Message Propagation Through Time: An Algorithm for Sequence Dependency Retention in Time Series Modeling

Shaoming Xu * Ankush Khandelwal * Arvind Renganathan * Vipin Kumar *

Abstract

Time series modeling, a crucial area in science, often encounters challenges when training Machine Learning (ML) models like Recurrent Neural Networks (RNNs) using the conventional mini-batch training strategy that assumes independent and identically distributed (IID) samples and initializes RNNs with zero hidden states. The IID assumption ignores temporal dependencies among samples, resulting in poor performance. This paper proposes the Message Propagation Through Time (MPTT) algorithm to effectively incorporate long temporal dependencies while preserving faster training times relative to the stateful algorithms. MPTT utilizes two memory modules to asynchronously manage initial hidden states for RNNs, fostering seamless information exchange between samples and allowing diverse mini-batches throughout epochs. MPTT further implements three policies to filter outdated and preserve essential information in the hidden states to generate informative initial hidden states for RNNs, facilitating robust training. Experimental results demonstrate that MPTT outperforms seven strategies on four climate datasets with varying levels of temporal dependencies.

Keywords: Time Series modeling, Long-Term Dependencies, Neural Networks, Mini-Batch Training

1 Introduction

Machine learning (ML) models have achieved remarkable success in commercial applications such as computer vision and natural language processing, prompting the scientific community to consider them as viable alternatives to physics-based models [1, 2]. Time series modeling, which seeks to uncover patterns, trends, and complex relationships in time-based data, is critical to scientific discovery [3]. To train ML models on long time series, a common practice is to divide the time series into shorter sequence samples to reduce computational complexity and memory usage, and mitigate vanishing/exploding gradient issues [4, 5].

However, the conventional random mini-batch algorithms treat these equal-sized sequences as IID and initialize RNNs with zero hidden states, leading to a loss of dependencies between the sequences. As a result, the trained RNNs cannot capture long-term temporal dependencies across multiple sequences and thus have limited performance for modeling long time series [6, 7].

State-based approaches have been proposed to use hidden states to transfer temporal information across sequences. Stateful RNNs [8] pass hidden states between mini-batches, but they treat sequences within each mini-batch as independent, necessitating smaller mini-batch sizes to maintain long-term dependencies [9, 10, 11]. This results in unstable training and extended training time [12]. The Sequential Stateful RNNs [7] can preserve long-term dependencies by passing hidden states both within and between mini-batches. However, it requires each sequence within the batch to be processed sequentially, which makes it slow to train.

Response-based approaches use response values as additional inputs to encode temporal dependencies between sequences [7, 13, 14]. These approaches are derived from ideas in dynamical control theory to adaptively let RNNs estimate responses as the initial system conditions for each sequence [13, 15, 16, 17, 18]. However, response-based approaches may suffer error accumulation during inference, as they primarily focus on learning responses' local changes in each timestep, impeding the learning of accumulated changes over multiple steps [7]. Moreover, they may struggle when responses encode limited temporal information.

To address these challenges, we propose a novel algorithm, Message Propagation Through Time (MPTT), which preserves long-term temporal dependencies by seamlessly transferring informative hidden states between sequences during training. First, MPTT utilizes two memory modules to enable asynchronous communication of hidden states between sequences. This approach supports shuffled sequences and diverse mini-batches across epochs, preserving long-term dependencies while improving computational efficiency relative to state-based approaches. Second, MPTT incorporates three policies to generate informative initial hidden states by filtering outdated information and retain-

*University of Minnesota. {xu000114, khand035,renga016,kumar001}@umn.edu

ing essential information in the hidden states throughout epochs. This facilitates more robust training than the stateful RNN approaches. Finally, MPTT addresses the error accumulation issue found in response-based approaches by utilizing informative hidden states, which possess a greater capacity than response values to encode the temporal effects and learn cumulative changes across timesteps. We highlight the advantage of these components in modeling long time series across four climate datasets with diverse temporal dependency levels. We have released our codes, datasets, pre-trained models, and appendix to include additional implementation details needed to reproduce our results [1].

2 Background and Preliminaries

2.1 Problem Statement. The goal is to accurately predict a sequence of outputs from the given time series of inputs. We primarily focus on the many-to-many prediction; however, the many-to-one is also possible but entails higher computational overhead. The overarching aim is to develop effective algorithms to transfer information across sequences.

2.2 Independent training and inference.

Random Mini-Batches (RMB) training (Figure 1 plot A) is a common mini-batch gradient descent method [19] for RNNs, which involves dividing longer time series into shorter sequences and treating each sequence independently. In each epoch, RMB randomly allocates sequences to multiple mini-batches, initializes the hidden states to $\vec{0}$ for each sequence, and trains the models on each mini-batch until all mini-batches have been processed.

Independent Inference (IIF) (Figure 1, plot F) is the inference algorithm that considers each test sequence IID and makes predictions using $\vec{0}$ as initial states. The combination of RMB and IIF algorithms represents a widely-used learning strategy in the ML community. However, it neglects the interactions between sequences, potentially leading to suboptimal performance in time series modeling. Both RMB and IIF generate RNNs' hidden states following the equation:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ &= f(x_t, f(x_{t-1}, f(x_{t-2}, \dots f(x_1, \vec{0}) \dots))) \end{aligned} \tag{2.1}$$

where x denotes the input sequence, and x_t and h_t denote the data and hidden state at the time step t of the sequence, respectively. W denotes the length of the sequence, and $0 < t < W$. Next, we describe existing strategies to incorporate temporal dependencies.

[1] https://github.com/XuShaoming/Message-Propagation-Through-Time

2.3 State-based approaches aim to transfer temporal information between sample sequences through the initial hidden state h_0 as follows:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ &= f(x_t, f(x_{t-1}, f(x_{t-2}, \dots f(x_1, h_0) \dots))) \end{aligned} \tag{2.2}$$

In equation 2.2, h_0 is the hidden state one step before the current input sequence x and is computed as the last state of the previous sequence using the same model f. In the implementation, h_0 is detached from the computational graph to prevent error backpropagation over sequences and reduce computational cost. The idea behind state-based approaches is that initial hidden states can summarize the temporal effects of previous sequences, thereby preserving sequence dependencies.

Stateful Mini-Batches (SMB) training (Figure 1 plot B) uses the last hidden states of RNN sequences from a mini-batch to initialize the sequences in the next mini-batch to train Stateful RNNs [8]. However, SMB still treats each sequence within the mini-batch as independent, limiting their ability to capture long-term dependencies. Consequently, researchers often have to compromise by using smaller mini-batch sizes to link more sequences for applications with long-term dependencies [9, 10, 11], which can result in a prolonged and unstable training process [12].

Sequential Stateful Mini-Batches (SSMB) training (Figure 1 plot C) can preserve long-term dependencies while allowing model training on larger mini-batches for stable training by passing hidden states both within and between mini-batches. However, the fixed sequence order required by both SMB and SSMB for hidden state transmission can potentially lead to overfitting. Moreover, SSMB requires additional computational cost, as it processes each sequence individually to pass hidden states sequentially [7].

Sequential Stateful inference (SSIF) (Figure 1, plot G) serves as the default inference algorithm for the state-based training algorithms. It passes hidden states between test sequences according to their temporal order to generate predictions in a single pass. As SSIF prevents RNNs from reconstructing hidden states from scratch for each sequence during inference, SSIF can also improve the predictions of RMB-trained models [7].

2.4 Response-based approaches encode the temporal relationship between sequences through response variables, akin to ideas in dynamical control theory to adaptively estimate responses as the system conditions for predictions [13, 15, 16, 17, 18]. These approaches employ previous response values as additional inputs, aiding the model in learning temporal variations.

Teacher Forcing training (TF) (Figure 1, plot

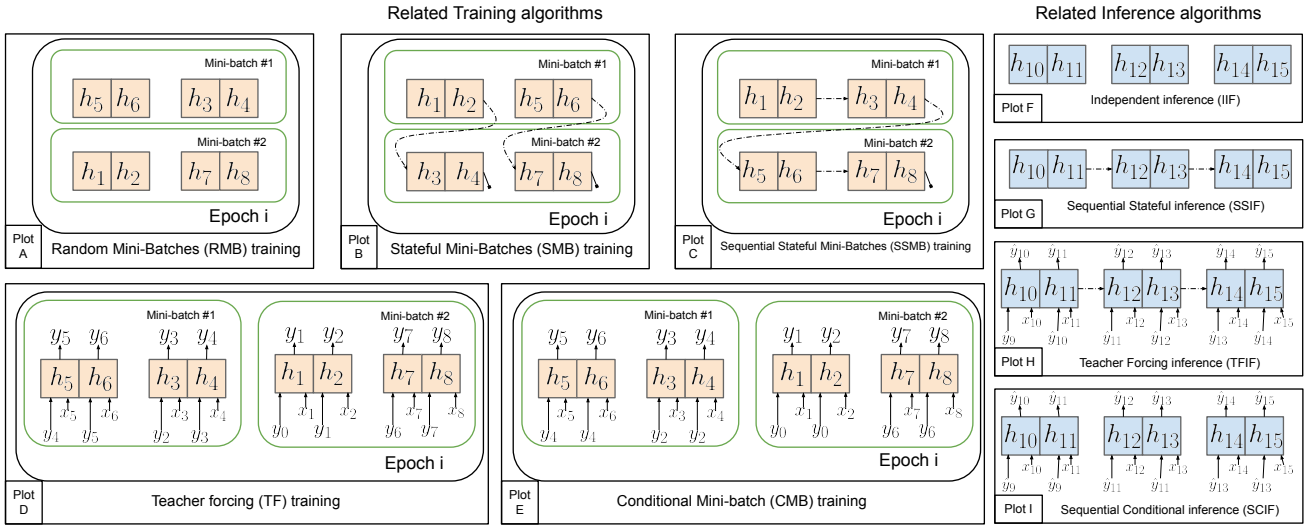

Figure 1: The related training and inference algorithms feed forward information flow between sequences, while backpropagation of losses across sequences is disallowed. For clarity, the sequence length is set to two, and inputs and outputs are omitted in some plots.

D) utilizes the observed response value from the previous timestep as an additional input for the next timestep, as shown in equation 2.3 [13]. TF uses the initial response value y_0 from the previous sequences to maintain dependencies between the current and previous sequences. However, TF struggles with time-series modeling because it focuses on learning local changes relative to previous response values, leading to error accumulation in the inference phase [7].

Scheduled sampling (SSPL) training [14] mitigates the exposure bias of TF by incrementally replacing the observed responses with the predictions as additional inputs during training. However, SSPL does not solve TF's inability to learn accumulated changes in time series modeling.

Teacher Forcing inference (TFIF) (Figure 1, plot H) serves as the inference algorithm for TF and SSPL trained RNNs. TFIF incorporates the previous timestep's predicted response as additional input for the next timestep [13]. However, since TF and SSPL cannot learn accumulated changes during training, any errors made by TFIF during inference can accumulate over time, leading to poor time-series predictions.

$$h_t = f(x_t, y_{t-1}, h_{t-1}) = f(x_t, y_{t-1}, f(x_{t-1}, y_{t-2}, f(x_{t-2}, y_{t-3}, \dots f(x_1, y_0, \vec{0}) \dots))) \tag{2.3}$$

Conditional Mini-Batches (CMB) training (Figure 1, plot E) mitigates the error accumulation issue in TF and SSPL by using a single response value as an initial condition for each sequence, specifically, the response value one timestep before the start of each training sequence. This initial response value is replicated across the sequence, maintaining the input vector's dimensionality and offering the corresponding RNN sequence a starting point to learn accumulated changes over time (as shown in equation 2.4) [7]. CMB is inspired by the multiple shooting method in control theory, where the entire time series is divided into sequences, and an initial condition is estimated for each sequence[17, 18]. CMB can effectively train RNNs for applications characterized by long-term dependencies, as their responses encapsulate the temporal effects of previous timesteps. However, for applications where the response variables encapsulate limited temporal information, CMB is not better than RMB [7].

Sequential conditional inference (SCIF) (Figure 1, plot I) serves as the inference algorithm for CMB. SCIF maintains the temporal order of sequences and sequentially uses the initial response predicted from the previous sequence as the additional input for CMB-trained models at the current sequence to obtain all predictions in one pass [7].

$$h_t = f(x_t, y_0, h_{t-1}) = f(x_t, y_0, f(x_{t-1}, y_0, f(x_{t-2}, y_0, \dots f(x_1, y_0, \vec{0}) \dots))) \tag{2.4}$$

3 Proposed Approach

The primary objective of the MPTT algorithm is to maintain the computational efficiency of mini-batch training while enforcing temporal dependencies between

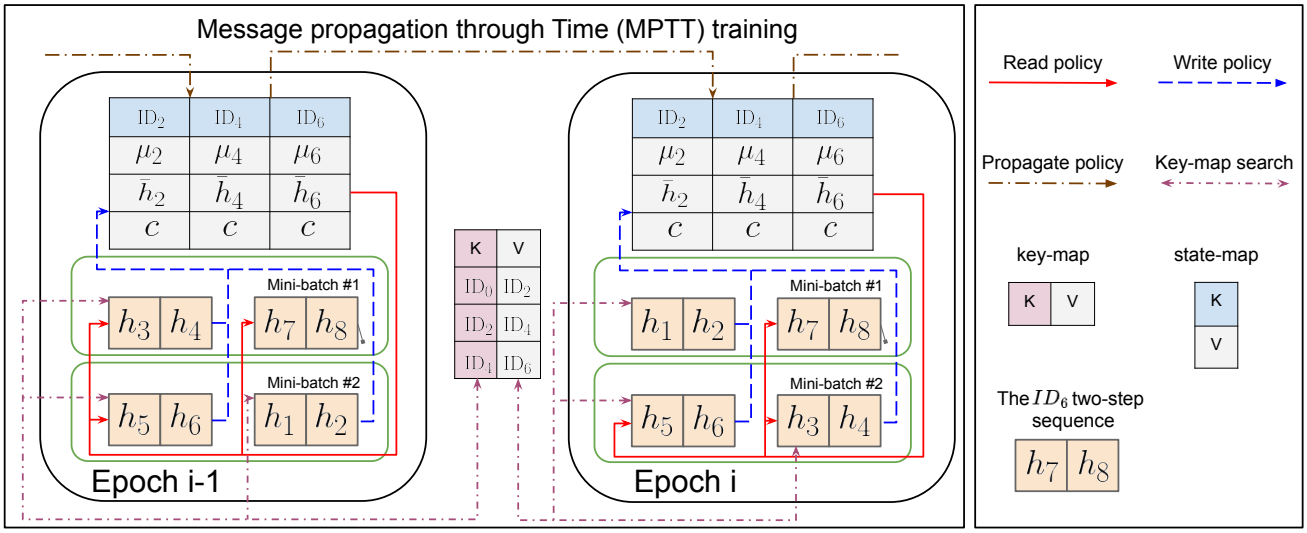

Figure 2: The proposed Message Propagation through Time (MPTT) algorithm that designs two memory modules and three policies to generate informative initial hidden states to facilitate seamless communication between RNN sequences during training.

sequences. The core idea in MPTT is to use informative hidden states, denoted as $\tilde{h}_0$, as the messages to transmit temporal information among sequences as follows:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ &= f(x_t, f(x_{t-1}, f(x_{t-2}, \dots f(x_1, \tilde{h}_0) \dots))) \end{aligned} \tag{3.5}$$

Next, we outline the memory modules and policies that MPTT employs to manage message propagation throughout the training procedure.

3.1 Memory modules In state-based approaches, subsequent sequences must await the generation of hidden states from preceding ones to initiate training, which dictates the sequential nature of the approaches and can lead to overfitting and suboptimal training [7, 8]. MPTT overcomes these issues by employing two memory modules, *key-map* and *state-map*, to enable asynchronous reading and writing of messages as the informative initial hidden states for RNNs, as illustrated in Figure 2. This facilitates seamless information sharing between sequences, allowing for shuffled sequences and diverse mini-batches to address overfitting and local minima during mini-batch training.

Key-map. MPTT assigns each sequence S a unique ID:$\{T\}$, where T denotes the initial step of S. The initial step refers to the timestep one step before S in the original time series. For instance, a sequence with ID:$\{365\}$ starts at timestep 366 in the time series.

The key-map comprises key-value pairs, where each key corresponds to the ID of a specific sequence S, and its value is a list of IDs representing the sequences whose initial steps are covered in S. A smaller stride of the sliding window can result in more overlapped sequences, leading to longer lists in the key-map. The key-map can be represented as follows:

$$\textbf{key-map}(\mathrm{ID}_i) = [\mathrm{ID}_j | \forall T_j, T_i < T_j \leq T_i + W] \tag{3.6}$$

where W is the sequence length, and ID_i and T_i represent S_i and its initial step, respectively.

Before starting the training process, a key-map is initialized for each sequence using Equation 3.6 and remains fixed without new training data. The key-map serves as a mediator that enables a given sequence S to identify other sequences that require its generated hidden states for initializing their hidden states.

State-map. We can represent the state-map as

$$\textbf{state-map}(\mathrm{ID}) = (\mu_0, \bar{h}_0, c), \tag{3.7}$$

The state-map maintains three variables, $(\mu_0, \bar{h}_0, c)$, for each sequence S, which are used to generate the initial message $\tilde{h}_0$ for initializing the hidden states. μ_0 encapsulates information from a collection of initial hidden states, h_0's, produced in prior epochs for a sequence S. $\bar{h}_0$ represents a running average of h_0's generated during the current epoch, while c denotes the count of h_0's in the current epoch for S. Before the start of training, a state-map is initialized for each sequence using Equation 3.7, but with $(\vec{0}, \vec{0}, 0)$ as the assigned value. The state-map functions as a repository for each sequence, storing pertinent information needed to generate the messages $\tilde{h}_0$'s as initial hidden states.

3.2 Policies MPTT employs read, write, and propagate policies to maintain the message $\tilde{h}_0$ and its $\{\mu_0, \bar{h}_0, c\}$ for each sequence S during training. In each epoch, MPTT randomly partitions the sequences into multiple mini-batches without replacement. It then iterates through each mini-batch to train the model.

Read policy. For each sequence in a mini-batch, MPTT reads its $\{\mu_0, \bar{h}_0, c\}$ from the *state-map* and generates $\tilde{h}_0$ as follows:

$$\tilde{h}_0 = \begin{cases} \mu_0 & \text{if } \delta = 0, c = 0 \\ \frac{\delta\mu_0 + c\bar{h}_0}{\delta + c} & \text{Otherwise} \end{cases} \tag{3.8}$$

The binary value $\delta = \{0, 1\}$ is called the message keeper. If $\delta = 0$, *read policy* will disregard μ_0 as soon as $\bar{h}_0$ is updated in the current epoch. If $\bar{h}_0$ has not been updated ($c = 0$), *read policy* will always utilize μ_0 as the initial hidden state for RNNs to generate the prediction $\hat{y}$ and hidden state h at each timestep of the sequence S. The *read policy* sets initial hidden states for each sequence at the beginning of the mini-batch.

Write Policy. After the optimization step, each sequence S in the current mini-batch uses the *key-map* to identify corresponding sequences that require its generated hidden states for their *state-map* updates. For a sequence ID_j in the key-map of the sequence ID_i, the hidden state generated during the forward pass at the $T_j - T_i$ timestep of sequence ID_i is used as h_0 to update the *state-map* for sequence ID_j as follows:

$$\mu_0 = \mu_0, \quad \bar{h}_0 = \frac{c\bar{h}_0 + h_0}{c + 1}, \quad c = c + 1 \tag{3.9}$$

Propagate policy. The *Propagate Policy* is executed at the end of each epoch, generating a new μ_0 and resetting both $\bar{h}_0$ and c in the *state-map* for each sequence as follows:

$$\mu_0 = \frac{\delta\mu_0 + c\bar{h}_0}{\delta + c}, \qquad \bar{h}_0 = \vec{0}, \qquad c = 0 \tag{3.10}$$

If message keeper $\delta = 0$, *propagate policy* will disregard μ_0 derived from previous epochs and rely solely on $\bar{h}_0$ from the current epoch as the new μ_0 to initialize hidden states in the succeeding epoch.

3.3 Forgetting mechanism We can expand equation 3.10 and check the μ_0 in each epoch E as follows:

$$\mu_0 = \begin{cases} \vec{0} & \text{if } E = 1 \\ \bar{h}_0^{<1>} & \text{if } E = 2 \\ \frac{\delta^{E-2}\bar{h}_0^{<1>}}{(\delta+c)^{E-2}} + \sum_{e=3}^{E} \frac{\delta^{E-e} c\bar{h}_0^{<e-1>}}{(\delta+c)^{E-e+1}} & \text{if } E > 2 \end{cases} \tag{3.11}$$

Where $E > 2$ case is summarized from this expanded form:

$$\mu_0 = \frac{\delta^{E-2}\bar{h}_0^{<1>}}{(\delta+c)^{E-2}} + \frac{\delta^{E-3}c\bar{h}^{<2>}}{(\delta+c)^{E-2}} + \frac{\delta^{E-4}c\bar{h}^{<3>}}{(\delta+c)^{E-3}} + \cdots + \frac{\delta c\bar{h}^{<E-2>}}{(\delta+c)^2} + \frac{c\bar{h}^{<E-1>}}{(\delta+c)} \tag{3.12}$$

Here, $\bar{h}_0^{<e>}$ is the average of h_0's, and c is the total count of the h_0's generated at the end of the epoch e for a sequence S. These equations reveal that MPTT utilizes three types of forgetting mechanisms to control the messages propagating over epochs.

Message keeper δ. MPTT employs the message keeper $\delta = \{0, 1\}$ to control the amount of information propagated. When $\delta = 0$, the *read policy* disregards μ_0 as soon as $\bar{h}_0$ has been updated in the current epoch. The *propagate policy* discards μ_0 derived from previous epochs and solely relies on $\bar{h}_0^{<E-1>}$ as the new μ_0 for the epoch E.

Exponential decay. Conversely, when $\delta = 1$, both the *read* and *propagate* policies utilize the full list of $\bar{h}_0$'s generated from previous epochs to obtain $\tilde{h}_0$ and μ_0. Equation 3.12 shows that this process is regulated by exponential decay on the low-quality $\bar{h}_0$'s generated by under-trained RNNs in early epochs, prioritizing $\bar{h}_0$'s from more recent epochs to obtain new μ_0 for the subsequent epoch.

Auto-adjusted forgetting. Equation 3.12 shows how the decay speed is automatically adjusted by the value of c. At the end of each epoch, c corresponds to the total number of h_0's generated for each sequence S. A larger c indicates more overlapping sequences and an augmented dataset for training the RNNs, resulting in a wider array of h_0's that are generated and aggregated to yield $\bar{h}_0$ with reduced variance in each epoch. Consequently, MPTT employs equation 3.12 to accelerate the decay process when c is larger, thereby emphasizing the importance of recently generated $\bar{h}_0$'s in producing new μ_0.

4 Results and Discussion

We evaluate the performance of MPTT algorithms by comparing them with RMB, state-based, and response-based algorithms for training RNN models on four climate datasets. Combined with five inference algorithms used during testing, this leads to nine learning strategies for comparison.

4.1 Synthetic data experiments The Soil & Water Assessment Tool (SWAT) is a river basin model for simulating surface and groundwater dynamics in complex watersheds and assessing climate change impacts [20, 21]. In this experiment, we evaluate our algo-

Table 1: The table displays experiment results where the sequence length is fixed at 366 steps. Models are trained using different percentages of the 500-year training set and evaluated on the testing set to measure their relative performance in RMSE. The results indicate that, when provided with more data, MPTTs reach lower RMSEs more quickly than other approaches, highlighting their ability to effectively leverage extra data for robust time-series modeling.

Algo \ Data		SWAT-SW				SWAT-SNO				SWAT-SF			
Train	**Inference**	**2%**	**16%**	**32%**	**100%**	**2%**	**16%**	**32%**	**100%**	**2%**	**16%**	**32%**	**100%**
RMB	IIF	41.1 ± 0.8	36.9 ± 0.4	34.8 ± 0.4	32.8 ± 0.2	7.24 ± 0.38	4.15 ± 0.29	3.14 ± 0.15	2.57 ± 0.14	1.52 ± 0.07	0.89 ± 0.03	0.79 ± 0.02	0.66 ± 0.02
RMB	SSIF	36.4 ± 1.5	27.3 ± 1.2	23.7 ± 1.6	20.1 ± 0.7	7.13 ± 0.36	3.75 ± 0.37	2.39 ± 0.21	1.67 ± 0.25	1.49 ± 0.08	0.81 ± 0.03	0.7 ± 0.03	0.53 ± 0.05
TF	TFIF	35.4 ± 2.6	27.9 ± 5.4	26 ± 7.5	27.7 ± 7.0	$\mathbf{6.58 \pm 0.68^*}$	$\mathbf{1.49 \pm 0.23^*}$	$\mathbf{1.22 \pm 0.17^*}$	1.61 ± 0.86	1.63 ± 0.04	0.83 ± 0.04	0.81 ± 0.09	0.67 ± 0.04
SSPL	TFIF	49.3 ± 8.4	34.8 ± 6.5	30 ± 5.7	34.1 ± 12.5	8.14 ± 1.41	$\mathbf{1.76 \pm 0.4}$	$\mathbf{1.31 \pm 0.23}$	$\mathbf{1.06 \pm 0.28}$	$\mathbf{1.48 \pm 0.08}$	0.83 ± 0.02	0.76 ± 0.06	0.61 ± 0.03
CMB	SCIF	35.6 ± 0.9	23.1 ± 0.6	19.1 ± 0.8	16.4 ± 0.3	6.96 ± 0.27	2.61 ± 0.2	1.95 ± 0.12	1.07 ± 0.05	1.61 ± 0.04	0.9 ± 0.02	0.81 ± 0.02	0.68 ± 0.01
SMB	SSIF	36 ± 1.6	27.2 ± 1.2	26.4 ± 0.5	18.2 ± 0.6	7.05 ± 0.23	3.73 ± 0.2	2.18 ± 0.24	1.4 ± 0.09	1.55 ± 0.06	0.86 ± 0.02	$0.77 \pm 0.$	0.54 ± 0.06
SSMB	SSIF	36.9 ± 1.2	$\mathbf{22.6 \pm 0.3^*}$	18.2 ± 1.1	14 ± 0.3	$\mathbf{6.74 \pm 0.23}$	3.25 ± 0.23	2.27 ± 0.22	1.39 ± 0.09	1.55 ± 0.06	0.77 ± 0.06	0.7 ± 0.08	0.49 ± 0.03
MPTT($\delta = 0$)	SSIF	$\mathbf{35 \pm 0.9^*}$	23.2 ± 1.3	$\mathbf{17.2 \pm 1.3}$	$\mathbf{11.5 \pm 0.6}$	7.19 ± 0.36	2.69 ± 0.24	1.91 ± 0.12	$\mathbf{1.04 \pm 0.1^*}$	1.51 ± 0.05	$\mathbf{0.76 \pm 0.05}$	$\mathbf{0.65 \pm 0.05}$	$\mathbf{0.48 \pm 0.07}$
MPTT($\delta = 1$)	SSIF	$\mathbf{35.4 \pm 0.7}$	$\mathbf{22.8 \pm 0.4}$	$\mathbf{14.1 \pm 0.8^*}$	$\mathbf{10.9 \pm 0.4^*}$	6.89 ± 0.33	2.62 ± 0.25	1.8 ± 0.18	1.12 ± 0.11	$\mathbf{1.38 \pm 0.02^*}$	$\mathbf{0.71 \pm 0.07^*}$	$\mathbf{0.62 \pm 0.06^*}$	$\mathbf{0.44 \pm 0.02^*}$

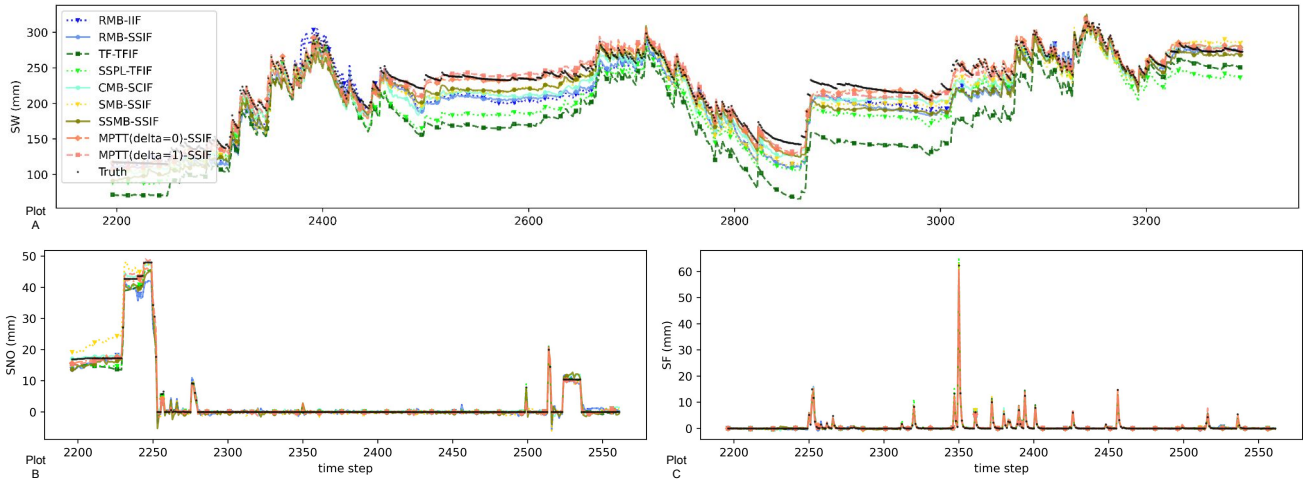

Figure 3: Comparisons between observed and predicted time series reveal algorithmic performance variations across datasets with different temporal dependencies. SW exhibits long-term dependencies, SNO has seasonal dependencies due to snowmelt, and SF features short-term dependencies driven by precipitation and snowmelt. The performance gap narrows from the SW to the SF tasks, indicating that MPTT algorithms excel particularly in scenarios with extended temporal dependencies.

rithms by training the Gated Recurrent Units (GRU) models [22] as SWAT surrogate models. We generate three 1000-year daily-scale time series datasets for a Minnesota watershed using the SWAT model. These datasets, *SWAT-SW*, *SWAT-SNO*, and *SWAT-SF*, represent soil water (SW), snowpack (SNO), and streamflow (SF) respectively. As Figure 3 shows, SW has a long-term temporal dependency, reflecting soil water content changes. SNO exhibits a seasonal dependency, as snow typically melts by summer's end. SF, with a short-term temporal dependency, is mainly influenced by precipitation and snowmelt events. These responses depend on input weather drivers like precipitation, temperature, solar radiation, wind speed, and relative humidity (see appendix B.1).

Experimental Setup. We split the time series into 500 years of training, 100 years of validation, and 400 years of testing sets in temporal order. The data is Gaussian-normalized before being segmented into sequences, each with a length of 366 days. To evaluate strategy stability, we use RMB-IIF optimized hyperparameters (Table 2) to train five GRUs separately for each strategy. The performance is then evaluated based on the mean RMSE of the five GRUs for each strategy.

Table 2: The RMB-IIF optimized hyperparameters.

Model	loss	H	η	optimizer	B	E	dropout
GRU	mse	32	0.01	Adam	64	500	0
LSTM	mse	256	0.001	Adam	64	200	0.4

mse: mean square error **H**: hidden state size **B**: batch size
η: learning rate **E**: maximum epoch

Impact of temporal dependency. Table 1 and Figure 3 show that the algorithms perform differently on datasets with varying temporal dependencies. Among response-based methods, TF and SSPL accumulate er-

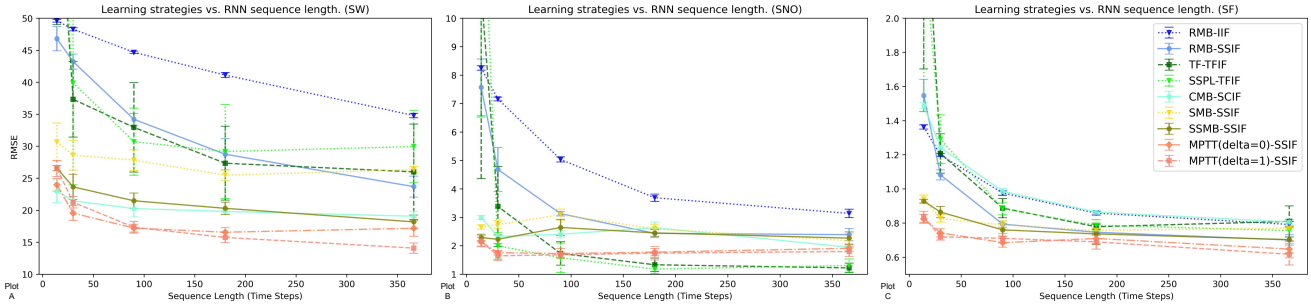

Figure 4: This figure displays experiment results where the training data size is fixed at 160 years. Models are trained with sequences having varying lengths (14, 30, 90, 180, 366 steps). Results indicate that all learning strategies perform better on longer sequences that retain more temporal dependencies.

rors in SW inference as they rely on the previous timestep response as input and only learn local changes between timesteps. This makes them unable to correct errors from earlier timesteps. They perform nicely on SNO since complete snowmelt by summer's end naturally prevents error accumulation. CMB-SCIF performs well on SW and SNO, as their initial responses provide a solid foundation for accumulating changes over time. The three response-based approaches perform similarly to basic RMB-IIF on SF, as its short temporal dependency means responses carry limited information. MPTT algorithms perform nicely in modeling both SW and SF and reasonably well with SNO, largely due to their efficient utilization of hidden states for message generation. This avoids error accumulation issues for variables with long-term dependencies and improves predictions for variables with short-term dependencies, where hidden states carry more information than responses.

Impact of training sequence length. Figure 4 suggests that all learning strategies yield improved performance with longer sequences, which capture more temporal dependencies. It further offers these insights: (a) As temporal dependencies decrease (from SW to SF), the performance gap among algorithms narrows, underscoring our proposed strategies' effectiveness in utilizing temporal dependencies when present. (b) Despite using the same RMB-trained models, RMB-SSIF consistently outperforms RMB-IIF, highlighting that even models trained using IID assumption can benefit from message passing between samples during inference. (c) The three response-based approaches exhibit distinct performances. CMB-SCIF performs well on SW and SNO. However, TF-TFIF and SSPL-TFIF perform poorly on SW, regardless of sequence length, and only perform well on SNO for longer sequences. All three strategies show no difference from RMB-IIF on SF predictions because of weak temporal auto-correlation in response values for streamflow. (d) In contrast, state-based approaches, SMB-SSIF and SSMB-SSIF, perform well on varying sequence lengths, as they use hidden states during both training and inference phases, reducing the loss of temporal dependencies due to shorter sequences. (e) MPTT algorithms consistently perform the best across all three datasets, regardless of sequence lengths, as they use high-quality hidden states and train models on heterogeneous mini-batches compared to state-based approaches.

Time efficiency. Table 3 presents the per-epoch training time for various algorithms on the sequence length of 366 on an NVIDIA A100 GPU. Results show MPTT uses around 0.06 seconds/epoch, considerably faster than SSMB and preserving full sample and temporal dependencies. MPTT lags behind RMB by 0.04 seconds per epoch, a delay partly attributed to their Python dictionary-based memory modules, indicating room for optimization.

Table 3: Per-epoch training time for various algorithms.

	RMB	TF	SSPL	CMB	SMB	SSMB	MPTT
sec/epoch	0.0172	0.0217	0.02	0.0151	0.0211	0.539	0.0614

4.2 Real world data experiments. CARAVAN, a comprehensive global hydrology dataset, integrates data from seven large-scale hydrology studies [23]. Long Short-Term Memory (LSTM) models has outperformed traditional models in rainfall-runoff predictions [24, 25]. In this paper, we use nine dynamic meteorological forcings and 20 static basin characteristics to train LSTMs for predicting streamflow across 191 Great Britain (GB) basins in the CARAVAN dataset. Our results affirm the efficacy of our algorithms in modeling real-world complex systems.

Experimental Settings. We partition the time series into training (1989/10/01-1997/09/30), validation (1997/10/01-1999/09/30), and testing (1999/10/01-

Table 4: The table displays the experiment results where each learning strategy is employed to train five separate LSTMs, utilizing hyperparameters outlined in Table 2. The final prediction for each basin in the CARAVAN test set is then obtained by averaging the predictions from these five LSTMs to evaluate the learning strategies.

	RMB-IIF	RMB-SSIF	TF-TFIF	SSPL-TFIF	CMB-SCIF	SMB-SSIF	MPTT($\delta = 0$)-SSIF	MPTT($\delta = 1$)-SSIF
RMSE	1.301 ± 0.817	1.285 ± 0.815	1.465 ± 1.000	1.273 ± 0.812	1.284 ± 0.822	1.368 ± 0.851	$\mathbf{1.255 \pm 0.814}$	$\mathbf{1.255 \pm 0.810}^*$
R^2	0.674 ± 0.120	0.686 ± 0.112	0.482 ± 0.531	0.692 ± 0.113	0.692 ± 0.102	0.625 ± 0.153	$\mathbf{0.705 \pm 0.117}^*$	$\mathbf{0.694 \pm 0.166}$

2009/09/30) periods. Data is normalized using the training set's mean and variance and then segmented into 365-day sequences with 183-day overlaps. Given the substantial variation in streamflow scales across basins, we use the Coefficient of Determination (R^2) in figures to evaluate predictions.

Top strategies for river basin streamflow prediction. Table 4 reveals that MPTT algorithms yield the best overall performance across all basins. Figure 5 Plot A specifically identifies the leading strategy for each river basin, leading to these observations: (a) RMB-IIF and RMB-SSIF only excel in two basins, while SMB-SSIF performs well in four. Their subpar performance underscores the importance of preserving sequence and long-term temporal dependencies and the need to avoid fixed mini-batches in training real-world applications. (b) Response-based algorithms run relatively well, with TF-TFIF and SSPL-TFIF achieving the best predictions in 49 basins. (c) MPTT algorithms excel in 113 out of 191 basins. Within MPTT algorithms, MPTTs(delta=1) yield slightly more best-predicted basins than MPTTs(delta=0).

A closer look at algorithm performance. Figure 5, Plot B, displays the ECDF curves for 191 basins and offers the following observations: (a) Response-based approaches exhibit varied results; TF-TFIF has 22 best-predicted basins, but its 40% basins have an R^2 lower than 0.6, while SSPL-TFIF and CMB-SCIF have only about 20% with R^2 lower than 0.6. This suggests that TF-based algorithms have a large variance in their performance across 191 basins. (b) SSPL-TFIF and CMB-SCIF display ECDF curves similar to RMB-IIF and RMB-SSIF, indicating no significant improvement in predictions. (c) MPTT-based algorithms outperform others, with almost 15% of basins having an R^2 above 0.8, compared to approximately 10% for other algorithms.

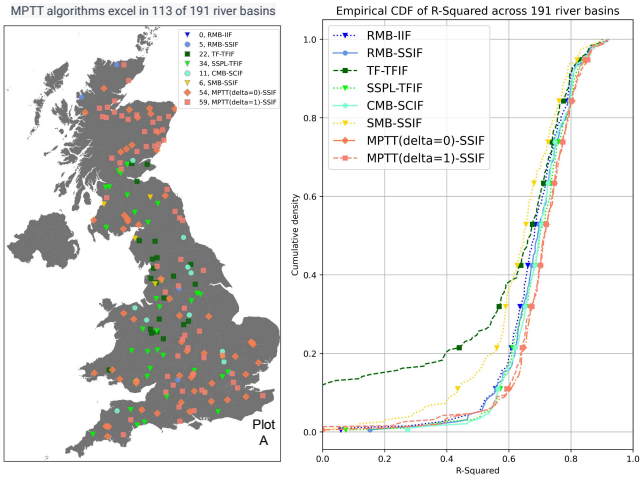

Figure 5: Plot A identifies the top-performing strategy for each basin. The results reveal that MPTT algorithms are the leading choice in 113 out of 191 basins. Plot B illustrates the empirical Cumulative Distribution Function (ECDF) of R^2 values. A curve leaning towards one on this graph suggests the model's effectiveness in explaining data variance across 191 basins. The results indicate that MPTT-based algorithms outperform their counterparts: only around 10% of basins show an R^2 value below 0.6, compared to approximately 20% for other algorithms.

5 Limitations and Future Directions

The MPTT algorithm has some limitations. First, MPTT is designed for state-based models; adapting it to transformers may require incorporating state information into the transformer architecture. While Transformers are known for their ability to capture long-range dependencies, they still consider each sequence sample independent. MPTT may help Transformers capture relations between sequences. Second, while the fixed $\delta = 1$ slightly outperforms $\delta = 0$ in experiments, the optimal choice for MPTT message keeper δ is still uncertain, leaving room for future exploration. Third, MPTT uses two memory modules, the key-map and state-map, which result in an additional asymptotic space complexity of $\Theta(M(H + \frac{W}{\Delta}))$, where M is the total number of sequences in the training set,H is hidden state size, W is the sliding window size, and Δ is the stride size. Fourth, MPTT is slower than RMB due to its frequent interactions with the memory modules. Despite potential optimizations, the extra computational cost is inherent to MPTT's design. Finally, this study highlights MPTT's strength in RNN training within environmental contexts. Future research could extend its applicability to diverse state-based models like Graph Neural Networks, aiming to improve optimization and information quality. Additionally, exploring MPTT's utility in

sectors like healthcare, finance, and language processing offers promising avenues for expansion.

6 Conclusion

This paper introduces novel MPTT algorithms designed to effectively capture sample and temporal dependencies when training sequential models for time series. MPTTs design two memory modules and three policies to generate informative initial hidden states as messages and facilitate seamless message communication between RNN sequences while allowing sample shuffling and diverse mini-batches across epochs during training. Experimental results show MPTTs consistently outperforming various competing algorithms on four climate datasets with multi-scale temporal dependencies, achieving the best predictions for 113 of 191 Great Britain river basins, demonstrating their effectiveness in modeling complex real-world dynamical systems.

7 Acknowledgement

This work was funded by the National Science Foundation through the Harnessing the Data Revolution award (1934721) and the U.S. Department of Energy's Advanced Research Projects Agency–Energy (ARPA-E) award (DE-AR0001382). We sincerely appreciate Zhenong Jin, Licheng Liu, Xiang Li, Xiaowei Jia, Rahul Ghosh, Haoyu Yang, and Michael Steinbach for their insightful feedback. We are also thankful for the access to advanced computing resources provided by the Minnesota Supercomputing Institute.

References

[1] Jared Willard et al. Integrating scientific knowledge with machine learning for engineering and environmental systems. *ACM Computing Surveys*, 2022.

[2] Christian Legaard et al. Constructing neural network based models for simulating dynamical systems. *ACM Computing Surveys*, 2023.

[3] Bryan Lim et al. Time-series forecasting with deep learning: a survey. *Philos. Trans. R. Soc. A*, 2021.

[4] Razvan Pascanu et al. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[5] Qingsong Wen et al. Transformers in time series: A survey. *arXiv:2202.07125*, 2022.

[6] Yoshua Bengio et al. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, 1993.

[7] Shaoming Xu et al. Mini-batch learning strategies for modeling long term temporal dependencies: A study in environmental applications. In *SIAM International Conference on Data Mining*, 2023.

[8] Antonio Gulli et al. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[9] M Akin Yilmaz et al. Effect of architectures and training methods on the performance of learned video frame prediction. In *ICIP*, 2019.

[10] Steven Elsworth et al. Time series forecasting using lstm networks: A symbolic approach. *arXiv:2003.05672*, 2020.

[11] Alexander Katrompas et al. Enhancing lstm models with self-attention and stateful training. In *SAI Intelligent Systems Conference*, 2021.

[12] Léon Bottou et al. The tradeoffs of large scale learning. *NeurIPS*, 2007.

[13] Ronald J Williams et al. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1989.

[14] Samy Bengio et al. Scheduled sampling for sequence prediction with recurrent neural networks. *NeurIPS*, 2015.

[15] Henry Abarbanel. *Predicting the future: completing models of observed complex systems*. Springer, 2013.

[16] Henry DI Abarbanel et al. Machine learning: Deepest learning as statistical data assimilation problems. *Neural computation*, 2018.

[17] Henning U Voss et al. Nonlinear dynamical system identification from uncertain and indirect measurements. *International Journal of Bifurcation and Chaos*, 2004.

[18] Jonas Mikhaeil et al. On the difficulty of learning chaotic dynamics with rnns. *NeurIPS*, 2022.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[20] Jeffrey G Arnold et al. Swat: Model use, calibration, and validation. *Journal of the ASABE*, 2012.

[21] Katrin Bieger et al. Introduction to swat+, a completely restructured version of the soil and water assessment tool. *JAWRA*, 2017.

[22] Kyunghyun Cho et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

[23] Frederik Kratzert et al. Caravan-a global community dataset for large-sample hydrology. *Scientific Data*, 2023.

[24] Frederik Kratzert et al. Towards learning universal, regional, and local hydrological behaviors via machine learning applied to large-sample datasets. *Hydrology and Earth System Sciences*, 2019.

[25] Martin Gauch et al. Rainfall–runoff prediction at multiple timescales with a single long short-term memory network. *Hydrology and Earth System Sciences*, 2021.