RITA: Group Attention is All You Need for Timeseries Analytics

JIAMING LIANG, University of Pennsylvania, USA
LEI CAO, U of Arizona/MIT, USA
SAMUEL MADDEN, Massachusetts Institute of Technology, USA
ZACHARY IVES, University of Pennsylvania, USA
GUOLIANG LI, Tsinghua University, China

Timeseries analytics is important 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: embeddings are key to the performance of various timeseries analytics tasks such as similarity-based timeseries queries within vector databases. However, 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.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Timeseries Analytics, Self-supervised, Attention, Efficient Transformers

ACM Reference Format:
Jiaming Liang, Lei Cao, Samuel Madden, Zachary Ives, and Guoliang Li. 2024. RITA: Group Attention is All You Need for Timeseries Analytics. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 62 (February 2024), 28 pages. https://doi.org/10.1145/3639317

1 INTRODUCTION

Many data-driven applications involve processing massive timeseries data, including IoT [14], medical AI [17], stock market [32], etc. As such, there is a great need for timeseries analytics, such as forecasting [9], classification [24], clustering [36], similarity search [45], and anomaly detection [57], with applications ranging from automatically diagnosing diseases [6], recognizing human activities [34], to stopping financial fraud [67].

In particular, in the database community, vector databases [61] are being used to store and index high-dimensional *feature embeddings* of unstructured and semi-structured data, which allows users

Authors' addresses: Jiaming Liang, University of Pennsylvania, Philadelphia, PA, USA, liangjm@seas.upenn.edu; Lei Cao, U of Arizona/MIT, Tucson, AZ, 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.

ACM 2836-6573/2024/2-ART62
https://doi.org/10.1145/3639317

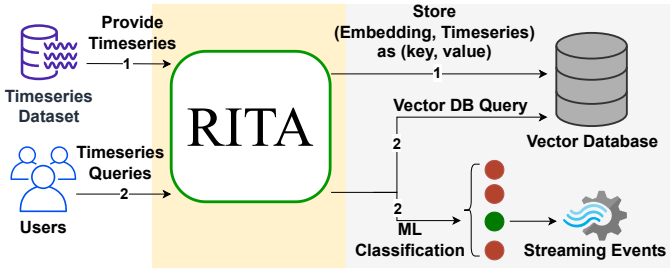

Fig. 1. RITA in a timeseries-based query system

to efficiently explore this data by conducting similarity searches in the embedding space. Effective feature representations are a key building-block to construct a valid vector database.

As a real-world example, we have been collaborating with a major US hospital [7] to develop a large-scale interactive system to label EEG segments (450 million segments, 30TB) with 6 classes representing different types of seizures. These labeled EEG segments are used to train a classifier which can then automatically detect seizures based on EEG signals collected during the clinical observation of patients. The goal of this system is to propagate the labels provided by the experts to similar segments, thus reducing the manual labeling efforts by the neurologists. Our timeseries feature embedding approach, RITA, functions as a core component in this system. More specifically, we use RITA to convert the EEG segments into feature embeddings and store them in a vector database which supports similarity search requests submitted by the neurologists to return the k nearest neighbors of the to-be-labelled segment, as depicted in Fig. 1.

Recently researchers [70] have started leveraging the *self-supervised pre-training* methodology of Transformers [5, 19, 59], 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 [70] confirm that Transformer-based methods outperform traditional timeseries analytics techniques.

Existing work [70] that directly applies Transformers to learn features from timeseries data has been shown not to scale to *long* timeseries [35]. The idea of self-attention [59] 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. This limits the model's scalability, especially when handling large-scale timeseries data, which is common in real-world applications such as IoT, medical AI, and finance [7, 39, 71]. Predictions about timeseries may need to look at hundreds of thousands of prior samples to achieve accuracy.

Referring again to our EEG project, seizures are brief, so we chunk EEG data into 2-second segments and detect seizures at the segment level. However, the classification of a particular segment depends on up to 12 hours of prior signal to consider long-term trends and determine if one 2-second segment indicates a seizure. There are more than 21,000 segments in 12 hours. This greatly exceeds the number of semantic units that typical NLP tasks expect. For example, BERT [19]

limits the number of units to 512 and even massive models like GPT-3 [5] limit the number of units to 2048.

Although in NLP some lower-complexity methods have been proposed to *approximately* compute self-attention [11, 31, 62], 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. Our goal is to significantly speed up execution versus prior methods while sacrificing minimal predictive accuracy.

Leveraging the similarity among long timeseries pieces, 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, enabling them to approximately share the computation of attention. As the timeseries increases in length, more sharing opportunities become available. RITA then computes 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.** 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, they may not share the same feature embedding. Using the group attention matrix, the existing self-attention mechanism will produce only 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 speedup and 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 phase, group attention should use different values of N in different iterations to adapt to the varying feature embeddings. This makes manually optimizing 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 (1) makes full use of GPU memory; (2) enables parallelism across the samples. Thus, RITA should dynamically adjust the 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 [59]. Together they ensure RITA able to directly use the compressed attention matrix to produce different feature embeddings for different segments. We theoretically show that the embeddings RITA produces in this way are identical to those produced by first re-storing the original large attention matrix. Thus RITA produces high-quality embeddings without introducing extra overhead.

Second, we design an *adaptive scheduler* that 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 predicts 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 is able to learn 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 [7] confirm that compared to existing self-attention mechanisms [11, 12, 46, 59, 62], our group attention mechanism achieves a 63X speedup with much less memory required with comparable or even better accuracy on various timeseries analytics tasks.

Contributions. The key contributions of this work include:

- Our group attention mechanism reduces 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 analytics tasks, demonstrating that RITA is 4 to 63 times faster than the existing Transformer-based approaches while achieving comparable or better accuracy when handling long timeseries (length ≥ 2000).

2 BACKGROUND

We provide some background on the canonical self-attention module in the Transformer [59]. *Self-attention* takes n hidden embedding vectors $H \in \mathbb{R}^{n*d_h}$ 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:

$$Q = HW_Q, K = HW_K, V = HW_V, O = AV = SoftMax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

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

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

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

3 RITA OVERVIEW

Given a collection of *unlabeled* timeseries, RITA first pre-trains a Transformer-style model to produce high quality feature embeddings. This pre-trained model is then used to support various downstream tasks with minimal modifications to the model architecture, aligned with the design philosophy of BERT [19].

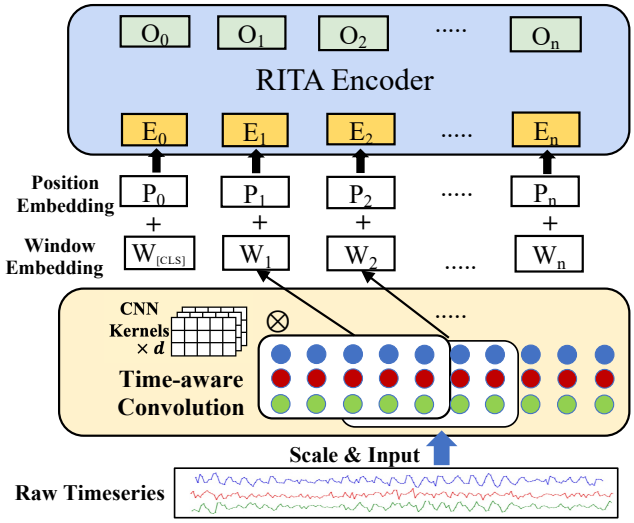

Fig. 2. RITA architecture

Next, we use an example to show how RITA supports a timeseries query system in Sec. 3.1, followed by an overview of the RITA model architecture in Sec. 3.2. We will provide a detailed illustration on how RITA addresses a range of downstream tasks in Sec. 6.

3.1 RITA in Timeseries Query Systems

Fig. 1 shows a Seizure Diagnosis System we have been developing for a major US hospital. RITA is used both in the training stage and the deployment stage. As described in the Introduction, we use RITA to encode the EEG segments and store the resulting (timeseries, embedding) pairs in a vector database as (key, value) pairs. This is used to for similarity-based labeling. Thereafter, we train a classification model by fine-tuning the RITA model and deploy it in the patient monitoring system to detect seizures at real time. Once a new EEG segment arrives, the system uses the RITA encoder to embed it into a feature embedding, and thereafter classifies it into one type of seizures.

3.2 Model Architecture

As shown in Fig. 2, RITA consists of two components: (1) *Time-aware Convolution Layer* and (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 [33] 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 $\mathbf{n}$ windows and produce one d-dimensional embedding per window using the convolution operation [33]. 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 takes the embeddings of n semantic units $X_1, X_2, ..., X_n (X_i \in R^d)$ as input (e.g. embeddings of n windows for a timeseries), then models the correlations between the semantic units and outputs $Y_1, ..., Y_n (Y_i \in R^d)$ 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 scalablity 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" pretraining 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. We use examples to explain the group attention mechanism, as depicted in Figure 3.

4.1 The Idea of Group Attention

As periodicity is a natural property of timeseries [64], 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 a key vector similar to that of 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 $k_j \approx k_x$.

Example 1. As shown in Figure 3 (Part 1), the first and third timeseries windows are notably similar, consequently yielding akin key vectors ($k_1 \approx k_3$). In vanilla self-attention (Part 2), this similarity leads to two closely related columns within the attention matrix, corresponding to key vectors k_1 and k_3.

This observation inspires our group attention mechanism. That is, we group the windows by their similarity in keys. Assuming that 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. Thus, this 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 this minimally impacts quality, as confirmed in our experiments (Sec. 7.2).

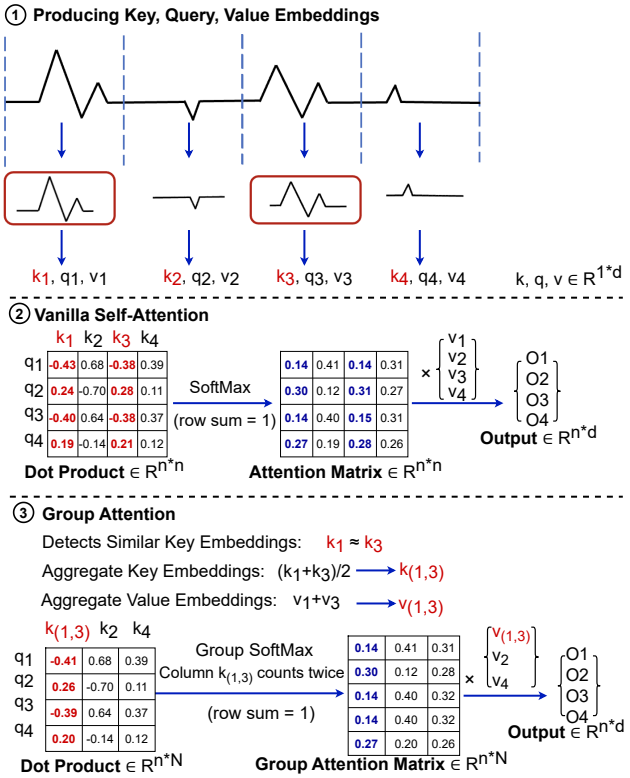

Fig. 3. Group Attention vs. Vanilla Self-Attention

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 with the Group Attention Matrix*

As described in the Background (Sec. 2), 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 $\widetilde{A}$, where N corresponds to the number of groups. In this case the matrix product will produce a $N \times d_v$ matrix $\widetilde{O}$. 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 $\widetilde{A}$. For example, given one group composed of win_i and win_j, we map its group attention vector in $\widetilde{A}$ 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.

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 $a_i =< a_i^1, a_i^2, \cdots, a_i^n >$ and $v_j =< v_j^1, v_j^2, \cdots, v_j^n >$, $O_{i,j} = \sum_{k=1}^{n} a_i^k v_j^k$.

Example 2. As shown in Fig. 3 (Part 3), assume win_1 and win_3 belong to the same group G_1. Then $a_i^1 = a_i^3 = \widetilde{a}_i^1$, where $\widetilde{a}_i^1 \in \widetilde{A}$ corresponds to the attention of group G_1 onto win_i. Therefore, $a_i^1 v_j^1 + a_i^3 v_j^3 = \widetilde{a}_i^1 (v_j^1 + v_j^3)$. So we aggregate $v_{(1,3)} = v_1 + v_3$.

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 $\widetilde{v}_j$ beforehand, we could directly use the group attention vector $\widetilde{a}_i$ and the group feature vector $\widetilde{v}_j$ to compute $O_{i,j}$.

Using embedding aggregation, RITA is able to produce the feature embedding $\widetilde{O}$ 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 $A = SoftMax(\frac{QK^T}{\sqrt{d_k}})$. To compute A, we have to first compute QK^T (denoted as P) which is an $n \times n$ matrix. Normalizing the P matrix with softmax produces the attention matrix A.

Group attention follows the same procedure. However, after grouping keys into $\widetilde{K}$, $Q\widetilde{K}^T$ produces an $n \times N$ matrix $\widetilde{P}$. Due to the non-linearity of the softmax function, applying softmax directly on $\widetilde{P}$ will result in a group attention matrix $\widetilde{A}$ from which we are not able to recover a full attention matrix that is identical to first restoring $\widetilde{P}$ 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 $n \times N$ $\widetilde{P}$ 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).

$$GroupSoftMax(\widetilde{P_{i,j}}) = \frac{exp(P_{i,j})}{\sum_{k=0}^{N-1} count_k exp(P_{i,k})} \tag{3}$$

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 $P_{i,k}$. For instance, in Fig. 3 (Part 3), we count the column corresponding to $k_{(1,3)}$ twice in GroupSoftMax because there are two elements (k_1, k_3) in this group. In this way, the group softmax function operating on the small $\widetilde{P}$ matrix will produce *exactly the same* result to the softmax function operating on the full P matrix.

Efficient Implementation. Next, we demonstrate an efficient implementation of the embedding aggregation operation and group softmax function in Alg. 1. We denote CNT_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 $\mathbf{r}_i$, BNG_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 emebddings $\{o_1, ..., o_n\}$, where o_i corresponds to the feature embedding of win_i.

Algorithm 1 Efficient Computation of Group Attention

Require: Q, V, R, CNT, BLG
Ensure: $Q, V \in \mathbb{R}^{n*d}$,$R \in \mathbb{R}^{N*d}$,$CNT \in \mathbb{N}^{N}$,$BLG \in \mathbb{N}^{n}$
1: **function** GROUP_ATTENTION(Q, V, R)
2: **for** $i = 0 \rightarrow N-1$ **do**
3: $\widetilde{v}_i \leftarrow \sum_{j=0}^{n-1}(BLG_j == i)v_j$
4: $\widetilde{P} \leftarrow QR^T$
5: **for** $i = 0 \rightarrow n-1$ **do**
6: **for** $j = 0 \rightarrow N-1$ **do**
7: $w_{i,j} \leftarrow exp(\widetilde{P}_{i,j})CNT_j$
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: $o_i \leftarrow \sum_{j=0}^{N-1} \frac{exp(\widetilde{P}_{i,j})}{s_i}\widetilde{v}_j$
12: **return** O

Lines 2-3 implement the embedding aggregation operation, while Lines 8-11 implement the group softmax function.

The Correctness Proof of the Group Attention Algorithm. Here we prove that our efficient group attention algorithm, i.e., Alg. 1, produces the same output feature embedding with the naive method that has to first restore the big full attention matrix.

Note in group attention's computation, we use a representative vector to represent all the key vectors in the i^{th} group, thus satisfying the assumption made in Lemma 3.

LEMMA 3. *Assuming the windows belonging to the same group G_i have the same key vector, i.e. $k_j = r_i(win_j \in G_i)$, 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 $\widetilde{k_j}$ to be the representative vectors of k_j, i.e. $\widetilde{k_j} = r_i = k_j(win_j \in G_i)$. Algorithm 1 gives that

$$\begin{aligned} \widetilde{v}_i &= \sum_{j=0}^{n-1}(BLG_j == i)\mathbf{v}_j,\ \widetilde{P}_{i,j} = \mathbf{q}_i \cdot \mathbf{r}_j \\ s_i &= \sum_{j=0}^{N-1} exp(\widetilde{P}_{i,j})CNT_j,\ \widetilde{o}_i = \sum_{j=0}^{N-1}\frac{\widetilde{P}_{i,j}}{s_i}\widetilde{v}_j \end{aligned} \tag{4}$$

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

$$P_{i,j} = \mathbf{q}_i \cdot \mathbf{k_j},\ A_{i,j} = \frac{exp(P_{i,j})}{\sum_{k=0}^{n-1} exp(P_{i,k})},\ \mathbf{o}_i = \sum_{j=0}^{n-1} A_{i,j}\mathbf{v}_j \tag{5}$$

With 4 and 5, we have

$$\begin{aligned} \sum_{j=0}^{n-1} exp(P_{i,j}) &= \sum_{j=0}^{n-1} exp(\mathbf{q}_i \cdot \mathbf{k}_j) = \sum_{j=0}^{N-1}\sum_{x=0}^{n-1}(BLG_x == j)exp(\mathbf{q}_i \cdot \mathbf{k}_x) \\ &= \sum_{j=0}^{N-1} exp(\mathbf{q}_i \cdot \mathbf{r}_j)\sum_{x=0}^{n-1}(BLG_x == j) = \sum_{j=0}^{N-1} exp(\widetilde{P}_{i,j})CNT_j = s_i \end{aligned} \tag{6}$$

Further,

$$\begin{aligned}\mathbf{o}_i &= \sum_{j=0}^{n-1} A_{i,j}\mathbf{v_j} = \sum_{j=0}^{N-1}\sum_{x=0}^{n-1}(BNG_x == j)A_{i,x}\mathbf{v}_x \\ &= \sum_{j=0}^{N-1}\sum_{x=0}^{n-1}(BLG_x == j)\frac{exp(\mathbf{q}_i\cdot\mathbf{k}_x)}{\sum_{k=0}^{n-1}exp(P_{i,k})}\mathbf{v}_x \\ &= \sum_{j=0}^{N-1}\frac{exp(\mathbf{q}_i\cdot\mathbf{r_j})}{\sum_{k=0}^{n-1}exp(P_{i,k})}\sum_{x=0}^{n-1}(BLG_x == j)\mathbf{v}_x = \sum_{j=0}^{N-1}\frac{exp(\mathbf{q}_i\cdot\mathbf{r_j})}{\sum_{k=0}^{n-1}exp(P_{i,k})}\widetilde{v}_j\end{aligned} \tag{7}$$

Combining (4), (6) (7), we have $\mathbf{o}_i = \sum_{j=0}^{N-1}\frac{\widetilde{P}_{i,j}}{s_i}\widetilde{\mathbf{v}}_j = \widetilde{o}_i$.
This concludes that the output of our group attention is identical to vanilla self-attention's. □

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 $\widetilde{A}$ which approximates the attention matrix A produced by the classical self-attention with a *bounded error*, as shown in Lemma 4.

Lemma 4. *Let R be the radius of the ball where all key vectors live; $\widetilde{k}_i$ be the representative of the group that contains key k_i. Let $\overline{A}$ denote the full attention matrix restored from $\widetilde{A}$. Suppose the distance between $\widetilde{k}_i$ and k_i ($||\widetilde{\mathbf{k}}_i - \mathbf{k}_i||$) satisfies: $||\widetilde{\mathbf{k}}_i - \mathbf{k}_i|| \le \mathbf{d}$.*

Then $\forall\ \epsilon > 1$, if $d \le \frac{\ln(\epsilon)}{2R}$, $\frac{1}{\epsilon} \le \frac{\overline{A}_{i,j}}{A_{i,j}} \le \epsilon$

Lemma 4 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.

Proof. We have

$$\begin{aligned}\frac{exp(\widetilde{P}_{i,j})}{exp(P_{i,j})} &= \frac{exp(\mathbf{q}_i\cdot\widetilde{\mathbf{k}}_j)}{exp(\mathbf{q}_i\cdot\mathbf{k}_j)} = exp(\mathbf{q}_i\cdot(\widetilde{\mathbf{k}}_j - \mathbf{k}_j)) \\ &= exp(||\mathbf{q}_i||||\widetilde{\mathbf{k}}_j - \mathbf{k}_j||cos(\mathbf{q}_i, \widetilde{\mathbf{k}}_j - \mathbf{k}_j))\end{aligned} \tag{8}$$

So

$$exp(-dR) \le \frac{exp(\widetilde{P}_{i,j})}{exp(P_{i,j})} \le exp(dR) \tag{9}$$

Then we have:

$$\begin{aligned}\frac{\widetilde{A}_{i,j}}{A_{i,j}} &= \frac{exp(\widetilde{P}_{i,j})}{\sum_{k=1}^{n}exp(\widetilde{P}_{i,k})} \Big/ \frac{exp(P_{i,j})}{\sum_{k=1}^{n}exp(P_{i,k})} \\ &= \frac{exp(\widetilde{P}_{i,j})}{exp(P_{i,j})}\frac{\sum_{k=1}^{n}exp(P_{i,k})}{\sum_{k=1}^{n}exp(\widetilde{P}_{i,k})}\end{aligned} \tag{10}$$

Combining (9) (10), the error is bounded by

$$exp(-2dR) \le \frac{\widetilde{A}_{i,j}}{A_{i,j}} \le exp(2dR) \tag{11}$$

Thus, if $d \le \frac{\ln(\epsilon)}{2R}$, $\frac{1}{\epsilon} \le \frac{\widetilde{A}_{i,j}}{A_{i,j}} \le \epsilon$. This proves Lemma 4. □

4.4 GPU-Friendly Grouping Method

We implement a grouping method that corresponds to K-means clustering [41], because K-means offers a tight distance bound between each key and its group representative. We design a GPU-friendly implementation of K-means compliant to the Transformer architecture. 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 $v_i - c_j$. Instead, we use a different formulation: $|v_i - c_j| = |v_i - c_j| = \sqrt{|v_i|^2 + |c_j|^2 - 2v_i \cdot c_j}, i \in [1, n], j \in [1, N]$. In this formulation, the performance bottleneck is $v_i \cdot c_j$, which could be implemented as a matrix product operation, while 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. 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 [30], thus typically 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 4 to translate ϵ to a distance threshold d. We believe this error bound is the most natural knob for users to specify based on the need of the domain. Compared to setting a distance threshold to indirectly influence the approximation error, this is more intuitive and accessible. As confirmed in our experiments (Table. 4, Sec. 7.5.1), RITA works well in a large range of error bound factors. Hence it is not a parameter that needs careful tuning.

RITA then uses Lemma 5 to determine if merging some given clusters still meets the error bound threshold ϵ.

Lemma 5. *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 4. If there exist m clusters, namely, $cluster_{k_1}, cluster_{k_2}, ..., cluster_{k_m}$, satisfying that:*

$$\max_{x \in cluster_{k_i}} |c_{k_i} - c_{k_j}| + |x - c_{k_i}| \leq d, i, j \in [1, m] \tag{12}$$

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

Proof. Denote the cluster size of $cluster_k$ to be n_k. After mergeing, 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:

$$\begin{aligned}
|x - c'| &\leq |x - c_{k_i}| + |c_{k_i} - c'| \ (Triangle\ inequality) \\
&= |x - c_{k_i}| + |\frac{\sum_{j=1}^{m} n_{k_j}}{\sum_{j=1}^{m} n_{k_j}} c_{k_i} - \frac{\sum_{j=1}^{m} n_{k_j} c_{k_j}}{\sum_{j=1}^{m} n_{k_j}}| \\
&= |x - c_{k_i}| + |\frac{\sum_{j=1}^{m} n_{k_j} (c_{k_i} - c_{k_j})}{\sum_{j=1}^{m} n_{k_j}}| \\
&= \frac{\sum_{j=1}^{m} n_{k_j} (|c_{k_i} - c_{k_j}| + |x - c_{k_i}|)}{\sum_{j=1}^{m} n_{k_j}} \leq \frac{\sum_{j=1}^{m} n_{k_j} d}{\sum_{j=1}^{m} n_{k_j}} = d
\end{aligned} \tag{13}$$

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:

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

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

In this scenario, finding the maximal number of *mergeable* clusters is equivalent to finding the minimal clique cover in the corresponding graph, which is an NP-hard problem [28]. 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:

$$\max_{x \in cluster_i} |c_i - c_j| + |x - c_i| \leq d, \max_{x \in cluster_j} |c_j - c_i| + |x - c_j| \leq \frac{d}{2} \tag{14}$$

$cluster_j$ is marked.

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

The Correctness Proof. In this solution, clusters in S_1 can be regarded as transfer nodes. If (14) 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,

$$\begin{aligned}
&\max_{x \in cluster_{j_1}} |c_{j_1} - c_{j_2}| + |x - c_{j_1}| \\
\leq &\max_{x \in cluster_{j_1}} |c_{j_1} - c_i| + |c_i - c_{j_2}| + |x - c_{j_1}| \\
\leq &\max_{x \in cluster_{j_1}} |c_{j_1} - c_i| + |c_i - c_{j_2}| + |x - c_{j_1}| + |x - c_{j_2}| \leq d
\end{aligned} \tag{15}$$

Thus (12) 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 [49] 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

When the model architecture and hardware are fixed, the batch size depends on the length of the timeseries L and the average group number $\overline{N}$ among all attention modules. Intuitively, given a batch size B and the number of groups N, if we could precisely calculate its GPU memory usage, it would be straightforward to determine the appropriate batch size based on the GPU memory size and L. However, this is infeasible due to the following: (1) RITA's dynamic grouping leading to

varying deep learning computational graphs [1], and (2) the utilization of memory management techniques by the deep learning infrastructure [3].

Therefore, we propose a learning-based method to predict the batch size. It models the correlation between the length of the timeseries L, the number of groups N, and the batch size B based on the actual GPU memory consumed by some sampled batches, which we efficiently measure during the training process.

RITA samples several $(L_i, \overline{N}_i)$ pairs and estimate a proper batch size for each pair. Treating these pairs as ground truth labels, we use function fitting [22] to learn the batch size predicting function $B = f(L, N)$, where B is a function of two variables L and N.

More specifically, given a user-defined timeseries maximal length L_{max}, we randomly sample integral points (L_i, N_i) from plane $\{1 \le L \le L_{max}, 1 \le N \le L\}$. 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).

Next, we discuss how to learn the prediction function using these sampled (L_i, N_i, B_i).

Learning the Prediction Function. We apply *curve fit* from SciPy [60] as the function fitting tool to fit the two-variable function $B_i = f(L_i, N_i)$ on plane $\{1 \le L \le L_{max}, 1 \le N \le L\}$.

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, i.e., Alg. 2. 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.

Algorithm 2 Dynamic Programming for Plane Division

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

The Optimality Proof of the Plane Division Algorithm . We describe Alg. 2 and intuitively show its optimality. We assume that Scipy [60] 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 $\{l_2 \le L \le l_1, N \le n\}$. In Lines 11-13, we enumerate all possible ways of cutting $\{l_2 \le L \le l_1, N \le n\}$ horizontally into two sub-plane $\{l_2 \le L \le l_1, N \le i\}$ and $\{l_2 \le L \le l_1, i \le N \le n\}$ 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 $\{l_2 \le L \le l_1, N \le l_1\}$, which is recorded as f_{l_1,l_2} in Line 14. $dp(l)$ denotes the minimal estimation error for sub-plane $\{L \le l\}$. We enumerate all the possible ways of cutting $\{L \le l\}$ vertically into two sub-plane $\{L \le i\}$ and $\{i \le L \le l\}$ 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.

6 SUPPORTING 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.

6.1 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: $y = Softmax(W_{cls}Z_{[CLS]} + B_{cls})$, 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 as the loss function for the classification task [16].

6.2 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, a data cleaning task.

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:

$$T_o(i,j) = \begin{cases} -1 & (i,j) \in M \\ T_r(i,j) & (i,j) \notin M \end{cases} \tag{16}$$

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., $Y = TransposeCNN(Z_1 \oplus Z_2 \oplus ... \oplus Z_n)$, 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 [58]: $L = \frac{1}{|M|}\sum_{(i,j)\in M}(Y(i,j) - T_r(i,j))^2$.

6.3 Forecasting

Forecasting can be regarded as a special case of imputation, in which all missing values are at the end of timeseries. Similarly to the imputation task, we scale the timeseries to non-negative and use

a special value (-1) to indicate the values to be predicted:

$$T_{observed}(i,j) = \begin{cases} T_{real}(i,j) & i \le t_{observed} \\ -1 & otherwise \end{cases} \tag{17}$$

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.

6.4 Other Unsupervised Tasks

RITA naturally supports other unsupervised tasks, such as similarity search and clustering [29, 36, 37], 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 [26, 27, 44] can be built on the embeddings, i.e., the time series query system we show in Fig. 1 and evaluate in Sec. 7.6.

7 EVALUATION

Our experimental study focuses on the following questions:

1. **Effectiveness and efficiency**: How does RITA compare with other Transformer-based methods and traditional timeseries representation learning methods in accuracy and efficiency?
2. **Similarity Search**: How well does RITA in supporting time series similarity search?
3. **Ablation Study**: How do the key techniques of RITA work?

7.1 Experimental Setup

Datasets. We evaluate RITA on classification, imputation, and similarity search tasks using 6 multi-variate and 3 uni-variate timeseries datasets.

- ***WISDM*** [63] 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 [53] 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 [55] covers 15 subjects performing 8 locomotion-style activities. Each subject wears the sensors for approximately ten minutes. The sampling rate is 50 Hz.
- ***ETT*** [71] dataset comprises 2 years of 2 electrical transformers' data collected from 2 stations, including the oil temperature and six features of the power load. We use ETTm1 where each timeseries lasts 15 minutes. We pre-process and split the data as in prior work [46].
- ***ECG*** dataset [39] 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*** [7] is an 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 univariate datasets derived by selecting 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 to 200 on small datasets (WISDM,

HHAR, RWHAR, ETT), 2000 on medium size dataset (ECG), and 10,000 on the large dataset (MGH). Table 1 shows the statics 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 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
ETT	34,265	11,425	200	7	N/A
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 deep learning-based representation learning methods, including the Transformer based **TST** [70] and the non-Transformer method **TS2VEC** [68]. 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 [59](referred to as **Vanilla**) and two SOTA methods that reduce the complexity of self-attention by approximation in NLP, namely, Performer [11] (referred to as **Performer**) and Linformer [62] (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. Finally, we compare against the SOTA Transformer-based methods for timeseries forecasting, **Triformer** [12] and **PatchTST** [46], wherein their decoders are substituted with output networks for the downstream tasks.

We also compare Group Attn. against **GRAIL** [47], 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 [15] or K-Nearest Neighbor [20] classifier. Note GRAIL only targets **uni-variate** timeseries and cannot support imputation tasks.

Experiment Methodology. We focus on three 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 a few labeled timeseries with/without pretraining on large scales of unlabeled timeseries. 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.

(2) **Imputation**. We run the imputation task on the datasets used in classification as well as the large unlabeled MGH dataset. We 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.

(3) **Similarity Search**. We run similarity search (kNN) task on the ECG dataset, which is the longest labeled dataset. We evaluate both query execution time and precision. The query execution time measures the *inference cost* of each feature embedding method. The query precision measures the percentage of the nearest neighbors that belong to the same class with the querying object, representing the quality of the produced embeddings.

To evaluate the **efficiency** of Group Attn., the total time of forward computation, backward propagation, and grouping are measured for all methods in all the experiments.

We first compare against the Transformer-based methods and TS2VEC on multi-variate datasets (sec. 7.2, 7.3), then compare against the non-deep learning method GRAIL on uni-variate datasets

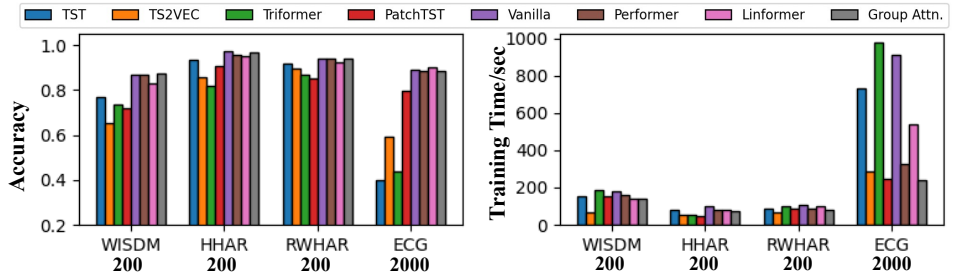

Fig. 4. Full-label classification results (multi-variate data).

Dataset	WISDM		HHAR		RWHAR		ECG	
Pretrain Size	62,231		68,294		63,599		561,358	
Method	Scratch	Pre.	Scratch	Pre.	Scratch	Pre.	Scratch	Pre.
TST [70]	49.13%	50.03%	72.56%	75.30%	69.46%	80.41%	20.98%	27.99%
TS2VEC [68]	54.30%	62.01%	69.03%	82.54%	79.41%	85.01%	39.38%	39.94%
Triformer [12]	34.28%	40.61%	71.51%	72.21%	76.23%	82.73%	22.74%	29.53%
PatchTST [46]	44.63%	50.51%	71.47%	73.12%	60.73%	60.83%	29.82%	32.61%
Vanilla	66.16%	**75.89%**	75.60%	81.35%	85.68%	91.14%	42.05%	46.16%
Performer	66.09%	73.97%	76.52%	80.70%	87.54%	**91.33%**	43.34%	45.58%
Linformer	50.12%	67.44%	65.94%	76.52%	81.03%	86.33%	27.19%	31.34%
Group Attn.	62.56%	75.06%	76.17%	**82.62%**	86.13%	89.63%	42.58%	**46.39%**

Table 2. Pretrain + few-label finetuning results. Metrics: Accuracy. The best results are marked with bold.

(sec. 7.4). Note that TS2VEC[68] doesn't support imputation tasks. It is thus not reported in this set of experiments.

We report the median result among 5 random seeds and data splits as here we observe a low standard error.

Configuration. All models were trained on an NVIDIA Tesla V100 16GB GPU. All methods are optimized with AdamW [42], with the starting learning rate and weight decay parameter set to $1e^{-4}$. 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 [70], we train the model for 50 epochs. The baselines use a *maximal batch* size within GPU's capacity during training: per [21], a larger batch size can expedite training without compromising accuracy. Therefore, with this setting, the baselines are shown with their best possible results.

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 based on our ablation study (Table 4). 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}. For Triformer, we enumerate the settings given by the authors [12] and choose the setting that balances accuracy and efficiency. For PatchTST, similarly, we choose an appropriate patch size among {16,64,256}, per the suggestion of the authors.

7.2 Classification

In this section, we evaluate the effectiveness and efficiency of RITA on classification tasks. We first compare RITA and the baselines by training them with full labels from scratch. We then show how pretraining RITA increases the accuracy on the downstream tasks.

Dataset	WISDM		HHAR		RWHAR		ETT		ECG		MGH	
Length	200		200		200		200		2,000		10,000	
Method	MSE	Time/s	MSE	Time/s	MSE	Time/s	MSE	Time/s	MSE	Time/s	MSE	Time/s
TST [70]	13.30	150.3	1.085	78.2	0.0882	83.9	0.1661	181.8	0.0905	696.3	N/A	N/A
Triformer [12]	11.20	163.2	2.468	87.9	0.4580	97.5	0.0777	197.2	0.0905	977.9	0.00079	2936
PatchTST [46]	5.568	**132.7**	0.7337	**69.6**	0.1330	**78.0**	0.0552	**160.5**	0.0101	235.8	N/A	N/A
Vanilla	**3.240**	178.1	**0.2968**	97.4	**0.0478**	108.1	**0.0530**	215.5	0.0037	857.9	N/A	N/A
Performer	3.449	162.6	0.2980	82.6	0.0489	89.1	0.0532	196.7	**0.0033**	270.2	**0.00014**	356.2
Linformer	3.852	141.9	0.3198	81.1	0.0572	98.4	0.0601	171.6	0.0035	291.38	0.00088	404.9
Group Attn.	3.277	136.7	0.2974	73.3	**0.0478**	81.3	0.0535	165.4	0.0038	**164.36**	0.00042	**54.4**

Table 3. Imputation results (multi-variate data). Metrics: MSE. The best results are marked with bold.

7.2.1 *full-label training (Multi-variate classification)*

Results shown in Figure 4 get us the following observations:

(1) Group Attn.'s advantage over TST. On all four datasets, Group Attn. outperforms TST in both accuracy and training time. In particular, Group Attn. outperforms TST by 49 percentage points (88.48% vs 39.93%) and is 3 times faster in training time per epoch (236.8s vs 731.0s) on the ECG dataset.

Three deficiencies may cause TST's 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. Thirdly, TST uses vanilla self-attention, which causes quadratic complexity.

(2) Group Attn.'s advantage over TS2VEC. Group Attn. is consistently more accurate than TS2VEC across the four datasets. In particular, on the ECG dataset which contains long timeseries – the scenario we focus on, Group Attn. outperforms it in accuracy by 28% (88.48% vs 59.17%), while using less training time per epoch. This can be attributed to our backbone model (Transformer) which effectively captures long-range correlations.

(3) Group Attn.'s advantage over Vanilla. 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.

Group Attn. is more scalable than Vanilla Self-Attention. When the series length is 200 (WISDM, HHAR, RWHAR), Group Attn. requires 25% less training time on average. When the length increases to 2,000 (ECG), Vanilla takes over 15 minutes per epoch, about 4 times slower than Group Attn.. Vanilla fails on the long MGH data when the length reaches 10,000 due to out of GPU memory.

(4) Group Attn.'s advantage over other efficient attention mechanisms. Group Attn. is more accurate than Performer and Linformer on 3 out of 4 datasets. Although Linformer works slightly better than Group Attn. on the ECG dataset (90.37% vs 88.84%), it is the worst in all other cases compared to any other RITA-based methods. In the meantime, Group Attn. is always faster than Performer and Linformer on all 6 multi-variate datasets, thus a **win-win**.

Group Attn. significantly outperforms PatchTST and Triformer in classification accuracy. This is because these two methods are tailored for timeseries forecasting, and the feature embeddings produced by their specialized designs may have adverse effects on other timeseries analytical tasks, as discussed in related work section. Moreover, Group Attn. is 1.3/4.4 times faster than PatchTST/Triformer on long timeseries, e.g., on the ECG dataset, because Group Attn. has more sharing opportunities when processing long timeseries.

7.2.2 *Pretraining + few label finetune (multi-variate classification)*

The results shown in Table 2 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) Group Attn.'s advantage over TST and TS2VEC. On the four classification datasets, our Group Attn. significantly outperforms TST by 15 percentage points on average, and outperforms TS2VEC by 6 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.

Triformer and PatchTST, the two timeseries forecasting methods have an accuracy much (34%) lower than Group Attn., for a similar reason to what we have discussed in Sec. 7.2.1.

7.3 Imputation

In this section, we compare our group attention and the baselines on imputation tasks, a typical data cleaning task in data management. We first show the results on all the 6 datasets in Sec. 7.3.1. We then vary the length of the timeseries on the MGH dataset to show group attention's scalability on long timeseries in Sec. 7.3.2.

7.3.1 Full-dataset training (Multi-variate imputation)

Similar to classification tasks, the results of **imputation tasks** (Table 3) show that Group Attn. consistently outperforms the baselines in training time while achieving comparable/better Mean Square Error (MSE). On the large dataset MGH (length = 10,000), TST, Vanilla and PatchTST fail due to out of memory (OOM) errors. Methods using RITA framework (Group Attn., Performer, Linformer) all achieve very low MSE and thus are highly accurate. Among them Linformer is the worst. The accuracy of Triformer and PatchTST in general is much lower, because they target forecasting task rather than generating high quality feature embeddings to support various tasks.

Here we observe that **the longer the timeseries, the larger the speedup is**. On the medium sized ECG dataset with a length of 2,000, Group Attn. has a speedup of 5.23/1.65/1.77/5.96/1.43 compared to Vanilla/Performer/Linformer/Triformer/PatchTST. When the length increases to 10,000, the speedup on the MGH dataset increases to 6.59/7.48/54.37 compared to Performer/Linformer/Triformer (Vanilla,TST and PatchTST failed due to out of memory) on imputation task (Table. 3). This is because when the length of the timeseries gets longer, Group Attn. gets more opportunities to find windows with similar properties. Note Triformer is very slow on long series (length = 10,000), because it involves a lot of sequential computation and thus cannot benefit from GPUs.

Even on the short WISDM, HHAR, RWHAR, and ETT datasets, Group Attn. still consistently outperforms all other methods except PatchTST. This confirms that it fully leverages the similarity among the timeseries windows, while not introducing much overhead. Although PatchTST has comparable speed with Group Attn. on short series, it is much slower on the longer ECG dataset (length = 2000) and fails on the very long EEG data (length = 10,000). This is because it concatenates many embeddings to form a very long embedding before making prediction, thus consuming much more GPU memory. This shows PatchTST does not scale to long series.

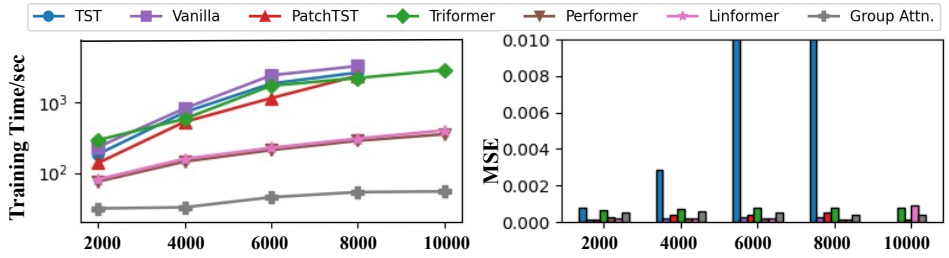

Fig. 5. Varying the lengths of timeseries.

7.3.2 *Training time: Varying the Length*

In this experiment, we truncate the original MGH timseries 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. 5 again show that *the longer the timeseries, the larger the speed up.* With comparable or better MSE, Group Attn. outperforms Vanilla/TST/PatchTST/Triformer/Performer/Linformer by 63/49/45.9/54.3/6.5/7.4x. Although PatchTST is fast on short series (length = 200), its training time increases very fast as the length increases and fails eventually when reaching 10,000. 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 timeseires becomes longer, there are more grouping opportunities because of the similarity of the timeseries segments.

7.4 Comparison to Non-deep Learning Methods

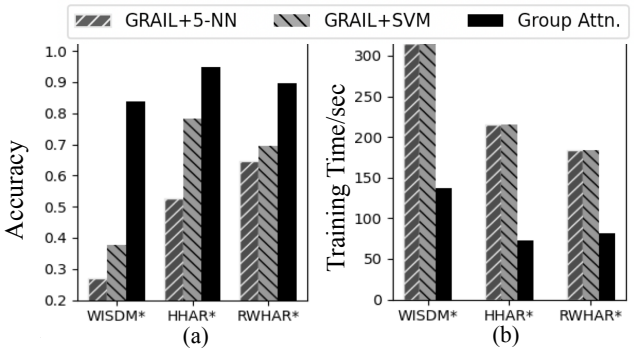

Fig. 6. Comparison to non-deep learning method (uni-variate data).

We compare against GRAIL, the SOTA of non-deep learning timeseries representation learning. We use the three uni-variate datasets, because GRAIL only targets uni-variate timeseries. Fig. 6 demonstrates 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, the GPU-friendly design of RITA makes it at least 2× faster than GRAIL in training time.

7.5 Ablation Study

7.5.1 *Adaptive Scheduler*

To evaluate the effectiveness of RITA's adaptive scheduler (denoted as **Dynamic**) (Sec. 5), we compare it against (1) baseline **Fixed**: a fixed group number N; (2) baseline **Heuristic**: if the validation loss gets lower than ever after the current epoch, we consider that the current N is sufficiently large for the current training stage, and set N_{new} to $N * Decay\_rate$ for the next epoch. Note $N * Decay\_rate$ is smaller than N, because we observe that the number of groups typically

should decrease gradually. We vary N for **Fixed**, $Decay\_rate$ for **Heuristic**, 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 the baselines. Training with Adaptive Scheduler achieves better or comparable performance compared to the best performing N or $Decay\_rate$. For $Fixed$, 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. For $Heuristic$, we have similar observations: on ECG dataset, dynamic scheduler consistently achieves better accuracy than $Heuristic$. On MGH dataset, dynamic scheduler achieves comparable MSE with less training time. Note that manually finding the best N or $Decay\_rate$ that balances the accuracy and running time requires careful tuning, while our adaptive scheduler does not need any 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 manually find an appropriate N or $Decay\_rate$ 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
		Heuristic	0.8	85.41%	240.36
			0.9	86.28%	253.57
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
		Heuristic	0.8	0.00041	102.0
			0.9	0.00040	104.0

Table 4. Adaptive Scheduling vs Baseline (Fixed N/Heuristics)

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.

7.5.2 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

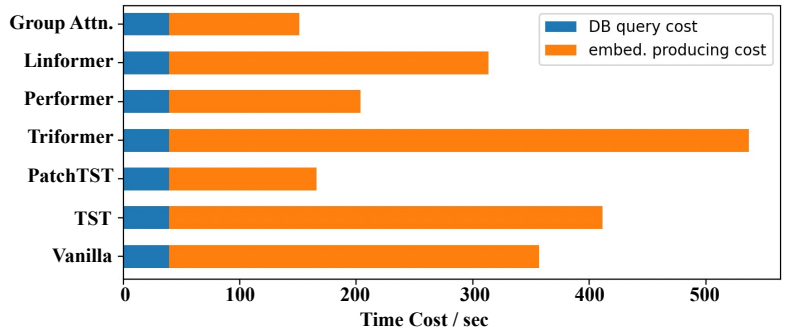

Fig. 7. Query Execution time for 10,000 10-NN queries. Precision: Vanilla (88.34%), Group Attn.(88.29%), Linformer(88.07%), Performer(88.23%/), Triformer(42.45%), PatchTST(80.84%), TST(40.72%).

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%).

7.6 Similarity Search

We develop a time-series query system to conduct this similarity search experiment. We first use RITA to extract feature embeddings from the training set of the ECG dataset and store them in a vector DB. We then use the timeseries in validation set as queries to find their kNN from the vector DB. Given a query, we use RITA to extract its feature embedding at online query time.

In our experiment, we use Postgres [13] as the vector DB, as it features the HNSW index to speed up the high dimensional similarity search. We set the k as 10 in kNN search and run 10,000 queries. The results, as depicted in Figure 7, show that RITA is faster than all baselines by 60%, indicating that RITA outperforms the baselines on *inference time* as well. The precision of RITA is on par to that of Vanilla Transformer. Note the precision of Vanillar Transform is considered to be the upper bound of the precision of all Transformer-based methods, because it uses the original expensive attention mechanism without any approximation. RITA outperforms all other baselines in precision, while faster than them.

8 RELATED WORK

Method	TST	Vanilla	Triformer	PatchTST	Performer	Linformer	Group Attn.
Time Complexity	$O(n^2 d)$	$O(n^2 d)$	$O(nd^2 \frac{m}{T})$	$O(nd^2 \frac{m}{S})$	$O(nd(d+P))$	$O(nd(d+L))$	$O(nd(d+N))$
Space Complexity	$O(n^2)$	$O(n^2)$	$O(nd \frac{m}{T})$	$O(nd \frac{m}{S})$	$O(n(d+P))$	$O(n(d+L))$	$O(n(d+N))$

Table 6. The time/space complexity of Transformer-based methods. n: timeseries length; d: embedding dimension; m: number of channels; P,L,N,S,T: method-specific parameters

8.1 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 [52], HIVE-COTE [38], ROCKET [18] 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.

In particular, as the SOTA of timeseries **representation learning**, GRAIL [47] extracts landmarks from data and computes the representations with the combination of the landmarks. However, GRAIL only supports uni-variate timeseries. Our experiments (Sec. 7.4) show that RITA outperforms GRAIL in both effectiveness and efficiency on uni-variate timeseries.

CNN/RNN-based Deep Learning Methods. CNN-based methods, such as InceptionTime [25] and Resnet [23], are good at classification tasks, but can not handle generative tasks such as forecasting because of the inductive bias of convolution networks. TS2VEC [68] is the SOTA non-Transformer representation learning method which uses CNN architecture. As confirmed in experiments, RITA consistently outperform it. RNN-based methods, such as Brit [8] and deepAR [51], support 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 [48] 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.

In timeseries forecasting, LogTrans [35] introduced a log sparsity assumption to attention computation. Informer [71] pushes LogTrans a step further and scales forecasting to multi-variate timeseries. Autoformer [65] performs forecasting by decomposing timeseries into the trend part and the seasonal part. FEDformer [72] proposed a Frequency Enhanced Attention with Fourier Transform. Pyraformer [40] and Triformer [12] used hierarchical attention structures. PatchTST [46] introduced a patching pattern in attention. In our experiments, among this big family, we choose Triformer and PatchTST as our baselines, because they experimentally outperform the others and are considered as SOTA in this field.

Because these methods are tailored for timeseries forecasting, the feature embeddings produced by their specialized designs may have adverse effects on other timeseries analytical tasks. For instance, Triformer incorporates sequence compression operations, which can be detrimental for imputation due to the potential loss of vital information on the entire timeseries. Moreover, Triformer employs variable-specific modeling, computing embeddings for each input channel separately. While this approach may be beneficial for forecasting tasks, it may not be suitable for classification, as the latter necessitates the consideration of information from all channels simultaneously.

For imputation tasks, CDSA [43] outperforms statistical methods and the SOTA RNN-based method Brit [8]. For timeseries classification, AutoTransformer [50] performs architecture search to adapt to tasks in different domains. For timeseries anomaly detection, Anomaly Transformer [66] outperforms many widely used methods such as OmniAnomaly [54], assuming the attention score maps show Gaussian distribution.

All of these works are designed for specific tasks, rather than a **representation learning** framework to serve different downstream tasks. To fill this gap, some researchers proposed a Transformer-based architecture, called TST [70]. 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. 7).

8.2 Efficient Transformers

The need to improve the scalability of Transformers has led to more efficient variations, especially for accommodating long text data in NLP [56]. A first step was to introduce fixed/random patterns to the self-attention mechanism. Sparse Transformer [10] and Longformer [4] only compute attention at fixed intervals. ETC [2] and BigBird [69] use global-local attention: the computation is limited within a fixed radius, while some auxiliary tokens are added to attend/get attended globally.

However, fixed attention patterns heavily depends on users to give an optimal setting.

Reformer [31] 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 [62] 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 [11] uses linear functions to approximate the kernel function *softmax*, making attention computation commutative. 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. 7, our group attention significantly outperforms them in both accuracy and efficiency (training time), because group attention fully leverages the periodicity of timeseries.

In Table 6, we summarize the time/space complexity of the works that target scale Transformers to long sequence. Note although several methods claim linear time/space complexity, their complexities all involve a method-specific constant, similar to the number of groups N in our group attention.

In the long time series scenarios, when $d \in [64, 128, 256]$ and $n \gg d \approx$ (common settings) where d denotes the dimension of feature embeddings and n represents the length of the timeseries, the theoretical complexities of PatchTST, Triformer, Linformer, and Performer are at the same magnitude to group attention.

However, our empirical experiments show that Group Attention consistently and significantly outperforms these methods in terms of efficiency, as shown in Fig. 4, Fig. 5, Table 3. This is because our group attention is well-suited to long time series. The longer the timeseries is, the more opportunity group attention has to find the similar segments and group them together. This leads to a slower growth rate in the number of groups 'N' – the constant in the time/space complexities of group attention.

9 CONCLUSION

In this work, we presented RITA, an automatic, self-supervised, and scalable timeseries embedding tool. RITA effectively adapts Transformer, popular in NLP, to embed timeseries segments into feature vectors. 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 existing attention mechanisms by 63X with a better accuracy.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants III-1910108 and DBI-2327954.

REFERENCES

[1] Martín 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] Lu Bai, Weixing Ji, Qinyuan Li, Xilai Yao, Wei Xin, and Wanyi Zhu. 2022. DNNAbacus: Toward Accurate Computational Cost Prediction for Deep Neural Networks. *arXiv preprint arXiv:2205.12095* (2022).

[4] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[5] 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.

[6] 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.

[7] 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.

[8] 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).

[9] Chris Chatfield. 2000. *Time-series forecasting*. Chapman and Hall/CRC.

[10] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[11] Krzysztof Choromanski, Valerii Likhosherstov, 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).

[12] Razvan-Gabriel Cirstea, Chenjuan Guo, Bin Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. 2022. Triformer: Triangular, Variable-Specific Attentions for Long Sequence Multivariate Time Series Forecasting–Full Version. *ICJAI* (2022).

[13] PostGIS Project Steering Committee et al. 2018. PostGIS, spatial and geographic objects for postgreSQL. https://postgis.net

[14] Andrew A Cook, Göksel Mısırlı, and Zhong Fan. 2019. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal* 7, 7 (2019), 6481–6494.

[15] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[16] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)* 20, 2 (1958), 215–232.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017). arXiv:1706.02677 http://arxiv.org/abs/1706.02677

[22] Philip George Guest and Philip George Guest. 2012. *Numerical methods of curve fitting*. Cambridge University Press.

[23] 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.

[24] 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.

[25] 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.
[26] 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.
[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[28] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*. Springer, 85–103.
[29] 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.
[30] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[31] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451* (2020).
[32] John Kraft and Arthur Kraft. 1977. Determinants of common stock prices: A time series analysis. *The journal of finance* 32, 2 (1977), 417–425.
[33] 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
[34] 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.
[35] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu 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).
[36] T Warren Liao. 2005. Clustering of time series data—a survey. *Pattern recognition* 38, 11 (2005), 1857–1874.
[37] Rake& Agrawal King-lp 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.
[38] 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.
[39] 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.
[40] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X Liu, and Schahram Dustdar. 2021. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*.
[41] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
[42] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).
[43] 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).
[44] 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.
[45] 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.
[46] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2022. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *The Eleventh International Conference on Learning Representations*.
[47] John Paparrizos and Michael J Franklin. 2019. Grail: efficient time-series representation learning. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1762–1777.
[48] 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.
[49] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.
[50] 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.

[51] 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.

[52] 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.

[53] 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 mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*. 127–140.

[54] 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.

[55] 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.

[56] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)* (2020).

[57] Mingyan Teng. 2010. Anomaly detection on time series. In *2010 IEEE International Conference on Progress in Informatics and Computing*, Vol. 1. IEEE, 603–608.

[58] Patrick A Thompson. 1990. An MSE statistic for comparing forecast accuracy across series. *International Journal of Forecasting* 6, 2 (1990), 219–227.

[59] 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.

[60] 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

[61] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[62] 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).

[63] 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.

[64] 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

[65] 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.

[66] 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).

[67] 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.

[68] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8980–8987.

[69] 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.

[70] 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.

[71] 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*.

[72] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. 2022. FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting. *arXiv preprint arXiv:2201.12740* (2022).

Received July 2024; revised October 2024; accepted November 2024