

RITA: Group Attention is All You Need for Timeseries Analytics

Jiaming Liang
University of Pennsylvania
Philadelphia, PA, USA
liangjm@seas.upenn.edu

Lei Cao*
Massachusetts Institute of Technology
Cambridge, MA, USA
lcao@csail.mit.edu

Samuel Madden
Massachusetts Institute of Technology
Cambridge, MA, USA
madden@csail.mit.edu

Zachary Ives
University of Pennsylvania
Philadelphia, PA, USA
zives@cis.upenn.edu

Guoliang Li
Tsinghua University
Beijing, China
liguoliang@tsinghua.edu.cn

ABSTRACT

Timeseries analytics is of great importance in many real-world applications. Recently, the Transformer model, popular in natural language processing, has been leveraged to learn high quality feature embeddings from timeseries, core to the performance of various timeseries analytics tasks. However, the quadratic time and space complexities limit Transformers’ scalability, especially for long timeseries. To address these issues, we develop a timeseries analytics tool, RITA, which uses a novel *attention* mechanism, named *group attention*, to address this scalability issue. Group attention dynamically clusters the objects based on their similarity into a small number of groups and approximately computes the attention at the coarse group granularity. It thus significantly reduces the time and space complexity, yet provides a theoretical guarantee on the quality of the computed attention. The dynamic scheduler of RITA continuously adapts the number of groups and the batch size in the training process, ensuring group attention always uses the fewest groups needed to meet the approximation quality requirement. Extensive experiments on various timeseries datasets and analytics tasks demonstrate that RITA outperforms the state-of-the-art in accuracy and is significantly faster — with speedups of up to 63X.

1 INTRODUCTION

Motivation. Many data driven applications involve processing massive timeseries data, including IoT [11], medical AI [14], stock market [27], and so on. As such, there is a great need for timeseries analytics, such as forecasting [8], classification [20], clustering [31], similarity search [39], and anomaly detection [50], with applications ranging from automatically diagnosing diseases [5], recognizing human activities [29], to stopping financial fraud [59].

Effective feature extraction [40] lies at the core of almost all these timeseries analytics tasks. Recently researchers [61] have started leveraging the *self-supervised pre-training* methodology of Transformers [4, 16, 52], which have proven remarkably successful in natural language processing (NLP), to automatically learn high quality feature embeddings from timeseries. In NLP, self-supervised pre-training exploits the sequential patterns (correlations) among the words in sentences to produce *contextualized* feature embeddings. Timeseries bear similarity to natural language, because in timeseries data the sequential order among the values (stock price, volume, etc.) over time matters. That is, each value is highly correlated with other values observed before or after it. Therefore,

pre-training a Transformer model which takes the correlations among different observations into account is a natural idea to learn feature embeddings from timeseries. Indeed, the experiments in [61] confirm that Transformer-based methods outperform traditional timeseries analytics techniques.

However, existing work [61] that directly applies Transformers to learn features from timeseries data have been shown not to be scalable to *long* timeseries [30]. The idea of self-attention [52] is central to pre-training methods in NLP: It computes pairwise correlations among different semantic units in a sequence (in NLP, a sentence); as such, it has *quadratic time and space* complexity in the length of the input sequence. Such an approach places limits on the model’s scalability, especially when handling large sequences, which are common in real-world timeseries applications such as IoT, medical AI, and finance [6, 34, 62]. Predictions about timeseries may need to look at months or years of historical data to make accurate predictions, spanning hundreds of thousands of samples. As an example, in collaboration with a research hospital we have been developing a seizure classifier that automatically detects seizures based on EEG signals (timeseries) collected during the clinical observation of patients. As seizures last only a few seconds, we chunk long EEG data into many 2 second segments and detect seizures at a segment level. However, the classification of a particular segment depends on up to 12 hours of prior signal to determine if one 2 second segment indicates seizure or not, because seizure diagnosis needs to consider long-term trends in the EEG data [6]. The number of segments in 12 hours is more than 21k. This is far larger than the number of semantic units the typical NLP tasks expect. For example, BERT [16] limits the number of units to 512 and even massive models like GPT-3 [4] limit the number of units to 2048.

Although in NLP some lower-complexity methods have been proposed to *approximately* compute self-attention [10, 26, 54], their performance degrades dramatically when used on timeseries, due to the gap between natural language and timeseries, as we will show in our experiments.

Proposed Approach. To tackle the aforementioned problem, we develop RITA, a Transformer-based timeseries analytics tool, which uses a novel attention mechanism, called **group attention**, to scale to long timeseries.

Leveraging the periodicity of timeseries, RITA chunks the input timeseries into segments and dynamically clusters the segments into a small number (denoted as N) of groups. Segments in the same group possess similar feature embeddings during the current training iteration, thus enabling them to approximately share the

*Corresponding Author

computation of attention. As the timeseries increases in length, more sharing opportunities become available. RITA then computes the self-attention at a group level and produces a *compressed group attention matrix*. In this way, group attention eliminates both computation and memory bottlenecks in Transformer-style models and thus more scalable to long timeseries.

However, making this idea effective and efficient in Transformer architectures is *challenging* for several reasons:

- **Efficiently Producing High Quality Feature Embeddings.** Although RITA computes the attention matrix at a group level, to preserve the quality of the feature embeddings, it still has to produce different embeddings for different segments. This is because even if some segments share the attention score temporally, it does not mean they should have the same feature embedding. However, using the group attention matrix, the existing self-attention mechanism will only produce a single feature vector for each group. A naive solution would be to restore the original attention matrix from the group attention matrix. However, in this case we again get an attention matrix with quadratic space complexity. Because GPUs have limited memory, GPU memory will remain a *bottleneck* in group attention.

- **The Number of Groups N .** In RITA, the number of groups N is a crucial factor that balances the speed up and the quality of attention approximation. A small N will lead to a large speedup, but the approximation errors can also be significant. On the other hand, although a large N tends to produce high-quality approximations, it inevitably slows down the training process. Therefore, an appropriate N is essential to the performance of group attention. However, N depends on the distributional properties of the dataset. Furthermore, like the classical transformer models, RITA stacks multiple attention layers to produce better embeddings. Ideally, different layers should also use different values of N . In addition, during the model training phrase, group attention should use different values of N at different iterations to adapt to the varying feature embeddings. This makes manually setting appropriate N almost impossible.

- **Batch Size.** Moreover, as we want to dynamically adjust N during training, a fixed batch size is sub-optimal: as N decreases, the memory usage of a single sample decreases. This allows a larger batch size which is beneficial, because: (1) it makes full use of GPU memory; (2) high-parallelism across the samples in a big batch brings better performance. Our experimental study shows that doubling the batch size reduces the training time by 30%, while still preserving the quality of the model. Thus, RITA should dynamically adjust batch size as N changes.

To address the above problems, we first propose an *embedding aggregation* strategy and a customized *group softmax function* to replace the classical softmax function [52]. Together they ensure RITA is able to directly use the compressed attention matrix to produce different feature embeddings for different segments. We theoretically show the embeddings RITA produces in this way are identical to those produced by first re-storing the original large attention matrix. Thus RITA is able to produce high quality embeddings without introducing extra overhead. Further, we design a GPU friendly algorithm to group the segments *in parallel*, effectively minimizing the grouping cost.

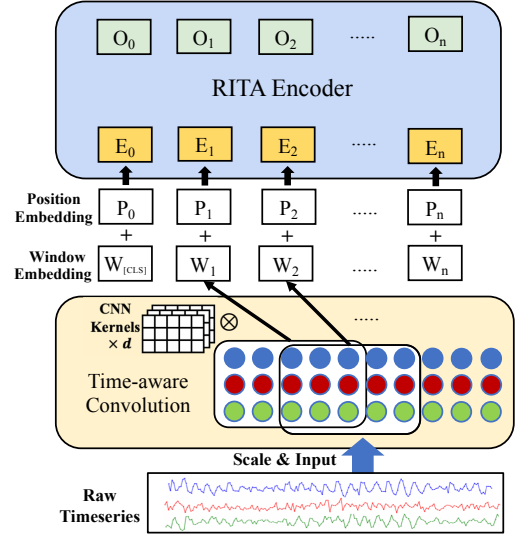


Figure 1: RITA Architecture

Second, we design an *adaptive scheduler* which dynamically decides an appropriate N for each group attention layer during the training process. It starts with a large N and iteratively merges groups that are similar to each other. Guided by an error bound on the approximated self-attention that users can tolerate, it automatically determines if two groups are mergeable, performing merging efficiently in a GPU-friendly way.

Moreover, we propose a *learning-based method* to model the correlation between the number of groups N and the batch size B . This model is used to predict B for a given N when training RITA. Specifically, we first sample some N values in a reasonable range. For each sampled N , we find a batch size that consumes up to a certain percentage of GPU memory in a cost-efficient way. Using a small set of mathematical functions as a prior, RITA learns a model with only a few $\langle N, B \rangle$ pairs as ground truth labels.

Our experiments on public timeseries benchmarks and the MGH EEG data [6] confirm that RITA outperforms state-of-the-art methods in accuracy on various timeseries analytics tasks, while our group attention mechanism achieves a 63X speedup with much less memory required, compared to existing self-attention mechanisms [10, 52, 54].

Contributions. The key contributions of this work include:

- Our group attention mechanism leverages the periodicity of timeseries, reducing the time and space complexity of the self-attention mechanism with accuracy guarantees, allowing RITA to scale to long timeseries data.
- Guided by an approximation error bound, our adaptive scheduler dynamically adapts the number of groups and the batch size to the distribution properties of the evolving feature embeddings, making group attention efficient and easily tunable.
- We conduct experiments on various datasets and different analytics tasks, demonstrating that RITA is 4 to 63 times faster than the state-of-the-art while achieving better accuracy when handling long timeseries (length ≥ 2000).

2 BACKGROUND

We provide some background on the canonical self-attention module in the Transformer[52]. A *self-attention* module takes n hidden embedding vectors $H \in \mathbb{R}^{n \times d_h}$ as input, then projects them to queries (Q), keys (K) and values (V) and performs Scaled-dot Product Attention, which given input hidden state H , is computed by:

$$\begin{aligned} Q &= HW_Q, K = HW_K, V = HW_V \\ O &= AV = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \end{aligned} \quad (1)$$

Where $W_Q \in \mathbb{R}^{d_h \times d_k}$, $W_K \in \mathbb{R}^{d_h \times d_k}$, $W_V \in \mathbb{R}^{d_h \times d_v}$ are projection matrices for generating Q, K, V . $Q \in \mathbb{R}^{n \times d_k}$ is also regarded as the packing of n query vectors $\{q_1, \dots, q_n\}$ with dimension d_k into a matrix. $K \in \mathbb{R}^{n \times d_k}$, $V \in \mathbb{R}^{n \times d_v}$ are regarded as the packing of key vectors $\{k_1, \dots, k_n\}$ and value vectors $\{v_1, \dots, v_n\}$ in the same way.

Given a matrix $M \in \mathbb{R}^{L \times n}$, the softmax function normalizes M to ensure the sum of each row equals to 1, as shown below.

$$\text{SoftMax}(M_{i,j}) = \frac{\exp(M_{i,j})}{\sum_{k=0}^{n-1} \exp(M_{i,k})} \quad (2)$$

Note the attention matrix A is an $n \times n$ matrix, where n represents the number of elements in the input sequence (e.g. words in NLP).

3 RITA OVERVIEW

Given a collection of *unlabeled* timeseries, RITA first pre-trains a Transformer-style model to produce high quality feature embeddings for timeseries data. This pre-trained model is then used to support various downstream tasks, similar to BERT [16]. Next, we overview the model architecture of RITA. We show how RITA supports various downstream tasks in Appendix A.7.

As shown in Fig. 1, RITA is consist of two components: (1) Time-aware Convolution Layer (2) RITA Encoder.

Time-aware Convolution Layer fills the gap between timeseries and natural language. Despite their high-level similarity, there is a big gap between timeseries and natural language. First, in natural language each word, as a discrete semantic unit, has an independent meaning, while each element in a timeseries is a continuous, numerical value and does not necessarily constitute an independent event. Furthermore, the input sequences are single-channeled in NLP, but often multi-channeled in timeseries (i.e., sensor data often consists of several related channels).

RITA leverages the classical convolution [28] strategy to solve this problem. Convolution is widely used to capture the local structures of an image. We use convolution to chunk one input timeseries into a sequence of windows and learn the local structure of each window, similar to the discrete semantic units in natural language. It also discovers the correlations across different channels, thus naturally solving the multi-channel problem.

More specifically, treating a multi-variate timeseries of length n and with m variables as an $n \times m$ matrix T , RITA uses d convolution kernels to chunk T into n windows and produce one d -dimensional embedding per window using the convolution operation [28]. Each convolution kernel corresponds to a $w \times m$ matrix, where w defines the number of timestamps that each convolution kernel covers, identical to the window size in sliding window.

RITA Encoder functions as Transformer Encoder as described in the original Transformer work[52]. It takes the embeddings of n

semantic units $X_1, X_2, \dots, X_n (X_i \in \mathbb{R}^d)$ as input (e.g. embeddings of n windows for a timeseries), then models the correlations between the semantic units and outputs $Y_1, \dots, Y_n (Y_i \in \mathbb{R}^d)$ as the context-aware embedding of each unit.

What makes RITA Encoder different from Transformer Encoder is that: at the core of Transformer Encoder lies self-attention mechanism which incurs a $O(n^2)$ time complexity and memory usage. This quadratic cost becomes prohibitive for long timeseries and limits the scalability of Transformer-based models. To make the attention computation efficient yet high-quality, we replace the canonical self-attention with our proposed **group attention**.

Self-supervised Pretraining. Inspired by the ‘‘cloze text’’ pre-training task in NLP, we designed a mask-and-predict task as the pretraining task for our model. The timeseries is randomly masked and the model should recover the masked values based on corresponding contextual information.

To be specific, we generate masks on time-stamps, with a mask rate p . The timeseries is scaled to be non-negative and the values across all the channels on the masked timestamps are set to be -1, an impossible value on normal timestamps. Then the masked timeseries is fed into RITA and the output representation is translated to the recovered timeseries by a Transpose Convolution layer.

4 GROUP ATTENTION MECHANISM

Group attention, a novel and efficient approximate attention mechanism, addresses the performance bottleneck of self-attention in the vanilla Transformer. In this section, we first introduce the framework of group attention and then theoretically establish the bound of its approximation error.

4.1 The Idea of Group Attention

As periodicity is a natural property of timeseries [56], similar windows frequently occur. Similar windows result in similar queries/keys for attention computation, bringing opportunities for saving computation.

As discussed in Sec. 2, A_{ij} , the attention score of window i onto window j , is determined by the inner product between the query vector of window i and the key vector of window j , that is, $q_i \cdot k_j$. Given another window x , if window x has the similar key vector to window j , that is, $k_j \approx k_x$, then $q_i \cdot k_j \approx q_i \cdot k_x$. In other words, $A_{ij} \approx A_{ix}$ when $k_j \approx k_x$.

This observation inspires our group attention mechanism. That is, we group the windows by their similarity in keys. Assuming all windows in the same group have the same attention score onto another window k , we then only compute the attention once by using *one single key* to represent this group, for example the centroid of the group of keys. This thus saves significant computation cost.

Better yet, after grouping n windows into N groups, group attention compresses the attention matrix from an $n \times n$ matrix to an $n \times N$ matrix. Because N (number of groups) tends to be much smaller than n (number of windows) due to the periodicity of timeseries, group attention consumes much less memory than the original self-attention mechanism, successfully eliminating the memory bottleneck. Note that it also doesn’t hurt quality all that much, as confirmed in our experiments (Sec. 6.2).

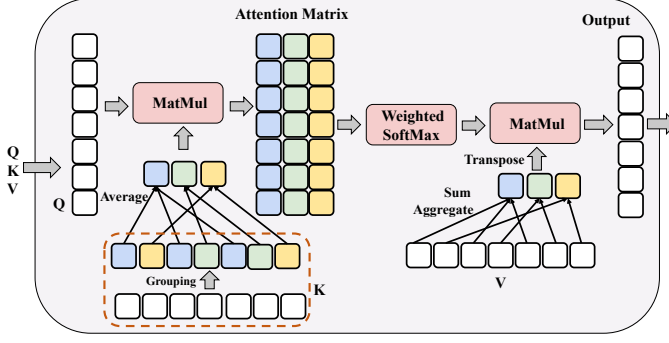


Figure 2: Group Attention

4.2 Computing the Output Feature Embedding

We now discuss how to efficiently compute the output feature embeddings using the small compressed group attention matrix.

4.2.1 Problem: Producing Embeddings w/ Group Attention Matrix As described in the Background, once we have acquired the attention matrix A , canonical self-attention computes the output embedding O as $O = AV$. Because A is an $n \times n$ matrix and V is an $n \times d_v$ matrix, the matrix product operation still produces an $n \times d_v$ matrix O . That is, it produces a d_v dimensional feature vector for each *window*. However, our group attention will produce an $n \times N$ attention matrix \tilde{A} , where N corresponds to the number of groups. In this case the matrix product will produce a $N \times d_v$ matrix \tilde{O} . That is, it produces a feature vector for each *group*. However, our goal is to produce different embeddings for different windows, because even if some windows share the attention score temporally, it does not mean they should have the same feature embedding.

A Naive Solution. A naive solution would be to restore the full attention matrix A from the group attention matrix \tilde{A} . For example, given one group composed of win_i and win_j , we map its group attention vector in \tilde{A} into two rows that correspond to win_i and win_j in A . However, in this case we again get a $n \times n$ attention matrix; and GPU memory remains a *bottleneck* in group attention.

4.2.2 Solution: Embedding Aggregation and Group SoftMax

Using an *embedding aggregation* operation and a *group softmax* function, RITA produces n embeddings without restoring the full attention matrix. Fig. 2 shows the workflow of group attention.

Embedding Aggregation. The idea is inspired by the observation on the matrix product operation $O = AV$ conducted on the fully restored attention matrix A .

Given an element $O_{i,j}$ of O corresponding to the j^{th} dimension of win_i 's feature vector, $O_{i,j} = a_i \cdot v_j$, where vector $a_i \in \mathbb{R}^n$ denotes the i^{th} row of the attention matrix A and vector $v_j \in \mathbb{R}^n$ denotes the j^{th} dimension of all the n feature vectors. Given $a_i = \langle a_i^1, a_i^2, \dots, a_i^n \rangle$ and $v_j = \langle v_j^1, v_j^2, \dots, v_j^n \rangle$, $O_{i,j} = \sum_{k=1}^n a_i^k v_j^k$.

As an example, assume win_1 and win_2 belong to the same group G_1 . Then $a_1^1 = a_2^1 = \tilde{a}_1^1$, where $\tilde{a}_1^1 \in \tilde{A}$ corresponds to the attention of group G_1 onto win_1 . Therefore, $a_1^1 v_j^1 + a_2^1 v_j^1 = \tilde{a}_1^1 (v_j^1 + v_j^2)$.

As an immediate generalization of the above analysis, if we aggregate up the windows that belong to the same group and convert the n -dimensional feature vector v_j into a N -dimensional group feature vector \tilde{v}_j beforehand, we could directly use the group attention vector \tilde{a}_i and the group feature vector \tilde{v}_j to compute $O_{i,j}$.

Using embedding aggregation, RITA is able to produce the feature embedding \tilde{O} that is identical to the embedding O produced by using the full attention matrix A and the embedding matrix V .

Group Softmax Function. In canonical self-attention the attention matrix A is computed as $A = \text{SoftMax}(\frac{QK^T}{\sqrt{d_k}})$. To compute A , we have to first compute QK^T (denoted as P) which is an $n \times n$ matrix. Then normalizing the P matrix with softmax produces the attention matrix A .

Group attention follows the same procedure. But after grouping keys into \tilde{K} , $Q\tilde{K}^T$ produces an $n \times N$ matrix \tilde{P} . Due to the non-linearity of the softmax function, applying softmax directly on \tilde{P} will result in a group attention matrix \tilde{A} from which we are not able to recover a full attention matrix that is identical to first restoring \tilde{P} to P and then applying softmax on P . The A matrix produced by the latter is desirable, as we want to approximate the original attention matrix as accurately as possible. However, restoring the small $n \times N$ \tilde{P} matrix is not memory efficient, as it will end up with a full $n \times n$ matrix P .

To solve the above problems, we introduce a new **group softmax** function to replace the original softmax function (Eq. 2).

$$\text{GroupSoftMax}(\tilde{P}_{i,j}) = \frac{\exp(P_{i,j})}{\sum_{k=0}^{N-1} \text{count}_k \exp(P_{i,k})} \quad (3)$$

In Eq. 3, count_k represents the number of windows that Group G_k contains. Compared to the original softmax, our group softmax considers each group G_k as count_k elements and counts it count_k times when summing up the exponential of each group's $P_{i,k}$. In this way, the group softmax function operating on the small \tilde{P} matrix will produce *exactly the same* result to the softmax function operating on the full P matrix.

Theoretical Guarantee. In Appendix A.4, we prove that the group softmax function and the embedding aggregation operation produce the same output feature embedding with the naive method that has to first restore the big full attention matrix.

We show an efficient implementation of the embedding aggregation operation and group softmax function in Appendix A.2, Alg. 1. **Time Complexity.** The time complexity of Alg. 1 is $O(nNd)$ and the space complexity is $O(nN)$, while the time and space complexity of the original self-attention mechanism are $O(n^2d)$ and $O(n^2)$.

4.3 Error Bound

Group attention produces a group attention matrix \tilde{A} which approximates the attention matrix A produced by the classical self-attention with a *bounded error*, as shown in Lemma 1.

LEMMA 1. Let R be the radius of the ball where all key vectors live; \tilde{k}_i be the representative of the group that contains key k_i . Let \tilde{A} denote the full attention matrix restored from \tilde{A} . Suppose the distance between \tilde{k}_i and k_i ($\|\tilde{k}_i - k_i\|$) satisfies: $\|\tilde{k}_i - k_i\| \leq d$.

Then $\forall \epsilon > 1$, if $d \leq \frac{\ln(\epsilon)}{2R}$, $\frac{1}{\epsilon} \leq \frac{\tilde{A}_{ij}}{A_{ij}} \leq \epsilon$

Lemma 1 shows that the error bound ϵ of the group attention is determined by the distance d . As discussed in Sec. 5.1, it inspires us to design a strategy to dynamically determine the number of groups N – the most critical parameter of group attention. Please refer to Appendix A.5 for the proof.

4.4 GPU Friendly Grouping Method

In this section, we discuss the implementation of a grouping method. To make group attention efficient and effective, the grouping method has to satisfy the following requirements:

(1) Tight distance bound: to ensure the approximation quality, the distance between each key and its group representative should be minimized according to Lemma 1.

(2) Lightweight: to ensure the performance gain, the grouping method must be lightweight, at worst not exceeding the complexity of group attention itself ($O(Nn)$).

(3) GPU friendly: to take advantage of GPUs, we prefer a grouping method that mainly consists of matrix operations, which can be efficiently executed on a GPU.

To satisfy the above requirements, after thorough investigation on various clustering algorithms, we design a GPU friendly K-means [35] as the grouping method.

First, K-means minimizes the overall distance between any object and its cluster center, hence naturally satisfying Requirement 1.

Second, given N centers, in each iteration the time and space complexity of K-means is $O(nN)$. Usually, the iteration goes until convergence. However, we observe that rather than seeking a perfect K-means clustering, training a few iterations is sufficient to get a good grouping for group attention, because typically the later iterations only slightly update the clustering and group attention is robust to such imperfection.

Third, we design a GPU-friendly implementation of K-means. The performance bottleneck of K-means comes from the distance computation between each vector and its center, that is, $|v_i - c_j| = \sqrt{(v_i - c_j)^2}$, $i \in [1, n], j \in [1, N]$. The performance bottleneck is $v_i - c_j$. We instead use a different formulation: $|v_i - c_j| = |v_i - c_j| = \sqrt{|v_i|^2 + |c_j|^2 - 2v_i \cdot c_j}$, $i \in [1, n], j \in [1, N]$. This is because in this formulation, the performance bottleneck is $v_i \cdot c_j$, which could be implemented as a matrix product operation. Although the complexity of the two formulations is the same, in GPUs matrix product is much more efficient than pairwise difference.

5 ADAPTIVE SCHEDULER

Next, we present the adaptive scheduler of RITA which addresses the challenges of determining an appropriate number of groups N and accordingly the batch size B , as described in Introduction. Using a dynamic scheduling method we propose, the scheduler automatically determines and adjusts N and B based on the distributional properties of the feature embeddings produced over the iterative training process, while guaranteed to produce high quality attention approximation that meets the requirement of users.

In Sec. 5.1 we show how RITA automatically determines N . Then we introduce in Sec. 5.2 the learning-based method which given an N , immediately predicts a good batch size.

5.1 Dynamically Determining the Number of Groups N

Without loss of generality, we use one group attention module as an example to show how RITA automatically gets an appropriate N . The adaptive scheduler of RITA starts with a large N and decreases it dynamically. This is because in the training process of RITA, the

feature embeddings produced epoch by epoch tend to get stabler and stabler and gradually converge, thus no need to increase N .

RITA reduces the number of groups by merging similar groups. Intuitively, given two groups, we could measure their similarity based on the distance of their centers. If the distance between their centers is smaller than a distance threshold, then the two groups could be merged. However, setting an appropriate distance threshold seems hard – as difficult as setting an appropriate N .

To solve this problem, RITA leverages the error bound of group attention introduced in Sec. 4.3. It only requires users to set an error bound ϵ , and then uses Lemma 1 to translate ϵ to a distance threshold d . RITA then uses Lemma 2 to determine if merging some given clusters still meets the error bound threshold ϵ .

LEMMA 2. Denote c_k to be the cluster center of $cluster_k$. Assume the existing grouping satisfies $\forall k, \max_{x \in cluster_k} |c_k - x| \leq d$, thus satisfying an error bound ϵ by Lemma 1. If there exist m clusters, namely, $cluster_{k_1}, cluster_{k_2}, \dots, cluster_{k_m}$, satisfying that:

$$\max_{x \in cluster_{k_i}} |c_{k_i} - c_{k_j}| + |x - c_{k_i}| \leq d, i, j \in [1, m] \quad (4)$$

merging them into one cluster still meets the error bound ϵ .

Please refer to Appendix A.6 for the proof.

Finding the Mergable Clusters. We formulate the problem of finding mergeable clusters using graph theory:

(1) each cluster is a node in the graph;

(2) if $cluster_i$ and $cluster_j$ satisfy:

$$\max_{x \in cluster_i} |c_i - c_j| + |x - c_i| \leq d, \text{ and } \max_{x \in cluster_j} |c_j - c_i| + |x - c_j| \leq d$$

there is an undirected edge between $node_i$ and $node_j$;

In this scenario, finding the maximum number of *mergeable* clusters is equivalent to finding the minimal clique cover in the corresponding graph, which is an NP-hard problem [24]. Such heavy computation overhead is not acceptable for RITA. We thus offer a simplified solution:

(1) Halve the clusters into two sets S_1, S_2 ;

(2) If $cluster_i \in S_1$ and $cluster_j \in S_2$ satisfy:

$$\max_{x \in cluster_i} |c_i - c_j| + |x - c_i| \leq d, \max_{x \in cluster_j} |c_j - c_i| + |x - c_j| \leq \frac{d}{2} \quad (5)$$

$cluster_j$ is marked.

(3) Decrease the number of clusters by counting the masks in S_2 .

In this solution, clusters in S_1 can be regarded as transfer nodes. If (5) holds for $(cluster_i \in S_1, cluster_{j_1} \in S_2)$ and $(cluster_i \in S_1, cluster_{j_2} \in S_2)$, respectively, we have,

$$\begin{aligned} & \max_{x \in cluster_{j_1}} |c_{j_1} - c_{j_2}| + |x - c_{j_1}| \\ & \leq \max_{x \in cluster_{j_1}} |c_{j_1} - c_i| + |c_i - c_{j_2}| + |x - c_{j_1}| \\ & \leq \max_{x \in cluster_{j_1}} |c_{j_1} - c_i| + |c_i - c_{j_2}| + |x - c_{j_1}| + |x - c_{j_2}| \leq d \end{aligned} \quad (6)$$

Thus (4) holds when merging several clusters in S_2 with one cluster in S_1 . As a result, we can greedily merge clusters in S_2 , as illustrated in step(3).

Assume the number of clusters decreases by D after merging, we apply a momentum update [42] on the number of clusters N , as is commonly used in machine learning to smooth the changing of N and avoid sample selection bias. To be specific: $N_{new} = \alpha(N - D) + (1 - \alpha)N$, where α is a hyper-parameter for momentum.

5.2 Dynamically Determining the Batch Size

Because of the dynamic grouping operation, the computational graph in deep learning training [1] varies from sample to sample. As a result, it is impossible to precisely compute a batch’s GPU memory usage without indeed feeding it into the model. To overcome this problem, RITA learns a batch size prediction function offline; then at the RITA training time, given a number of groups N , RITA uses this function to predict a proper batch size.

When the model architecture and hardware are fixed, the batch size depends on the length of the timeseries L and the average group number among all attention module \bar{N} . So RITA samples several (L_i, \bar{N}_i) pairs and estimate a proper batch size for each pair.

More specifically, given a user-defined timeseries maximal length L_{max} , we randomly sample integral points (L_i, N_i) from plane $\{1 \leq L \leq L_{max}, 1 \leq N \leq L\}$. Then we use a binary search based algorithm to find the maximal batch size B_i that consumes less than 90% available GPU memory, aiming to avoid wasting GPU memory and the risks of out of memory (OOM).

Treating these pairs as ground truth labels, we use function fitting [18] to learn the batch size predicting function $B = f(L, N)$, where B is a function of two variables L and N .

Learning the Prediction Function. We apply *curve fit* from SciPy [53] as the function fitting tool to fit the two-variable function $B_i = f(L_i, N_i)$ on plane $\{1 \leq L \leq L_{max}, 1 \leq N \leq L\}$.

We observe that applying one function to the whole plane incurs a huge estimation error. So we develop a dynamic-programming (DP) method to divide the plane into several sub-planes and apply a distinct function to each sub-plane respectively. It is **optimal** in minimizing the total estimation error on all sub-planes

With the learned prediction function f , we can estimate a proper batch size for any (L, N) during training, even if it is not seen in the sampled (L_i, N_i) pairs.

The Algorithms and Optimality Proof. Please refer to Appendix A.3 for the pseudo code of the binary search-based algorithm and the description of the DP method for plane-division and the proof for its optimality.

6 EVALUATION

Our experimental study focuses on the following questions:

1. **Effectiveness and efficiency of RITA:** How does RITA compare with other Transformer-based methods and traditional timeseries representation learning methods in accuracy and efficiency?
2. **Ablation Study:** How do the key techniques of RITA work?

6.1 Experimental Setup

Datasets. We evaluate RITA on classification and imputation tasks using 5 multi-variate and 3 uni-variate timeseries datasets.

- **WISDM** [55] is a popular multivariate timeseries dataset generated from the accelerometer in the mobile phone. The subjects performed 18 daily activities (e.g. walking, jogging). The dataset was collected from 51 subjects and the sampling rate is 20 Hz.
- **HHAR** dataset [46] contains sensing data of accelerometer collected from 9 users performing 5 activities with 12 different smartphones (varying in sampling rate). This increases the complexity of the task and thus can test the model’s robustness.

- **RWHAR** *RealWorld HAR* dataset [48] covers 15 subjects performing 8 locomotion-style activities. Each subject wears the sensors for approximately ten minutes. The sampling rate is 50 Hz.

- **ECG** dataset [34] consists of 10,000 EEG recordings for arrhythmia classification. Each recording has an uncertain length ranging from 6 to 60 seconds sampled at 500 Hz. The ECG recordings correspond to 9 types of heart problems such as atrial fibrillation (AF) and premature atrial contraction (PAC), etc.

- **MGH** [6] is a EEG dataset collected by Mass. General Hospital. Each timeseries corresponds to the EEG data observed from one patient during their stay in ICU for a couple of days. The EEG monitoring produced data with 20 channels. The sampling rate is 200 HZ. So it produces very long timeseries.

- **WISDM*/HHAR*/RWHAR*** are three uni-variate datasets derived by picking one channel from WISDM/HHAR/RWHAR.

Training/Validation Data Generation. We apply a sliding window on the raw timeseries to get training/validation samples. The size of the sliding window is set as 200 on small datasets (WISDM, HHAR, RWHAR), 2000 on medium size dataset (ECG), and 10,000 on the large dataset (MGH). Table 1 shows the statistics of the generated datasets. They are randomly split into training/validation set in a proportion of 0.9/0.1. In “pretraining + few-label finetuning” scenario, we use 100 labeled data per class for finetuning. We guarantee that training set does not overlap with the validation set.

Dataset	Train. Size	Valid. Size	Length	Channel	Classes
WISDM	28,280	3,112	200	3	18
HHAR	20,484	2,296	200	3	5
RWHAR	27,253	3,059	200	3	8
ECG	31,091	3,551	2000	12	9
MGH	8,550	950	10000	21	N/A

Table 1: The statistics of the datasets

Alternative Methods. We compare RITA against the SOTA Transformer based timeseries representation learning method TST [61]. To evaluate our group attention (referred to as **Group Attn.**), we develop three baselines by replacing the group attention component in RITA with the classic vanilla Self-Attention [52](referred to as **Vanilla**) and two SOTA methods that reduce the complexity of self-attention by approximation in NLP, namely, Performer [10] (referred to as **Performer**) and Linformer [54] (referred to as **Linformer**). Similar to our proposed Group Attn., Vanilla, Performer, Linformer all use RITA’s time-aware convolution operation (Sec. 3) to turn timeseries segments into input feature vectors.

We also compare Group Attn. against **GRAIL** [40], which is the SOTA of the non-deep learning methods for timeseries representation learning. GRAIL supports classification tasks by feeding the learned representations into a Support-Vector Machine [12] or K-Nearest Neighbor [17] classifier. Note GRAIL only targets **uni-variate** timeseries and cannot support imputation tasks.

Methodology. We mainly focus on two downstream tasks:

(1) **Classification.** First, we train Group Attn. and the baselines with full labels from scratch to test the effectiveness of RITA framework and the approximation quality of our group attention.

Second, to measure the effectiveness of self-supervised pretraining, we evaluate the accuracy of training on few labeled timeseries with/without pretraining on large scales of unlabeled timeseries. To be specific, we split the training set into a *pretraining* set and a *fine-tuning* set, with very few data in the latter (100 labeled samples per

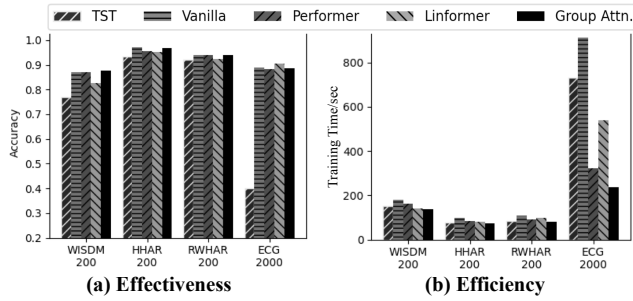


Figure 3: Full-label classification results (multi-variate data).

class in our experiment). We train the model on the cloze pretraining task with a mask rate $p = 0.2$. Then we train two classification models using the finetuning set, either based on the pretrained version or from scratch. We repeat the experiment 5 times with random data splits and report the median accuracy.

(2) **Imputation.** We run the imputation task on the datasets used in classification as well as the large unlabeled MGH dataset, and measure the mean square error and absolute imputation error. To get timeseries with missing values, we randomly mask the values with an expected mask rate of $p = 0.2$. The masked values are replaced with a special value.

Finally, to evaluate Group Attn.’s benefit on **efficiency**, the total time of forward computation, backward propagation, and grouping are measured for all methods in all the experiments.

To save space, we only report the average training time per epoch here and refer readers to Appendix A.8 for the inference time.

We first compare against the Transformer-based methods on multi-variate datasets (sec. 6.2, 6.3), then compare against the non-deep learning method GRAIL on uni-variate datasets (sec. 6.4).

Configuration. Please refer to Appendix A.1 for the experiment configuration and hyper-parameter settings.

6.2 Effectiveness: Transformer-Based Methods

We first evaluate the quality of the models trained with full labels from scratch. We then show how the pretraining of RITA increases the accuracy of the downstream tasks.

6.2.1 full-label training (Multi-variate classification)

Results shown in Figure 3(a) get us the following observations:

(1) **RITA’s advantage over TST.** On all four datasets for the classification tasks, Group Attn. and the other three baselines that use RITA architecture (Vanilla, Performer, and Linformer) outperform TST. In particular, Group Attn. outperforms TST by 49 percentage points on the ECG dataset (88.48% vs 39.93%) with long timeseries. Two deficiencies in TST may cause its poor performance on the long timeseries. Firstly, TST concatenates the output embedding vector of each time stamp, then uses a linear classifier to do classification on the concatenated vector. When the timeseries is long, the linear classifier has so many parameters that it tends to overfit easily. Secondly, TST replaces Layer Normalization in vanilla Transformer with Batch Normalization. When the timeseries is long, it can only accommodate a small number of timeseries in each batch, leading to bias in Batch Normalization.

(2) **Group-attention’s advantage over other attention mechanisms.** Group Attn. is better than Performer and Linformer on

3 out of 4 datasets for classification. Although Linformer works slightly better than Group Attn. on the ECG dataset (90.37% vs 88.84%), its performance is the worst in all other cases compared to any other RITA-based methods. Vanilla computes the attention scores precisely. Thus it is expected to work well. However, Group Attn. outperforms Vanilla on WISDM (87.50% vs 86.95%) and is very close to it on other 3 datasets. This suggests that group attention’s approximation quality is good.

6.2.2 pretraining + few label finetune (Multi-variate classification)

The results shown in Table 3 get us the following observation:

(1) **Pretraining is effective.** Pretraining always leads to better accuracy than training with a few labels from scratch. In particular, on WISDM data all the methods using RITA architecture increase the accuracy by at least 10%. This is impressive considering we do not have a very large unlabeled pre-training set to use.

(2) **RITA’s advantage over TST.** our Group Attn. and other three baselines using RITA architecture (Vanilla, Performer, and Linformer) significantly outperform TST on all four classification datasets by 25 percentage points.

(3) **Group Attention’s advantage over other attention mechanisms.** Group Attn. is better than Performer and Linformer on 3 out of 4 datasets. When compared to Vanilla, Group Attn. is better on HHAR and ECG, and comparable on the other two, further confirming its high quality on approximation. Further, we notice that Linformer struggles in this setting: in average its accuracy is worse than Vanilla by 8.22% and Group Attn. by 8.01%. This is because the low-rank projection operation introduces extra model parameters, making Linformer more easily overfit, while overfitting is especially harmful when there are only a few labeled training samples.

6.2.3 full-dataset training (Multi-variate imputation)

Similar to classification tasks, the results of **imputation tasks** (Table.2) show that Group Attn. consistently outperforms the baselines in training time while achieving comparable/better MSE. Again, on the large dataset MGH (length = 10,000), TST and Vanilla fail due to out of memory (OOM) errors. Methods using RITA framework (Group Attn., Performer, Linformer) all achieve very low MSE (are highly accurate). Among them Linformer is the worst.

6.3 Efficiency: Transformer-based Methods

We measure the efficiency by the average training time per epoch including the cost of the forward computation + backward propagation and the grouping overhead. We first show the results on all the 5 datasets in Sec. 6.3.1. We then vary the length of the timeseries on the MGH dataset to show group attention’s scalability on long timeseries in Sec. 6.3.2.

6.3.1 Training Time: All Multi-variate Datasets

The results in Fig. 3(b) and Table 2 lead to the below observations:

(1) **Vanilla Self-Attention is not scalable.** In average, it takes 2-3 minutes to train one epoch when the length of the timeseries is only 200 (WISDM, HHAR, RWTHAR), takes over 15 minutes when the length increases to 2,000 (ECG), and fails on the long MGH data when the length reaches 10,000 due to out of GPU memory.

(2) **Group Attn.’s advantage over all other attention mechanisms.** As we have shown in Sec. 6.2, Group Attn. is more accurate

Dataset	Length	TST [61]		Vanilla		Performer		Linformer		Group Attn.	
		MSE	Time/s	MSE	Time/s	MSE	Time/s	MSE	Time/s	MSE	Time/s
WISDM	200	13.30	150.3	3.240	178.1	3.449	162.6	3.852	141.9	3.277	136.7
HHAR	200	1.085	78.2	0.2968	97.4	0.2980	82.6	0.3198	81.1	0.2974	73.3
RWHAR	200	0.0882	83.9	0.0478	108.1	0.0489	89.1	0.0572	98.4	0.0478	81.3
ECG	2000	0.0905	696.3	0.0037	857.9	0.0033	270.2	0.0035	291.38	0.0038	164.36
MGH	10000	N/A	N/A	N/A	N/A	0.00014	356.2	0.00088	404.9	0.00042	54.4

Table 2: Imputation results (multi-variate data). The best results are marked with **bold**.

Dataset	Pretrain Size	TST [61]		Vanilla		Performer		Linformer		Group Attn.	
		Scratch	Pre.	Scratch	Pre.	Scratch	Pre.	Scratch	Pre.	Scratch	Pre.
WISDM	62,231	49.13%	50.03%	66.16%	75.89%	66.09%	73.97%	50.12%	67.44%	62.56%	75.06%
HHAR	68,294	72.56%	75.30%	75.60%	81.35%	76.52%	80.70%	65.94%	76.52%	76.17%	82.62%
RWHAR	63,599	69.46%	80.41%	85.68%	91.14%	87.54%	91.33%	81.03%	86.33%	86.13%	89.63%
ECG	561,358	20.98%	27.99%	42.05%	46.16%	43.34%	45.58%	27.19%	31.34%	42.58%	46.39%

Table 3: Pretrain + few-label finetuning results. The best results are marked with **bold**.

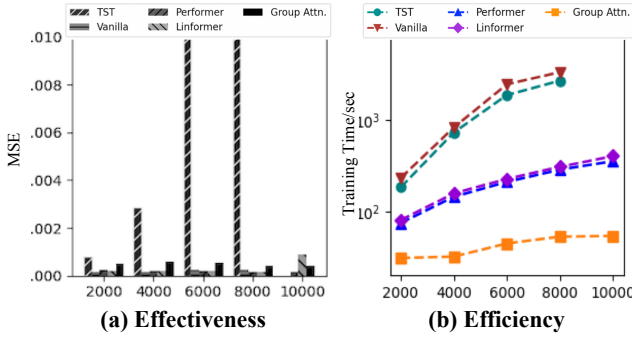


Figure 4: Varying the lengths of timeseries.

than Performer and Linformer in classification and imputation tasks, while Group Attn. is always faster than Performer, Linformer, and all other baselines on all 5 multi-variate datasets, thus a **win-win**.

(3) **The longer the timeseries, the larger the speedup.** On the medium sized ECG dataset with a length of 2,000, Group Attn. has a speedup of 3.86/1.36/2.27 compared to Vanilla/Performer/Linformer. When the length increases to 10,000, the speedup on the MGH dataset increases to 6.59/7.48 compared to Performer/Linformer (Vanilla and TST failed in this case) on imputation task (Table. 2). However, even on the short WISDM, HHAR, RWHAR datasets, Group Attn. still consistently outperforms other methods, confirming that it does not introduce much overhead. This is because when the length of the timeseries gets longer, Group Attn. gets more opportunities to find windows with similar properties.

6.3.2 Training time: Varying the Length

In this experiment, we truncate the original MGH timeseries into sequences with the lengths at 2000/4000/6000/8000/10000, and compare Group Attn. against Vanilla and other attention mechanisms. Vanilla cannot handle sequences longer than 8000.

The results in Fig. 4 again show that *the longer the timeseries, the larger the speed up*. With comparable MSE, Group Attn. outperforms Vanilla by 63X. Moreover, as the length increases from 2000 to 10000, the training time of Group Attn. only increases from 31.2 seconds to 54.4 seconds per epoch. The reason is that as the timeseries becomes longer, there are more grouping opportunities because of the similarity of the timeseries segments.

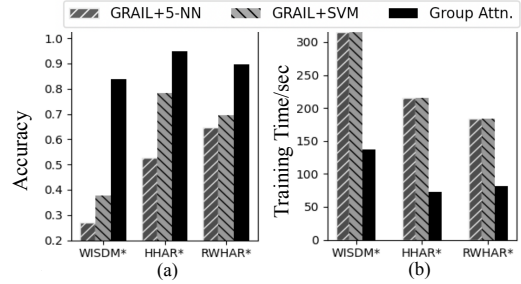


Figure 5: Comparison to non-deep learning method (uni-variate data).

6.4 Comparison to Non-deep Learning Methods

We compare against GRAIL, the SOTA of non-deep learning time-series representation learning. We use the three uni-variate datasets, because GRAIL only targets uni-variate timeseries.

Results in Fig. 5 show that on all 3 datasets RITA significantly outperforms GRAIL in accuracy by 45, 16, and 21 percentage points because of the expressive power of Transformer. Moreover, thanks to the GPU-friendly design of RITA, it is at least 2 \times faster than GRAIL in training time.

6.5 Ablation Study

6.5.1 Adaptive Scheduler

To evaluate the effectiveness of RITA’s adaptive scheduler (Sec. 5), we compare it against a baseline using a fixed group number N . We vary N and the error bound threshold ϵ used by RITA.

From the results in Table 4 we get the following observations:

(1) **Adaptive Scheduler is better than fixed N .** Training with Adaptive Scheduler already achieves better or comparable performance compared to the best performing N . More specifically, on the MGH dataset, dynamic scheduler always achieves better accuracy and is much faster compared to fixed N . On the ECG dataset, although fixed N is slightly better than adaptive scheduler in accuracy when setting the N as 512, it runs much slower than adaptive scheduler. Of course, finding the best N that balances the accuracy and running time requires careful tuning.

(2) **Adaptive Scheduler is tuning free.** It is robust on both accuracy and running time when ϵ varies, while the results of fixed N vary significantly when the value of N changes. Therefore, Adaptive Scheduler frees the users from tuning the ϵ threshold, while it is hard to find an appropriate N for a given dataset.

Dataset	Task	Scheduler	Parameter	Metric	Time
ECG	Class.	Dynamic	1.5	88.34%	292.5
			2	88.48%	236.8
			3	87.83%	216.8
		Fixed	64	87.50%	255.2
			128	88.96%	297.2
			256	88.82%	414.1
			512	90.03%	662.6
			1024	88.65%	873.7
MGH	Imput.	Dynamic	1.5	0.00041	60.7
			2	0.00040	57.9
			3	0.00042	54.4
		Fixed	128	0.00054	128.6
			256	0.00053	190.2
			512	0.00049	240.8
			1024	0.00046	323.3

Table 4: Adaptive Scheduling VS Fixed N.

Pretrain Data size	Few-label Accuracy
N/A	62.56%
12,446	72.94%
24,892	72.78%
37,338	74.10%
49,784	74.22%
62,231	75.06%

Table 5: RITA Pretraining: increasing sizes of pretrain set.

6.5.2 The Sizes of the Pretraining Data

Next, we evaluate how the number of unlabeled data influences the effectiveness of pretraining. To get empirical results, we pretrain RITA on WISDM dataset with 20%/40%/60%/80% of the pretraining data and finetune each pretrained model with 100 labels per class. The results in Table 5 show that: **(1) The more pretraining data, the larger the improvement.** The accuracy increases with the sizes of the pretraining data; **(2) Marginal utility diminishing.** The first 20% pretraining data gives a 10.38% improvement in accuracy (72.94% vs 62.56%), while the remaining 80% pretraining data only gives an additional improvement of 2.12% (75.06% vs 72.94%).

7 RELATED WORK

7.1 Timeseries Analytics

There is a great deal of prior work on timeseries analytics methods. This work can be divided into three categories: (1) non-deep learning methods; (2) CNN/RNN-based deep learning methods; and (3) Transformer-based deep learning methods.

Traditional Methods. These methods, such as TS-CHIEF [45], HIVE-COTE [33], ROCKET [15] have achieved notable performance on public datasets. Despite that, traditional methods suffer from one or more issues: they (1) rely on expert knowledge for feature extraction; (2) incur heavy computation cost and are inappropriate for GPU devices; (3) support only uni-variate timeseries; (4) perform classification solely. Some work [61] shows that the transformed methods outperform these traditional methods especially on multi-variate timeseries.

In particular, as the SOTA of timeseries **representation learning**, GRAIL [40] extracts landmarks from data and computes the representations with the combination of the landmarks. However, GRAIL only supports uni-variate timeseries. Our experiments (Sec. 6.4) show that RITA significantly outperforms GRAIL in both effectiveness and efficiency on uni-variate timeseries.

CNN/RNN-based Deep Learning Methods. CNN-based methods, such as InceptionTime [21] and Resnet [19], are good at classification tasks, but can not handle generative tasks such as forecasting because of the inductive bias of convolution networks. RNN-based methods, such as Brit [7] and deepAR [44], are capable for classification, regression and generation. However, the recurrent structure brings a lot of problems: (1) limiting the model’s ability in capturing long-range correlation; (2) notoriously difficult to train [41] because of gradient vanishing and exploding problem. As a result, such methods can hardly scale to very long timeseries.

Transformer-based Deep Learning Methods. Given that Transformer is the best choice for backbone in almost all sequence modeling tasks, some effort has been made to apply Transformer to timeseries analytics. Targeting forecasting of uni-variate timeseries, LogTrans [30] introduced a log sparsity assumption to attention computation. Informer [62] pushes LogTrans a step further and scales forecasting to multi-variate timeseries. Autoformer [57] performs forecasting by decomposing timeseries into two parts, i.e. the trend part and the seasonal part.

For imputation tasks, CDSA [37] outperforms statistical methods and the SOTA of RNN-based method Brit [7] on 3 public and 2 competition datasets. For timeseries classification, AutoTransformer [43] performs architecture search to adapt to the tasks in different domains. For timeseries anomaly detection, Anomaly Transformer [58] outperforms many widely-used methods such as OmniAnomaly [47], assuming the attention score maps show Gaussian distribution.

All of these works are designed for specific tasks, rather than functioning as a **representation learning** framework to serve different downstream tasks. To fill this gap, some researchers proposed a Transformer-based architecture, called TST [61]. Like RITA, TST supports regression, classification, and unsupervised learning through the “cloze test” pretraining task on timeseries. However, TST directly uses the classical Vanilla self-attention, thus not scalable to long timeseries as shown in our experiments (Sec. 6.3.2).

7.2 Efficient Transformers

The need of improving the scalability of Transformers has led to more efficient variations of Transformers, especially for accommodating long text data in NLP [49].

Introducing fixed/random patterns to self-attention mechanism is an intuitive idea. Sparse Transformer [9] and Longformer [3] only compute attention at fixed intervals. ETC [2] and BigBird [60] use global-local attention: the attention computation is limited within a fixed radius, while some auxiliary tokens are added to attend/get attended globally. The deficiencies of fixed attention patterns are obvious: it heavily depends on users to give an optimal setting.

To decrease the reliance on human labor, some works seek to introduce learnable/adaptive attention patterns instead of fixed patterns. Reformer [26] proposed only computing the dominant attention terms based on their observation of sparsity in attention matrix from language/image data. Such sparsity is intuitive in language data, in which a word’s attention mainly focuses on the nearby sentences. However, attention in timeseries data shows strong seasonal patterns rather than sparse patterns, mainly as

result of the periodicity of timeseries data. Therefore, such works do not work well for timeseries.

Apart from introducing attention patterns, some works seek to solve this problem with applied mathematics techniques. Linformer [54] performs a projection to decrease the size of query, key and value matrices before attention computation, because the attention matrix tends to be low-ranked. Performer [10] uses linear functions to approximate the kernel function *softmax*, making attention computation commutative. When the sequence length is far greater than the dimension of embedding vectors, Performer benefits from changing the order of matrix multiplication. Linformer and Performer do not depend on the unique properties of language data, thus potentially fitting timeseries better than other techniques, which is why we compared against them in our experiments. However as shown in Sec. 6, our group attention significantly outperforms them in both accuracy and efficiency (training time), because group attention fully leverages the periodicity of timeseries.

8 CONCLUSION

In this work, we presented RITA, an automatic, self-supervised, and scalable timeseries analytics tool. RITA effectively adapts Transformer, popular in NLP, into timeseries analytics. As the key component of RITA, group attention eliminates the performance bottleneck of the classical self-attention mechanisms, thus successfully scaling RITA to highly complex, long timeseries data. Our experiments confirm that RITA significantly speeds up the state-of-the-art by 63X with a better accuracy.

REFERENCES

- [1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).
- [2] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. *arXiv preprint arXiv:2004.08483* (2020).
- [3] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).
- [4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [5] C Bui, N Pham, A Vo, A Tran, A Nguyen, and T Le. 2017. Time series forecasting for healthcare diagnosis and prognostics with the focus on cardiovascular diseases. In *International conference on the development of biomedical engineering in Vietnam*. Springer, 809–818.
- [6] Lei Cao, Wenbo Tao, Sungtae An, Jing Jin, Yizhou Yan, Xiaoyu Liu, Wendong Ge, Adam Sah, Leilani Battle, Jimeng Sun, Remco Chang, M. Brandon Westover, Samuel Madden, and Michael Stonebraker. 2019. Smile: A System to Support Machine Learning on EEG Data at Scale. *Proc. VLDB Endow.* 12, 12 (2019), 2230–2241.
- [7] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. Brits: Bidirectional recurrent imputation for time series. *Advances in neural information processing systems* 31 (2018).
- [8] Chris Chatfield. 2000. *Time-series forecasting*. Chapman and Hall/CRC.
- [9] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).
- [10] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794* (2020).
- [11] Andrew A Cook, Göksel Misirlı, and Zhong Fan. 2019. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal* 7, 7 (2019), 6481–6494.
- [12] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.
- [13] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)* 20, 2 (1958), 215–232.
- [14] Benjamin F Crabtree, Subhash C Ray, Priscilla M Schmidt, Patrick T O’Connor, and David D Schmidt. 1990. The individual over time: time series applications in health care research. *Journal of clinical epidemiology* 43, 3 (1990), 241–260.
- [15] Angus Dempster, François Petitjean, and Geoffrey I. Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.* 34, 5 (2020), 1454–1495.
- [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, 2019, Volume 1 (Long and Short Papers)*. 4171–4186.
- [17] Evelyn Fix and Joseph Lawson Hodges. 1989. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique* 57, 3 (1989), 238–247.
- [18] Philip George Guest and Philip George Guest. 2012. *Numerical methods of curve fitting*. Cambridge University Press.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
- [20] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.
- [21] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. 2020. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery* 34, 6 (2020), 1936–1962.
- [22] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.
- [23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
- [24] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.
- [25] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems* 3, 3 (2001), 263–286.
- [26] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451* (2020).
- [27] John Kraft and Arthur Kraft. 1977. Determinants of common stock prices: A time series analysis. *The journal of finance* 32, 2 (1977), 417–425.
- [28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>
- [29] Oscar D Lara and Miguel A Labrador. 2012. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials* 15, 3 (2012), 1192–1209.
- [30] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyu Zhou, Wenhui Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems* 32 (2019).
- [31] T Warren Liao. 2005. Clustering of time series data—a survey. *Pattern recognition* 38, 11 (2005), 1857–1874.
- [32] Rake & Agrawal King-Ip Lin and Harpreet S Sawhney Kyuseok Shim. 1995. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceeding of the 21th International Conference on Very Large Data Bases*. 490–501.
- [33] Jason Lines, Sarah Taylor, and Anthony Bagnall. 2018. Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles. *ACM Trans. Knowl. Discov. Data* 12, 5, Article 52 (jul 2018), 35 pages.
- [34] Feifei Liu, Chengyu Liu, Lina Zhao, Xiangyu Zhang, Xiaoling Wu, Xiaoyan Xu, Yulin Liu, Caiyun Ma, Shoushui Wei, Zhiqiang He, et al. 2018. An open access database for evaluating the algorithms of electrocardiogram rhythm and morphology abnormality detection. *Journal of Medical Imaging and Health Informatics* 8, 7 (2018), 1368–1373.
- [35] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
- [36] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).
- [37] Jiawei Ma, Zheng Shou, Alireza Zareian, Hassan Mansour, Anthony Vetro, and Shih-Fu Chang. 2019. CDSA: cross-dimensional self-attention for multivariate, geo-tagged time series imputation. *arXiv preprint arXiv:1905.09904* (2019).

- [38] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
- [39] Tripti Negi and Veena Bansal. 2005. Time series: Similarity search and its applications. In *Proceedings of the International Conference on Systemics, Cybernetics and Informatics: ICSCI-04, Hyderabad, India*. 528–533.
- [40] John Paparrizos and Michael J Franklin. 2019. Grail: efficient time-series representation learning. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1762–1777.
- [41] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 1310–1318.
- [42] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.
- [43] Yankun Ren, Longfei Li, Xinxing Yang, and Jun Zhou. 2022. AutoTransformer: Automatic Transformer Architecture Design for Time Series Classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 143–155.
- [44] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.
- [45] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. 2020. TS-CHIEF: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery* 34 (2020), 742–775.
- [46] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. 2015. Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*. 127–140.
- [47] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2828–2837.
- [48] Timo Sztyler and Heiner Stuckenschmidt. 2016. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 1–9.
- [49] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)* (2020).
- [50] Mingyan Teng. 2010. Anomaly detection on time series. In *2010 IEEE International Conference on Progress in Informatics and Computing*, Vol. 1. IEEE, 603–608.
- [51] Patrick A Thompson. 1990. An MSE statistic for comparing forecast accuracy across series. *International Journal of Forecasting* 6, 2 (1990), 219–227.
- [52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5998–6008.
- [53] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- [54] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).
- [55] Gary M Weiss, Kenichi Yoneda, and Thaier Hayajneh. 2019. Smartphone and smartwatch-based biometrics using activities of daily living. *IEEE Access* 7 (2019), 133190–133202.
- [56] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. 2021. RobustPeriod: Robust Time-Frequency Mining for Multiple Periodicity Detection. In *Proceedings of the 2021 International Conference on Management of Data (Virtual Event, China) (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2328–2337. <https://doi.org/10.1145/3448016.3452779>
- [57] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems* 34 (2021), 22419–22430.
- [58] Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. 2021. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. *arXiv preprint arXiv:2110.02642* (2021).
- [59] Dianmin Yue, Xiaodan Wu, Yunfeng Wang, Yue Li, and Chao-Hsien Chu. 2007. A review of data mining-based financial fraud detection research. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. Ieee, 5519–5522.
- [60] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems* 33 (2020), 17283–17297.
- [61] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. A Transformer-based Framework for Multivariate Time Series Representation Learning. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*. 2114–2124.
- [62] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of AAAI*.

A APPENDIX: SUPPLEMENTARY MATERIAL

A.1 Experiment Configuration and Hyper-parameter Settings

Configuration. All models were trained on an NVIDIA Tesla V100 16GB GPU. All the methods are optimized with AdamW [36] of which the starting learning rate and weight decay parameter are both $1e^{-4}$. In full-label training scenario, we train the models for 100 epochs. In “pretraining + few-label finetuning scenario”, as the pretrained models require fewer epochs to converge [61], we train the model for 50 epochs. For a fair comparison, the baselines use a maximal batch size within GPU’s capacity during training.

As for model hyper-parameter setting, RITA and the baselines use a Transformer structure balancing *Vanilla*’s accuracy and efficiency: 8-layer stack of 2-head attention with hidden vectors in dimension of 64. Convolution kernel size is set to 5 by default. We set the error bound threshold (ϵ , Sec. 5.1) of Group Attention to 2, as it balances the accuracy and the efficiency in general on all datasets. Because Linformer requires the users to set the sizes of projection matrix, in different settings we choose an accuracy-efficiency balancing one among {64,128,256,512}.

A.2 Efficient Computation of Group Attention

Algorithm 1 Efficient Computation of Group Attention

Require: $Q, V, R, COUNT, BELONG$

Ensure: $Q, V \in \mathbb{R}^{n \times d}, R \in \mathbb{R}^{n \times d}, COUNT \in \mathbb{N}^N, BELONG \in \mathbb{N}^N$

```

1: function GROUP_ATTENTION( $Q, V, R$ )
2:   for  $i = 0 \rightarrow N - 1$  do
3:      $\tilde{v}_i \leftarrow \sum_{j=0}^{N-1} (BELONG_j == i) v_j$ 
4:    $\tilde{P} \leftarrow QR^T$ 
5:   for  $i = 0 \rightarrow n - 1$  do
6:     for  $j = 0 \rightarrow N - 1$  do
7:        $w_{i,j} \leftarrow \exp(\tilde{P}_{i,j}) COUNT_j$ 
8:   for  $i = 0 \rightarrow n - 1$  do
9:      $s_i \leftarrow \sum_{j=0}^{N-1} w_{i,j}$ 
10:  for  $i = 0 \rightarrow n - 1$  do
11:     $o_i \leftarrow \sum_{j=0}^{N-1} \frac{\exp(\tilde{P}_{i,j})}{s_i} \tilde{v}_j$ 
12:  return  $O$ 
```

In Alg. 1, we denote $COUNT_i$ to be the size of the i^{th} group, N to be the number of groups, \mathbf{r}_i to be the representative key of the i^{th} group and \mathbf{R} to be the matrix consisting of all \mathbf{r}_i , $BELONG_i$ to be the group that \mathbf{k}_i belongs to. Q, V are the packing matrices of query vectors and value vectors as described in Sec.2. Alg. 1 outputs the

packing matrix O for new feature embeddings $\{o_1, \dots, o_n\}$, where o_i corresponds to the feature embedding of win_i . Lines 2-3 implement the embedding aggregation operation, while Lines 8-11 implement the group softmax function.

A.3 The Algorithms and Optimality Proof for Dynamically Determining Batch Size

Algorithm 2 Binary Search for Batch Size

Require: L, N

Ensure: $1 \leq L \leq L_{\max}, 1 \leq N \leq L$

```

1: function BINARY_SEARCH( $L, N$ )
2:    $L \leftarrow 1$ 
3:    $R \leftarrow \text{MaxBatchSize}$ 
4:    $\text{data} \leftarrow \text{RandomTimeSeries in length } L$ 
5:    $B_{\text{temporal}} \leftarrow R$ 
6:   while  $L \leq R$  do
7:      $\text{Input} \leftarrow \text{data} \times B_{\text{temporal}}$ 
8:      $\text{ModelForward}(\text{Input})$ 
9:      $\text{ModelBackward}$ 
10:     $u \leftarrow \frac{\text{PeakMemoryUsage}}{\text{TotalMemory}}$ 
11:    if  $0.9 > u$  then
12:       $L \leftarrow B_{\text{temporal}} + 1$ 
13:       $B \leftarrow B_{\text{temporal}}$ 
14:    else
15:       $R \leftarrow B_{\text{temporal}} - 1$ 
16:     $B_{\text{temporal}} \leftarrow \lfloor \frac{L+R}{2} \rfloor$ 
17:  return  $B$ 
```

Algorithm 3 Dynamic Programming for Plane Division

Require: L_i, N_i, B_i, L_{\max}

Ensure: $1 \leq L_i \leq L_{\max}, 1 \leq N_i \leq L_i$

```

1: function COST( $S$ )
2:   if  $|S| < M$  then return  $+\infty$ 
3:    $L, N, B \leftarrow \text{points in } S$ 
4:    $f \leftarrow \text{function fitting}(B|L, N)$ 
5:   return  $E(B, L, N|f)$ 
6: function DYNAMIC_PROGRAMMING( $L_i, N_i, L_{\max}$ )
7:   for  $l_1 = 1 \rightarrow L_{\max}$  do
8:     for  $l_2 = 1 \rightarrow l_1$  do
9:       for  $n = 1 \rightarrow l_1$  do
10:         $S \leftarrow \text{points set in } \{l_2 \leq L \leq l_1, N \leq n\}$ 
11:         $g(n) \leftarrow \text{COST}(S)$ 
12:        for  $i = 1 \rightarrow n$  do
13:           $S \leftarrow \text{points set in } \{l_2 \leq L \leq l_1, i \leq N \leq n\}$ 
14:           $g(n) \leftarrow \min(g(n), g(i) + \text{COST}(S))$ 
15:         $f_{l_2, l_1} \leftarrow g(l_1)$ 
16:   for  $l = 1 \rightarrow L_{\max}$  do
17:      $dp(l) \leftarrow f(1, l)$ 
18:     for  $i = 1 \rightarrow l$  do
19:        $dp(l) \leftarrow \min(dp(l), dp(i) + f(i, l))$ 
20:   return  $dp(L_{\max})$ 
```

We describe Alg. 3 and intuitively show its optimality. We assume that Scipy [53] learns an optimal function in Line 4 so that function COST gives the optimal estimation error when fitting the points in set S . When fitting very few points, we assign an infinite cost to prevent a biased fitting function (Line 2). $g(n)$ denotes the minimal

estimation error for points in sub-plane $\{l_2 \leq L \leq l_1, N \leq n\}$. In Lines 11-13, we enumerate all possible ways of cutting $\{l_2 \leq L \leq l_1, N \leq n\}$ horizontally into two sub-plane $\{l_2 \leq L \leq l_1, N \leq i\}$ and $\{l_2 \leq L \leq l_1, i \leq N \leq n\}$ by iterating i from 1 to n . Choosing the cutting strategy that minimizes estimation error gets us a $g(l_1)$ with minimal estimation error for sub-plane $\{l_2 \leq L \leq l_1, N \leq l_1\}$, which is recorded as f_{l_1, l_2} in Line 14. $dp(l)$ denotes the minimal estimation error for sub-plane $\{L \leq l\}$. We enumerate all the possible ways of cutting $\{L \leq l\}$ vertically into two sub-plane $\{L \leq i\}$ and $\{i \leq L \leq l\}$ by iterating i from 1 to l (Line 17-19). Finally, we have the minimal estimation error for the whole plane as $dp(L_{\max})$. Based on the above discussion, this algorithm guarantees to not miss any better solution, hence optimal.

A.4 The Correctness of Group Attention

LEMMA 3. Assuming the windows belonging to the same group G_i have the same key vector, i.e. $k_j = r_i(\text{win}_j \in G_i)$, then the feature embedding O produced by the original self-attention mechanism is identical to the output of our group attention mechanism implemented in Algorithm 1.

PROOF. Denote \tilde{k}_j to be the representative vectors of k_j , i.e. $\tilde{k}_j = r_i = k_j(\text{win}_j \in G_i)$. Algorithm 1 gives that

$$\begin{aligned} \tilde{v}_i &= \sum_{j=0}^{n-1} (\text{BELONG}_j == i) \mathbf{v}_j, \quad \tilde{P}_{i,j} = \mathbf{q}_i \cdot \mathbf{r}_j \\ s_i &= \sum_{j=0}^{N-1} \exp(\tilde{P}_{i,j}) \text{COUNT}_j, \quad \tilde{o}_i = \sum_{j=0}^{N-1} \frac{\tilde{P}_{i,j}}{s_i} \tilde{v}_j \end{aligned} \quad (7)$$

By the canonical self-attention mechanism introduced in Sec. 2, we get:

$$P_{i,j} = \mathbf{q}_i \cdot \mathbf{k}_j, \quad A_{i,j} = \frac{\exp(P_{i,j})}{\sum_{k=0}^{n-1} \exp(P_{i,k})}, \quad \mathbf{o}_i = \sum_{j=0}^{n-1} A_{i,j} \mathbf{v}_j \quad (8)$$

With 7 and 8, we have

$$\begin{aligned} \sum_{j=0}^{n-1} \exp(P_{i,j}) &= \sum_{j=0}^{n-1} \exp(\mathbf{q}_i \cdot \mathbf{k}_j) \\ &= \sum_{j=0}^{N-1} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \exp(\mathbf{q}_i \cdot \mathbf{k}_x) \\ &= \sum_{j=0}^{N-1} \exp(\mathbf{q}_i \cdot \mathbf{r}_j) \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \\ &= \sum_{j=0}^{N-1} \exp(\mathbf{q}_i \cdot \mathbf{r}_j) \text{COUNT}_j \\ &= \sum_{j=0}^{N-1} \exp(\tilde{P}_{i,j}) \text{COUNT}_j \\ &= s_i \end{aligned} \quad (9)$$

Further,

$$\begin{aligned}
\mathbf{o}_i &= \sum_{j=0}^{n-1} A_{i,j} \mathbf{v}_j \\
&= \sum_{j=0}^{N-1} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) A_{i,x} \mathbf{v}_x \\
&= \sum_{j=0}^{N-1} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \frac{\exp(P_{i,x})}{\sum_{k=0}^{n-1} \exp(P_{i,k})} \mathbf{v}_x \\
&= \sum_{j=0}^{N-1} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \frac{\exp(\mathbf{q}_i \cdot \mathbf{k}_x)}{\sum_{k=0}^{n-1} \exp(P_{i,k})} \mathbf{v}_x \\
&= \sum_{j=0}^{N-1} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \frac{\exp(\mathbf{q}_i \cdot \mathbf{r}_j)}{\sum_{k=0}^{n-1} \exp(P_{i,k})} \mathbf{v}_x \\
&= \sum_{j=0}^{N-1} \frac{\exp(\mathbf{q}_i \cdot \mathbf{r}_j)}{\sum_{k=0}^{n-1} \exp(P_{i,k})} \sum_{x=0}^{n-1} (\text{BELONG}_x == j) \mathbf{v}_x \\
&= \sum_{j=0}^{N-1} \frac{\exp(\mathbf{q}_i \cdot \mathbf{r}_j)}{\sum_{k=0}^{n-1} \exp(P_{i,k})} \tilde{\mathbf{v}}_j
\end{aligned} \tag{10}$$

Combining (7), (9) (10), we have $\mathbf{o}_i = \sum_{j=0}^{N-1} \frac{\tilde{P}_{i,j}}{s_i} \tilde{\mathbf{v}}_j = \tilde{\mathbf{o}}_i$.

This concludes that the output of our group attention is identical to vanilla self-attention's. \square

A.5 The Proof of Error Bound (Lemma 1)

PROOF. We have

$$\begin{aligned}
\frac{\exp(\bar{P}_{i,j})}{\exp(P_{i,j})} &= \frac{\exp(\mathbf{q}_i \cdot \tilde{\mathbf{k}}_j)}{\exp(\mathbf{q}_i \cdot \mathbf{k}_j)} = \exp(\mathbf{q}_i \cdot (\tilde{\mathbf{k}}_j - \mathbf{k}_j)) \\
&= \exp(|\mathbf{q}_i| \cdot \|\tilde{\mathbf{k}}_j - \mathbf{k}_j\| \cdot \cos(\mathbf{q}_i, \tilde{\mathbf{k}}_j - \mathbf{k}_j))
\end{aligned} \tag{11}$$

So

$$\exp(-dR) \leq \frac{\exp(\bar{P}_{i,j})}{\exp(P_{i,j})} \leq \exp(dR) \tag{12}$$

Then we have:

$$\begin{aligned}
\frac{\bar{A}_{i,j}}{A_{i,j}} &= \frac{\exp(\bar{P}_{i,j})}{\sum_{k=1}^n \exp(\bar{P}_{i,k})} / \frac{\exp(P_{i,j})}{\sum_{k=1}^n \exp(P_{i,k})} \\
&= \frac{\exp(\bar{P}_{i,j})}{\exp(P_{i,j})} \frac{\sum_{k=1}^n \exp(P_{i,k})}{\sum_{k=1}^n \exp(\bar{P}_{i,k})}
\end{aligned} \tag{13}$$

Combining (12) (13), the error is bounded by

$$\exp(-2dR) \leq \frac{\bar{A}_{i,j}}{A_{i,j}} \leq \exp(2dR) \tag{14}$$

Thus, if $d \leq \frac{\ln(\epsilon)}{2R}$, $\frac{1}{\epsilon} \leq \frac{\bar{A}_{i,j}}{A_{i,j}} \leq \epsilon$. This proves Lemma 1.

A.6 The Proof of Merge Operation (Lemma 2)

PROOF. Denote the cluster size of $cluster_k$ to be n_k . After merging, the new center will be:

$$c' = \frac{\sum_{i=1}^m n_{k_i} c_{k_i}}{\sum_{i=1}^m n_{k_i}}$$

For $\forall i \in [1, m], \forall x \in cluster_{k_i}$, it holds that:

$$\begin{aligned}
|x - c'| &\leq |x - c_{k_i}| + |c_{k_i} - c'| \quad (\text{Triangle inequality}) \\
&= |x - c_{k_i}| + \left| \frac{\sum_{j=1}^m n_{k_j} c_{k_j}}{\sum_{j=1}^m n_{k_j}} - c_{k_i} \right| \\
&= |x - c_{k_i}| + \left| \frac{\sum_{j=1}^m n_{k_j} (c_{k_j} - c_{k_i})}{\sum_{j=1}^m n_{k_j}} \right| \\
&= |x - c_{k_i}| + \frac{|\sum_{j=1}^m n_{k_j} (c_{k_j} - c_{k_i})|}{\sum_{j=1}^m n_{k_j}} \\
&\leq |x - c_{k_i}| + \frac{\sum_{j=1}^m n_{k_j} |c_{k_i} - c_{k_j}|}{\sum_{j=1}^m n_{k_j}} \\
&= \frac{\sum_{j=1}^m n_{k_j} (|c_{k_i} - c_{k_j}| + |x - c_{k_i}|)}{\sum_{j=1}^m n_{k_j}} \\
&\leq \frac{\sum_{j=1}^m n_{k_j} d}{\sum_{j=1}^m n_{k_j}} = d
\end{aligned} \tag{15}$$

A.7 Downstream Tasks

RITA supports a variety of downstream tasks. In this section, we show that with minimal modification RITA can effectively support classification, imputation and forecasting tasks. Other unsupervised tasks such as similarity search or clustering are naturally supported by extracting feature embeddings from RITA.

A.7.1 Classification

To classify timeseries, we input timeseries to the model as described in Sec. 3 and attach a special token **[CLS]** as the first input embedding. **[CLS]**'s embedding acts as the embedding for the entire timeseries, and the output representation of **[CLS]** is fed into a classifier: $y = \text{Softmax}(W_{cls} Z_{[CLS]} + B_{cls})$, where $Z_{[CLS]} \in \mathbb{R}^d$ is the output representation of **[CLS]**, C is the number of classes, and $W_{cls} \in \mathbb{R}^{C \times d}, B_{cls} \in \mathbb{R}^C$ are learnable parameters for classification task. The result vector $y \in \mathbb{R}^C$ represents the possibility that the input timeseries belongs to each class.

We apply Cross Entropy Loss as the loss function of the classification task [13]: $L = \frac{1}{C} \sum_{i=1}^C -\hat{y}(i) \log(y(i))$, where \hat{y} is a binary indicator for ground truth label:

$$\hat{y}(i) = \begin{cases} 1 & i \text{ is ground truth label} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

A.7.2 Imputation

Timeseries are mainly generated by sensors, a common problem of which is missing values. This becomes a challenge when many downstream analytics require the missing values to be recovered. The recovering task is imputation.

Denote the real timeseries as $T_r \in \mathbb{R}^{t \times m}$, the observed timeseries with missing values as $T_o \in \mathbb{R}^{t \times m}$, and the set of missing values' positions as M . We scale the values of all timeseries to non-negative and use a special value (-1) to indicate missing values:

$$T_o(i, j) = \begin{cases} -1 & (i, j) \in M \\ T_r(i, j) & (i, j) \notin M \end{cases} \tag{17}$$

T_o is fed into the RITA as input, and the output representations are concatenated and fed into a *Transpose Convolution* layer which decodes the output embedding vectors from hidden space to timeseries values, corresponding to the convolution operation in

the input stage, i.e., $Y = \text{TransposeCNN}(Z_1 \oplus Z_2 \oplus \dots \oplus Z_n)$, where $Y \in \mathbb{R}^{t \times m}$ is the recovered timeseries, and $Z_i \in \mathbb{R}^d$ is the output of each position.

Here Mean Square Error is chosen as the loss function [51]: $L = \frac{1}{|M|} \sum_{(i,j) \in M} (Y(i,j) - T_r(i,j))^2$.

A.7.3 Forecasting

Forecasting can be regarded as a special case of imputation, in which all missing values are at the end of timeseries.

So like in imputation task, we scale the timeseries to non-negative and use a special value (-1) to indicate the values to be predicted:

$$T_{observed}(i,j) = \begin{cases} T_{real}(i,j) & i \leq t_{observed} \\ -1 & otherwise \end{cases} \quad (18)$$

Where $t_{observed}$ is the observed timestamp. Then the output representations are fed into a Transpose Convolution layer using Mean Squared Error as loss function, as described above.

A.7.4 Other Unsupervised Tasks

RITA naturally supports other unsupervised tasks, such as similarity search and clustering [25, 31, 32], by producing the embedding of one timeseries (output representation of the special token [CLS]). Clustering can be performed on the embeddings with flexible choice of distance metrics. Similarly, a high dimensional similarity search system [22, 23, 38] can be built on the embeddings.

A.8 Inference Time

Dataset	Length	TST[61]	Vanilla	Performer	Linformer	Group Attn.
WISDM	200	2.18	2.26	2.35	2.22	2.17
HHAR	200	1.19	1.23	1.28	1.21	1.18
RWHAR	200	1.32	1.37	1.42	1.34	1.31
ECG	2000	18.44	15.26	5.80	6.08	5.16

Table 6: Inference time: Classification on multi-variate data (seconds).

Dataset	Length	TST[61]	Vanilla	Performer	Linformer	Group Attn.
WISDM	200	2.03	2.11	2.19	2.07	2.02
HHAR	200	1.11	1.14	1.19	1.12	1.10
RWHAR	200	1.23	1.27	1.32	1.25	1.22
ECG	2000	17.22	14.32	4.73	4.99	4.11
MGH	10000	N/A	N/A	6.58	6.88	1.35

Table 7: Inference time: Imputation on multi-variate data (seconds).

In this section, we present the average inference time on validation sets. The results in Table. 6 and 7 correspond to the average inference time on validation sets of classification and imputation tasks, respectively. Consistent with the results in Section. 6.3, our method Group Attn. outperforms the baselines on both classification and imputation tasks, particularly on the datasets comprising long timeseries (ECG and MGH).