Test Time Learning for Time Series Forecasting

Panayiotis Christou
Tandon School of Engineering
New York University
Brooklyn, NY 11201
pc2442@nyu.edu

Shichu Chen
Courant Institute of Mathematical Sciences
New York University
Manhattan, NY 10012
sc10740@nyu.edu

Xupeng Chen
Tandon School of Engineering
New York University
Brooklyn, NY 11201
xc1490@nyu.edu

Parijat Dube
IBM Research
Yorktown Heights, NY 10598
pdube@us.ibm.com

Abstract

We propose the use of Test-Time Training (TTT) modules in a cascade architecture to enhance performance in long-term time series forecasting. Through extensive experiments on standard benchmark datasets, we demonstrate that TTT modules consistently outperform state-of-the-art models, including Mamba-based TimeMachine, particularly in scenarios involving extended sequence and prediction lengths. Our results show significant improvements, especially on larger datasets such as Electricity, Traffic, and Weather, underscoring the effectiveness of TTT in capturing long-range dependencies. Additionally, we explore various convolutional architectures within the TTT framework, showing that even simple configurations like 1D convolution with small filters can achieve competitive results.

1 Introduction

Long Time Series Forecasting (LTSF) is a crucial task in various fields, including energy [5], industry [6], defense [3], and atmospheric sciences [13]. LTSF uses a historical sequence of observations, known as the look-back window, to predict future values through a learned or mathematically induced model. However, the stochastic nature of real-world events makes LTSF challenging. Deep learning models have been widely adopted for tasks in engineering and scientific fields, including time series forecasting. Early approaches employed Recurrent Neural Networks (RNNs) to capture long-range dependencies in sequential data like time series. However, recurrent architectures like RNNs have limited memory retention, are difficult to parallelize, and have constrained expressive capacity. Transformers [20], with the ability to efficiently process sequential data in parallel and capture contextual information, have significantly improved performance on the time series prediction task [14, 21]. Yet, due to the quadratic complexity of attention mechanisms with respect to the context window (or look-back window in LTSF), Transformers are limited in their ability to capture very long dependencies.

In recent years, State Space Models (SSMs) such as Mamba [7], a gated linear RNN variant, have revitalized RNNs for LTSF. These models efficiently capture much longer dependencies while reducing computational costs and enhancing expressive power and memory retention. More recently, a new class of linear RNNs, known as Test Time Training (TTT) modules [19], has emerged. These modules use expressive hidden states and provide theoretical guarantees for capturing long-range dependencies, positioning them as one of the most promising architectures for LTSF.

NeurIPS 2024 Workshop on Time Series in the Age of Large Models.

In this work, we explore the potential of TTT modules in Long-Term Series Forecasting (LTSF) by integrating them into novel model configurations to surpass the current state-of-the-art (SOTA) models. Below are our key contributions:

- We propose a new model architecture utilizing quadruple TTT modules, inspired by the TimeMachine model [1], which currently holds SOTA performance. By replacing the Mamba modules with TTT modules, our model effectively captures longer dependencies and predicts larger sequences.
- We evaluate the model on benchmark datasets, exploring the original look-back window and prediction lengths to identify the limitations of the SOTA architecture. We demonstrate that the SOTA model achieves its performance primarily by constraining look-back windows and prediction lengths, thereby not fully leveraging the potential of LTSF.
- We extend our evaluations to significantly larger sequence and prediction lengths, showing that our TTT-based model consistently outperforms the SOTA model using Mamba modules, particularly in scenarios involving extended look-back windows and long-range predictions.
- We conduct an ablation study to assess the performance of various hidden layer architectures within our model. By testing five different convolutional configurations, we quantify their impact on model performance and provide insights into how they compare with the SOTA model.

2 Related Work

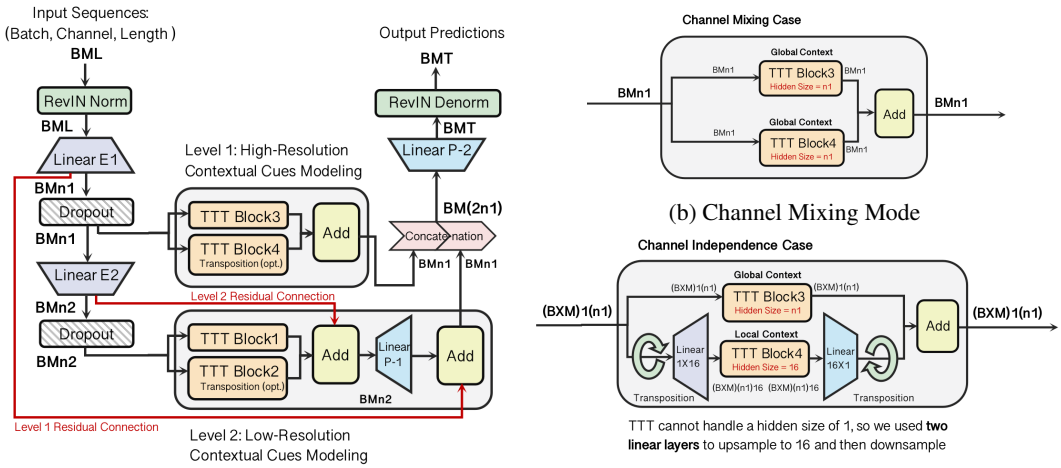
Transformer-based models [14, 16, 17, 22, 24] have significantly advanced long-term time series forecasting (LTSF). Informer [23] reduces complexity with sparse self-attention but can miss finer details in multivariate data; Autoformer [22] captures periodicity and trends but struggles with non-periodic data; Pyraformer [14] handles multi-scale patterns via a hierarchical structure but increases computational cost; Fedformer [24] combines time and frequency representations to reduce overhead but may underperform on noisy time series; iTransformer [16] reliance on multimodal data can raise computational costs when such interactions are absent; PatchTST's [17] performance depends on choosing the correct patch size. Each model improves LTSF in specific ways but also introduces its own limitations. Structured State Space Models [8, 9, 10] efficiently leverage hidden state vectors for LTSF tasks but struggle with time-invariance issues. Mamba [7] generates time-dependent parameters to adapt the model to varying temporal contexts. TimeMachine [1] extends S4 [8] capabilities by employing a quadruple Mamba setup that handles both channel mixing and channel independence scenarios. TTT [19] dynamically adjusts model weights during test time thereby providing an efficient, linear-complexity alternative to self-attention, and is designed to parallelize effectively. We compare the computational complexity of the TTT, Transformer, Mamba and Convolutional (ModernTCN blocks) in Appendix D and we compare the computational complexity of our model (TTT for LTSF), PatchTST, TSMixer, iTransformer and TimeMachine in Appendix E.

3 Model Architecture

Our model architecture builds upon the TimeMachine model [1], introducing key modifications, as shown in Figure 1a, 1b and 1c. Specifically, we replace the Mamba modules in TimeMachine with the TTT modules [19], which retain compatibility since both are linear RNNs [18]. TTT offers superior long-range dependency modeling due to its adaptive nature and theoretically infinite context window. We evaluated our approach with various setups, including a sequence modeling block of TTT with different backbones (Mamba Backbone and Transformer Backbone) and TTT layer configurations (TTT-Linear and TTT-MLP). Additionally, we introduced convolutional hidden layers before the sequence modeling block and conducted experiments with different context lengths and prediction lengths. A detailed visualization of the TTT block and the different proposed configurations of hidden layer is in Figure 3 in Appendix B. Our model employs hierarchical embeddings along with two-level contextual cue modeling and two channel modes. Detailed descriptions on how these work are given in Appendix B along with motivation on the effectiveness of TTT for non-stationary data.

4 Experiments and Evaluation

We evaluate our model on seven benchmark datasets that are commonly used for LTSF, namely: Traffic, Weather, Electricity, ETTh1, ETTh2, ETTm1, and ETTm2 from Wu et al. [22], and Zhou et



(a) TimeMachine incoporating TTT-Blocks

(b) Channel Mixing Mode

(c) Channel Independence Mode

Figure 1: Our model architecture. (a) We replace the four Mamba Block in TimeMachine with four TTT(Test-Time Training) Block. (b) There are two modes of TimeMachine, the channel mixing mode for capturing strong between-channel correlations, and the channel independence mode for modeling within-channel dynamics. For the channel independence scenario, the inputs are first transposed, and then we integrate two linear layers (1×16 and 16×1) to provide the TTT Block with a sufficiently large hidden size.

al. [23] and present the average over 5 runs. Among these, the Traffic and Electricity datasets are significantly larger, with 862 and 321 channels, respectively, containing tens of thousands of temporal points. Table 1 in the Appendix summarizes the dataset details.

For all experiments, we adopted the same setup as in [16], fixing the look-back window $L = 96$ and testing four different prediction lengths $T = 96, 192, 336, 720$. We compared our TTT model against 12 state-of-the-art (SOTA) models. For a detailed comparison refer to Figure 2, and for the full results refer to Table 2 in the Appendix.

Across all seven benchmark datasets, our TTT model consistently demonstrated superior performance compared to SOTA models. In the Weather dataset, TTT achieved leading performance at longer horizons (336 and 720), with MSEs of 0.246 and 0.339, respectively, outperforming TimeMachine, which recorded MSEs of 0.256 and 0.342. The Traffic dataset, with its high number of channels (862), also saw TTT outperform TimeMachine and iTransformer at both medium (336-step MSE of 0.430 vs. 0.433) and long horizons (720-step MSE of 0.456 vs. 0.467), highlighting the model's ability to handle multivariate time series data.

In the Electricity dataset, TTT showed dominant results across all horizons, achieving an MSE of 0.135, 0.153, 0.166 and 0.199 at horizons 96, 192, 336, and 720 respectively, outperforming TimeMachine and PatchTST. For ETTh1, TTT was highly competitive, with strong short-term results (MSE of 0.352 at horizon 96) and continued dominance at medium-term horizons like 336, with an MSE of 0.412. For ETTh2, TTT beat TimeMachine on horizon 96 (MSE of 0.274), TTT also closed the gap at longer horizons (MSE of 0.448 at horizon 720 compared to 0.411 for TimeMachine).

For the ETTm1 dataset, TTT outperformed TimeMachine at nearly every horizon, recording an MSE of 0.309 at horizon 96, 0.381 on horizon 336 and 0.431 at horizon 720, confirming its effectiveness for long-term industrial forecasting. Similarly, in ETTm2, TTT performed well at shorter horizons (MSE of 0.177 at horizon 96) and remained highly competitive at longer horizons, maintaining its lead over TimeMachine at horizon 720 (MSE of 0.364 vs. 0.371).

5 Prediction & Sequence Length Analysis and Ablation Study

For the first part of our experiments we tested the following configurations (Figure 3): (1) **Conv 3**: 1D Convolution with kernel size 3, (2) **Conv 5**: 1D Convolution with kernel size 5, (3) **Conv Stack 3**: two 1D Convolutions with kernel size 3 in cascade, (4) **Conv Stack 5**: two 1D Convolutions with

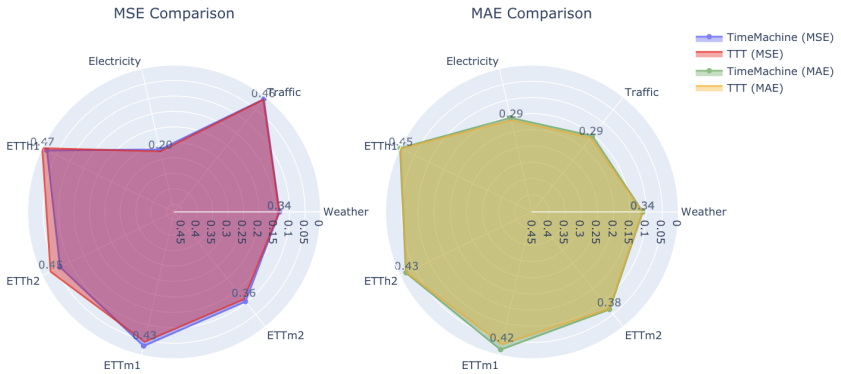


Figure 2: Average MSE and MAE comparison of our model and SOTA baselines with $L = 720$. The circle center represents the maximum possible error. Closer to the boundary indicates better performance.

kernel sizes 5 and 3 in cascade, and (5) **Inception**: an Inception Block combining 1D Convolutions with kernel sizes 5 and 3, followed by concatenation and reduction to the original size.

For the second part of our experiments, we extended the sequence and prediction lengths beyond the parameters tested in previous studies. We used the same baseline architectures (MLP and Linear) with the Mamba backbone as in the original TimeMachine paper, but this time also included the best-performing 1D Convolution architecture with kernel size 3. We tested the following sequence and prediction lengths, with $L = 2880$ and 5760, far exceeding the original length of $L = 96$.

TTT-based models, particularly Conv Stack 5, demonstrated clear advantages in capturing long-range dependencies in the ablation study with the original experimental setup. Find more details for the experimental setup in Appendix C. Conv Stack 5 consistently showed a reduction in MSE compared to TimeMachine, especially at shorter horizons (e.g., 96), where it achieved an MSE of 0.259 versus TimeMachine's 0.262. As prediction lengths increased, such as at 720, TTT achieved better accuracy rates, with Conv Stack 5 achieving an MAE of 0.373 compared to TimeMachine's 0.378.

As the sequence and prediction lengths increased above the original values used in TimeMachine, the Conv 3 architecture (used in increased sequence and prediction length experiments) showed superior performance to Time Machine on all prediction lengths except 4320 and on all sequence lengths. On prediction length 720, TTT records an MSE of 0.517 compared to TimeMachine's 0.535. The data can be seen in Tables 5 and 6 in the appendix. TTT recorded lower MSE and MAE values, demonstrating better scalability and adaptability to larger contexts. Moreover, even at lower prediction lengths but with increased sequence length TTT remains at lower error rates.

6 Conclusion and Future Work

In this work, we improved the state-of-the-art (SOTA) model for time series forecasting by replacing the Mamba modules in the original TimeMachine model with Test-Time Training (TTT) modules, which use linear RNNs to capture long-range dependencies. Extensive experiments showed that the TTT architectures—MLP and Linear—performed well, with MLP slightly outperforming Linear. Exploring alternative architectures, particularly *Conv Stack 5*, improved performance at longer prediction horizons. The most significant gains came from increasing sequence and prediction lengths, where our TTT models consistently matched or outperformed the SOTA model, particularly on larger datasets like Electricity, Traffic, and Weather, highlighting the model's strength in handling long-range dependencies. While convolutional stacks showed promise, further improvement is possible by refining hidden layer configurations and exploring architectural diversity. Overall, this work demonstrates the potential of TTT modules in long-term forecasting, especially with larger datasets and longer horizons.

References

[1] Md Atik Ahamed and Qiang Cheng. Timemachine: A time series is worth 4 mambas for long-term forecasting, 2024. URL https://arxiv.org/abs/2403.09898.

[2] Sercan O. Arik, Nathanael C. Yoder, and Tomas Pfister. Self-adaptive forecasting for improved deep learning on non-stationary time-series, 2022. URL https://arxiv.org/abs/2202.02403.

[3] Jonathan Z. Bakdash, Steve Hutchinson, Erin G. Zaroukian, Laura R. Marusich, Saravanan Thirumuruganathan, Charmaine Sample, Blaine Hoffman, and Gautam Das. Malware in the future? forecasting of analyst detection of cyber events. *arXiv preprint arXiv:1707.03243*, 2017.

[4] Muxi Chen, Zhijian Xu, Ailing Zeng, and Qiang Xu. Fraug: Frequency domain augmentation for time series forecasting, 2023. URL https://arxiv.org/abs/2302.09292.

[5] John Doe and Jane Smith. Time series forecasting for energy consumption. *Energies*, 15(3):773, 2023. doi: 10.3390/en15030773. URL https://www.mdpi.com/1996-1073/15/3/773.

[6] John Doe and Jane Smith. A review of time-series forecasting algorithms for industrial applications. *Machines*, 12(6):380, 2024. doi: 10.3390/machines12060380. URL https://www.mdpi.com/2075-1702/12/6/380.

[7] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

[8] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

[9] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models, 2022. URL https://arxiv.org/abs/2206.11893.

[10] Ankit Gupta, Harsh Mehta, and Jonathan Berant. Simplifying and understanding state space models with diagonal linear rnns, 2023. URL https://arxiv.org/abs/2212.00768.

[11] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

[12] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=cGDAkQo1C0p.

[13] Bryan Lim and Stefan Zohren. Time series forecasting with deep learning: A survey. *arXiv preprint arXiv:2004.13408*, 2020.

[14] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2022.

[15] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. *arXiv preprint arXiv:2310.06625*, 2023.

[16] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting, 2024. URL https://arxiv.org/abs/2310.06625.

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

[18] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[19] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2024. URL https://arxiv.org/abs/2407.04620.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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

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

[23] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021. URL https://arxiv.org/abs/2012.07436.

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

A Appendix A: TTT vs Mamba

Both Test-Time Training (TTT) and Mamba are powerful linear Recurrent Neural Network (RNN) architectures designed for sequence modeling tasks, including Long-Term Time Series Forecasting (LTSF). While both approaches aim to capture long-range dependencies with linear complexity, there are key differences in how they handle context windows, hidden state dynamics, and adaptability. This subsection compares the two, focusing on their theoretical formulations and practical suitability for LTSF.

A.1 Mamba: Gated Linear RNN via State Space Models (SSMs)

Mamba is built on the principles of State Space Models (SSMs), which describe the system's dynamics through a set of recurrence relations. The fundamental state-space equation for Mamba is defined as:

$$h_k = \bar{A}h_{k-1} + \bar{B}u_k, \quad v_k = Ch_k,$$

where:

- h_k represents the hidden state at time step k.
- u_k is the input at time step k.
- $\bar{A}$ and $\bar{B}$ are learned state transition matrices.
- v_k is the output at time step k, and C is the output matrix.

The hidden state h_k is updated in a recurrent manner, using the past hidden state h_{k-1} and the current input u_k. Although Mamba can capture long-range dependencies better than traditional RNNs, its hidden state update relies on fixed state transitions governed by $\bar{A}$ and $\bar{B}$, which limits its ability to dynamically adapt to varying input patterns over time.

In the context of LTSF, while Mamba performs better than Transformer architectures in terms of computational efficiency (due to its linear complexity in relation to sequence length), it still struggles to fully capture long-term dependencies as effectively as desired. This is because the fixed state transitions constrain its ability to adapt dynamically to changes in the input data.

A.2 TTT: Test-Time Training with Dynamic Hidden States

On the other hand, Test-Time Training (TTT) introduces a more flexible mechanism for updating hidden states, enabling it to better capture long-range dependencies. TTT uses a trainable hidden state that is continuously updated at test time, allowing the model to adapt dynamically to the current input. The hidden state update rule for TTT can be defined as:

$$z_t = f(x_t; W_t), \quad W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t),$$

where:

- z_t is the hidden state at time step t, updated based on the input x_t.
- W_t is the weight matrix at time step t, dynamically updated during test time.
- $\ell(W; x_t)$ is the loss function, typically computed as the difference between the predicted and actual values: $\ell(W; x_t) = \|f(\tilde{x}_t; W) - x_t\|^2$.
- η is the learning rate for updating W_t during test time.

The key advantage of TTT over Mamba is the dynamic nature of its hidden states. Rather than relying on fixed state transitions, TTT continuously adapts its parameters based on new input data at test time. This enables TTT to have an infinite context window, as it can effectively adjust its internal representation based on all past data and current input. This dynamic adaptability makes TTT particularly suitable for LTSF tasks, where capturing long-term dependencies is crucial for accurate forecasting.

A.3 Comparison of Complexity and Adaptability

One of the major benefits of both Mamba and TTT is their linear complexity with respect to sequence length. Both models avoid the quadratic complexity of Transformer-based architectures, making them efficient for long-time series data. However, TTT offers a distinct advantage in terms of adaptability:

- Mamba:
  $$\mathcal{O}(L \times D^2),$$
  where L is the sequence length and D is the dimension of the state space. Mamba's fixed state transition matrices limit its expressiveness over very long sequences.
- TTT:
  $$\mathcal{O}(L \times N \times P),$$
  where N is the number of dynamic parameters (weights) and P is the number of iterations for test-time updates. The dynamic nature of TTT allows it to capture long-term dependencies more effectively, as it continuously updates the weights W_t during test time.

Theoretically, TTT is more suitable for LTSF due to its ability to model long-range dependencies dynamically. By continuously updating the hidden states based on both past and present data, TTT effectively functions with an infinite context window, whereas Mamba is constrained by its fixed state-space formulation. Moreover, TTT is shown to be theoretically equivalent to self-attention under certain conditions, meaning it can model interactions between distant time steps in a similar way to Transformers but with the added benefit of linear complexity. This makes TTT not only computationally efficient but also highly adaptable to the long-term dependencies present in time series data.

In summary, while Mamba provides significant improvements over traditional RNNs and Transformer-based models, its reliance on fixed state transitions limits its effectiveness in modeling long-term dependencies. TTT, with its dynamic hidden state updates and theoretically infinite context window, is better suited for Long-Term Time Series Forecasting (LTSF) tasks. TTT's ability to adapt its

parameters at test time ensures that it can handle varying temporal patterns more flexibly, making it the superior choice for capturing long-range dependencies in time series data.

B Appendix B: Model Components and Motivation

B.1 Motivation of TTT on Non-Stationary Data

Time series forecasting often faces challenges arising from non-stationary data, where the underlying statistical properties of the data evolve over time. Traditional models struggle with such scenarios, as they are typically trained on static distributions and are not inherently equipped to handle distribution shifts at inference time. Test-Time Training (TTT) has gained attention as a robust paradigm to mitigate this issue, enabling models to adapt dynamically during inference by leveraging self-supervised learning tasks. For example, the work on self-adaptive forecasting introduced by Google in [2] demonstrates how incorporating adaptive backcasting mechanisms allows models to adjust their predictions to evolving patterns in the data, improving accuracy and robustness under non-stationary conditions. Similarly, FrAug [4] explores data augmentation in the frequency domain to bolster model performance in distributionally diverse settings. While not explicitly a TTT method, FrAug's augmentation principles align with TTT's objectives by enhancing model resilience to dynamic changes in time series characteristics. These studies collectively highlight the potential of adaptive methods like TTT to address the unique challenges posed by non-stationary time series data, making them well-suited for applications where robustness and flexibility are paramount.

B.2 Hierarchical Embedding

Our model employs a two-level hierarchical architecture that captures both high- and low-resolution temporal patterns. The input sequence BML (Batch, Channel, Length) is first passed through Reversible Instance Normalization [11] (RevIN), which stabilizes the model by normalizing the input data and helps mitigate distribution shifts. This operation is essential for improving generalization across datasets.

After normalization, the sequence passes through two linear embedding layers. Linear E1 and Linear E2 are used to map the input sequence into two embedding levels: higher resolution and lower resolution. The embedding operations $E_1 : \mathbb{R}^{M \times L} \to \mathbb{R}^{M \times n_1}$ and $E_2 : \mathbb{R}^{M \times n_1} \to \mathbb{R}^{M \times n_2}$ are achieved through MLP. n_1 and n_2 are configurations that take values from $\{512, 256, 128, 64, 32\}$, satisfying $n_1 > n_2$. Dropout layers are applied after each embedding layer to prevent overfitting, especially for long-term time series data. As shown in Figure 1a.

B.3 Two Level Contextual Cue Modeling

At each of the two embedding levels, a contextual cues modeling block processes the output from the Dropout layer following E1 and E2. This hierarchical architecture captures both fine-grained and broad temporal patterns, leading to improved forecasting accuracy for long-term time series data.

In Level 1, **High-Resolution Contextual Cues Modeling** is responsible for modeling high-resolution contextual cues. TTT Block 3 and TTT Block 4 process the input tensor, focusing on capturing fine-grained temporal dependencies. The TTT Block3 operates directly on the input, and transposition may be applied before TTT Block4 if necessary. The outputs are summed and then concatenated with the Level 2 output. There is no residual connection summing in Level 1 modeling.

In Level 2, **Low-Resolution Contextual Cues Modeling** handles broader temporal patterns, functioning similarly to Level 1. TTT Block 1 and TTT Block 2 process the input tensor to capture low-resolution temporal cues and add them together. A linear projection layer (P-1) is then applied to map the output (with dimension RM×n2) to a higher dimension RM×n1, preparing it for concatenation. Additionally, the Level 1 and Level 2 Residual Connections ensure that information from previous layers is effectively preserved and passed on.

B.4 Two Channel Modes

The architecture adapts to two modes: Channel Mixing and Channel Independence. The Channel Mixing Mode(Figure 1a and 1b) processes all channels of a multivariate time series together, al-

lowing the model to capture potential correlations between different channels and understand their interactions over a longer time. Figure 1a illustrates an example of the channel mixing case, but there is also a channel independence case corresponding to Figure 1a, which we have not shown here. Figures 1b and 1c demonstrate the channel mixing and independence modes of the Level 1 High-Resolution Contextual Cues Modeling part with TTT Block 3 and TTT Block 4. Similar versions of the two-channel modes for Level 2 Low-Resolution Contextual Cues Modeling are quite similar to those in Level 1, which we have also omitted here.

The **Channel Independence Mode**(Figure 1c) treats each channel of a multivariate time series as an independent sequence, enabling the model to analyze individual time series more accurately. This mode focuses on learning patterns within each channel without considering potential correlations between them.

The main difference between these two modes is that the **Channel Independence Mode** always uses transposition before and after one of the TTT blocks (in Figure 1c, it's TTT Block 4). This allows the block to capture contextual cues from local perspectives while the other block focuses on modeling the global context. However, in the **Channel Mixing Mode**, both TTT Block 3 and TTT Block 4 model the global context.

The hidden size value for TTT Blocks in global context modeling is set to n_1 since the input shape is BMn_1 for Channel Mixing and $(B \times M)1n_1$ for Channel Independence. To make the TTT Block compatible with the local context modeling scenario—where the input becomes $(B \times M)n_1 1 \leftarrow Transpose((B \times M)1n_1)$ after transposition—we add two linear layers: one for upsampling to $(B \times M)n_1 16$ and another for downsampling back. In this case, the hidden size of TTT Block 4 is set to 16.

B.5 TTT Block and Proposed Architectures

In Figure 3 we illustrate the components of the TTT block and the proposed architectures we used in our ablation study for the model based on convolutional blocks.

B.6 Prediction

The prediction process in our model works as follows. During inference, the input time series $(\mathbf{x}_1, \ldots, \mathbf{x}_L)$, where L is the look-back window length, is split into M univariate series $\mathbf{x}^{(i)} \in \mathbb{R}^{1 \times L}$. Each univariate series represents one channel of the multivariate time series. Specifically, an individual univariate series can be denoted as:

$$\mathbf{x}_{1:L}^{(i)} = \left(x_1^{(i)}, \ldots, x_L^{(i)}\right) \quad \text{where } i = 1, \ldots, M.$$

Each of these univariate series is fed into the model, and the output of the model is a predicted series $\hat{\mathbf{x}}^{(i)}$ for each input channel. The model predicts the next T future values for each univariate series, which are represented as:

$$\hat{\mathbf{x}}^{(i)} = \left(\hat{x}_{L+1}^{(i)}, \ldots, \hat{x}_{L+T}^{(i)}\right) \in \mathbb{R}^{1 \times T}.$$

Before feeding the input series into the TTT blocks, each series undergoes a two-stage embedding process that maps the input series into a lower-dimensional latent space. This embedding process is crucial for allowing the model to learn meaningful representations of the input data. The embedding process is mathematically represented as follows:

$$\mathbf{x}^{(1)} = E_1(\mathbf{x}^{(0)}), \quad \mathbf{x}^{(2)} = E_2(DO(\mathbf{x}^{(1)})),$$

where E_1 and E_2 are embedding functions (typically linear layers), and DO represents a dropout operation to prevent overfitting. The embeddings help the model process the input time series more effectively and ensure robustness during training and inference.

Dataset	Channels	Time Points	Frequencies
Weather	21	52696	10 Minutes
Traffic	862	17544	Hourly
Electricity	321	26304	Hourly
ETTh1	7	17420	Hourly
ETTh2	7	17420	Hourly
ETTm1	7	69680	15 Minutes
ETTm2	7	69680	15 Minutes

Table 1: Details of each dataset

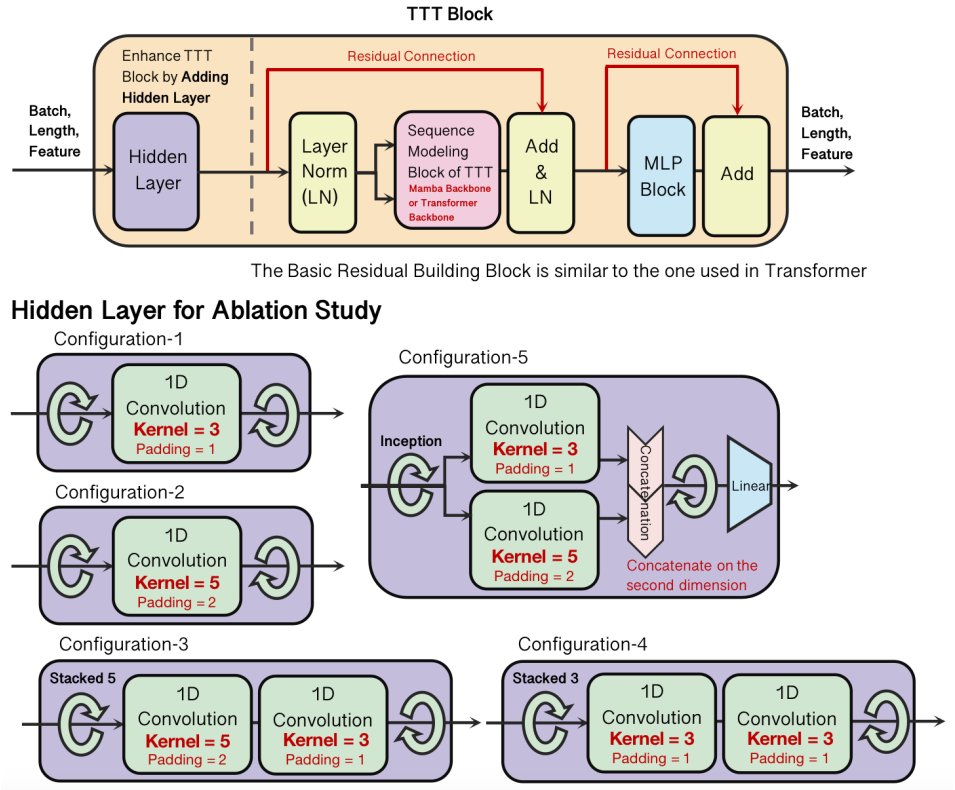


Figure 3: Convolutional Hidden Layer Added to the Beginning of the TTT Block. This basic residual building block is similar to the one used in Transformer models. We use the Hidden Layer as part of an ablation study to evaluate the effects of different hidden layer architectures on model performance. The five configurations are detailed below: (1) 1D Convolution with kernel size 3. (2) 1D Convolution with kernel size 5. (3) Two 1D Convolutions with kernel sizes 5 and 3 in cascade.(4) Two 1D Convolutions with kernel size 3 in cascade. (5) An Inception Block combining 1D Convolutions with kernel sizes 5 and 3, followed by concatenation and reduction to the original size. The Sequence Modeling Block of TTT can be used with two different backbones: the Mamba Backbone and the Transformer Backbone.

Table 2: Results in MSE and MAE (the lower the better) for the long-term forecasting task averaged over 5 runs. We compare extensively with baselines under different prediction lengths, $T = \{96, 192, 336, 720\}$ following the setting of iTransformer [15]. The length of the input sequence (L) is set to 96 for all baselines. TTT (ours) is our TTT block with the Conv Stack 5 architecture. The best results are in **bold** and the second best are underlined.

Methods→		TTT(ours)		TimeMachine		iTransformer		RLinear		PatchTST		Crossformer		TiDE		TimesNet		DLinear		SCINet		FEDformer		Stationary	
$\mathcal{D}$	T	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	0.165	0.214	0.164	**0.208**	0.174	0.214	0.192	0.232	0.177	0.218	**0.158**	0.230	0.202	0.261	0.172	0.220	0.196	0.255	0.221	0.306	0.217	0.296	0.173	0.223
	192	0.225	0.263	0.211	**0.250**	0.221	0.254	0.240	0.271	0.225	0.259	**0.206**	0.277	0.242	0.298	0.219	0.261	0.237	0.296	0.261	0.340	0.276	0.336	0.245	0.285
	336	**0.246**	**0.275**	0.256	0.290	0.278	0.296	0.292	0.307	0.278	0.297	0.272	0.335	0.287	0.335	0.280	0.306	0.283	0.335	0.309	0.378	0.339	0.380	0.321	0.338
	720	**0.339**	**0.343**	0.342	**0.343**	0.358	0.349	0.364	0.353	0.354	0.348	0.398	0.418	0.351	0.386	0.365	0.359	0.345	0.381	0.377	0.427	0.403	0.428	0.414	0.410
Traffic	96	0.397	**0.268**	0.397	**0.268**	**0.395**	**0.268**	0.649	0.389	0.544	0.359	0.522	0.290	0.805	0.493	0.593	0.321	0.650	0.396	0.788	0.499	0.587	0.366	0.612	0.338
	192	0.434	0.287	**0.417**	**0.274**	**0.417**	0.276	0.601	0.366	0.540	0.354	0.530	0.293	0.756	0.474	0.617	0.336	0.598	0.370	0.789	0.505	0.604	0.373	0.613	0.340
	336	**0.430**	0.283	0.433	**0.281**	0.433	0.283	0.609	0.369	0.551	0.358	0.558	0.305	0.762	0.477	0.629	0.336	0.605	0.373	0.797	0.508	0.621	0.383	0.618	0.328
	720	**0.456**	**0.286**	0.467	0.300	0.467	0.302	0.647	0.387	0.586	0.375	0.589	0.328	0.719	0.449	0.640	0.350	0.645	0.394	0.841	0.523	0.626	0.382	0.653	0.355
Electricity	96	**0.135**	**0.230**	0.142	0.236	0.148	0.240	0.201	0.281	0.195	0.285	0.219	0.314	0.237	0.329	0.168	0.272	0.197	0.282	0.247	0.345	0.193	0.308	0.169	0.273
	192	**0.153**	0.254	0.158	**0.250**	0.162	0.253	0.201	0.283	0.199	0.289	0.231	0.322	0.236	0.330	0.184	0.289	0.196	0.285	0.257	0.355	0.201	0.315	0.182	0.286
	336	**0.166**	**0.255**	0.172	0.268	0.178	0.269	0.215	0.298	0.215	0.305	0.246	0.337	0.249	0.344	0.198	0.300	0.209	0.301	0.269	0.369	0.214	0.329	0.200	0.304
	720	**0.199**	**0.285**	0.207	0.298	0.225	0.317	0.257	0.331	0.256	0.337	0.280	0.363	0.284	0.373	0.220	0.320	0.245	0.333	0.299	0.390	0.246	0.355	0.222	0.321
ETTh1	96	**0.352**	**0.375**	0.364	0.387	0.386	0.405	0.386	0.395	0.414	0.419	0.423	0.448	0.479	0.464	0.384	0.402	0.386	0.400	0.654	0.599	0.376	0.419	0.513	0.491
	192	**0.412**	0.418	0.415	**0.416**	0.441	0.436	0.437	0.424	0.460	0.445	0.471	0.474	0.525	0.492	0.436	0.429	0.437	0.432	0.719	0.631	0.420	0.448	0.534	0.504
	336	0.479	0.446	**0.429**	**0.421**	0.487	0.458	0.479	0.446	0.501	0.466	0.570	0.546	0.565	0.515	0.491	0.469	0.481	0.459	0.778	0.659	0.459	0.465	0.588	0.535
	720	0.478	0.454	**0.458**	**0.453**	0.503	0.491	0.481	0.470	0.500	0.488	0.653	0.621	0.594	0.558	0.521	0.500	0.519	0.516	0.836	0.699	0.506	0.507	0.643	0.616
ETTh2	96	**0.274**	**0.328**	0.275	0.334	0.297	0.349	0.288	0.338	0.302	0.348	0.745	0.584	0.400	0.440	0.340	0.374	0.333	0.387	0.707	0.621	0.358	0.397	0.476	0.458
	192	0.373	0.379	**0.349**	**0.381**	0.380	0.400	0.374	0.390	0.388	0.400	0.877	0.656	0.528	0.509	0.402	0.414	0.477	0.476	0.860	0.689	0.429	0.439	0.512	0.493
	336	0.403	0.408	**0.340**	**0.381**	0.428	0.432	0.415	0.426	0.426	0.433	1.043	0.731	0.643	0.571	0.452	0.452	0.594	0.541	1.000	0.744	0.496	0.487	0.552	0.551
	720	0.448	0.434	**0.411**	**0.433**	0.427	0.445	0.420	0.440	0.431	0.446	1.104	0.763	0.874	0.679	0.462	0.468	0.831	0.657	1.249	0.838	0.463	0.474	0.562	0.560
ETTm1	96	**0.309**	**0.348**	0.317	0.355	0.334	0.368	0.355	0.376	0.329	0.367	0.404	0.426	0.364	0.387	0.338	0.375	0.345	0.372	0.418	0.438	0.379	0.419	0.386	0.398
	192	0.371	0.389	**0.357**	**0.378**	0.377	0.391	0.391	0.392	0.367	0.385	0.450	0.451	0.398	0.404	0.374	0.387	0.380	0.389	0.439	0.450	0.426	0.441	0.459	0.444
	336	**0.381**	**0.401**	0.379	**0.399**	0.426	0.420	0.424	0.415	0.399	0.410	0.532	0.515	0.428	0.425	0.410	0.411	0.413	0.413	0.490	0.485	0.445	0.459	0.495	0.464
	720	**0.433**	**0.423**	0.445	0.436	0.491	0.459	0.487	0.450	0.454	0.439	0.666	0.589	0.487	0.461	0.478	0.450	0.474	0.453	0.595	0.550	0.543	0.490	0.585	0.516
ETTm2	96	0.180	**0.253**	**0.175**	0.256	0.180	0.264	0.182	0.265	**0.175**	0.259	0.287	0.366	0.207	0.305	0.187	0.267	0.193	0.292	0.286	0.377	0.203	0.287	0.192	0.274
	192	0.242	0.301	**0.239**	**0.299**	0.250	0.309	0.246	0.304	0.241	0.302	0.414	0.492	0.290	0.364	0.249	0.309	0.284	0.362	0.399	0.445	0.269	0.328	0.280	0.339
	336	0.302	0.341	**0.287**	**0.332**	0.311	0.348	0.307	0.342	0.305	0.343	0.597	0.542	0.377	0.422	0.321	0.351	0.369	0.427	0.637	0.591	0.325	0.366	0.334	0.361
	720	**0.364**	**0.384**	0.371	0.385	0.412	0.407	0.407	0.398	0.402	0.400	1.730	1.042	0.558	0.524	0.408	0.403	0.554	0.522	0.960	0.735	0.421	0.415	0.417	0.413

B.7 Normalization

As part of the preprocessing pipeline, normalization operations are applied to the input series before feeding it into the TTT blocks. The input time series $\mathbf{x}$ is normalized into $\mathbf{x}^0$, represented as:

$$\mathbf{x}^0 = \left[\mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_L^{(0)}\right] \in \mathbb{R}^{M \times L}.$$

We experiment with two different normalization techniques:

- **Z-score normalization**: This normalization technique transforms the data based on the mean and standard deviation of each channel, defined as:
$$x_{i,j}^{(0)} = \frac{x_{i,j} - \text{mean}(x_{i,:})}{\sigma_j},$$
where σ_j is the standard deviation of channel j, and $j = 1, \ldots, M$.
- **Reversible Instance Normalization (RevIN)** [12]: RevIN normalizes each channel based on its mean and variance but allows the normalization to be reversed after the model prediction, which ensures the output predictions are on the same scale as the original input data. We choose to use RevIN in our model because of its superior performance, as demonstrated in [1].

Once the model has generated the predictions, RevIN Denormalization is applied to map the normalized predictions back to the original scale of the input data, ensuring that the model outputs are interpretable and match the scale of the time series used during training.

C Appendix C: Analysis on Increased Prediction and Sequence Length

C.1 Experimental Setup with Enhanced Architectures

To assess the impact of enhancing the model architecture, we conducted experiments by adding hidden layer architectures before the sequence modeling block in each of the four TTT blocks. The

Seq Length	2880	2880	2880	2880	5760	5760	5760	5760	720	720	720	720
Pred Length	192	336	720	96	192	336	720	96	192	336	720	96

Table 3: Testing parameters for sequence and prediction lengths.

	Conv stack 5		TimeMachine		Conv 3		Conv 5		Conv stack 3		Inception		Linear		MLP	
horizon	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
96	**0.259**	**0.289**	0.262	0.292	0.269	0.297	0.269	0.297	0.272	0.300	0.274	0.302	0.268	0.298	0.271	0.301
192	0.316	0.327	**0.307**	**0.321**	0.318	0.329	0.320	0.331	0.319	0.330	0.321	0.330	0.326	0.336	0.316	0.332
336	0.344	0.344	**0.328**	**0.339**	0.348	0.348	0.347	0.347	0.359	0.358	0.361	0.359	0.357	0.358	0.358	0.357
720	0.388	**0.373**	**0.386**	0.378	0.406	0.389	0.400	0.389	0.399	0.387	0.404	0.390	0.414	0.393	0.394	0.393

Table 4: MSE and MAE performance metrics for TimeMachine, TTT, and different convolutional architectures across prediction horizons.

goal was to improve performance by enriching feature extraction through local temporal context. As shown in Figure 3.

Ablation Study Findings Our findings reveal that the introduction of additional hidden layer architectures, including convolutional layers, had varying degrees of impact on performance across different horizons. The best-performing setup was the Conv Stack 5 architecture, which achieved the lowest MSE and MAE at the 96 time horizon, with values of 0.261 and 0.289, respectively, outperforming the TimeMachine model at this horizon. At longer horizons, such as 336 and 720, Conv Stack 5 continued to show competitive performance, though the gap between it and the TimeMachine model narrowed. For example, at the 336 horizon, Conv Stack 5 showed an MSE of 0.344, while TimeMachine had an MSE of 0.328.

However, other architectures, such as Conv 3 and Conv 5, provided only marginal improvements over the baseline TTT architectures (Linear and MLP). While they performed better than Linear and MLP, they did not consistently outperform more complex setups like Conv Stack 3 and Conv Stack 5 across all horizons. This suggests that, although hidden layer expressiveness can enhance model performance, the added complexity from shallow convolutional layers does not always lead to substantial gains relative to the base MLP and Linear TTT hidden layer architectures.

Further exploration into more sophisticated architectures, such as varying convolutional depths or integrating more advanced attention mechanisms, may provide additional opportunities for performance improvements. The current findings indicate that while convolutional layers can be beneficial, more diverse or deeper hidden layers may offer greater potential in future iterations of the model.

C.2 Results and Statistical Comparisons for Proposed Architectures

General Performance The proposed architectures—TTT Linear, TTT MLP, Conv Stack 3, Conv Stack 5, Conv 3, Conv 5, and Inception—demonstrate varying levels of performance across prediction horizons. TTT Linear performs well at shorter horizons, with an MSE of 0.268 and MAE of 0.298 at horizon 96, but experiences increasing error at longer horizons, with an MSE of 0.357 at horizon 336. TTT MLP follows a similar trend but with slightly worse overall performance. Conv 3 and Conv 5 outperform the Linear and MLP models at shorter horizons, achieving comparable MSE (0.269) and MAE (0.297) at horizon 96, but fall behind the more complex Conv Stack models at longer horizons. Conv Stack 5 performs best at shorter and longer horizons. The Inception architecture provides stable performance across horizons, closely following the Conv Stack models, with an MSE of 0.361 at horizon 336.

Impact of Architectures The Conv Stack 5 architecture demonstrates the best overall performance among all convolutional models. Conv 3 and Conv 5 perform better than the simpler Linear and MLP models but are consistently outperformed by the more complex stacked architectures. At horizon 720, Conv 5 shows a marginal improvement over Conv 3, with an MSE of 0.400 compared to 0.406. The Inception architecture performs similarly to Conv Stack 5, offering consistent results across all horizons and proving particularly effective for complex, long-term predictions. Across all horizons, MSE and MAE increase as the horizon lengthens, but Conv Stack 3, Conv Stack 5, and Inception

	TTT		TimeMachine	
Prediction Length	MSE	MAE	MSE	MAE
96	**0.283**	**0.322**	0.309	0.337
192	**0.332**	**0.356**	0.342	0.359
336	**0.402**	**0.390**	0.414	0.394
720	**0.517**	**0.445**	0.535	0.456
1440	**0.399**	**0.411**	0.419	0.429
2880	**0.456**	**0.455**	0.485	0.474
4320	0.580	0.534	**0.564**	**0.523**

Table 5: Average MSE and MAE for different prediction lengths comparing TimeMachine and TTT architectures.

	TTT		TimeMachine	
Sequence Length	MSE	MAE	MSE	MAE
720	**0.312**	**0.336**	0.319	0.341
2880	**0.366**	**0.384**	0.373	0.388
5760	**0.509**	**0.442**	0.546	0.459

Table 6: Average MSE and MAE for different sequence lengths comparing TimeMachine and Conv 3 architectures.

handle this increase more effectively than TimeMachine and other simpler models, which exhibit approximately 50% higher errors at longer horizons.

C.3 Results and Statistical Comparisons for Increased Prediction and Sequence Lengths

Both shorter and longer sequence lengths affect model performance differently. Shorter sequence lengths (e.g., 2880) provide better accuracy for shorter prediction horizons, with the TTT model achieving an MSE of 0.332 and MAE of 0.356 at a 192-step horizon, outperforming TimeMachine. Longer sequence lengths (e.g., 5760) result in higher errors, particularly for shorter horizons, but TTT remains more resilient, showing improved performance over TimeMachine. For shorter prediction lengths (96 and 192), TTT consistently yields lower MSE and MAE compared to TimeMachine. As prediction lengths grow to 720, both models experience increasing error rates, but TTT maintains a consistent advantage. For instance, at a 720-step horizon, TTT records an MSE of 0.517 compared to TimeMachine's 0.535. Overall, TTT consistently outperforms TimeMachine across most prediction horizons, particularly for shorter sequences and smaller prediction windows. As the prediction length increases, TTT's ability to manage long-term dependencies becomes increasingly evident, with models like Conv Stack 5 showing stronger performance at longer horizons.

C.4 Effect of Sequence Length

Shorter Sequence Lengths (e.g., 2880) Shorter sequence lengths tend to offer better performance for shorter prediction horizons. For instance, with a sequence length of 2880 and a prediction length of 192, the TTT model achieves an MSE of 0.332 and an MAE of 0.356, outperforming TimeMachine, which has an MSE of 0.342 and an MAE of 0.359. This indicates that shorter sequence lengths allow the model to focus on immediate temporal patterns, improving short-horizon accuracy.

Longer Sequence Lengths (e.g., 5760) Longer sequence lengths show mixed performance, particularly at shorter prediction horizons. For example, with a sequence length of 5760 and a prediction length of 192, the TTT model's MSE rises to 0.509 and MAE to 0.442, which is better than TimeMachine's MSE of 0.546 and MAE of 0.459. While the performance drop for TTT is less severe than for TimeMachine, longer sequence lengths can introduce unnecessary complexity, leading to diminishing returns for short-term predictions.

C.5 Effect of Prediction Length

Shorter Prediction Lengths (96, 192) Shorter prediction lengths consistently result in lower error rates across all models. For instance, at a prediction length of 96 with a sequence length of 2880, the TTT model achieves an MSE of 0.283 and an MAE of 0.322, outperforming TimeMachine's MSE of 0.309 and MAE of 0.337. This demonstrates that both models perform better with shorter prediction lengths, as fewer dependencies need to be captured.

Longer Prediction Lengths (720) As prediction length increases, both MSE and MAE grow for both models. At a prediction length of 720 with a sequence length of 2880, the TTT model records an MSE of 0.517 and an MAE of 0.445, outperforming TimeMachine, which has an MSE of 0.535 and MAE of 0.456. This shows that while error rates increase with longer prediction horizons, TTT remains more resilient in handling longer-term dependencies than TimeMachine.

D Computational Complexity Comparison

D.1 Complexity Derivation

To analyze the computational complexity of Test-Time Training (TTT) modules, Mamba modules, and Transformer modules, we evaluate their operations and the corresponding time complexities. Let:

- T denote the sequence length.
- d denote the dimensionality of hidden representations.
- N denote the total number of model parameters.
- U denote the number of test-time updates for TTT modules.
- h denote the number of attention heads in Transformer modules.
- k denote the kernel size in convolution operations for Mamba modules.

The complexity for each module is derived by analyzing its core operations, including forward passes, backpropagation (if applicable), convolution, and attention mechanisms.

D.2 Computational Complexity Analysis of Modules

D.2.1 TTT Modules

Test-Time Training modules perform two main tasks at inference:

1. A forward pass through the main model.
2. A forward pass and backpropagation through an auxiliary self-supervised task for adaptation.

Let $O_{\text{forward}}(T, d, N)$ represent the complexity of the forward pass and $O_{\text{backward}}(T, d)$ represent the complexity of backpropagation. The total complexity for TTT modules can be expressed as:

$$O_{\text{TTT}}(T, d, N, U) = O_{\text{forward}}(T, d, N) + U \cdot O_{\text{backward}}(T, d) \tag{1}$$

$$= O(T \cdot d \cdot N) + O(U \cdot T \cdot d^2), \tag{2}$$

where $O(T \cdot d \cdot N)$ accounts for the main forward pass, and $O(U \cdot T \cdot d^2)$ captures the repeated backpropagation steps for U updates.

D.2.2 Mamba Modules

Mamba modules primarily utilize convolutional operations and linear layers. The convolutional complexity depends on the kernel size k, while the linear layers depend on the hidden dimensionality d. The total complexity is given by:

$$O_{\text{Mamba}}(T, d, k) = O(T \cdot k \cdot d) + O(T \cdot d^2), \tag{3}$$

where $O(T \cdot k \cdot d)$ represents the convolution operations, and $O(T \cdot d^2)$ represents the cost of the linear layers.

D.2.3 Transformer Modules

Transformer modules consist of two main components:

1. Multi-head self-attention, which requires matrix multiplication of dimension $T \times d$ with $T \times d$ to compute attention scores, leading to $O(T^2 \cdot d)$ complexity.
2. A feedforward network, which processes the sequence independently, contributing $O(T \cdot d^2)$ complexity.

The total complexity of Transformer modules is therefore:

$$O_{\text{Transformer}}(T, d) = O(T^2 \cdot d) + O(T \cdot d^2). \tag{4}$$

D.2.4 Convolutional Block in ModernTCN

ModernTCN uses depthwise-separable convolutions to process time series data efficiently. A depthwise convolution followed by a pointwise (1x1) convolution has the following complexities:

- Depthwise convolution: $O(T \cdot k \cdot C_{\text{in}})$, where k is the kernel size.
- Pointwise convolution: $O(T \cdot C_{\text{in}} \cdot C_{\text{out}})$.

The total complexity of the convolutional block is:

$$O_{\text{ModernTCN}}(T, C_{\text{in}}, C_{\text{out}}, k) = O(T \cdot k \cdot C_{\text{in}}) + O(T \cdot C_{\text{in}} \cdot C_{\text{out}}). \tag{5}$$

D.3 Comparison of Complexities

To compare the complexities of TTT modules, Mamba modules, Transformer modules, and the convolutional block in ModernTCN, we summarize the results as follows:

$$O_{\text{TTT}}(T, d, N, U) = O(T \cdot d \cdot N) + O(U \cdot T \cdot d^2), \tag{6}$$

$$O_{\text{Mamba}}(T, d, k) = O(T \cdot k \cdot d) + O(T \cdot d^2), \tag{7}$$

$$O_{\text{Transformer}}(T, d) = O(T^2 \cdot d) + O(T \cdot d^2), \tag{8}$$

$$O_{\text{ModernTCN}}(T, C_{\text{in}}, C_{\text{out}}, k) = O(T \cdot k \cdot C_{\text{in}}) + O(T \cdot C_{\text{in}} \cdot C_{\text{out}}). \tag{9}$$

From these equations:

- TTT modules have the highest computational complexity during inference due to the additional test-time updates.
- Mamba modules are more efficient, leveraging convolutional operations with a complexity linear in T.
- Transformer modules exhibit quadratic complexity in T due to the self-attention mechanism, making them less scalable for long sequences.

E Computational Complexity Analysis of Models

E.1 Test-Time Learning for Time Series Forecasting (TTT-LTSF)

Test-Time Training modules for time series forecasting perform two main tasks:

1. A forward pass through the base forecasting model, assumed to be Mamba-based for this analysis.
2. Test-time updates using a self-supervised auxiliary task.

Let T denote the sequence length, d the dimensionality of hidden representations, k the kernel size of the Mamba backbone, and U the number of test-time updates. The computational complexity of the Mamba backbone is:

$$O_{\text{Mamba}}(T, d, k) = O(T \cdot k \cdot d) + O(T \cdot d^2), \tag{10}$$

where $O(T \cdot k \cdot d)$ represents convolutional operations and $O(T \cdot d^2)$ accounts for linear layers.

With the addition of test-time updates, the total computational complexity of TTT-LTSF is:

$$O_{\text{TTT-LTSF}}(T, d, k, U) = O(T \cdot k \cdot d) + O(T \cdot d^2) + O(U \cdot T \cdot d^2), \tag{11}$$

where $O(U \cdot T \cdot d^2)$ captures the overhead introduced by test-time optimization.

E.2 TimeMachine

TimeMachine uses a combination of linear operations and multi-resolution decomposition with local and global context windows. Its computational complexity is:

$$O_{\text{TimeMachine}}(T, d) = O(T \cdot d) + O(T \cdot d^2), \tag{12}$$

where $O(T \cdot d)$ represents linear operations, and $O(T \cdot d^2)$ arises from context-based decomposition.

E.3 PatchTST

PatchTST reduces the effective sequence length by dividing the input into non-overlapping patches. Let patch_size denote the size of each patch, resulting in an effective sequence length $T_p = T/\text{patch\_size}$. The complexity is:

$$O_{\text{PatchTST}}(T, d, \text{patch\_size}) = O(T \cdot d) + O(T_p^2 \cdot d) + O(T_p \cdot d^2) \tag{13}$$

$$= O(T \cdot d) + O\left(\left(\frac{T}{\text{patch\_size}}\right)^2 \cdot d\right) + O\left(\frac{T}{\text{patch\_size}} \cdot d^2\right). \tag{14}$$

E.4 TSMixer

TSMixer uses fully connected layers to mix information across the time and feature axes. Its complexity is:

$$O_{\text{TSMixer}}(T, d) = O(T \cdot d^2) + O(d \cdot T^2), \tag{15}$$

where $O(T \cdot d^2)$ represents time-axis mixing and $O(d \cdot T^2)$ represents feature-axis mixing.

E.5 ModernTCN

ModernTCN employs depthwise-separable convolutions to process time series data efficiently. Let C_{in} and C_{out} denote the input and output channel dimensions, and k the kernel size. The complexity is:

$$O_{\text{ModernTCN}}(T, C_{\text{in}}, C_{\text{out}}, k) = O(T \cdot k \cdot C_{\text{in}}) + O(T \cdot C_{\text{in}} \cdot C_{\text{out}}), \tag{16}$$

where $O(T \cdot k \cdot C_{\text{in}})$ is for depthwise convolutions and $O(T \cdot C_{\text{in}} \cdot C_{\text{out}})$ for pointwise convolutions.

E.6 iTransformer

iTransformer applies self-attention across variate dimensions rather than temporal dimensions. Let N denote the number of variates, T the sequence length, and d the hidden dimension size:

$$O_{\text{iTransformer}}(T, N, d) = O(T \cdot N^2 \cdot d) + O(T \cdot N \cdot d^2), \tag{17}$$

where $O(T \cdot N^2 \cdot d)$ arises from self-attention across variates and $O(T \cdot N \cdot d^2)$ from the feedforward network.

E.7 Comparison of Complexities

The complexities of the models analyzed are as follows:

$$O_{\text{TTT-LTSF}}(T, d, k, U) = O(T \cdot k \cdot d) + O(T \cdot d^2) + O(U \cdot T \cdot d^2), \tag{18}$$

$$O_{\text{TimeMachine}}(T, d) = O(T \cdot d) + O(T \cdot d^2), \tag{19}$$

$$O_{\text{PatchTST}}(T, d, \text{patch\_size}) = O(T \cdot d) + O(T_p^2 \cdot d) + O(T_p \cdot d^2), \tag{20}$$

$$O_{\text{TSMixer}}(T, d) = O(T \cdot d^2) + O(d \cdot T^2), \tag{21}$$

$$O_{\text{ModernTCN}}(T, C_{\text{in}}, C_{\text{out}}, k) = O(T \cdot k \cdot C_{\text{in}}) + O(T \cdot C_{\text{in}} \cdot C_{\text{out}}), \tag{22}$$

$$O_{\text{iTransformer}}(T, N, d) = O(T \cdot N^2 \cdot d) + O(T \cdot N \cdot d^2). \tag{23}$$

E.8 Summary of Model Complexities

- **TTT-LTSF**: Incorporates the complexity of the Mamba backbone ($O(T \cdot k \cdot d + T \cdot d^2)$) with additional overhead for test-time updates ($O(U \cdot T \cdot d^2)$).
- **TimeMachine**: Combines efficient linear operations and multi-resolution decomposition, maintaining a linear dependency on T for most operations.
- **PatchTST**: Reduces sequence length via patch embedding, resulting in a complexity dependent on $T_p = T/\text{patch\_size}$.
- **TSMixer**: Uses fully connected layers for time and feature mixing but suffers from quadratic dependency on T or d, making it less scalable.
- **ModernTCN**: Relies on depthwise-separable convolutions, achieving linear complexity in T while maintaining flexibility in channel dimensions (C_{in}, C_{out}).
- **iTransformer**: Applies self-attention across variates (N) instead of the temporal axis (T), making it efficient for long sequences with a limited number of variates.

E.9 Key Insights

- **Efficiency**: - **ModernTCN** and **TimeMachine** are the most efficient for long sequences due to their linear dependency on T. - **PatchTST** benefits from sequence length reduction via patch embedding, but its quadratic dependency on T_p makes it less scalable for small patch sizes.
- **Robustness**: - **TTT-LTSF** (with Mamba) introduces additional adaptability through test-time updates, enhancing robustness to distribution shifts. The use of a Mamba backbone keeps the complexity manageable compared to Transformer-based backbones.
- **Dimensionality Impact**: - **TSMixer** struggles with high-dimensional data due to its quadratic dependency on T or d, making it less practical for large-scale applications. - **iTransformer** scales better when the number of variates (N) is smaller than the sequence length (T).
- **Scalability**: - **ModernTCN** and **TimeMachine** remain scalable for both long sequences and high-dimensional data. - **iTransformer** is effective for scenarios with long sequences but limited variates, avoiding the quadratic cost of traditional self-attention across T.