Neural Networks 168 (2023) 549–559

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

STTRE: A Spatio-Temporal Transformer with Relative Embeddings for multivariate time series forecasting

Azad Deihim [a,*], Eduardo Alonso [b], Dimitra Apostolopoulou [a]

[a] Department of Engineering, City University of London, Northampton Square, London, EC1V 0HB, England, United Kingdom
[b] Department of Computer Science, City University of London, Northampton Square, London, EC1V 0HB, England, United Kingdom

ARTICLE INFO

Keywords:
Multivariate time series
Transformer
Forecasting
Attention
Embeddings
Spatio-temporal

ABSTRACT

The prevalence of multivariate time series data across several disciplines fosters a demand and, subsequently, significant growth in the research and advancement of multivariate time series analysis. Drawing inspiration from a popular natural language processing model, the Transformer, we propose the Spatio-Temporal Transformer with Relative Embeddings (STTRE) to address multivariate time series forecasting. This work primarily focuses on developing a Transformer-based framework that can fully exploit the spatio-temporal nature of a multivariate time series by incorporating several of the Transformer's key components, but with augmentations that allow them to excel in multivariate time series forecasting. Current Transformer-based models for multivariate time series often neglect the data's spatial component(s) and utilize absolute position embeddings as their only means to detect the data's temporal component(s), which we show is flawed for time series applications. The lack of emphasis on fully exploiting the spatio-temporality of the data can incur subpar results in terms of accuracy. We redesign relative position representations, which we rename to relative embeddings, to unveil a new method for detecting latent spatial, temporal, and spatio-temporal dependencies more effectively than previous Transformer-based models. We couple these relative embeddings with a restructuring of the Transformer's primary sequence learning mechanism, multi-head attention, in a way that allows for full utilization of relative embeddings, thus achieving up to a 24% improvement in accuracy over other state-of-the-art multivariate time series models on a comprehensive selection of publicly available multivariate time series forecasting datasets.

1. Introduction

A multivariate time series is a group of time-dependent variables, where each variable is represented as a sequence of historical data indexed in chronological order; each element in the sequence can exhibit dependencies on its historical values and other interlinked variables (Silvestrini & Veredas, 2008). For example, increased precipitation may be linked to decreased car traffic volume on a motorway. Multivariate time series data are prevalent across several domains, such as energy, finance, transportation, and healthcare (Lu et al., 2022, Patel et al. 2022, Lee et al. 2021, Merdjanovska and Rashkovska 2022), calling attention to the need for research and advancement of multivariate time series forecasting. Multivariate time series forecasting can be defined as the prediction of a future value of a target variable based on historical values of that variable and other related variables (Box et al., 2008).

The ideal model for multivariate time series forecasting should be able to fully exploit latent spatio-temporal dependencies to extract as much relevant information from the data as possible. Models developed for univariate time series can often be adapted slightly for multivariate time series and still achieve notable accuracy without a means for detecting spatial dependencies. Still many of the top-performing models for multivariate time series utilize mechanisms to identify both spatial and temporal dependencies (Ruiz et al., 2021; Wen et al., 2022). The current state-of-the-art in time series is generally composed of neural networks that employ advanced methods that can capture these relationships, such as recurrent units, convolution, attention, graph learning, and many others. Before the rise of these deep learning methods, more rudimentary statistical analysis was commonly used for time series forecasting, such as autoregressive integrated moving average (Box & Pierce, 1970).

Although most commonly associated with computer vision applications, convolutional architectures have also gained widespread notoriety in time series research. Their ability to detect dependencies within image pixels is easily transferable to time series, demonstrated

* Corresponding author.
E-mail address: azaddeihim@gmail.com (A. Deihim).

https://doi.org/10.1016/j.neunet.2023.09.039
Received 21 February 2023; Received in revised form 28 July 2023; Accepted 24 September 2023
Available online 30 September 2023

in InceptionTime (Fawaz et al., 2020) and XceptionTime (Rahimian et al., 2019). InceptionTime is an ensemble of deep convolutional networks designed for univariate time series classification. XceptionTime utilizes depthwise separable convolutions and a novel normalization technique to operate on multivariate time series data but was only used for hand gesture classification. A non-deep convolutional model, Rocket (Dempster et al., 2020), currently scores high accuracy on many publicly available datasets by combining a novel method of random convolutional kernels and a linear classifier, outperforming or matching other time series models, including InceptionTime, across 85 University of California, Riverside (UCR) datasets for univariate time series classification. MiniRocket (Dempster et al., 2021), the successor to Rocket, utilizes very similar techniques and can produce comparable results to Rocket across 109 UCR datasets — univariate and multivariate time series classification.

Recurrent Neural Networks, namely long short-term memory (LSTM), have long been a staple in time series analysis. They utilize a hidden state to recall information from prior inputs to influence the current and future inputs and outputs, granting the ability to detect temporal behavior. LSTM-FCN (Karim et al., 2018) combines long short-term memory and a fully convolutional network to achieve improved accuracy over earlier models, such as ResNet (He et al., 2016) and COTE (Bagnall et al., 2015). ALSTM-FCN, an extension of LSTM-FCN, gains minor accuracy improvement with an added attention module. Initially designed for univariate time series, these models were later augmented for use in multivariate time series classification (Karim et al., 2019).

Despite their effectiveness, many of the aforementioned models are outclassed by ensemble models HIVE-COTE (Lines et al., 2018) and TS-CHEIF (Shifaz et al., 2020), who perform time series classification by combining various sophisticated classifiers. HIVE-COTE utilizes an ensemble of phase-independent shapelets, bag-of-words-based dictionaries, and phase-dependent intervals, while TS-CHEIF utilizes an ensemble of tree classifiers. Although these methods are powerful, they suffer from extremely high computational complexity, coupled with the inability to exploit GPU hardware well. Additionally, these models were built for univariate time series classification but did not consider multivariate time series or regression. HIVE-COTE 2.0 (Middlehurst et al., 2021), introducing Rocket classifiers, and novel dictionary and forest classifiers, can perform multivariate time series classification, but, similar to its predecessor, it cannot be used for regression and suffers from computational complexity issues.

Graph neural networks have become increasingly popular in recent years as they are effective in analyzing data with graph-like structures. These networks typically require the data to have a defined set of vertices, edges, and adjacency matrix.MTGNN (Wu et al., 2020a) introduces a novel implementation of a graph learning layer to use a graph neural network for multivariate time series. The graph learning layer allows the model to connect elements with strong relationships, allowing it to uncover spatio-temporal dependencies without needing a preexisting graph structure. MTGNN addresses single-step and multi-step forecasting across four multivariate time series datasets. MTPool (Duan et al., 2022) introduces a novel graph-pooling framework to hierarchically aggregate node information, addressing a significant limitation of graph neural networks, but is only evaluated on multivariate time series classification.

Similarly to the graph neural network, the Transformer (Vaswani et al., 2017), an attention-based neural network, has also increased in popularity since its inception. While initially developed for natural language translation, the Transformer is universally powerful across many sequence learning tasks, including time series (Qi et al., 2021, Li et al., 2019, Zhou et al., 2021, Wu et al., 2020b, Wu et al., 2021, Zhou et al., 2022, Liu et al., 2022). While the structure of the Transformer's attention mechanism is invariant to a sequence's order, positional embeddings are used to encode position information into each element in the sequence, informing attention about the sequence's order, thus allowing the Transformer to be effective for sequence learning.

Many Transformer-based models have been developed for multivariate time series applications. BAT (Lei et al., 2022) utilizes a Transformer-based framework for speech emotion recognition. Time Series Transformer (TST) (Zerveas et al., 2021) performs both multivariate time series classification and forecasting across 17 datasets using a Transformer encoder-based architecture, achieving higher accuracy than Rocket on a majority of studied datasets. TST used absolute position embeddings but did not specify an explicit means for capturing the data's spatial components. Spacetimeformer (Grigsby et al., 2021) couples absolute position embeddings with variable embeddings, informing the attention mechanism about both position information and variable information of each element, allowing it to capture spatio-temporal dependencies. This model achieves high accuracy across four multivariate time series datasets, outperforming MTGNN. Chen et al. (2022) combines a Transformer with a graph learning layer and graph convolution to perform anomaly detection in an Internet of Things system. The Gated Transformer Network (Liu et al., 2021) utilizes two transformer encoders for multivariate time series classification: one encoder that attends to time steps and one encoder that attends to variables across a single time step. Transformers have also been widely adopted in video analysis tasks, which have similar spatio-temporal properties to MTS. Learning videos using transformers (Neimark et al., 2021, Ye & Bilodeau, 2023, Qu et al., 2023) demonstrates the ability of transformers to be effective across a wide range of tasks that require spatio-temporal learning.

Notwithstanding the success of recent Transformer-based models for multivariate time series, many did not utilize a structure that could fully exploit the spatio-temporal nature of a multivariate time series. Inspired by the Transformer, we propose Spatio-temporal Transformer with Relative Embeddings (STTRE). This model utilizes three different modules to learn spatial, temporal, and spatio-temporal dependencies by re-purposing relative positional representations to exploit these dependencies. STTRE is the first Transformer-based model for multivariate time series to introduce relative positional representations. Relative positional representations (Shaw et al., 2018), which we will refer to as relative embeddings for the remainder of this paper, were introduced to replace the absolute positional embeddings in the Transformer — for natural language processing (NLP); this was a minor improvement. Still, it could be far more advantageous in time series applications. Using only absolute position to describe an element's position in a sequence can be misleading for many types of sequences. For time series, where the sequence is tied to observable time such as time of day, or day of the week, it can be arbitrary and misinformative to denote any time step in a sequence with a specific integer position, as that position could represent dissimilar times and/or days in different sequences. For example, position i could represent 00:00 and 06:00 in two different sequences, but they would be given the same position embedding. A visualization for this problem is provided in Fig. 1. Many Transformer-based models use absolute positional embeddings as their only means of informing the attention mechanism about position information, but it is flawed in time series applications.

The contributions of our work are as follows:

1. We create a novel three-module Transformer-based architecture to extract and isolate temporal, spatial, and spatio-temporal dependencies.
2. We introduce a novel implementation of relative embeddings to extract spatial and temporal dependencies in a multivariate time series. We also showcase a method of structuring multi-head attention to exploit our relative embedding implementation.
3. We implement the first Transformer-based model for multivariate time series to utilize relative embeddings.
4. We demonstrate that STTRE achieves the best accuracy across six experiments compared to other state-of-the-art multivariate time series models.

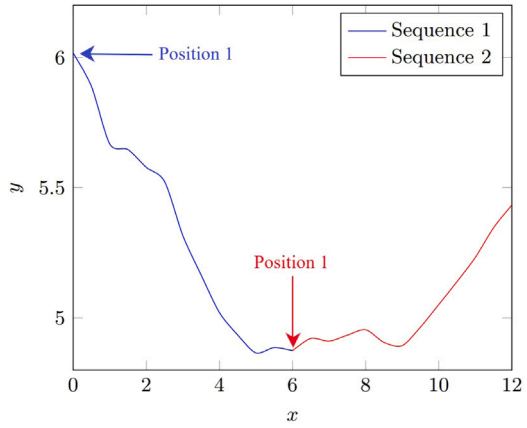

Fig. 1. Two sequence segments from a toy dataset. The x-axis denotes the time of day in hours, while the y-axis denotes the value of a toy variable. Position 1 in sequence 1 represents the time 0:00, while position 1 in sequence 2 represents the time 6:00. While dissimilar, these positions will be given the same absolute position embedding.

These contributions primarily focus on augmentations to the Transformer encoder, allowing our model to uncover latent relationships between elements more effectively than other Transformer-based implementations, granting a significant advantage for multivariate time series forecasting. We apply STTRE to perform forecasting across five multivariate time series datasets, covering energy consumption, traffic, finance, and air pollution — studying STTRE's accuracy on each. We compare performance with five baseline models, including top-performing models for multivariate time series, and demonstrate that STTRE outperforms all using various standard metrics. We also analyze the contribution of each component of our model by performing ablation studies, which adds to the interpretability of our results.

The rest of the paper is structured as follows: Section 2 will cover background and related work, outlining details of the Transformer and its primary components; Section 3 will detail the proposed model, outlining the three core modules of the architecture and their characteristics; Section 4 will detail the experimental set-up and the datasets used, present the results, and analyze the performance of STTRE against the state-of-the-art; we shall conclude with a discussion and outline of future work in Section 5.

2. Background

In this section, we review background material and related work that supports STTRE. This includes a brief outline of the Transformer, embeddings, multi-head attention, and normalization.

The Transformer ((Vaswani et al., 2017)) is an attention-based neural network developed for natural language translation. Attention mechanisms in neural networks imitate cognitive function, amplifying parts of the input deemed essential while diminishing those less essential. The use of attention in neural networks across several areas of machine learning has been widely studied (Chaudhari et al., 2021). While the Transformer was initially developed for natural language translation, many have studied its application outside of NLP, discovering merit across many sequence learning tasks and beyond sequence learning (Han et al., 2023). Many of the modules and mechanisms utilized in the Transformer were designed solely for sequence-to-sequence processing of words and are incompatible with other tasks. As a result, architectures tend to differ significantly from that of the original Transformer in non-NLP applications.

The Transformer uses an encoder to process an input sequence to create a new encoded representation of the input, which a decoder can process and translate into a desired output sequence. The primary learning mechanism in the transformer decoder, masked multi-head attention, requires the output to be a sequence. For multivariate time series forecasting, we will predict one value rather than a sequence; thus, using a decoder is neither necessary nor applicable in our work, so we will focus on the encoder.

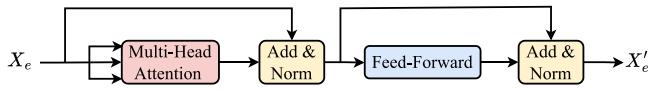

Fig. 2. Diagram of the Transformer encoder. The encoder, which processes X_e to compute X'_e, comprises multi-head attention and a feed-forward layer, with normalization after each.

The Transformer encoder comprises the following components: multi-head attention, feed-forward, and normalization layers. Fig. 2 shows a diagram of the encoder.

Firstly, the raw input X is embedded. In the multivariate case, $X \in \mathbb{R}^{l \times m}$ is cast to $X_e \in \mathbb{R}^{l \times m \times d}$, where l is the sequence length, m is the number of variables, and d is the size of the embedding dimension (in the univariate case, m is 1). Next, multi-head attention processes X_e to produce a new matrix $z \in \mathbb{R}^{l \times m \times d}$, a matrix of attention scores. A skip connection is used to sum z with X_e. The resultant matrix is then normalized, producing $z_n \in \mathbb{R}^{l \times m \times d}$. A two-layer feed-forward followed by an activation function is used to transform z_n into a new matrix with the same dimension; the elected activation function will be LeakyReLU:

$$\mathrm{FF}(z_n) = \mathrm{LeakyReLU}(W_1 z_n + b_1)W_2 + b_2, \tag{1}$$

$$\mathrm{LeakyReLU}(x) = \left\{ \begin{array}{ll} x, & \text{if } x \geq 0 \\ 0.01x, & \text{if } x < 0 \end{array} \right\}, \tag{2}$$

where $W_{(\cdot)}$ and $b_{(\cdot)}$ are weight matrices and bias vectors respectively. The feed-forward output is summed with the input, z_n. The sum is then normalized, producing the final output of the encoder, $X'_e \in \mathbb{R}^{l \times m \times d}$, highlighting important features and dependencies within X_e.

The encoder also utilizes a layering mechanism, where the encoder is duplicated k times: the input of the i'th encoder sublayer will be the output of encoder sublayer i — 1, and the output of the k'th layer will be the final output of the encoder. Note that learned weights are not shared across sublayers.

As a standalone module, the encoder cannot generate a prediction; it simply creates an encoded data representation. With the removal of the decoder, an alternative learning mechanism is required to translate the encoded representation into a prediction.

2.1. Embeddings

The encoder operates on an embedded sequence; the embedding layer casts each sequence's elements into a higher dimensional space, where each dimension can represent a different characteristic, feature, or contextual meaning regarding its respective element. Elements closer to each other in this multidimensional space are more likely to be closely related. Element embeddings are more intuitive when operating on words, as it is widely recognized that words can have different meanings depending on their context, but this is also true for elements in a time series, as specific values may have different contextual meanings based on their surrounding values. These element embeddings are learned during the model's training via a set of learned weights. In the original Transformer, which had a discrete input space, an embedding vector for each unique word is stored in a dictionary. In the continuous case, where an infinitely large dictionary is infeasible, a learned weight matrix $W_e \in \mathbb{R}^{l \times dl}$ and bias vector $b_e \in \mathbb{R}^l$ are used to linearly transform the sequence via $W_e X + b_e$, thus casting it into the embedding dimension. Fig. 3 provides a visualization of the element embedding process.

Another type of embedding, position embedding, is used to implant position information into each element. Since multi-head attention has

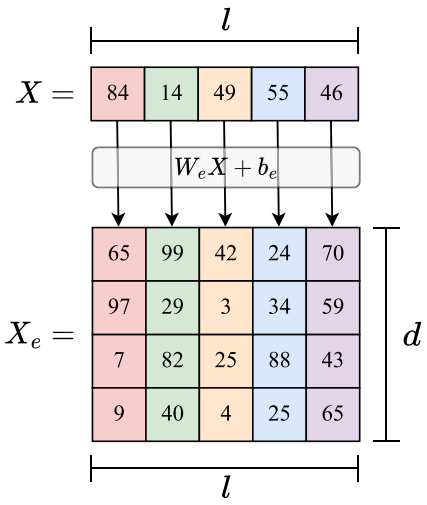

Fig. 3. Visualization of the element embedding process on an l length sequence.

no built-in mechanism to understand the order of a sequence, absolute position embeddings are necessary to understand temporal information. Each position t_i in $T = \{t_1, t_2, t_3, \cdots, t_l\}$ will be translated into a higher dimensional vector, resulting in the absolute position embedding matrix $PE \in \mathbb{R}^{l \times d}$, where each position t_i will be represented by a vector of d different values. The Transformer introduced sinusoidal position embeddings of varying frequencies taking the form:

$$PE(t_i, j) = \sin(\frac{t_i}{10000^{2j/d}}) \qquad \forall_t \in T, \tag{3}$$

$$PE(t_i, j) = \cos(\frac{t_i}{10000^{(2j+1)/d}}) \qquad \forall_t \in T, \tag{4}$$

where $j = \{1, 2, 3, \ldots, \frac{2}{d}\}$. Sinusoidal embeddings work well for sequence learning tasks with sequences of varying lengths, as they can generalize to sequence lengths that are unseen during training. Their flaw is that they may overpower or be overpowered by element embeddings when summed together, as orthogonality is not guaranteed to remain intact after they are added. Some have experimented with concatenating element embeddings and positional embeddings rather than summing them to ensure that they occupy orthogonal spaces (Huang et al., 2018), but this is an impractical approach in many applications as it increases memory requirements significantly. Time2Vec (Kazemi et al., 2019) experiments with learned sinusoidal position representations to alleviate orthogonality issues. Since then, many tried fully learnable absolute position embeddings, where an embedding for every possible position is stored in a dictionary (Devlin et al., 2018). A benefit of fully learnable absolute position embeddings is that they can learn to occupy a virtually orthogonal space in relation to the element embeddings, mimicking the effect of concatenation without the added memory constraints. However, fully learnable absolute position embeddings do not work well in tasks with varying sequence lengths, as they cannot generalize to sequence lengths unseen during training — the learned dictionary will not contain entries for positions beyond the maximum position seen during training. Also, higher positions seen infrequently during training may have undertrained weights. In time series tasks, where there is more control over the length of sequences, varying sequence lengths will not be a complication.

For use in multivariate time series, Grigsby et al. (2021) experimented with variable embeddings, utilizing a similar method to fully learnable absolute position embeddings to embed variable information into each element, thus informing multi-head attention about variable information regarding each element in the sequence.

In 2018, relative embeddings were created as a novel way to represent positional information for multi-head attention by using an embedding matrix of relative distances between all sequence element pairs, granting minor accuracy improvement over absolute position embeddings (Shaw et al., 2018). Note that relative embeddings are more representative of the relationship between two elements than an actual numerical distance between their positions. They function by calculating a matrix of pairwise embeddings between any two elements in the input sequence, informing attention about the relationship between two positions. The memory requirements associated with relative embeddings made them infeasible for long sequence lengths, but they were later optimized to reduce memory requirements significantly (Huang et al., 2018).

Each type of embedding offers a unique benefit when computing a new representation of the input data. Many of the aforementioned embeddings are utilized in STTRE.

2.2. Multi-head attention

At the core of the Transformer encoder is multi-head attention, a type of self-attention mechanism that computes an attention function several times in parallel, concatenating each computation. Self-attention describes a type of attention mechanism that can relate different positions of a sequence and compute a new representation of that sequence; this new representation highlights parts of the sequence and its embedding space that warrant more attention.

Multi-head attention is divided into h heads, each attending to $\frac{1}{h}$ of the input space. The output is $z \in \mathbb{R}^{l \times d}$, or in the multivariate case, $z \in \mathbb{R}^{l \times m \times d}$, a matrix that contains attention scores. Attention scores in z inform the model of what elements are more important and what embedding information is more important. Multi-head attention can exploit GPU power by parallelizing heads, which is prominent in its appeal.

When dividing X_e between heads, it is commonly divided along the d dimension — resulting in a $(l, \frac{d}{h})$ dimensional space for each head, allocating a portion of the embedding space to each head. The input to each head is duplicated thrice, forming the values, keys, and queries. Each value, key, and query matrix first undergo a linear transformation:

$$\begin{aligned} V &= W_V X_e + b_V, \\ K &= W_K X_e + b_K, \\ Q &= W_Q X_e + b_Q, \end{aligned} \tag{5}$$

where W are learned weight matrices and b are learned bias vectors — values, keys, and queries have separate weights and biases, as denoted by the subscripts. Reference to heads is removed from the equations for simplicity. The values, keys, and queries are named as such because the structure of the multi-head attention is analogous to that of a database retrieval system. The naming is representative of how they interact with each other. It is hypothesized that they could hold information representative of their named functions since their interactions will affect their learned weights. Following the linear transformation, attention is computed using the values, keys, and queries. The attention function most commonly used is scaled dot-product attention, which will be the attention function used in this paper. Scaled dot-product attention takes the form:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d/h}})V, \tag{6}$$

$$\text{Softmax}(a) = \frac{\exp(a)}{\sum_{j=1}^{K} \exp(a)}. \tag{7}$$

An upper left triangle mask is applied to QK^T to prevent the queries from attending to keys that occur later in the sequence. The output of the attention function for each head is then concatenated together, followed by a linear transformation, producing the resultant matrix z:

$$z = W_A A + b_A, \tag{8}$$

where A is the output of the attention function in (6).

2.3. Normalization

Introduced to mitigate the presence of internal covariate shift in neural networks, batch normalization (Ioffe & Szegedy, 2015) is used between layers of the encoder. Batch normalization decreases training time by allowing the user to set higher learning rate values and mitigates the variance of random initialization of weights. Batch normalization uses mean-standard deviation normalization:

$$n_i' = \frac{n_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{9}$$

where n_i is the value at the i'th neuron, μ is the mean of the batch, σ^2 is the variance of the batch, and ϵ is a small constant value added to the variance for numerical stability.

Layer normalization (Ba et al., 2016) was introduced shortly following the discovery of batch normalization. Layer normalization is the transpose of batch normalization, computing mean and variance across all neurons in the layer within a single training case rather than an entire batch. While the original Transformer uses layer normalization for their normalization layers, we use batch normalization in our study. The superiority of layer normalization in the Transformer can be mainly attributed to the varying sequence lengths found in sentences (Shen et al., 2020). However, this does not apply to the datasets examined in this study. Additionally, batch normalization can help dampen outliers found in time series (Zerveas et al., 2021), which is not an issue in natural language processing.

3. Proposed methodology

In this section, we address multivariate time series forecasting. Let $X = \{x_0, x_1, \cdots, x_m\}$ define a multivariate time series with m variables, where each x_i represents a univariate time series with l historical observations, i.e, $x_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,l}\}$, where $x_{i,j} \in \mathbb{R}$ denotes the value of the i'th variable at the j'th time step. Given X, the objective is to predict $Y = \{x_{i,l+1}\}$.

3.1. Model

STTRE[1] incorporates three modules: the temporal module, the spatial module, and the spatio-temporal module. Each module is designed to capture different latent dependencies within a multivariate time series. Each module will accept a flattened and embedded multivariate time series segment $X_e \in \mathbb{R}^{lm \times d}$. The inputs are flattened into a lm length vector, and then each element is cast into the embedding dimension via linear transformation. Absolute position embeddings are still used in conjunction with relative embeddings, as they offer a minor improvement in performance at a minimal computational cost — these are summed with element embeddings to produce X_e. In the spatial module, the absolute position embeddings are replaced with variable embeddings, which will embed information regarding the variable of an element rather than embedding information regarding the position of an element. For the position and variable embeddings, fully learnable positional embeddings are used.

Each module utilizes a structure similar to the Transformer's encoder but with augmentations that we hypothesize will improve its ability to perform its delegated task. We also adopt the encoder's layering mechanism, setting the number of layers to three.

The output of each module, $X_e' \in \mathbb{R}^{lm \times d}$ is concatenated, producing a new matrix $X_{st} \in \mathbb{R}^{3lm \times d}$, a spatio-temporal encoded representation of X_e. A feed-forward layer consolidates the embedding dimension such that X_{st} is reduced to a vector of length $3lm$. A linear regression layer reduces this vector into a single value, representing the final prediction, Y. A diagram of STTRE's architecture can be found in Fig. 4.

[1] https://github.com/AzadDeihim/STTRE

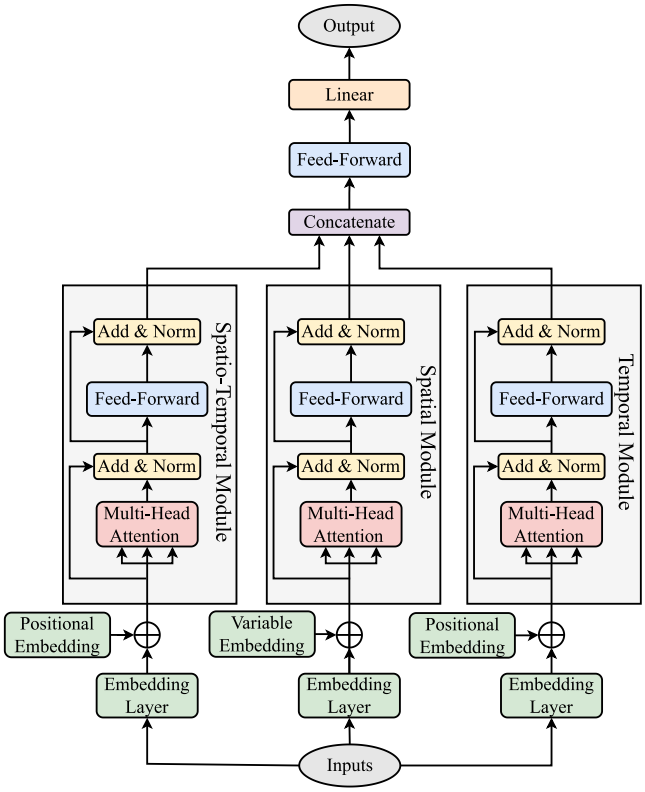

Fig. 4. Diagram of STTRE model.

Learned weights in the architecture were optimized using a mean squared error loss function:

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{10}$$

where n is the batch size, Y is the observed target value, and $\hat{Y}$ is the predicted target value.

In most cases, STTRE requires relatively small values of d for optimal performance, ranging from 8 to 32. Many Transformer implementations require significantly larger values of d for optimal results; for instance, Time Series Transformer, a Transformer-based framework for multivariate time series (Zerveas et al., 2021), uses embedding sizes ranging from 256 to 512. We found that using a smaller embedding size increases accuracy significantly while also greatly decreasing computation time and memory requirements. We demonstrate the trade-off between accuracy and embedding size in Appendix B.

3.1.1. Multi-head attention structure

While the architecture of each module appears the same, the structure of the multi-head attention in each differs. The three core modules, temporal, spatial, and spatio-temporal, are designed to learn information and detect dependencies related to their named functions; thus, each module's multi-head attention structure is designed to improve its ability to perform its assigned function.

The temporal module is designed to detect strictly temporal dependencies while minimizing its ability to capture spatial dependencies. In this module, multi-head attention will have exactly m heads, one head to attend to each variable. Also, X_e will be divided along the m dimension rather than the d dimension, resulting in a (l, d) space for each head so that heads cannot attend to variables other than the one allocated to it. Lastly, weight matrices will have separate weights for each head. There are no learned weights in the temporal module that are shared across multiple variables, so each weight will only be

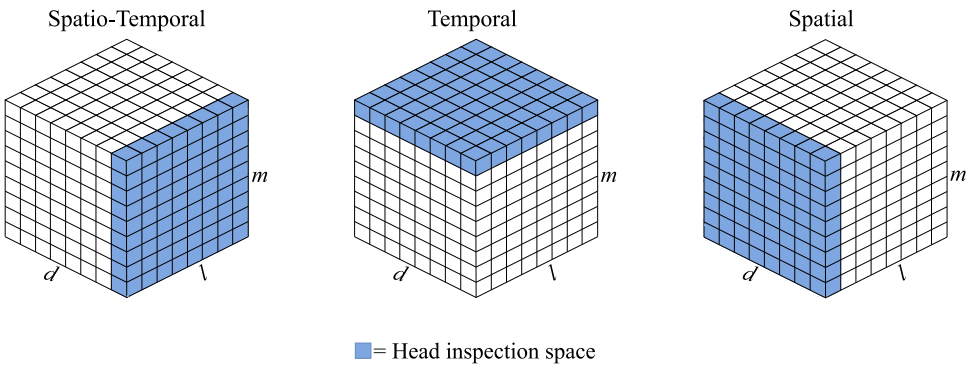

Fig. 5. Head inspection space for each module. This diagram is meant to visualize the dimensions each module attends to. For simplicity, only the inspection space for the first head is shown.

capable of learning information regarding the variable it was delegated to.

The spatial module behaves similarly to the temporal module but transposed — X must be transposed before transforming into X_e. In this module, multi-head attention will use exactly l heads, one head to attend to each time step. X_e is divided along the l dimension, resulting in a (m, d) space for each head, so each head cannot attend to time steps other than the one allocated. Weight matrices will have separate weights for each head — weights will be capable of learning information across variables but only within their assigned time step.

Multi-head attention in the spatio-temporal module will be constructed differently than in the spatial and temporal modules — it is more similar to that of a standard implementation. It will have four heads, with each attending to $\frac{1}{h}$ of the embedding space, resulting in a $(lm, \frac{d}{h})$ dimensional space for each head. Unlike the other modules, the heads in the multi-head attention will be granted access to elements across different variables and timesteps but are confined to their assigned portion of the embedding space. Weight matrices will share weights across heads. A visual representation of the multi-head attention's functionality for each module is provided in Fig. 5.

3.2. Relative embeddings

Relative embeddings are employed to aid each module in constructing an encoded spatio-temporal representation of X_e. To obtain relative embeddings, we must first generate a matrix of learnable relative embedding weights E_r. The operation shown in (12) is performed using Q, the queries, and E_r to obtain S, a square matrix containing an entry for all pairs of elements in X, denoting a relationship between the elements' locations in X. Next, the scaled dot-product attention function is augmented to factor relative embedding information:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T + S}{\sqrt{d/h}})V, \tag{11}$$

$$S = \text{skew}(QE_r^T). \tag{12}$$

An upper left triangle mask is applied to QE_r^T before the skewing function. The skew function is crucial in this calculation as S must match up with the indexing in QK^T. Otherwise, the addition calculation in (11) will add relative embeddings to the incorrect indices. Each (i, j) entry in QK^T contains the dot product of the query for element i in X and key for element j in X, making QK^T absolute-by-absolute indexed. However, When QE_r^T is computed, each (i, r) contains the dot product of the query at position i and the relative embedding r between elements i and j in X, making QE_r^T absolute-by-relative indexed. The skew function's purpose is to shift columns in QE_r^T to make the indexing absolute-by-absolute. (Huang et al., 2018). Algorithm 1 describes the skewing function, defined in the context of the temporal module. Dimensionality will vary in other modules, but the algorithm's structure will remain unchanged. Fig. 6 visualizes how the skewing function operates.

Algorithm 1 Skew

Input: $S_0 \in \mathbb{R}^{l \times l}$, the resultant matrix of QE_r^T with an upper left triangle mask.
Output: $S \in \mathbb{R}^{l \times l}$, the skewed representation of S_0.

$S_p \leftarrow \text{pad}(S_0)$ {Pad column of zeros on left}
$S_r \leftarrow \text{reshape}(S_p)$ {Such that upper left triangle mask is now an upper right triangle mask}
$S \leftarrow \text{slice}(S_r)$ {Remove first row}

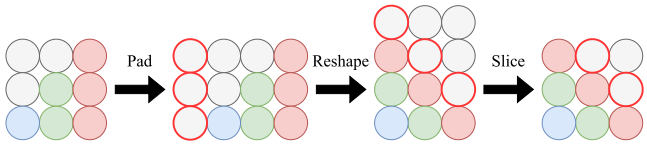

Fig. 6. Diagram of skewing function — gray circles represent masked values. Firstly, we pad a column vector before the leftmost column. Next, we reshape the matrix such that the upper triangle mask is now on the right. Lastly, we remove the first row to produce the desired S.

In each module, relative embeddings will capture pairwise relationships between elements within their respective head inspection spaces. In the temporal module, the relative embedding weight matrix $E_t \in \mathbb{R}^{h \times l \times d}$, have a separate (l, d) dimensional weight matrix for each head as the relative embeddings of each head should only be able to acknowledge pairs within its assigned variable, and not be able to acknowledge pairs across variables. The resulting matrix $S_t \in \mathbb{R}^{h \times l \times l}$ will be a matrix of pairwise temporal relationships.

In the spatial module, $E_s \in \mathbb{R}^{h \times m \times d}$ has a separate (m, d) dimensional weight matrix for each head, as the relative embeddings of each head should only be able to acknowledge pairs within its assigned time step, and not be able to acknowledge pairs across time steps. Thus $S_s \in \mathbb{R}^{h \times m \times m}$ will be a matrix of pairwise spatial relationships.

In the spatio-temporal module, $E_{st} \in \mathbb{R}^{lm \times \frac{d}{h}}$ has shared weights across heads. This is done to reduce memory requirements but cannot be done in other modules as it may obstruct their ability to perform their delegated task. Relative embeddings in this module will be capable of recognizing pairs across variables and time steps, thus $S_{st} \in \mathbb{R}^{lm \times lm}$ will be a matrix of pairwise spatio-temporal relationships.

Fig. 7 provides a visual representation of each module's inspection space of relative embeddings. Algorithm 2 describes multi-head attention with relative embeddings for the temporal module. The algorithm's structure is similar for the other modules, but the dimensions must be adjusted accordingly.

Algorithm 2 Temporal Multi-head Attention With Relative Embeddings

Input: $X_e \in \mathbb{R}^{lm \times d}$, the flattened and embedded input multivariate time series; h, the number of heads.
Output: $z \in \mathbb{R}^{lm \times d}$, the new attention-weighted temporal representation of X_e
Parameters: Linear transformation weights and biases for the values, keys, and queries,
$W_V \in \mathbb{R}^{d \times d}$, $b_V \in \mathbb{R}^d$
$W_K \in \mathbb{R}^{d \times d}$, $b_K \in \mathbb{R}^d$
$W_Q \in \mathbb{R}^{d \times d}$, $b_Q \in \mathbb{R}^d$
Relative embeddings weights, $E_r \in \mathbb{R}^{h \times l \times d}$

$V \leftarrow W_V X_e + b_V$
$K \leftarrow W_K X_e + b_K$
$Q \leftarrow W_Q X_e + b_Q$
$S_0 \leftarrow \text{mask}(Q E_r^T)$ {Upper left triangle mask}
$S \leftarrow \text{skew}(S)$
$z \leftarrow \text{Softmax}(\frac{QK^T + S}{\sqrt{d/h}})V$

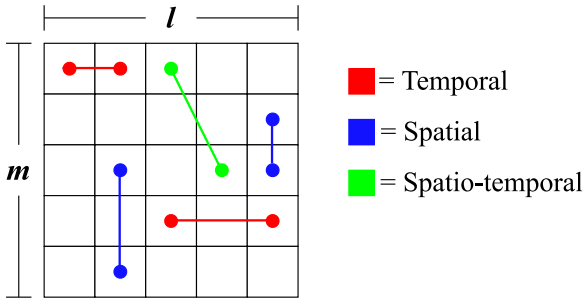

Fig. 7. Visual representation of the span of relative embeddings in each module. Note that not all possible pairwise connections are represented in this figure.

4. Experiments

The results shown in this section are produced by training models on a predefined set of training data and evaluating them on a predefined set of testing data. The training data and testing data will be the same for all models. All models are trained until convergence. The number of epochs until convergence varied between models and datasets. Experiments are conducted using Python 3.7 on an NVIDIA A100 40 GB GPU. For each epoch, the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE),[2] and mean absolute error (MAE) of the predictions on testing data is recorded. After each epoch, the average of each metric across the five most recent epochs is also recorded — the lowest average MSE will be used to determine which epoch to extract results from. The results will be presented as an average across five epochs.

4.1. Data

Five publicly accessible datasets are used for this study. Datasets are primarily gathered from the UCI repository (Dua & Graff, 2017), except for one aggregated from Yahoo! Finance.

- **Metro Interstate Traffic Volume**: a dataset from the UCI repository used to predict car traffic volume on an interstate highway.
- **Uber Stock**: a dataset aggregated from publicly available Yahoo! Finance data (Yahoo!Finance, 2022). Used to predict the stock price of Uber.
- **Appliances Energy**: a dataset from the UCI repository used to predict household energy usage. This dataset contains two target variables: energy consumption of appliances and energy consumption of lights — these target variables will be evaluated separately, denoted by 1 and 2, respectively.
- **BeijingPM2.5**: a dataset from the UCI repository used to predict air pollution in Beijing.
- **Istanbul Stock Exchange**: a dataset from the UCI repository used to predict Istanbul stock exchange returns.

[2] MAPE is only considered for datasets where the zero values do not exist in the target variable, as this will produce divide-by-zero errors.

Table 1
Dataset characteristics according to sampling rates, number of samples, sequence lengths, and number of variables. The heterogeneous dataset selection is designed to provide STTRE with an extensive assessment.

Dataset	Sampling rate	Total samples	Sequence length	Variables
MetroInterstate	1 h	48205	24	17
Uber Stock	1 day	726	60	5
Appliances Energy	10 min	19736	144	26
BeijingPM2.5	1 h	15901	24	6
Istanbul Stock	1 day	536	40	8

Table 1 describes the characteristics of each examined dataset. While the datasets cover various domains, they also provide diverse characteristics, allowing for a comprehensive examination. We used a 50%–50% split of training and testing data for each dataset. 20% of the training data was used as validation data; validation data is used strictly to tune hyperparameters for the model(s). The datasets are not randomized prior to the split. Chronologically, all data in the testing set occurs after all data in the training set. This is a more practical approach to time series problems, as real-world applications will always exhibit this characteristic. Also, this mitigates the presence of high degrees of autocorrelation between sequences in the training set and sequences in the test set since chronologically neighboring sequences will have significant overlap and will generally be easy targets to predict accurately.

Multivariate time series data can often exhibit different behavior based on long-term trends, i.e., seasonal trends. The train and test split used in this study allows the models to be evaluated on a more diverse set of data, encompassing various monthly or seasonal trends. If the test set is too small, we may only find that models are trained on data from various months or seasons but are only evaluated on data from one season, thus providing only a limited assessment of the model(s).

4.2. Baseline models

STTRE's performance is compared with a diverse selection of 5 baseline models. These models are primarily selected from recently published literature which include multivariate time series analysis in their study and hence are representative of the state-of-the-art.

- **XceptionTime**: A convolutional neural network for multivariate time series (Rahimian et al., 2019).
- **TST**: A transformer-based neural network for multivariate time series (Zerveas et al., 2021).
- **MiniRocket**: Random convolutional kernels with linear regression layer for prediction (Dempster et al., 2021).
- **LSTM**: Long short-term memory (Hochreiter & Schmidhuber, 1997).
- **LSTM-FCN**: Long short-term memory with a fully convolutional network (Karim et al., 2018).

Table 2
Results of each experiment — Bold values indicate the best score for each category, and underlined values represent the second-best score for that category.

	RMSE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**895.6**	**2.947**	**90.74**	**6.474**	**36.02**	**0.013**
XceptionTime	2011.6	4.342	108.7	8.473	40.16	0.016
TST	936.7	5.926	108.1	7.937	38.91	0.036
MiniRocket	1206.1	6.458	103.2	8.639	47.85	0.047
LSTM	974.4	4.678	109.4	9.244	41.26	0.016
LSTM-FCN	1100.8	5.190	103.8	8.213	47.61	0.018

	MAE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**556.7**	**2.216**	**48.65**	**3.649**	**20.96**	**0.008**
XceptionTime	1769.5	2.758	58.64	4.970	24.25	0.012
TST	569.6	3.714	62.30	4.671	24.14	0.029
MiniRocket	888.5	4.531	55.43	5.884	32.10	0.036
LSTM	617.3	3.088	60.10	6.072	25.65	0.012
LSTM-FCN	802.2	3.236	62.95	5.301	31.34	0.014

	MAPE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	–	**0.069**	**0.513**	–	–	**0.731**
XceptionTime	–	0.085	0.672	–	–	0.973
TST	–	0.115	0.776	–	–	6.546
MiniRocket	–	0.148	0.641	–	–	8.623
LSTM	–	0.094	0.708	–	–	1.438
LSTM-FCN	–	0.103	0.845	–	–	2.276

4.3. Results

We conducted six experiments using the five previously outlined datasets. Table 2 shows the RMSE, MAE, and MAPE scores earned by each model on each dataset. The winner in each experiment is denoted in bold font, while an underline denotes the runner-up. We observe that STTRE outperforms all baselines across all datasets, achieving an average rank of 1. The second-best-performing model, XceptionTime, earns an average rank of 3.4, directly followed by TST. Compared with XceptionTime, STTRE achieves a median 27% improvement in RMSE, 23% improvement in MAE, and 23% improvement in MAPE.

With the comprehensive dataset selection considered in this study, we observe STTRE's ability to forecast multivariate time series in various environments, including datasets from different domains with vastly differing sizes, sampling rates, sequence lengths, and number of variables. STTRE's performance remained consistent across all experiments, demonstrating that it can excel across a wide range of datasets, regardless of their characteristics. On the contrary, while TST and XceptionTime performed well on many of the datasets, both models obtained significantly lower accuracy than other models on at least one dataset: TST ranked 5th on the Istanbul stock exchange, and XceptionTime ranked 6th on the Metro Interstate Traffic Volume. This suggests that these models are less robust and have a more narrow usability range than STTRE.

For each experiment, we compare STTRE's accuracy against the second best performing model, XceptionTime, and the median accuracy of all baseline models, calculating the improvement in accuracy as a percentage. This is shown in Table 3. Although the margins are notable in all experiments, STTRE scores most decisively on Uber Stock and Istanbul Stock Exchange, two small financial datasets. While this could indicate that STTRE works best on small datasets, the more likely hypothesis is that the other models underperform on small datasets, as this is a complication for many deep learning models.

TST, the other Transformer-based model, is the runner-up in three of the six experiments. It utilizes a relatively simple architecture, incorporating one Transformer encoder rather than three and does not utilize relative embeddings. Still, the parameter count in TST outnumbers that of STTRE in most experiments. With a better allocation of learned weights, STTRE can use fewer weights on average to yield state-of-the-art accuracy. Fig. 8 provides a visual comparison of parameter counts of TST and STTRE. Information regarding hyperparameter settings, training, and convergence is provided in Appendix A.

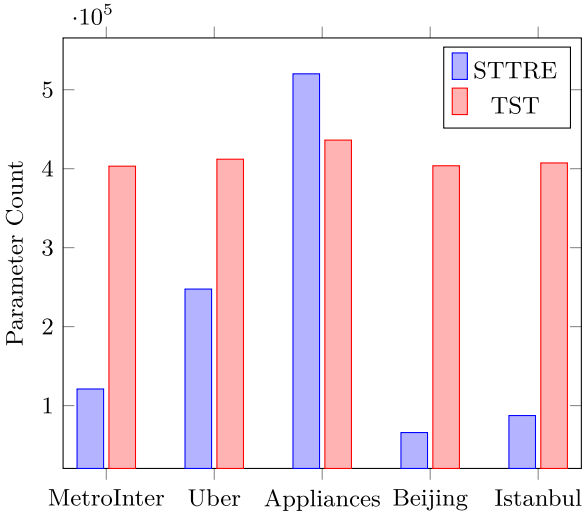

Fig. 8. Parameter count of STTRE compared with parameter count of TST.

Table 3
STTRE's improvement in accuracy over the runner-up, XceptionTime, and the median accuracy of all baseline models.

Experiment	Improvement	
	vs. XceptionTime	vs. median
MetroInterstate	3.3%	17.6%
Uber Stock	24.0%	33.4%
Appliances 1	14.7%	19.3%
Appliances 2	20.1%	25.6%
BeijingPM2.5	10.3%	13.7%
Istanbul Stock	23.7%	39.1%

Table 4
Ablation study — bold values indicate the best score for each category, and underlined values represent the second-best score for each category.

	RMSE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**895.6**	**2.947**	**90.74**	**6.474**	**36.02**	**0.013**
w/o RE	903.7	3.033	93.54	6.873	37.33	**0.013**
w/o STM	919.6	4.022	93.23	6.727	36.42	0.016
w/o SM	927.5	4.939	94.98	6.922	37.22	0.016
w/o TM	933.2	4.898	91.80	7.328	36.14	0.014

	MAE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**556.7**	**2.216**	48.65	**3.649**	**20.96**	**0.008**
w/o RE	563.8	2.222	52.64	3.981	21.46	**0.008**
w/o STM	578.5	2.621	51.05	3.843	21.29	0.012
w/o SM	570.6	3.412	51.59	4.022	21.20	0.011
w/o TM	577.4	3.371	**47.27**	4.193	21.01	0.009

	MAPE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	–	**0.069**	0.513	–	–	**0.731**
w/o RE	–	0.071	0.625	–	–	0.838
w/o STM	–	0.079	0.579	–	–	1.237
w/o SM	–	0.101	0.576	–	–	1.189
w/o TM	–	0.097	**0.501**	–	–	0.826

Table 5
Improvement in accuracy and standard deviation (SD) of accuracy improvement given by each component.

Component	Improvement	SD
Relative Embeddings	4.6%	4.9%
Spatio-Temporal Module	11.8%	11.5%
Spatial Module	15.4%	13.8%
Temporal Module	11.3%	13.0%

4.4. Ablation study

In this section, we conduct an ablation study to demonstrate the indispensability of key components in this architecture. The following models will be evaluated:

- **w/o RE**: STTRE without relative embeddings.
- **w/o STM**: STTRE without the spatio-temporal module.
- **w/o SM**: STTRE without spatial module.
- **w/o TM**: STTRE without the temporal module.

The ablation study is conducted using the same datasets which we examined previously. Table 4 contains the results of the ablation study, displaying the RMSE, MAE, and MAPE earned by each model on each dataset. The winner in each experiment is denoted in bold font, while an underline denotes the runner-up. The ablation study demonstrates that with all components and modules intact, the base model achieves the best results, earning an average rank of 1.3. We calculate the significance of each removed component across all datasets, given as a percentage, denoting the improvement of STTRE's accuracy compared to the other versions in the ablation study. In Table 5, we report the average improvement in accuracy granted by each component and the standard deviation of improvement in accuracy.

On average, w/o SM scores the lowest accuracy, which tends to remain consistent across all experiments. This suggests that the spatial module could be the STTRE's most influential component. Performance degradation w/o SM does not coincide with the number of variables in the datasets or any other dataset characteristic. This also appears to be the case for the other modules.

On Uber Stock and Istanbul Stock Exchange, STTRE performed relatively well without relative embeddings. We hypothesize that this is due to the small amount of data used for training in these datasets, as both of these datasets are two orders of magnitude smaller than the other datasets used in this study. It is likely that relative embeddings require larger amounts of data for full efficacy and may be less advantageous on smaller datasets. We find that on the subset of datasets containing larger amounts of data, relative embeddings provide an average 5.6% improvement in accuracy.

In some cases, STTRE can still obtain high accuracy w/o TM, demonstrated on Appliances Energy 1, BeijingPM2.5, and Istanbul Stock Exchange. This could be due to redundancy in the spatio-temporal module. For example, if the temporal module is removed, STTRE will still have the means to undercover temporal dependencies via the spatio-temporal module. Still, it is important to note that since the heads in these modules attend to different dimensions, they will both hold different interpretations of the uncovered temporal dependencies. Alternatively, this could result from those datasets having weak temporal dependencies.

5. Conclusion

In this work, we proposed STTRE, a novel Transformer-based model for spatio-temporal learning of multivariate time series. STTRE addresses shortcomings of Transformer-based models for multivariate time series by improving upon the encoder in a way that enhances recognition of spatio-temporal dependencies, namely the novel implementation of relative embeddings, coupled with a novel restructuring multi-head attention to fully exploit latent spatio-temporal dependencies in a multivariate time series. We evaluated STTRE on a comprehensive set of publicly available multivariate time series forecasting datasets encompassing an expansive range of characteristics. We demonstrate significantly improved performance over several state-of-the-art models, defining STTRE as the current best-performing multivariate time series forecasting model.

The innovations introduced in this study focused primarily on computing an encoded spatio-temporal representation of a multivariate time series. This was done via the three core modules in the architecture. The complementary task of computing a prediction given the encoded representation of the data was originally a task delegated to the decoder, which is absent in our work and replaced with feed-forward and linear layers. In the future, we aim to explore the use of alternative mechanisms which could aid in computing a prediction given the encoded representation of the data in lieu of the decoder.

Table A.6
Hyperparameter settings for all experiments. This table includes settings for learning rate, embedding size, batch size, number of epochs until convergence, and total training time in minutes.

Dataset	Learning rate	d	Batch size	Epochs	Time
MetroInterstate	0.0001	32	256	1070	328
Uber Stock	0.0001	32	64	1320	29
Appliances Energy 1	0.0001	8	32	47	371
Appliances Energy 2	0.0001	8	32	51	396
BeijingPM2.5	0.0001	32	256	1063	86
Istanbul Stock	0.00005	32	268	485	5

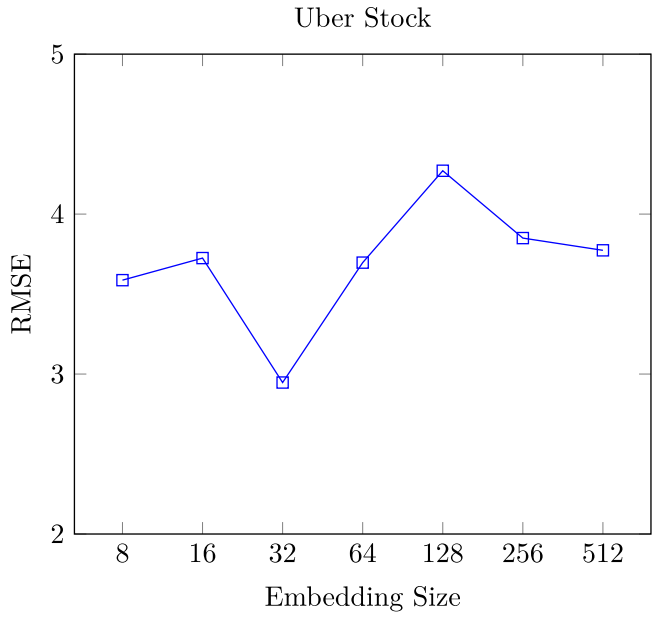

Fig. B.9. RMSE versus embedding size on Uber Stock dataset.

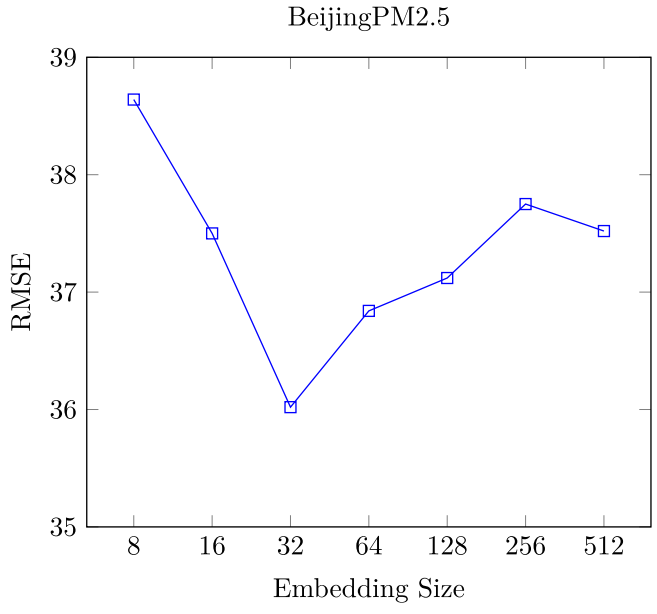

Fig. B.10. RMSE versus embedding size on BeijingPM2.5 dataset.

Additionally, although STTRE tends to be exceptionally memory efficient when compared to other Transformer-based models, due to the nature of relative embeddings, memory requirements can scale poorly as sequence lengths and the number of variables increase. In the future, we aim to find a remedy for this issue when faced with larger inputs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data is publicly available and will be included in the referenced GitHub link.

Appendix A. Reproduciblity

For reproducibility of our experiments, we outline hyperparameter settings and the number of epochs and total computation time for convergence for each experiment, shown in Table A.6. We set dropout to 0.1 for regularization for all experiments and used the Adam optimizer. The weight matrices in the feed-forward layer of each module are (d, d) dimensional.

Appendix B. Embedding size and accuracy trade-off

We demonstrate the trade-off between embedding size and accuracy to validate our claim that smaller embedding sizes work better for STTRE. Figs. B.9 and B.10 display the trade-off between RMSE and embedding size on the Uber Stock and BeijingPM2.5 datasets, respectively. These figures show that commonly used embedding sizes, which generally range between 256 and 512, do not work well for STTRE.

References

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. http://dx.doi.org/10.48550/ARXIV.1607.06450.

Bagnall, A., Lines, J., Hills, J., & Bostrom, A. (2015). Time-series classification with COTE: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, *27*(9), 2522–2535. http://dx.doi.org/10.1109/TKDE.2015.2416723.

Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (2008). *Time series analysis: Forecasting and control* (4). Hoboken, N.J: J. Wiley & Sons.

Box, G. E. P., & Pierce, D. A. (1970). Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, *65*(332), 1509–1526. http://dx.doi.org/10.1080/01621459.1970.10481180.

Chaudhari, S., Mithal, V., Polatkan, G., & Ramanath, R. (2021). An attentive survey of attention models.

Chen, Z., Chen, D., Zhang, X., Yuan, Z., & Cheng, X. (2022). Learning graph structures with transformer for multivariate time-series anomaly detection in IoT. *IEEE Internet of Things Journal*, *9*(12), 9179–9189. http://dx.doi.org/10.1109/JIOT.2021.3100509.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, *34*(5), 1454–1495. http://dx.doi.org/10.1007/s10618-020-00701-z.

Dempster, A., Schmidt, D. F., & Webb, G. I. (2021). MiniRocket: A very fast (almost) deterministic transform for time series classification. In *Data mining and knowledge discovery* (pp. 248–257). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3447548.3467231.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, URL http://arxiv.org/abs/1810.04805.

Dua, D., & Graff, C. (2017). *UCI machine learning repository*. University of California, Irvine, School of Information and Computer Sciences, URL http://archive.ics.uci.edu/ml.

Duan, Z., Xu, H., Wang, Y., Huang, Y., Ren, A., Xu, Z., Sun, Y., & Wang, W. (2022). Multivariate time-series classification with hierarchical variational graph pooling. *Neural Networks*, *154*, 481–490. http://dx.doi.org/10.1016/j.neunet.2022.07.032.

Fawaz, H. I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., Webb, G. I., Idoumghar, L., Muller, P.-A., & Petitjean, F. (2020). InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962. http://dx.doi.org/10.1007/s10618-020-00710-y.

Grigsby, J., Wang, Z., & Qi, Y. (2021). Long-range transformers for dynamic spatiotemporal forecasting. *CoRR*, URL https://arxiv.org/abs/2109.12218.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y., & Tao, D. (2023). A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *45*(1), 87–110. http://dx.doi.org/10.1109/TPAMI.2022.3152247.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition* (pp. 770–778). http://dx.doi.org/10.1109/CVPR.2016.90.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Huang, C. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A. M., Hoffman, M. D., & Eck, D. (2018). An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, URL http://arxiv.org/abs/1809.04281.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. http://dx.doi.org/10.48550/ARXIV.1502.03167.

Karim, F., Majumdar, S., Darabi, H., & Chen, S. (2018). LSTM fully convolutional networks for time series classification. *IEEE Access*, *6*, 1662–1669. http://dx.doi.org/10.1109/ACCESS.2017.2779939.

Karim, F., Majumdar, S., Darabi, H., & Harford, S. (2019). Multivariate LSTM-FCNs for time series classification. *Neural Networks*, *116*, 237–245. http://dx.doi.org/10.1016/j.neunet.2019.04.014.

Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., & Brubaker, M. A. (2019). Time2Vec: Learning a vector representation of time. *CoRR*, URL http://arxiv.org/abs/1907.05321.

Lee, K., Eo, M., Jung, E., Yoon, Y., & Rhee, W. (2021). Short-term traffic prediction with deep neural networks: A survey. *IEEE Access*, *9*, 54739–54756. http://dx.doi.org/10.1109/ACCESS.2021.3071174.

Lei, J., Zhu, X., & Wang, Y. (2022). BAT: Block and token self-attention for speech emotion recognition. *Neural Networks*, *156*, 67–80. http://dx.doi.org/10.1016/j.neunet.2022.09.022.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Neural Information Processing Systems*, 5243–5253. http://dx.doi.org/10.5555/3454287.3454758.

Lines, J., Taylor, S., & Bagnall, A. (2018). Time series classification with HIVE-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, *12*(5), http://dx.doi.org/10.1145/3182382.

Liu, M., Ren, S., Ma, S., Jiao, J., Chen, Y., Wang, Z., & Song, W. (2021). Gated transformer networks for multivariate time series classification. *CoRR*, URL https://arxiv.org/abs/2103.14438.

Liu, S., Yu, H., Liao, C., Li, J., Lin, W., Liu, A. X., & Dustdar, S. (2022). Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International conference on learning representations*. URL https://openreview.net/forum?id=0EXmFzUn5I.

Lu, C., Li, S., & Lu, Z. (2022). Building energy prediction using artificial neural networks: A literature survey. *Energy and Buildings*, *262*, Article 111718. http://dx.doi.org/10.1016/j.enbuild.2021.111718.

Merdjanovska, E., & Rashkovska, A. (2022). Comprehensive survey of computational ECG analysis: Databases, methods and applications. *Expert Systems with Applications: An International Journal*, *203*(C), http://dx.doi.org/10.1016/j.eswa.2022.117206.

Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., & Bagnall, A. (2021). HIVE-COTE 2.0: A new meta ensemble for time series classification. *Machine Learning*, *110*, 3211–3243. http://dx.doi.org/10.1007/s10994-021-06057-9.

Neimark, D., Bar, O., Zohar, M., & Asselmann, D. (2021). Video transformer network. In *Proceedings of the IEEE/CVF international conference on computer vision*. arXiv:2102.00719.

Patel, N. P., Parekh, R., Thakkar, N., Gupta, R., Tanwar, S., Sharma, G., Davidson, I. E., & Sharma, R. (2022). Fusion in cryptocurrency price prediction: A decade survey on recent advancements, architecture, and potential future directions. *IEEE Access*, *10*, 34511–34538. http://dx.doi.org/10.1109/ACCESS.2022.3163023.

Qi, X., Hou, K., Liu, T., Yu, Z., Hu, S., & Ou, W. (2021). From known to unknown: Knowledge-guided transformer for time-series sales forecasting in alibaba. *CoRR*, URL https://arxiv.org/abs/2109.08381.

Qu, M., Deng, G., Di, D., Cui, J., & Su, T. (2023). Dual attentional transformer for video visual relation prediction. *Neurocomputing*, *550*, Article 126372. http://dx.doi.org/10.1016/j.neucom.2023.126372.

Rahimian, E., Zabihi, S., Atashzar, S. F., Asif, A., & Mohammadi, A. (2019). XceptionTime: A novel deep architecture based on depthwise separable convolutions for hand gesture classification. *CoRR*, URL http://arxiv.org/abs/1911.03803.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *35*(2), 401–449. http://dx.doi.org/10.1007/s10618-020-00727-3.

Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-attention with relative position representations. *CoRR*, URL http://arxiv.org/abs/1803.02155.

Shen, S., Yao, Z., Gholami, A., Mahoney, M. W., & Keutzer, K. (2020). PowerNorm: Rethinking batch normalization in transformers. In *Proceedings of the 37th international conference on machine learning*. JMLR.org, http://dx.doi.org/10.1007/s10994-021-06057-9.

Shifaz, A., Pelletier, C., Petitjean, F., & Webb, G. I. (2020). TS-CHIEF: A scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, *34*(3), 742–775. http://dx.doi.org/10.1007/s10618-020-00679-8.

Silvestrini, A., & Veredas, D. (2008). Temporal aggregation of univariate and multivariate time series models: A survey. *Journal of Economic Surveys*, *22*(3), 458–497. http://dx.doi.org/10.1111/j.1467-6419.2007.00538.x.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems, Vol. 30*. Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2022). Transformers in time series: A survey. http://dx.doi.org/10.48550/ARXIV.2202.07125.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., & Zhang, C. (2020). Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Data mining and knowledge discovery* (pp. 753–763). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3394486.3403118.

Wu, S., Xiao, X., Ding, Q., Zhao, P., Wei, Y., & Huang, J. (2020). Adversarial sparse transformer for time series forecasting. In *Advances in neural information processing systems, Vol. 33* (pp. 17105–17115). Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2020/file/c6b8c8d762da15fa8dbbdfb6baf9e260-Paper.pdf.

Wu, H., Xu, J., Wang, J., & Long, M. (2021). Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in neural information processing systems, Vol. 34* (pp. 22419–22430). Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2021/file/bcc0d400288793e8bdcd7c19a8ac0c2b-Paper.pdf.

Yahoo!Finance (2022). URL https://finance.yahoo.com/quote/UBER/history?p=UBER.

Ye, X., & Bilodeau, G.-A. (2023). Video prediction by efficient transformers. *Image and Vision Computing*, *130*, Article 104612. http://dx.doi.org/10.1016/j.imavis.2022.104612.

Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A., & Eickhoff, C. (2021). A transformer-based framework for multivariate time series representation learning. In *Data mining and knowledge discovery* (pp. 2114–2124). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3447548.3467401.

Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., & Jin, R. (2022). Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *Proceedings of machine learning research: vol. 162, Proceedings of the 39th international conference on machine learning* (pp. 27268–27286). URL https://proceedings.mlr.press/v162/zhou22g.html.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. *Association for the Advancement of Artificial Intelligence*, *35*, 11106–11115. http://dx.doi.org/10.1609/aaai.v35i12.17325.