An Efficient Content-based Time Series Retrieval System

Chin-Chia Michael Yeh
Huiyuan Chen
Xin Dai
Visa Research
California, USA

Yan Zheng
Junpeng Wang
Visa Research
California, USA

Vivian Lai
Yujie Fan
Visa Research
California, USA

Audrey Der
University of California, Riverside
California, USA

Zhongfang Zhuang
Liang Wang
Wei Zhang
Visa Research
California, USA

Jeff M. Phillips
University of Utah
Utah, USA

ABSTRACT

A *Content-based Time Series Retrieval* (CTSR) system is an information retrieval system for users to interact with time series emerged from multiple domains, such as finance, healthcare, and manufacturing. For example, users seeking to learn more about the source of a time series can submit the time series as a query to the CTSR system and retrieve a list of relevant time series with associated metadata. By analyzing the retrieved metadata, users can gather more information about the source of the time series. Because the CTSR system is required to work with time series data from diverse domains, it needs a high-capacity model to effectively measure the similarity between different time series. On top of that, the model within the CTSR system has to compute the similarity scores in an efficient manner as the users interact with the system in real-time. In this paper, we propose an effective and efficient CTSR model that outperforms alternative models, while still providing reasonable inference runtimes. To demonstrate the capability of the proposed method in solving business problems, we compare it against alternative models using our in-house transaction data. Our findings reveal that the proposed model is the most suitable solution compared to others for our transaction data problem.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Data mining**; • **Computing methodologies** → **Neural networks**; • **Applied computing** → **Electronic funds transfer**.

KEYWORDS

time series, information retrieval, neural network, fintech

ACM Reference Format:
Chin-Chia Michael Yeh, Huiyuan Chen, Xin Dai, Yan Zheng, Junpeng Wang, Vivian Lai, Yujie Fan, Audrey Der, Zhongfang Zhuang, Liang Wang, Wei Zhang, and Jeff M. Phillips. 2023. An Efficient Content-based Time Series Retrieval System. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3583780.3614655

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614655

1 INTRODUCTION

Time series is a common data type analyzed for a variety of applications [9]. For example, time series from different sensors on manufacturing machines are examined by engineers for identifying ways to improve factories' efficiency, various biometric time series are studied by doctors for medical research, and multiple streams of time series from operating payment network are monitored for unusual activities. As a large volume of time series data are becoming available from various sources, it is essential to develop an effective system to help users browse time series databases. In this paper, we refer to such a system as a *Content-based Time Series Retrieval* (CTSR) system. This system is designed to retrieve relevant time series from a database when given a query time series.

To understand what a CTSR system is and how it can help users, let us consider the use case presented in Fig. 1. Suppose a user comes across a time series without any associated meta data. The time series could be a power consumption time series or data records from other sensors. The user wants to identify the possible source of the time series and recover the missing information. To solve this problem, the user queries the CTSR system[1] with the time series, and the system returns a ranked list of similar time series with associated meta data. In this example, five out of the top six returned time series are power consumption signatures for microwave ovens. Consequently, the user is able to infer that the unknown time series is most likely a microwave oven's power consumption signature. Hence, the CTSR system helps the user in recovering the missing information about time series.

We have two primary design goals when building our CTSR system: 1) to effectively capture various concepts in time series from different domains, and 2) to be efficient during inference, given the real-time interactions of users with the system. To identify the most suitable distance or similarity function for our system, we conducted a benchmark experiment using two distance functions and five neural network models (see Section 3). From the seven

[1]The query time series may or may not exist in the database of the CTSR system.

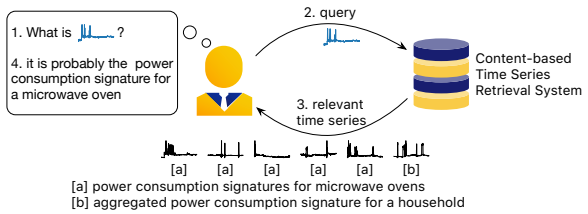

Figure 1: The use case for a *Content-based Time Series Retrieval* (CTSR) system where the database consists of time series from multiple domains.

tested methods, the *Residual Network 2D* (RN2D) outperforms the other methods with statistical significance. However, the RN2D method failed to meet the second design goal. Its average query time was 32 seconds, while most of the other rival methods' query times were below 0.04 seconds (refer to Table 1). In other words, while the RN2D method proved to be the best model in terms of accuracy, it may not be ideal for building our system if we aim to guarantee a reasonable response latency.

The main reason for the drastic difference in inference time is the difference in the role of the neural network. In faster methods (i.e., the methods with query time less than 0.04 seconds), the neural network serves purely as a *feature extractor*, and the distance is computed using Euclidean distance (see Fig. 2.a). Therefore, we only need to project each time series in the database to Euclidean space once using the neural network before inference. During query time, the neural network only needs to project the query time series to the same Euclidean space, and the distance computation can be efficiently performed in this space. On the other hand, the RN2D model serves as both the *feature extractor* and *distance function* as shown in Fig. 2.b. Thus, the RN2D model is invoked each time the distance is computed. In cases where there are n time series in the database, the faster method only requires one invocation of the neural network for the query. In contrast, the RN2D model invokes n times, drastically increasing the runtime.

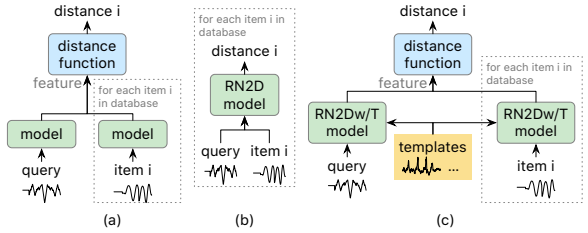

Figure 2: The design of the neural network can greatly impact the inference time.

To leverage the advantages of the best available model (RN2D) for our CTSR system, we propose a novel model architecture based on the RN2D model with improved efficiency, called *Residual Network 2D with Template Learning* (RN2Dw/T). As illustrated in Fig. 2.c, we incorporate a template (landmark) learning mechanism into the input of the RN2D model (Fig. 2.b) and modify the model to output feature vectors instead of distance values. Unlike the RN2D method, the RN2Dw/T model functions solely as a *feature extractor*. Our proposed RN2Dw/T achieves comparable effectiveness to the RN2D method while maintaining an average query time of less than 0.04 seconds (see Table 1). Our results provide strong evidence that an effective and efficient CTSR system with the proposed RN2Dw/T model can be a valuable tool for businesses in various industries.

2 BACKGROUND

In this section, we will begin by providing the problem statement, and then we will review relevant works from the literature.

2.1 Problem Statement

The CTSR problem is formulated as follows:

Problem 1. Given a set of time series $\mathcal{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$ and any query time series $\mathbf{q}$, we want to obtain a relevance score function $f(\cdot,\cdot)$, which satisfies the property that $f(\mathbf{x}_i, \mathbf{q}) > f(\mathbf{x}_j, \mathbf{q})$ if $\mathbf{x}_i$ is more relevant to $\mathbf{q}$ than $\mathbf{x}_j$.

The scoring function can be either a predefined similarity function or a trainable function that is optimized using the metadata associated with each time series in $\mathcal{X}$.

2.2 Related Work

There are two major ways to formulate the time series retrieval problem [13, 24, 28, 32, 46]. The first is known as the time series similarity search problem, which focuses on finding the top k time series that are most similar to a given query based on a fixed distance function [3, 24, 25, 28, 46]. For efficiency, techniques such as lower bounding [28], early abandoning [28], and/or indexing [25] are commonly used. This type of research differs from our goal (i.e., Problem 1).

The second type of problem formulation is more aligned with our problem statement, where the objective is to develop a model or scoring function to aid users in retrieving relevant time series from a database based on the query time series submitted. We searched for relevant literature and found only one paper [32]. Their proposed model was intended for multivariate time series analysis. If we were to adapt it for our univariate time series problem, the resulting model would essentially be a standard long short-term memory network [15]. Consequently, we expanded our literature review further to identify effective distance measures and models for time series data, regardless of their specific application.

The Euclidean distance and dynamic time warping distance are popular tools for analyzing time series data. They are widely used in various tasks such as similarity search, classification, and anomaly detection [9, 13, 22, 28, 44]. We can readily apply both distance functions to our problem, and therefore, we include them in our experiments. Another family of methods that can be used in our problem is neural networks. For example, long short-term memory networks [15, 17, 18, 20, 45], gated recurrent unit networks [7, 20, 45], transformers [5, 19, 20, 35, 45], and convolutional neural networks [16, 29, 39] have shown effectiveness in tasks such as time series classification, forecasting, and anomaly detection. Thus, we also include these neural network models in our experiments.

3 METHOD

In this section, we begin by presenting six common baseline methods. Following that, we introduce the Residual Network 2D (RN2D) method and contrast its benefits with the other baseline methods.

Once we have introduced the RN2D method, we will present the proposed Residual Network $2D$ with Template learning (RN2Dw/T) method, which solves the efficiency issue associated with the design of RN2D. The six common baseline methods are:

- **Euclidean Distance (ED)**: We compute the Euclidean distance between the query time series and the time series in the collection. Then, we sort the collection based on the distances. This is the simplest approach for solving the CTSR problem.
- **Dynamic Time Warping (DTW)**: This method is similar to the ED baseline, but uses the DTW distance instead. The DTW distance is considered as a simple yet effective baseline for time series classification problems [2, 9, 28]. Therefore, it is crucial to include this method in our CTSR benchmark.
- **Long Short-Term Memory network (LSTM)**: The LSTM is one of the most popular Recurrent Neural Networks (RNNs) used for modeling sequential data [15, 20, 45]. In this work, we optimize LSTM models using the Siamese network architecture [6] (see Fig. 2.a).
- **Gated Recurrent Unit network (GRU)**: The GRU is another popular RNN architecture widely used for modeling sequential data [7, 20, 45]. To optimize the GRU model, we applied a similar approach as for the LSTM model, wherein we replaced the LSTM cells in the RNN architecture with GRU cells.
- **Transformer (TF)**: The transformer is an alternative to the RNNs for sequence modeling [5, 19, 20, 35, 45]. To learn the hidden representation for the input time series, we utilized the transformer encoder proposed in [35]. We replaced the RNNs used in the previous two methods (i.e., LSTM and GRU) with transformer encoders, resulting in a transformer-based Siamese network architecture instead of an RNN-based one.
- **Residual Network 1D (RN1D)**: The RN1D is a time series classification model inspired by the success of residual networks in computer vision [14, 39]. It employs $1D$ convolutional layers instead of $2D$ convolutional layers [14, 39] and is first proposed in [39]. Extensive evaluations conducted by [16] have demonstrated that the RN1D design is among the strongest models for time series classification. We once again optimize this model in a Siamese network fashion [6] (see Fig. 2.a).

Both the ED and DTW methods require no training phase as they have no parameters to optimize. The DTW method is the more effective method of the two for time series data, because it considers all alignments between the input time series [28]. The computation of DTW distance can be abstracted into a two-stage process. In the first stage, a pair-wise distance matrix $D \in \mathbb{R}^{w \times h}$ is computed from the input time series $\mathbf{a} = [a_1, \cdots, a_w]$ (where w is the length of $\mathbf{a}$) and $\mathbf{b} = [b_1, \cdots, b_h]$ (where h is the length of $\mathbf{b}$) as $D[i, j] = |a_i - b_j|$. In the second stage, a fixed recursion function is applied to D, i.e., $D[i, j] \leftarrow D[i, j] + \textsc{min}(D[i-1, j], D[i, j-1], D[i-1, j-1])$, for each element in D. Consequently, the DTW method can be viewed as running a predefined function on the pair-wise distance matrix between the input time series.

The remaining four baseline methods use the Siamese network distance learning framework (refer to Fig. 2.a) and employ high-capacity[2] neural network models, i.e., LSTM, GRU, TF, and RN1D, to learn the hidden representation of the input time series. These representations are used to compute the distance between two time series, and the models are learned using the optimization procedure detailed in Section 3.3. Once the model is optimized, the hidden representations of each time series in the database are extracted before deployment. When a user submits a query, we only need to apply the model to the query time series to extract its hidden representation because the hidden representations of each time series in the database are already extracted before query time. Then, the distances between the query and each item in the database are computed using the Euclidean distance.

3.1 Residual Network 2D

The Residual Network 2D (RN2D) method (Fig. 3) combines the best of both worlds (i.e., DTW and neural network). The RN2D method takes advantage of the rich alignment information from the pair-wise distance matrix, similar to the DTW method. However, instead of using a fixed function, the RN2D method uses a high-capacity neural network as the function, making use of an expressive model like the four neural network baselines. Fig. 3.a shows the building block, while Fig. 3.b shows the overall model. Please see [34] for a detailed description of each module in Fig. 3.

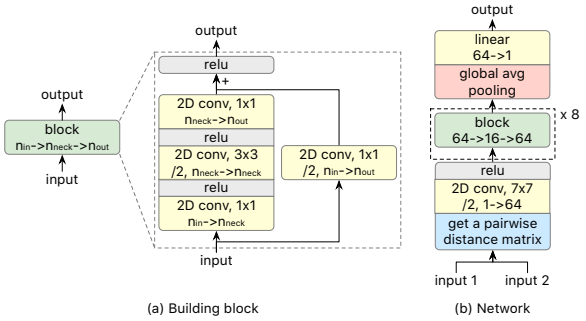

Figure 3: The building block and network designs for the Residual Network $2D$ (RN2D) model are shown in Fig. 3.a and Fig. 3.b, respectively.

As shown in Fig. 2.b, we cannot extract the hidden representations of each time series in the database before deployment when using RN2D, unlike the methods using the Siamese network framework. If there are n time series in the database, we need to run RN2D n times during query time to compute the distance between the query time series and each time series in the database. In contrast, the methods using the Siamese network framework only need to run the model once during query time, making the RN2D method an order of magnitude slower[3]. To address the efficiency issue of RN2D, we propose the Residual Network $2D$ with Template Learning method, which we will introduce in the next section.

3.2 Residual Network 2D with Template Learning

We propose the Residual Network $2D$ with Template Learning (RN2Dw/T) to address the efficiency issue of RN2D. This method is designed to be as effective as the RN2D method while being

[2]We use the term *capacity* to describe the *expressiveness* [23] of a model.

[3]The claim is based on our observation of the experimental results presented in Table 1.

an order of magnitude faster. Fig. 4 illustrates the design of the RN2Dw/T model.

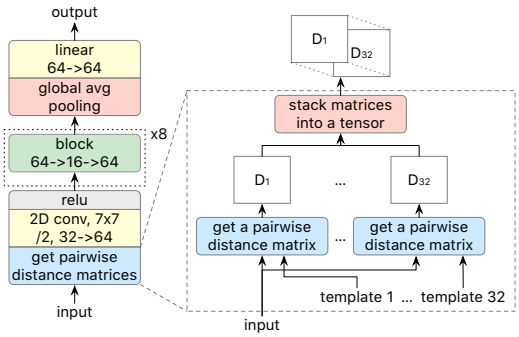

Figure 4: The proposed Residual Network $2D$ with Templates Learning (RN2Dw/T) model.

The proposed RN2Dw/T model begins by computing a set of pairwise distance matrices between the input time series and a set of templates. These distance matrices are computed as follows: Given an input time series $\mathbf{a} = [a_1, \cdots, a_w]$ and the kth template $\mathbf{t}_k = [t_{k,1}, \cdots, t_{k,w}]$, the kth distance matrix $D_k \in \mathbb{R}^{w \times h}$ is computed with $D_k[i, j] = |a_i - t_{k,j}|$. We compute the distance matrix for each of the 32 templates, resulting in 32 $w \times h$ matrices. The 32 templates are learned during the training phase as part of model parameters and are reference time series that help the model project the input time series to Euclidean space using the RN2D design. Next, we stack the 32 $w \times h$ matrices together to create a tensor, $D \in \mathbb{R}^{w \times h \times 32}$. This tensor is the output of the pairwise distance matrix computation step.

Similar to the RN2D model (as shown in Fig. 3.a), we employ a 7×7 convolutional layer with a step size of two to project the tensor D onto $\mathbb{R}^{w/2 \times h/2 \times 64}$ space. After applying a ReLU layer, the resulting intermediate representation passes through eight building blocks with the 64⟶16⟶64 configuration. Next, we apply a global average pooling layer to reduce the spatial dimensions, resulting in a size-64 vector. Finally, we use a linear layer to further process this vector into a size-64 vector, which serves as the final representation of the input time series.

For RN2Dw/T model, the feature of the input time series is computed based on the pairwise distance matrices between the input time series and the learned templates. This allows us to use the trained RN2Dw/T model to extract the feature vectors for all the items in the database before deployment, and we only need to do this feature extraction step once. After deployment, when a query time series is obtained, we only need to extract its feature vector, after which we can efficiently compute the Euclidean distance between the query feature vector and the pre-computed feature vectors from the database. However, if the RN2D model is used in the system, we would need to reprocess all items in the database whenever we receive a new query, which would considerably increase the query time.

The proposed RN2Dw/T method bears some resemblance to the Nyström approximation method for kernel learning [12, 40], in that both involve using a subset of samples to replace the training dataset. This similarity highlights how RN2Dw/T can achieve similar performance to RN2D, just as the Nyström method can approximate the exact kernel. The RN2Dw/T model has a similar capacity to the RN2D model, but it also enables a more efficient query mechanism, which is crucial when designing a real-world CTSR system.

3.3 Optimization

The loss function used in our study is the BPR loss [31], which is appropriate for the CTSR problem since it is a *learning-to-rank* problem. Given a batch of training data $\mathcal{B} = [\mathcal{T}_0, \cdots, \mathcal{T}_m]$, the loss function is defined as: $\sum_{(\mathbf{t}_i, \mathbf{t}_{i,+}, \mathbf{t}_{i,-}) \in \mathcal{B}} -\log \sigma(f_\theta(\mathbf{t}_i, \mathbf{t}_{i,+}) - f_\theta(\mathbf{t}_i, \mathbf{t}_{i,-}))$, where m is the batch size, $\sigma(\cdot)$ is the sigmoid function, and $f_\theta(\cdot, \cdot)$ is the model. Each sample in the batch is a tuple $\mathcal{T}_i$ that contains the query (or anchor) time series $\mathbf{t}_i$, the positive time series $\mathbf{t}_{i,+}$, and the negative time series $\mathbf{t}_{i,-}$. We used the AdamW optimizer [21] to train our models.

4 EXPERIMENT

In this section, we present the results of our experiments on a CTSR benchmark dataset created from the UCR Archive [9] and an in-house transaction dataset based on a real business problem (see Fig. 9). Neural network-based methods were implemented using PyTorch [27], and the model with the best average NDCG@10 score on the validation data was selected for testing. We used SciPy [36] to compute ED and Tslearn [33] to compute DTW. Further details including source code and hyper-parameter settings can be found in the project website [34].

4.1 UCR Archive

We created the CTSR benchmark dataset from the UCR Archive [9], which is a collection of 128 time series classification datasets from various domains such as motion, power demand, and traffic. The UCR Archive is widely used for benchmarking time series classification algorithms [2, 16, 39]. We convert the UCR Archive to a CTSR benchmark dataset using the steps listed on the project website [34]. The resulting dataset consists of 136,377 training time series, 17,005 test queries, and 17,005 validation queries.

To measure the performance of different retrieval methods, we computed common information retrieval metrics, including precision at k (Prec@k), average precision at k (AP@k), and normalized discounted cumulative gain at k (NDCG@k), for each query. The performance measurements at $k = 10$ for each of the 17,005 test queries are averaged and presented in Table 1. When comparing different methods, we also conducted two-sample t-tests with $\alpha = 0.05$ to test for statistical significance. Please refer to [34] for the complete results of the significance tests. The reported query time is the average time taken to compute relevant scores between a query and the 136,377 time series in the training dataset. We computed the average query time by using 1,000 different time series from the test data as the query. Table 1 allows for easy comparison of different methods based on different performance measures.

Firstly, we focus our discussion on the three performance measurements: PREC@10, AP@10, and NDCG@10. When comparing the performance of the two non-neural network baselines (ED and DTW), we observed that DTW significantly outperforms ED

Table 1: The proposed RN2Dw/T method is both effective and efficient. The query times are measured in seconds.

Method	PREC@10	AP@10	NDCG@10	Query Time
ED	0.7316	0.7655	0.7499	0.0353
DTW	0.8562	0.8792	0.8693	36.1224
LSTM	0.9205	0.9277	0.9260	0.0169
GRU	0.9221	0.9285	0.9273	0.0084
TF	0.9146	0.9212	0.9199	0.0029
RN1D	0.9086	0.9164	0.9146	0.0366
RN2D	0.9266	0.9342	0.9325	32.0752
RN2Dw/T	**0.9286**	**0.9343**	**0.9336**	0.0181

in all three performance measurements. This suggests that using alignment information helps with the CTSR problem, and similar conclusions have been drawn for the time series classification problem [2].

When considering the first four neural network baselines (i.e., LSTM, GRU, TF, and RN1D), all of them significantly outperform the DTW method. This demonstrates that using a high-capacity model helps with the CTSR problem. One possible reason for this is that the CTSR dataset consists of time series from many different domains [9], and higher capacity models are required for learning diverse patterns within the data. Among the four methods, GRU and LSTM outperform the other two methods significantly in all three performance measurements; GRU performs slightly better compared to LSTM, but the difference is not statistically significant.

The RN2D method, a high-capacity model utilizing alignment information, significantly outperforms all other methods according to the t-test results. When comparing the RN2Dw/T method with the RN2D method, the RN2Dw/T method achieves higher performance in all three performance measurements, although the difference is not significant. Thus, both the proposed RN2Dw/T method and the RN2D method can be considered the best performing methods for the CTSR dataset in terms of the three performance measurements.

When considering the query time, the eight tested methods can be grouped into two categories: slower methods (i.e., DTW and RN2D) with a query time of over 30 seconds, and faster methods (i.e., ED, LSTM, GRU, TF, RN1D, and RN2Dw/T) where each query takes less than 0.1 seconds. The main difference between the faster and slower groups is that all fast methods compute the relevance score in Euclidean space, while the slower methods compute the scores in non-Euclidean spaces. Overall, the proposed RN2Dw/T method is the best method as it is effective in retrieving relevant time series and efficient in terms of query time.

Following many prior works in time series classification [2, 16], we constructed a critical difference (CD) diagram to compare the performance of different methods. The CD diagram (see Fig. 5) shows the average rank of each method based on a performance measurement and indicates whether two methods exhibit a significant difference in performance based on the Wilcoxon signed-rank test ($\alpha = 0.05$). The conclusion is consistent with the findings presented in Table 1.

In Fig. 6, we present the results of the performance measurements using various values of k ranging from 5 to 15. This is done to ensure that the conclusions drawn from Table 1 and Fig. 5 are not limited to our particular choice of k. To improve readability, we have omitted ED and DTW from the figures since their performance is much worse than the other methods.

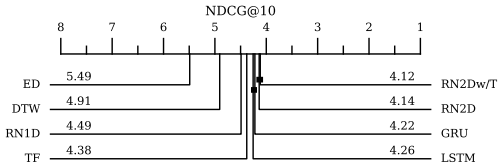

Figure 5: We also construct the CD diagram for PREC@10 and PR@10, the conclusion remains the same.

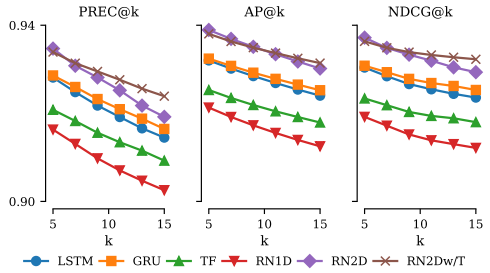

Figure 6: The proposed method outperforms the others with different settings of k for each performance measure.

As shown in Fig. 6, the proposed RN2Dw/T method achieves the best performance across different values of k for all three performance measurements. The order of the remaining methods from best to worst is: RN2Dw/T, RN2D, GRU, LSTM, TF, and RN1D. These results are consistent with the findings presented in Table 1 and Fig. 5.

The proposed RN2Dw/T method introduces an additional hyper-parameter: the number of templates used to compute the pairwise distance matrices (see Fig. 4). We experimented with various settings, including 8, 16, 24, 32, 40, and 48, and the results are presented in Fig. 7. The performance of RN2Dw/T is similar to that of the RN2D method, regardless of the hyper-parameter setting. One potential reason for the lack of sensitivity of RN2Dw/T to this hyper-parameter setting is that the performance of the learned representation is not influenced by the specific shape of templates. A similar phenomenon can be observed in various shapelet-based methods [10, 30, 41]. We chose to set the number of templates to 32 based on its performance on the validation set.

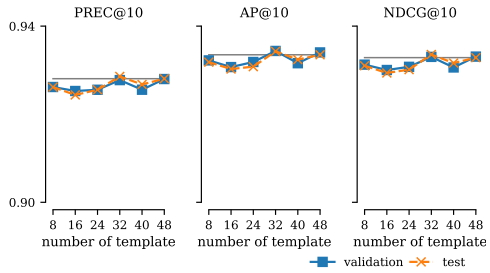

Figure 7: The proposed RN2Dw/T method is not sensitive to the number of template settings. The gray horizontal line marks the test performance of the RN2D method.

Next, in Fig. 8, we present the top eight retrieved time series using different methods. We selected two queries with different levels of complexity from the test dataset. The simpler query consists of a single cycle of a pattern, while the complex query contains periodic signals. Periodic signals in complex queries typically require shift-invariant distance measures [26] to retrieve relevant items correctly. This figure demonstrates that the CTSR problem is challenging, as even irrelevant time series are visually similar to the query.

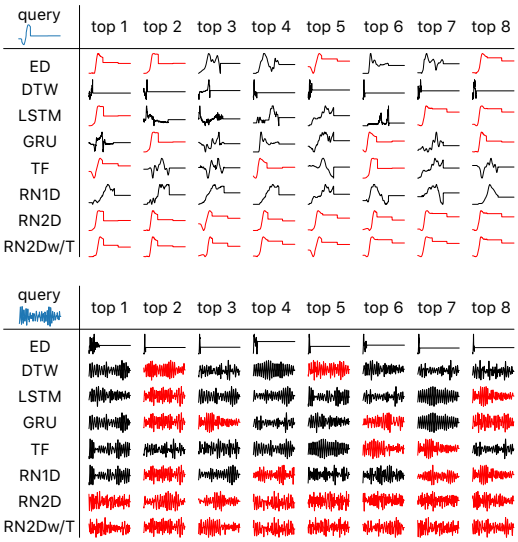

Figure 8: The top eight items of a given query time series were retrieved by different methods. The retrieved time series is colored red if it is relevant and black if it is irrelevant.

Examining the retrieved time series for different methods, we make the following observations: The ED method struggles with the more complex query because it cannot align the query to relevant time series. The DTW method outperforms the ED method on the complex query, but its alignment freedom hurts its performance on the simple query. The four neural network baselines (i.e., LSTM, GRU, TF, and RN1D) perform better than both ED and DTW methods when considering both queries. However, none of these baselines outperforms both the RN2Dw/T and RN2D methods, which reliably retrieve relevant items.

4.2 Transaction Time Series

To evaluate the effectiveness and efficiency of different CTSR system designs in addressing the problem presented in Fig. 9, we constructed a transaction time series dataset for testing these solutions.

The dataset comprises 160,014 training time series, 19,993 test queries, and 19,992 validation queries, each representing the time series signature for a merchant, with a length of 168. For a retrieved time series given a query time series, we consider the retrieved time series a relevant item from the database if it belongs to merchants of the same business type as the query time series. If the retrieved time series is from another business type than the query time series, it is considered an irrelevant item. We calculate the performance measurements at $k = 10$ for each of the 19,993 test queries, and present the average results in Table 2. Only faster deep

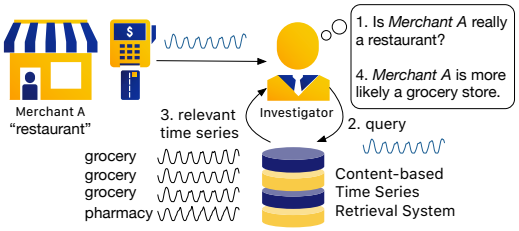

Figure 9: The use case for a CTSR system with transaction time series.

learning methods (i.e., LSTM, GRU, TF, RN1D, and RN2Dw/T) were tested on the transaction time series, as these methods were the clear winners from the experiments conducted on the UCR archive CTSR dataset, considering both effectiveness and efficiency. The proposed RN2Dw/T method is the best performing method, with the difference in performance between it and the second-best RN1D method appearing small. However, the difference is statistically significant based on two-sample t-tests with $\alpha = 0.05$. All methods have similar query times (in seconds).

Table 2: The proposed method outperforms the others in all performance measurements.

Method	PREC@10	AP@10	NDCG@10	Query Time
LSTM	0.8798	0.8857	0.8830	0.0194
GRU	0.8503	0.8580	0.8545	0.0093
TF	0.8622	0.8678	0.8652	0.0040
RN1D	0.8941	0.8967	0.8955	0.0687
RN2Dw/T	**0.8963**	**0.8999**	**0.8982**	0.0197

To provide a better user experience, we can further reduce the query time of the proposed RN2Dw/T system by replacing the exact nearest neighbor search algorithm with an approximate one. We compared the performance of the proposed system using both the exact and approximate nearest neighbor search algorithms. For the approximate search, we utilized the nearest neighbor descent method as described in [11], and implemented it using the PyNNDescent library [8]. Our results indicate that the query time for the proposed method is 19.65 milliseconds for exact nearest neighbor search and 0.45 milliseconds for approximate nearest neighbor search. Moreover, the method's performances (i.e., PREC@10, AP@10, and NDCG@10) remain exactly the same as presented in Table 2. In other words, we were able to improve the throughput by 43X without compromising the method's performance.

5 CONCLUSION

In this paper, we investigated the Content-based Time Series Retrieval (CTSR) problem and tested eight methods (ED, DTW, LSTM, GRU, TF, RN1D, RN2D, and RN2Dw/T). Our results show that the proposed RN2Dw/T method outperformed the other methods in terms of both effectiveness and efficiency. As part of our future work, we aim to enhance the efficiency of the proposed model by incorporating low-bit representation techniques, such as those presented in [1, 4, 37, 38, 43]. Moreover, we have plans to expand the method's scope to accommodate multi-dimensional time series data. Additionally, we aim to tailor the method for application in an unsupervised setting with pre-training methods [42].

REFERENCES

[1] Alexandr Andoni and Daniel Beaglehole. 2022. Learning to hash robustly, guaranteed. In *International Conference on Machine Learning*. PMLR, 599–618.

[2] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery* 31, 3 (2017), 606–660.

[3] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. isax 2.0: Indexing and mining one billion time series. In *2010 IEEE International Conference on Data Mining*. IEEE, 58–67.

[4] Huiyuan Chen, Xiaoting Li, Kaixiong Zhou, Xia Hu, Chin-Chia Michael Yeh, Yan Zheng, and Hao Yang. 2022. TinyKG: Memory-Efficient Training Framework for Knowledge Graph Neural Recommender Systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 257–267.

[5] Huiyuan Chen, Yusan Lin, Menghai Pan, Lan Wang, Chin-Chia Michael Yeh, Xiaoting Li, Yan Zheng, Fei Wang, and Hao Yang. 2022. Denoising Self-Attentive Sequential Recommendation. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 92–101.

[6] Davide Chicco. 2021. Siamese neural networks: An overview. *Artificial Neural Networks* (2021), 73–94.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[8] Contributions. 2023. A Python nearest neighbor descent for approximate nearest neighbors. https://github.com/lmcinnes/pynndescent.

[9] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305.

[10] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[11] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.

[12] Petros Drineas, Michael W Mahoney, and Nello Cristianini. 2005. On the Nyström Method for Approximating a Gram Matrix for Improved Kernel-Based Learning. *journal of machine learning research* 6, 12 (2005).

[13] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 1–34.

[14] 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.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] 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.

[17] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. 2017. LSTM fully convolutional networks for time series classification. *IEEE access* 6 (2017), 1662–1669.

[18] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. 2019. Multivariate LSTM-FCNs for time series classification. *Neural networks* 116 (2019), 237–245.

[19] 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).

[20] Bryan Lim and Stefan Zohren. 2021. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A* 379, 2194 (2021), 20200209.

[21] Ilya Loshchilov and Frank Hutter. 2018. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

[22] Yue Lu, Renjie Wu, Abdullah Mueen, Maria A Zuluaga, and Eamonn Keogh. 2022. Matrix profile XXIV: scaling time series anomaly detection to trillions of datapoints and ultra-fast arriving data streams. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1173–1182.

[23] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems* 30 (2017).

[24] Muhammad Marwan Muhammad Fuad and Pierre-François Marteau. 2010. Multi-resolution approach to time series retrieval. In *Proceedings of the Fourteenth International Database Engineering & Applications Symposium*. 136–142.

[25] Themis Palpanas. 2020. Evolution of a Data Series Index: The iSAX Family of Data Series Indexes: iSAX, iSAX2. 0, iSAX2+, ADS, ADS+, ADS-Full, ParIS, ParIS+, MESSI, DPiSAX, ULISSE, Coconut-Trie/Tree, Coconut-LSM. In *Information Search, Integration, and Personalization: 13th International Workshop, ISIP 2019, Heraklion, Greece, May 9–10, 2019, Revised Selected Papers 13*. Springer, 68–83.

[26] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1855–1870.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[28] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 262–270.

[29] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 3009–3017.

[30] Xavier Renard, Maria Rifqi, Gabriel Fricout, and Marcin Detyniecki. 2016. EAST representation: fast discovery of discriminant temporal patterns from time series. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*.

[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 452–461.

[32] Dongjin Song, Ning Xia, Wei Cheng, Haifeng Chen, and Dacheng Tao. 2018. Deep r-th root of rank supervised joint binary embedding for multivariate time series retrieval. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2229–2238.

[33] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. 2020. Tslearn, a machine learning toolkit for time series data. *J. Mach. Learn. Res.* 21, 118 (2020), 1–6.

[34] The Author(s). 2023. Supplementary materials. https://sites.google.com/view/efficient-ctsr-system.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[36] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[37] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. 2015. Learning to hash for indexing big data—A survey. *Proc. IEEE* 104, 1 (2015), 34–57.

[38] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. 2017. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2017), 769–790.

[39] Zhiguang Wang, Weizhong Yan, and Tim Oates. 2017. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*. IEEE, 1578–1585.

[40] Christopher Williams and Matthias Seeger. 2000. Using the Nyström method to speed up kernel machines. *Advances in neural information processing systems* 13 (2000).

[41] Chin-Chia Michael Yeh. 2018. *Towards a near universal time series data mining tool: Introducing the matrix profile*. University of California, Riverside.

[42] Chin-Chia Michael Yeh, Xin Dai, Huiyuan Chen, Yan Zheng, Yujie Fan, Audrey Der, Vivian Lai, Zhongfang Zhuang, Wang Junpeng, Liang Wang, and Wei Zhang. 2023. Toward a Foundation Model for Time Series Data. In *Proceedings of the 32nd ACM International Conference on Information & Knowledge Management*.

[43] Chin-Chia Michael Yeh, Mengting Gu, Yan Zheng, Huiyuan Chen, Javid Ebrahimi, Zhongfang Zhuang, Junpeng Wang, Liang Wang, and Wei Zhang. 2022. Embedding Compression with Hashing for Efficient Representation Learning in Large-Scale Graph. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4391–4401.

[44] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th international conference on data mining (ICDM)*. Ieee, 1317–1322.

[45] 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 the AAAI Conference on Artificial Intelligence*, Vol. 35. 11106–11115.

[46] Dixian Zhu, Dongjin Song, Yuncong Chen, Cristian Lumezanu, Wei Cheng, Bo Zong, Jingchao Ni, Takehiko Mizoguchi, Tianbao Yang, and Haifeng Chen. 2020. Deep unsupervised binary coding networks for multivariate time series retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1403–1411.