	*0.237*	*0.238*	*0.314*	*0.249*	*0.329*	*0.354*	*0.405*
	192	*+2.58*	*+0.00*	***0.189***	***0.241***	*0.209*	*0.263*	*0.194*	***0.241***	*0.220*	*0.282*	*0.275*	*0.329*	*0.325*	*0.370*	*0.419*	*0.434*
	336	*+0.82*	*+0.00*	***0.243***	***0.282***	*0.254*	*0.301*	*0.245*	***0.282***	*0.265*	*0.319*	*0.339*	*0.377*	*0.351*	*0.391*	*0.583*	*0.543*
	720	*+0.64*	*+0.00*	***0.311***	***0.334***	*0.313*	*0.340*	*0.314*	***0.334***	*0.323*	*0.362*	*0.389*	*0.409*	*0.415*	*0.426*	*0.916*	*0.705*
Exchange	*96*	*-1.23*	*+2.01*	*0.082*	***0.199***	-	-	-	-	***0.081***	*0.203*	*0.148*	*0.278*	*0.197*	*0.323*	*0.847*	*0.752*
	192	*-10.19*	*-0.68*	*0.173*	*0.295*	-	-	-	-	***0.157***	***0.293***	*0.271*	*0.380*	*0.300*	*0.369*	*1.204*	*0.895*
	336	*-6.88*	*+0.72*	*0.326*	***0.411***	-	-	-	-	***0.305***	*0.414*	*0.460*	*0.500*	*0.509*	*0.524*	*1.672*	*1.036*
	720	*-34.68*	*-16.31*	*0.866*	*0.699*	-	-	-	-	***0.643***	***0.601***	*1.195*	*0.841*	*1.447*	*0.941*	*2.478*	*1.310*
ILI	*24*	*-42.15*	*-18.44*	*1.875*	*0.893*	-	-	***1.319***	***0.754***	*2.215*	*1.081*	*2.624*	*1.095*	*2.906*	*1.182*	*4.657*	*1.449*
	36	*-12.67*	*-1.49*	*1.779*	*0.883*	-	-	***1.579***	***0.870***	*1.963*	*0.963*	*2.516*	*1.021*	*2.585*	*1.038*	*4.650*	*1.463*
	48	*-41.21*	*-19.02*	*2.193*	*0.970*	-	-	***1.553***	***0.815***	*2.130*	*1.024*	*2.505*	*1.041*	*3.024*	*1.145*	*5.004*	*1.542*
	60	*-25.92*	*-15.99*	*1.851*	*0.914*	-	-	***1.470***	***0.788***	*2.368*	*1.096*	*2.742*	*1.122*	*2.761*	*1.114*	*5.071*	*1.543*
ETTh1	*96*	*-0.53*	*-0.50*	*0.377*	*0.400*	***0.375***	***0.398***	*0.379*	*0.401*	***0.375***	*0.399*	*0.376*	*0.415*	*0.435*	*0.446*	*0.941*	*0.769*
	192	*-0.49*	*-0.24*	*0.414*	*0.421*	***0.412***	*0.422*	*0.413*	*0.429*	***0.412***	***0.420***	*0.423*	*0.446*	*0.456*	*0.457*	*1.007*	*0.786*
	336	*+0.69*	*-0.23*	***0.432***	*0.434*	*0.435*	***0.433***	*0.435*	*0.436*	*0.439*	*0.443*	*0.444*	*0.462*	*0.486*	*0.487*	*1.038*	*0.784*
	720	*-0.90*	*+0.00*	*0.450*	***0.464***	*0.454*	*0.465*	***0.446***	***0.464***	*0.472*	*0.490*	*0.469*	*0.492*	*0.515*	*0.517*	*1.144*	*0.857*
ETTh2	*96*	*-2.22*	*-1.19*	*0.276*	*0.340*	***0.270***	***0.336***	*0.274*	*0.337*	*0.289*	*0.353*	*0.332*	*0.374*	*0.332*	*0.368*	*1.549*	*0.952*
	192	*-1.81*	*-1.33*	*0.338*	*0.381*	***0.332***	*0.380*	*0.338*	***0.376***	*0.383*	*0.418*	*0.407*	*0.446*	*0.426*	*0.434*	*3.792*	*1.542*
	336	*+8.06*	*+2.77*	***0.331***	***0.386***	*0.360*	*0.407*	*0.363*	*0.397*	*0.448*	*0.465*	*0.400*	*0.447*	*0.477*	*0.479*	*4.215*	*1.642*
	720	*+2.04*	*+0.93*	***0.385***	***0.426***	*0.419*	*0.451*	*0.393*	*0.430*	*0.605*	*0.551*	*0.412*	*0.469*	*0.453*	*0.490*	*3.656*	*1.619*
ETTm1	*96*	*+0.68*	*-0.58*	***0.291***	*0.344*	*0.306*	*0.349*	*0.293*	*0.346*	*0.299*	***0.343***	*0.326*	*0.390*	*0.510*	*0.492*	*0.626*	*0.560*
	192	*-0.30*	*-1.37*	*0.334*	*0.370*	*0.335*	*0.366*	***0.333***	*0.370*	*0.335*	***0.365***	*0.365*	*0.415*	*0.514*	*0.495*	*0.725*	*0.619*
	336	*-1.10*	*-1.30*	*0.368*	*0.389*	***0.364***	***0.384***	*0.369*	*0.392*	*0.369*	*0.386*	*0.392*	*0.425*	*0.510*	*0.492*	*1.005*	*0.741*
	720	*-1.69*	*-1.94*	*0.420*	*0.421*	***0.413***	***0.413***	*0.416*	*0.420*	*0.425*	*0.421*	*0.446*	*0.458*	*0.527*	*0.493*	*1.133*	*0.845*
ETTm2	*96*	*-3.11*	*-1.59*	*0.166*	*0.255*	***0.161***	***0.251***	*0.166*	*0.256*	*0.167*	*0.260*	*0.180*	*0.271*	*0.205*	*0.293*	*0.355*	*0.462*
	192	*-3.26*	*-1.73*	*0.222*	*0.294*	***0.215***	***0.289***	*0.223*	*0.296*	*0.224*	*0.303*	*0.252*	*0.318*	*0.278*	*0.336*	*0.595*	*0.586*
	336	*-3.75*	*-2.15*	*0.277*	*0.333*	***0.267***	***0.326***	*0.274*	*0.329*	*0.281*	*0.342*	*0.324*	*0.364*	*0.343*	*0.379*	*1.270*	*0.871*
	720	*-0.85*	*-0.78*	*0.355*	*0.386*	***0.352***	***0.383***	*0.362*	*0.385*	*0.397*	*0.421*	*0.410*	*0.420*	*0.414*	*0.419*	*3.001*	*1.267*

[a]Percentage represents the performance gain of the BRTST with patching on the previous SOTA.

TABLE III
MULTIVARIATE LTSF RESULTS OF BRTST WITHOUT PATCHING

		Percentage Change[a]		***BRTST w/o p***		***Crossformer***		***LSTMa***		***LSTnet***		***Transformer***	
Metrics		***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***	***MSE***	***MAE***
Weather	*24*	*+66.67*	*+60.64*	***0.098***	***0.135***	*0.294*	*0.343*	*0.546*	*0.570*	*0.615*	*0.545*	*0.349*	*0.397*
	48	*+66.22*	*+58.39*	***0.125***	***0.171***	*0.370*	*0.411*	*0.829*	*0.677*	*0.660*	*0.589*	*0.386*	*0.433*
	168	*+59.41*	*+51.82*	***0.192***	***0.238***	*0.473*	*0.494*	*1.038*	*0.835*	*0.748*	*0.647*	*0.613*	*0.582*
	336	*+47.88*	*+43.69*	***0.258***	***0.290***	*0.495*	*0.515*	*1.657*	*1.059*	*0.782*	*0.683*	*0.707*	*0.634*
	720	*+36.50*	*+36.53*	***0.334***	***0.344***	*0.526*	*0.542*	*1.536*	*1.109*	*0.851*	*0.757*	*0.834*	*0.741*
ILI	*24*	*+28.97*	*+21.42*	***2.160***	***0.932***	*3.041*	*1.186*	*4.220*	*1.335*	*4.975*	*1.660*	*3.954*	*1.323*
	36	*+42.75*	*+28.81*	***1.950***	***0.877***	*3.406*	*1.232*	*4.771*	*1.427*	*5.322*	*1.659*	*4.167*	*1.360*
	48	*+38.55*	*+23.51*	***2.181***	***0.934***	*3.459*	*1.221*	*4.945*	*1.462*	*5.425*	*1.632*	*4.746*	*1.463*
	60	*+46.65*	*+29.73*	***1.942***	***0.917***	*3.640*	*1.305*	*5.176*	*1.504*	*5.477*	*1.675*	*5.219*	*1.553*
ETTh1	*24*	*-12.79*	*-4.90*	*0.344*	*0.385*	***0.305***	***0.367***	*0.650*	*0.624*	*1.293*	*0.901*	*0.620*	*0.577*
	48	*-8.24*	*-3.05*	*0.381*	*0.406*	***0.352***	***0.394***	*0.720*	*0.675*	*1.456*	*0.960*	*0.692*	*0.671*
	168	*-9.51*	*-1.13*	*0.449*	*0.446*	***0.410***	***0.441***	*1.212*	*0.867*	*1.997*	*1.214*	*0.947*	*0.797*
	336	*-9.51*	*-0.22*	*0.482*	*0.462*	***0.440***	***0.461***	*1.424*	*0.994*	*2.655*	*1.369*	*1.094*	*0.813*
	720	*+7.71*	*+8.97*	***0.479***	***0.477***	*0.519*	*0.524*	*1.960*	*1.322*	*2.143*	*1.380*	*1.241*	*0.917*
ETTm1	*24*	*-4.74*	*-3.07*	*0.221*	*0.302*	***0.211***	***0.293***	*0.621*	*0.629*	*1.968*	*1.170*	*0.306*	*0.371*
	48	*+4.00*	*+1.71*	***0.288***	***0.346***	*0.300*	*0.352*	*1.392*	*0.939*	*1.999*	*1.215*	*0.465*	*0.470*
	96	*+0.63*	*+7.24*	***0.318***	***0.346***	*0.320*	*0.373*	*1.339*	*0.913*	*2.762*	*1.542*	*0.681*	*0.612*
	288	*+3.96*	*+5.15*	***0.388***	***0.405***	*0.404*	*0.427*	*1.740*	*1.124*	*1.257*	*2.076*	*1.162*	*0.879*
	672	*+21.44*	*+17.05*	***0.447***	***0.438***	*0.569*	*0.528*	*2.736*	*1.555*	*1.917*	*2.941*	*1.231*	*1.103*

[a]Percentage represents performance gain of the BRTST without patching on the previous SOTA.

performance in 16 of its 56 test cases, and second-best performance in 27 cases. Despite these achievements, it is important to note that the margin of improvement over the previous SOTA is generally small. Specifically, in 10 instances, the improvement over the previous SOTA is less than 1%. Moreover, the macro-average performance increase in SOTA performance is 1.34%.

Furthermore, the BRTST exhibits a close similarity to the PatchTST model. Notably, 30 out of the 48 shared test cases are within a 1% range, and 13 of the test cases where the BRTST achieves SOTA is when the previous SOTA is PatchTST. This suggests that the BRTST is leveraging the channel-independent patching method associated with PatchTST, essentially fine-tuning rather than significantly enhancing performance.

In terms of specific dataset performance, the BRTST exhibits its best performance on the ETTh2 dataset with a prediction length of 336, achieving an 8.06% improvement in MSE over the previous SOTA. Notably, this is the only improvement of over 5% on the previous SOTA. Conversely, the weakest performance is observed on the ILI dataest, where the macro-average performance decrease is 30.49% on the SOTA. Given ILI's smaller size, this could suggest that the BRTST may require more training data to learn the relationships of the dataset. Another explanation could be due to ILI's short context and forecasting horizons, suggesting the BRTST may perform poorly when working with a short forecasting horizon.

Excluding the Exchange and ILI datasets, the BRTST shows a macro-average performance decrease of 0.37% on the SOTA (or previous SOTA where the BRTST is the SOTA) for each test-case. This result coupled with the its only marginal improvement of 1.34% on the previous SOTA, where it achieves SOTA, and its notable similarity to PatchTST, raises questions of whether the pursuit in Recurrent Transformers in LTSF is justifiable. Given that the SOTA models already achieve impressive results without the introduction of recurrence, this line of research may offer limited benefits in the context of LTSF.

BRTST without patching. The BRTST without patching emerges as the top-performing model in its experiments, achieving the best results in 28 out of 38 test cases and securing the second-best position in the remaining 10. It outperforms the Crossformer and the other comparison models. The BRTST's macro-average performance increase across all test-cases is 21.2% compared to the second-best model, the Crossformer. This suggests that introducing recurrence to the Transformer yields a better performance boost than employing a more intricate Attention mechanism, like the TSA of the Crossformer.

In terms of specific datasets, the BRTST excels on the Weather dataset, boasting a remarkable macro-average performance increase of 52.78% on the Crossformer. Conversely, its weakest performance is observed on the ETTh1 dataset, showing a modest macro-average performance decrease of 3.27% on relative to the Crossformer. It is worth noting, the performance on the ILI, ETTh1, and ETTm1 datasets relative to the Crossformer appears to improve as prediction length increases, while the opposite is true for the Weather dataset. This observation suggests that incorporating recurrence in the Transformer might be beneficial for long prediction length when compared to solely Attention-based models like the Crossformer.

The BRTST's performance in this experiment is encouraging, surpassing models that solely rely on Attention or a Recurrent state. This coupled with the fact that the BRTST's performance tends to increase as prediction length increases relative to the Crossformer provides evidence supporting the necessity of Recurrent Transformers, especially when temporal information is important, like in LTSF.

VII. Conclusion and Further Studies

The main objective of this paper was to examine the claim that incorporating Recurrence within Transformers would improve sequence modelling. Our investigation was carried out within the context of Long-Term Time-Series Forecasting (LTSF), a domain where temporal information holds significance. To facilitate this, we introduce the Block-Recurrent Time-Series Transformer (BRTST).

Our paper demonstrates the BRTST's effectiveness in introducing Recurrence to the Transformer by surpassing models that rely solely on Recurrence or Attention. Furthermore, from a theoretical standpoint we achieve this while addressing the deficiencies of both Recurrent models and Attention models. Additionally, we show that when combined with PatchTST's channel-independent patching, the BRTST yields SOTA results in 16 test cases. However, it is important to acknowledge that this performance increase over prior SOTAs is modest, with a macro-average of 1.34%. These findings suggest that there is a clear advantage in fusing Recurrence with Transformers in LTSF. Nevertheless, the existence of models like PatchTST and TiDE, achieving SOTA status without recurrence, questions the necessity of this line of research.

Further work could be done to validate the theoretical effectiveness of the BRTST in addressing the deficiencies of Recurrent models and Attention-based models. Moreover, an in-depth analysis into the hyper-parameters of the BRTST could help determine which level of recurrence is the most beneficial. Additionally, expanding the landscape of recurrent Transformers in LTSF could further enrich our understanding in the necessity of recurrence in situations with significant sequence information.

Acknowledgements

I would like to thank my supervisor, Dr. Fredrik Dahlqvist, for his continued support and insights throughout the project.

References

Bengio, Y., Simard, P. & Frasconi, P. (1994), 'Learning long-term dependencies with gradient descent is difficult', *IEEE Transactions on Neural Networks* **5**(2), 157–166.

Das, A., Kong, W., Leach, A., Mathur, S., Sen, R. & Yu, R. (2023), 'Long-term forecasting with tide: Time-series dense encoder'.

Fan, A., Lavril, T., Grave, E., Joulin, A. & Sukhbaatar, S. (2021), 'Addressing some limitations of transformers with feedback memory'.

Hao, J., Wang, X., Yang, B., Wang, L., Zhang, J. & Tu, Z. (2019), 'Modeling recurrence for transformer'.

Hutchins, D., Schlag, I., Wu, Y., Dyer, E. & Neyshabur, B. (2022), 'Block-recurrent transformers'.

Katrompas, A., Ntakouris, T. & Metsis, V. (2022), Recurrence and self-attention vs the transformer for time-series classification: A comparative study, *in* M. Michalowski, S. S. R. Abidi & S. Abidi, eds, 'Artificial Intelligence in Medicine', Springer International Publishing, Cham, pp. 99–109.

Kim, T., Kim, J., Tae, Y., Park, C., Choi, J.-H. & Choo, J. (2022), Reversible instance normalization for accurate time-series forecasting against distribution shift, *in* 'International Conference on Learning Representations'.
URL: *https://openreview.net/forum?id=cGDAkQo1C0p*

Lai, G., Chang, W.-C., Yang, Y. & Liu, H. (2018), 'Modeling long- and short-term temporal patterns with deep neural networks'.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X. & Yan, X. (2020), 'Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting'.

Liu, Y., Wu, H., Wang, J. & Long, M. (2022), 'Non-stationary transformers: Exploring the stationarity in time series forecasting'.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2018), 'The m4 competition: Results, findings, conclusion and way forward', *International Journal of Forecasting* **34**, 802–808.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2022), 'M5 accuracy competition: Results, findings, and conclusions', *International Journal of Forecasting* **38**.

Nie, Y., Nguyen, N. H., Sinthong, P. & Kalagnanam, J. (2023), 'A time series is worth 64 words: Long-term forecasting with transformers'.

Shi, J., Jain, M. & Narasimhan, G. (2022), 'Time series forecasting (tsf) using various deep learning models'.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B. & Liu, Y. (2022), 'Roformer: Enhanced transformer with rotary position embedding'.

Toumbas, P. (2017), 'History of competitions – mofc'.
URL: *https://mofc.unic.ac.cy/history-of-competitions/*

Ulyanov, D., Vedaldi, A. & Lempitsky, V. (2017), 'Instance normalization: The missing ingredient for fast stylization'.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2023), 'Attention is all you need'.

Wang, Z., Ma, Y., Liu, Z. & Tang, J. (2019), 'R-transformer: Recurrent neural network enhanced transformer'.

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J. & Sun, L. (2023), 'Transformers in time series: A survey'.

Wu, H., Xu, J., Wang, J. & Long, M. (2022), 'Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting'.

Wu, N., Green, B., Ben, X. & O'Banion, S. (2020), 'Deep transformer models for time series forecasting: The influenza prevalence case'.

Zeng, A., Chen, M., Zhang, L. & Xu, Q. (2022), 'Are transformers effective for time series forecasting?'.

Zhang, Y. & Yan, J. (2023), Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting, *in* 'The Eleventh International Conference on Learning Representations'.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H. & Zhang, W. (2021), 'Informer: Beyond efficient transformer for long sequence time-series forecasting'.

Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L. & Jin, R. (2022), 'Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting'.

Appendix A
Multi-Head Attention

Vaswani et al. (2023) states that multi-head Attention enhances the models learning capabilities by allowing the Attention to focus on different aspects of the input at the same time without introducing much computational overhead. This is achieved by using a set of trainable linear projections, resulting in multiple Attention outputs which are then concatenated:

$$\text{MultiHead}(Q, K, V) = [\text{head}_1, ..., \text{head}_h]W_0 \tag{5}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, h is the number of heads and all the W's are trainable weights.

Appendix B
RevIN

RevIN Norm is a standard instance normalization layer which normalizes each time-series, $X = (x_1, x_2, ..., x_N)$ to have 0-mean and unit variance. Then an affine transformation is applied to each time-step

$$\hat{x}_i = \gamma(\frac{x_i - \mathbb{E}[X]}{\sqrt{\mathbb{V}ar[X] + \epsilon}}) + \beta \tag{6}$$

where $\gamma, \beta \in \mathbb{R}$ are learnable parameters, $\mathbb{E}[X]$ is the time-series mean, $\mathbb{V}ar[X]$ is the variance, and ϵ is a small integer to prevent numerical instabilities.

RevIN DeNorm reverts the instance normalized time-series to its original mean and variance, using the parameters from equations (6), to obtain the final time-series prediction

$$\hat{y}_i = \sqrt{\mathbb{V}ar[X] + \epsilon} \cdot (\frac{\widetilde{y}_i - \beta}{\gamma}) + \mathbb{E}[X] \tag{7}$$

where $\widetilde{y}_i$ is the normalized time-series prediction for time-step i and $\hat{y}_i$ is the final time-series prediction for time-step i, where $\hat{Y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_M)$ is the predicted time-series.

APPENDIX C
LSTM-STYLE GATE

We adopt a fixed LSTM-style gate, visually represented in Figure 6. In the following equations, $h_t \in \mathbb{R}^{W \times d_s}$ corresponds to the operated state, while $c_t \in \mathbb{R}^{W \times d_s}$ refers to the unoperated state. The first gate, f_t, determines the extent to which the unoperated state should be forgotten based on the operated state:

$$f_t = \sigma(h_t W_f + b_f + 1) \tag{8}$$

where $W_f \in \mathbb{R}^{d_s \times d_s}$ and $b_f \in \mathbb{R}^{d_s}$ are trainable parameters. The next gate, z_t, computes a candidate hidden state based on the operated state:

$$z_t = \tanh(h_t W_z + b_z) \tag{9}$$

where $W_z \in \mathbb{R}^{d_s \times d_s}$ and $b_z \in \mathbb{R}^{d_s}$ are trainable parameters. The final gate, the input gate, decides the contribution of the candidate hidden state to the new state:

$$i_t = \sigma(h_t W_i + b_i - 1) \tag{10}$$

where $W_i \in \mathbb{R}^{d_s \times d_s}$ and $b_i \in \mathbb{R}^{d_s}$ are trainable parameters. Finally, the new state is calculated as follows:

$$c_{t+1} = c_t \odot f_t + z_t \odot i_t \tag{11}$$

where $\odot$ indicates an element-wise multiplication. The weights and biases are initialized to small values. The input gate and forget gate have -1 and $+1$ added, respectively. Hutchins et al. (2022) emphasizes that this initialization is crucial to stabilize training and prevent a failure mode where the recurrent states are ignored.

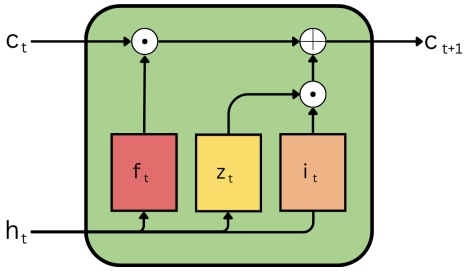

Fig. 6. LSTM-Style Gate where f_t is the forget gate, z_t is the new candidate state, i_t is the input gate.

APPENDIX D
MLP

Our MLP layer follows the same reasoning as the Position-wise Feed-Forward Network of the Canoncial Transformer. Specifically, it expands the number of channels to a higher dimension, $d_f f$, followed by the application of an ELU activation function. Subsequently, it projects the features back to the original dimension, d, while incorporating a Dropout and instance normalization. This design serves two primary purposes. First, it introduces a non-linearity after the Attention computations of the Cell. Second, it enables the dense representation of the most relevant features. The expansion to d_{ff} aids in capturing the complex relationships of the input, while the subsequent compression to dimension d ensures the retention of the most relevant information.

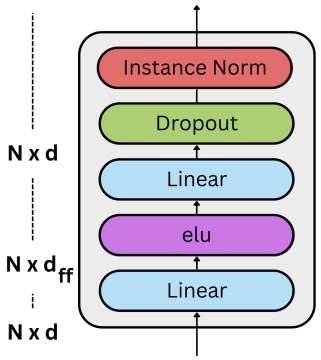

Fig. 7. MLP layer.

APPENDIX E
ROBUSTNESS ANALYSIS

The results of the BRTST, as presented in Tables II, III, and VI, are computed as the averages of multiple experiments conducted using the random seeds 2021, 2022, and 2023 [2]. In this section, we perform a robustness analysis of the BRTST. Tables IV and V provide the mean and standard deviation of the experiments carried out with the random seeds 2021, 2023, and 2023, for the BRTST with patching and without patching, respectively.

BRTST with patching. The mean values and standard deviations are documented in table IV. Notably, for all datasets except Exchange, the model exhibits a lower uncertainty at prediction length 720 than 96 (and at 60 and 24 for ILI). However this is not the case for the Exchange dataset, where the opposite is observed.

BRTST without patching. The mean value and standard deviations are provided in Table V. The uncertainty in MSE and MAE in the ETTh1 dataset tends to rise as the prediction length increases. For the ILI and ETTm1 datsets, a pattern emerges where uncertainty initially increases and then decreaes as the prediction length increases. However, even in these cases, the longest prediction length assessed still exhibits more uncertainty than the shortest prediction length, except for the MSE on the ILI dataset where the longest prediction length exhibits a smaller uncertainty than the shortest prediction length. It is also worth noting, the ILI dataset exhibits the highest level of uncertainty with the standard deviation being 7% of the mean MSE on the prediction length 48.

[2]The results for the Weather dataset are exclusively obtained using the random seed 2021.

TABLE IV
ROBUSTNESS ANALYSIS FOR BRTST WITH PATCHING

Forecast Length		*BRTST with patching*	
Metrics		*MSE*	*MAE*
Exchange	96	0.0816 ± 0.0002	0.1989 ± 0.0004
	192	0.1733 ± 0.0009	0.2945 ± 0.0007
	336	0.3263 ± 0.0027	0.4110 ± 0.0018
	720	0.8658 ± 0.0117	0.6989 ± 0.0054
ILI	24	1.8754 ± 0.1247	0.8928 ± 0.0250
	36	1.7794 ± 0.2194	0.8832 ± 0.0470
	48	2.1931 ± 0.1284	0.9705 ± 0.0038
	60	1.8507 ± 0.0725	0.9144 ± 0.0191
ETTh1	96	0.3771 ± 0.0040	0.3998 ± 0.0032
	192	0.4143 ± 0.0010	0.4213 ± 0.0004
	336	0.4318 ± 0.0026	0.4343 ± 0.0028
	720	0.4503 ± 0.0015	0.4643 ± 0.0012
ETTh2	96	0.2759 ± 0.0026	0.3396 ± 0.0009
	192	0.3377 ± 0.0010	0.3813 ± 0.0018
	336	0.3309 ± 0.0019	0.3861 ± 0.0017
	720	0.3850 ± 0.0007	0.4257 ± 0.0002
ETTm1	96	0.2911 ± 0.0039	0.3438 ± 0.0028
	192	0.3344 ± 0.0011	0.3704 ± 0.0019
	336	0.3675 ± 0.0010	0.3886 ± 0.0008
	720	0.4202 ± 0.0016	0.4210 ± 0.0011
ETTm2	96	0.1657 ± 0.0008	0.2549 ± 0.0009
	192	0.2218 ± 0.0006	0.2945 ± 0.0007
	336	0.2772 ± 0.0006	0.3327 ± 0.0009
	720	0.3554 ± 0.0003	0.3856 ± 0.0004

TABLE V
ROBUSTNESS ANALYSIS FOR BRTST WITHOUT PATCHING

Forecast Length		*BRTST without patching*	
Metrics		*MSE*	*MAE*
ILI	24	2.1600 ± 0.0843	0.9319 ± 0.0105
	36	1.9500 ± 0.0772	0.8771 ± 0.0205
	48	2.1806 ± 0.1536	0.9343 ± 0.0047
	60	1.9415 ± 0.0505	0.9174 ± 0.0270
ETTh1	24	0.3443 ± 0.0014	0.3854 ± 0.0015
	48	0.3811 ± 0.0018	0.4061 ± 0.0007
	168	0.4490 ± 0.0019	0.4461 ± 0.0010
	336	0.4823 ± 0.0032	0.4625 ± 0.0030
	720	0.4794 ± 0.0038	0.4775 ± 0.0032
ETTm1	24	0.2210 ± 0.0018	0.3025 ± 0.0004
	48	0.2878 ± 0.0042	0.3458 ± 0.0026
	96	0.3180 ± 0.0046	0.3661 ± 0.0038
	288	0.3884 ± 0.0038	0.4054 ± 0.0038
	672	0.4474 ± 0.0029	0.4378 ± 0.0021

APPENDIX F
WINDOW SIZE ANALYSIS

In this section, we perform an analysis on the BRTST without patching with varying window sizes. The results are presented in Table VI. on the ETTm1 dataset, it seems that the larger window size of 48 performs better as the prediction length increases. It is also worth noting that the model with a large window size is also significantly faster than the model with a shorter window size. This is likely due to the fact that a shorter window size means a more recurrence.

TABLE VI
VARYING THE WINDOW SIZE OF BRTST WITHOUT PATCHING

Prediction Length		*Window size: 12*		*Window size: 48*	
Metrics		*MSE*	*MAE*	*MSE*	*MAE*
ETTm1	24	**0.216**	**0.299**	0.221	0.302
	48	**0.278**	**0.341**	0.288	0.346
	96	**0.311**	0.361	0.318	**0.346**
	288	0.392	0.407	**0.388**	**0.405**
	672	**0.447**	**0.438**	**0.447**	**0.438**

Prediction Length		*Window size: 24*		*Window size: 42, 48*	
Metrics		*MSE*	*MAE*	*MSE*	*MAE*
ETTh1	24	0.345	0.387	**0.344**	**0.385**
	48	**0.380**	**0.406**	0.381	**0.406**
	168	**0.449**	**0.446**	**0.449**	**0.446**
	336	**0.481**	**0.462**	0.482	**0.462**
	720	**0.479**	**0.477**	**0.479**	**0.477**

APPENDIX G
HYPER-PARAMETERS

The choice of hyper-parameters was either determined by our hyper-parameter search [3] (See supporting material) or through similar analyses done in other papers which informed our decision.

The hyper-parameters slightly differ between the BRTST with patching and the BRTST without patching. Here I list a few of the most important: they both use a single Block-Recurrent Coder layer; batch size is 16 for ILI and 128[4] for the rest; the number of heads of the attention is 16; the hidden dimension of the MLP layer, Appendix D, is 512; and the dropout rate is 0.2.

BRTST with patching. Through hyper-parameter tuning, we determined that the best combinations of the projection dimension, state dimension, and the embedding dimension was 128, 64, and 32, respectively. We also determined that only using a residual connection on the cell and not the linear layer of the Coder yielded the best results. For channel-independent patching, we use the same hyper-parameters as Nie et al. (2023). Notably, a patch length of 16 and a stride of 8 [5]. Following Hutchins et al. (2022), we attempt to make the window size 8x smaller than the context length. But when working with varying context lengths and channel-independent patching, which changes the shape of the input, this was difficult to determine. Despite this, we attempt to emulate this as best as possible, focusing on a context length of 336, which after channel-independent patching with our hyper-parameters becomes 42. As such, to we choose our window size to be 6, which is 7x smaller[6].

[3]It is worth noting that the majority of our hyper-parameter search was conducted on the ETTm1 and ETTh1 datasets, both without and without patching.

[4]Early experimentation showed that increasing the batch size resulted in an inflated performance increase for both the BRTST and PatchTST. To keep the comparison fair, we adopt the same batch size as the PatchTST model.

[5]For ILI this is 4 and 2, respectively.

[6]For ILI, we choose a window size of 2.

BRTST without patching. Through hyper-parameter tuning, we determined that the best combinations of the projection dimension, state dimension, and the embedding dimension was 128, 32, and 128, respectively[7]. We also determined that using only using a residual connection on the linear layer of the Coder and not the cell, yielded the best results, which is contrary to the BRTST with patching. Again, we attempt to emulate the window size and context length size difference of Hutchins et al. (2022). Like the BRTST with patching, we focus on a context length of 336. On the Etth1 and Weather datasets, we use a window length of 42 with context length 168, and a window length of 48 on context lengths 336 and 720. For ILI the context length is 12, and for ETTm1 it is 48.

[7]For the weather dataset, the following context and prediction length pairs were run with dimensions 32, 32, and 32, respectively, as we ran out of GPU memory: (336,720), (720, 336), (720, 720).