Fast Training of a Transformer for Global Multi-horizon Time Series Forecasting on Tensor Processing Units

J. Luis García-Nava (jose.garcia@umich.mx)
Universidad Michoacana de San Nicolás de Hidalgo
Juan J. Flores
Universidad Michoacana de San Nicolás de Hidalgo
Victor M. Tellez
Universidad Michoacana de San Nicolás de Hidalgo
Felix Calderon
Universidad Michoacana de San Nicolás de Hidalgo

Research Article

Keywords: Deep Learning, Self-attention, Cloud Computing, TPU

Posted Date: August 11th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1813490/v1

Fast Training of a Transformer for Global Multi-horizon Time Series Forecasting on Tensor Processing Units

García-Nava J-Luis[1*], Flores Juan J[1,2], Tellez Victor M[1] and Calderon Felix[1]

[1*]School of Electrical Engineering, Universidad Michoacana de San Nicolás de Hidalgo, Morelia, 58030, Michoacán, México.
[2]University of Oregon, Eugene, 97403, OR, USA.

*Corresponding author(s). E-mail(s): jose.garcia@umich.mx;
Contributing authors: juan.flores@umich.mx;
0906214j@umich.mx; felix.calderon@umich.mx;

Abstract

Time Series Forecasting (TSF) is essential to key domains, and the transformer neural network has advanced the state-of-the-art on global, multi-horizon TSF benchmarks. The quadratic time and memory complexity of the vanilla transformer (VT) hinders its application to big data environments; therefore, multiple efficient variants of the VT that lower complexity via sparse self-attention have been proposed. However, less complex algorithms do not directly produce faster executions, and machine learning models for big data are typically trained on accelerators designed for dense-matrix computation that render slower performance with sparse matrices. To better compare the accuracy-speed trade-off of the VT and its variants, it is essential to test them on such accelerators. We implemented a cloud-based VT on Tensor Processing Units to address this task. Experiments on large-scale datasets show that our transformer outperforms two reference models on accuracy while reducing training times from hours to under two minutes.

Keywords: Deep Learning, Self-attention, Cloud Computing, TPU

1 Introduction

Time Series Forecasting (TSF) is essential to decision-making in various crucial domains such as science, engineering, business, and economics. Over the last two decades Machine Learning (ML) algorithms consolidated as an effective approach to TSF. Two factors contributed to this situation: (1) the increasing size, quality, and availability of time series historical datasets, and (2) the sustained development of powerful ML-oriented computing frameworks. Aditionally, progressive advances in Deep Learning (DL) provided innovative neural network architectures able to produce state-of-the-art results in TSF [1], including Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory networks (LSTM), attention-based mechanisms and, among the latter, transformers. The transformer is a novel neural network architecture based on an effective self-attention mechanism that entirely leaves out convolutional or recurrent components. Transformers have consistently shown the ability to produce state-of-the-art results in fields like natural language processing (NLP) and computer vision (CV). Based on their success, the number of projects based on transformers has increased exponentially since its first entrance in [2], leading to the publication of numerous papers and multiple specialized surveys [3], [4], [5]. Concerning TSF, self-attention enables transformers to build predictions by focusing on the most significant parts of the time series history. Moreover, as self-attention does not involve sequential processing, TSF transformers are entitled to a parallelization extent that is out of the reach of other architectures.

As a result of the Big Data era, the canonical TSF problem evolved from predicting one step ahead on a single time series to making predictions for multiple time steps into the future (*multi-horizon* forecasting) over large sets of related time series (*multi-series* or *global* forecasting) [6]. Figure 1 shows the general process of global multi-horizon forecasting: for each element i in a set of related time series, previous observations z_i, associated time-based covariates x_i, and static covariates s_i are passed as inputs to a single forecast model. Once trained, the model can predict a given number of time steps into the future of any time series in the set. Time-dependent covariates (i.e. hour, weekday, or month-based encodings) generate a temporal alignment, which can be used to find patterns across time series dynamics. Static covariates (i.e. geographical location, customer identity, or item category), in conjunction with proper embeddings, enable the model to group time series into sub-sets with similar behavior.

Recently, transformer architectures have been extensively applied to the global multi-horizon TSF problem [8], [9], [7]. A lot of this effort addresses two well known concerns that hinder the application of the original vanilla transformer (VT) [2] to practical TSF environments. First, the full self-attention mechanism makes the VT's speed and memory complexities quadratic on the input sequence length $L - \mathcal{O}(L^2)$. Second, concerning multi-step-ahead forecasting strategies [10], the VT inference over a multi-horizon is *iterative* (also

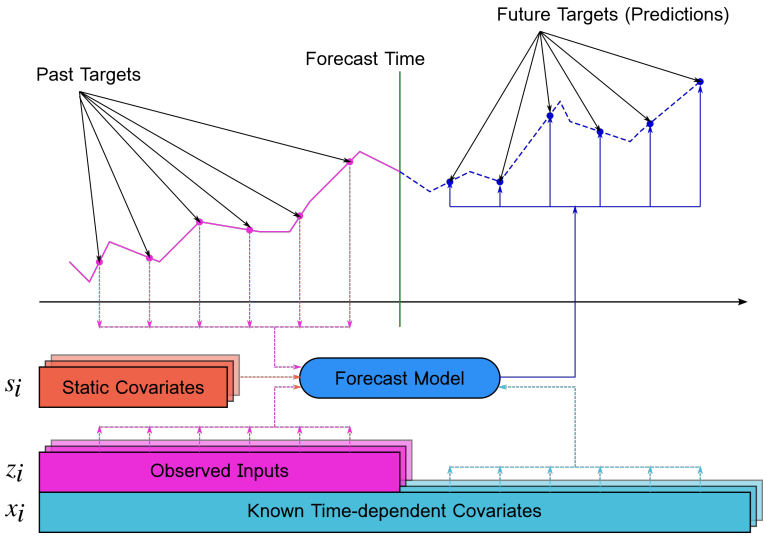

Fig. 1 Global Multi-horizon Forecasting. For each element in a set of related time series, observed inputs z_i, time-dependent covariates x_i, and static covariates s_i are passed as inputs to a single model. The model is able to predict multiple time steps into the future for any time series in the set. Figure based on [7].

called *recursive* or *multi-stage*), thus considerable slower than the *MIMO* inference available in other architectures. A significant proportion of recent research in TSF transformers is devoted to addressing the first challenge by designing variants of self-attention that apply more advanced attention strategies and simultaneously lower the quadratic-complexity computational and memory cost. However, less complexity does not directly traslate into more speed, and the fact that low-complexity, efficient transformers perform faster than dense-matrix-based transformers in practical TSF situations remains unclear. It is worth to notice the specialized hardware accelerators that usually execute ML workloads are designed for dense-matrix computation, and sparsity, a crucial characteristic of ligther self-attention mechanisms, often makes matrix multiplication slower on these devices[11]. Interesting research lines propose to adapt dense-matrix calculation devices for efficient sparse-matrix computation [12]. Still, it is crucial to investigate the direct implementation and execution of dense-matrix TSF transformers on well-suited hardware accelerators. The results of such a research work will serve as a baseline for future comparison with low-complexity, sparse-attention transformers when both types are trained on predominantly used ML hardware.

This work addresses the task by implementing a dense, full-self-attention, multi-layer, encoder-decoder transformer on a complete computational ecosystem for ML hardware-acceleration based on Tensor Processing Units (TPU). We train a global, multi-horizon TSF model based on the VT architecture

on two widely-known datasets using a specific cloud computing services integration to ensure TPU performance. The main contributions of this research are:

- To implement a fast transformer for global, multi-horizon TSF on a cloud computing ecosystem in full accordance with TPU operation.
- Our transformer generates results close to the state-of-the-art, at a small fraction of the computation time reported by the reference models.
- To our knowledge, this is the first implementation of a dense, full-self-attention, encoder-decoder, multi-layer transformer for TSF on TPUs.

The remainder of this paper is organized as follows. Section 2 discusses related research work concerning two fields: the application of transformers to TSF, and the application of TPUs to transformers or TSF. Section 3 describes the materials and methods employed in implementing the proposed architecture. Section 4 reports the experimental results. Finally, Section 5 presents conclusions and future work.

2 Related Work

In this section we discuss previous work that is related to our research concerning two specific fields: the application of transformer-based deep neural networks to TSF problems and TPU-accelerated computation to implement forecasting applications or transformers. With the intersection of these two research areas, we approximate a proper context for our work since we have not found in the literature projects that exhibit a closer or more direct relation to it.

2.1 Transformers for Time Series Forecasting

The `Deep Transformer` [13] implements the basic architecture of the original NLP transformer [2] adapted to the TSF domain, providing a practical context to discuss further model improvements. This basic (*vanilla*) transformer, is a sequence-to-sequence model with an encoder-decoder architecture, multiple layers featuring multi-head self-attention, layer normalization and residual connections, and an iterative strategy for multi-horizon prediction. The `Temporal Fusion Transformer` [7] implements an interpretable multi-head attention block on top of a sequence-to-sequence, LSTM-based encoder-decoder. This two-layer architecture for temporal processing learns both short- and long-term dependencies on time series data. It uses a direct strategy for multi-horizon forecasting and can learn regime-specific temporal dynamics from time series. A vital research line includes projects that improve the transformer performance by replacing the canonical full self-attention mechanism with a sparse mechanism. The `LogSparse Transformer` [8] implements a decoder-only architecture that employs convolutional self-attention to provide queries and keys with enhanced locality knowledge. It also

introduces *LogSparse* self-attention, which reduces the transformer complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(L(\log L)^2)$, where L is the input sequence length. The `Adversarial Sparse Transformer` (AST) [14] implements a sparse attention mechanism by replacing the $softmax$ activation function in the attention heads with the α-$entmax$ activation. It uses an encoder-decoder architecture for primary forecasting tasks and further utilizes it as the *generator* of a Generative Adversarial Network (GAN). The *discriminator* of the GAN is trained with predicted and ground truth time series that classifies them as generated or real. A composite loss function, including the generator and the discriminator losses, is optimized during *adversarial training* of the complete network. This training mitigates the error accumulation on further iterative inference. The `Informer` [9] proposes *ProbSparse* self-attention which reduces the transformer complexity to $\mathcal{O}(L(\log L))$. It also introduces a self-attention distilling operation across the transformer layers. This self-attention mechanism decreases memory usage, and a generative-style decoder produces all the prediction outputs with only one forward step. A recent research line includes projects implementing series decomposition as a built-in block of the transformer architecture and entirely replacing the canonical or sparse self-attention mechanism. The `Autoformer` [15] introduces time series decomposition to seasonal and trend-cyclical subseries as an internal operation of the TSF model. It also replaces self-attention with an Auto-Correlation mechanism, which focuses on period-based relationships at the sub-series level. The `ETSformer` [16] replaces self-attention with exponential smoothing attention and frequency attention and also redesigns the transformer architecture with modular decomposition blocks that allow the model to learn to decompose the time series into interpretable components.

2.2 Applications of Tensor Processing Units to Transformers and Forecasting

The massive computing requirements placed by ML workloads led to the continuous development of specialized hardware, like high-performance CPUs, GPUs, and more recently, TPUs. Since its public release, TPU accelerators have been applied to various research topics that demand very high processing and memory capabilities. Examples of those domains are DNN training subject to large batch sizes and specialized learning rate algorithms [17], [18], distributed evolution strategies for meta-learning [19], acceleration of explainable machine learning [20], and simulation of quantum physics [21], to mention a few. Regarding the transformer, NLP and CV projects constitute the majority of TPU-assisted research using this architecture, partially due to the high availability of pre-trained models and TPU-ready software implementations. Examples of the application of TPU-assisted transformers can be found in machine translation [22], language modeling [23], image generation [24], and visual recognition [25]. Regarding TSF research, TPUs have been applied to weather forecasting via convolutional networks enriched with a one-layer self-attention encoder block [26], and short-term wind speed forecasting via symbolic regression [27]. However, to the extent of our knowledge, our

project is the first implementation of a complete, multi-layer, encoder-decoder transformer for TSF on TPUs.

3 Materials and Methods

This section covers the materials and the methodology employed in our research. First, the global, multi-horizon TSF problem is defined. On this basis, the transformer neural network is proposed to model the prediction function. Next, we present TPUs as a convenient computation resource to deal with the huge workload required by the transformer-based TSF model. Afterward, this section explains the transformer implementation based on the previous elements. Finally, we present our experimental study in detail.

3.1 Global Multi-horizon Time Series Forecasting

This subsection defines the global, multi-horizon TSF problem based on the background provided in [8], with minor adjustments to fit our transformer implementation. Let $\{\mathbf{z}_{i,1:t_0}\}_{i=1}^{N}$ be a collection of N related univariate time series, where $\mathbf{z}_{i,t} \in \mathbb{R}$ is the value of time series i at time t. Let $\mathbf{z}_{i,1:t_0} = [\mathbf{z}_{i,1}, \mathbf{z}_{i,2}, \cdots, \mathbf{z}_{i,t_0}]$ be the *conditioning range* for time series i, and let $\tau \in \mathbb{N}$ be the *forecast horizon*. We will predict the next τ time steps (or *prediction range*) for all time series, i.e. $\{\mathbf{z}_{i,t_0+1:t_0+\tau}\}_{i=1}^{N}$ Also, let $\{\mathbf{x}_{i,1:t_0+\tau}\}_{i=1}^{N}$ be a set of associated time-based covariate vectors with dimension d_x that are known over the entire time period, e.g. hour-of-the-day or day-of-the-week. Finally, let $\{\mathbf{s}_i\}_{i=1}^{N}$ be a set of associated static covariate vectors of dimension d_s that are known and constant over the entire time period, e.g. a time series identifier. We intend to model the following conditional distribution:

$$p(\mathbf{z}_{i,t_0+1:t_0+\tau} \mid \mathbf{z}_{i,1:t_0}, \mathbf{x}_{i,1:t_0+\tau}, \mathbf{s}_i;\ \mathbf{\Phi})^{1} = \prod_{t=t_0+1}^{t_0+\tau} p(\mathbf{z}_{i,t} \mid \mathbf{z}_{i,1:t-1}, \mathbf{x}_{i,1:t}, \mathbf{s}_i;\ \mathbf{\Phi}) \tag{1}$$

where $\mathbf{\Phi}$ denotes the learnable parameters shared by all time series in the collection. The problem defined in (1) is reduced to learning a one-step-ahead prediction model $p(\mathbf{z}_{i,t} \mid \mathbf{z}_{i,1:t-1}, \mathbf{x}_{i,1:t}, \mathbf{s}_i;\ \mathbf{\Phi})$, as multiple steps ahead can be iteratively calculated by adding previously predicted values as new inputs. To fully use observations and covariates, they are concatenated into an augmented matrix as follows:

$$\mathbf{y}_{i,t} = [\mathbf{z}_{i,t-1} \circ \mathbf{x}_{i,t} \circ \mathbf{s}_i] \in \mathbb{R}^{d_x+d_s+1}, \qquad \mathbf{Y}_t = [\mathbf{y}_1, \cdots, \mathbf{y}_t]^T \in \mathbb{R}^{t\times(d_x+d_s+1)}{}^{2} \tag{2}$$

where $[\cdot \circ \cdot]$ represents concatenation along the t axis.

[1] For simplicity we omit the fact that this is the conditional distribution of the prediction range of time series i given the conditioning range of time series i, as well as the conditioning and prediction ranges of all other time series in the collection. We consider this situation implicit since the parameters $\mathbf{\Phi}$ are learned jointly from all the time series in the collection.

[2] Since the model applies to all time series, the subscript i is omitted for simplicity.

3.2 Transformer Architecture

We use a transformer-based model f to predict $\mathbf{z}_t$ given $\mathbf{Y}_t$ as $\mathbf{z}_t \sim f(\mathbf{Y}_t)$ (see Figure 2). First an internal dimension d_{model} is defined for the transformer and the augmented matrix $\mathbf{Y}_t$ is linearly projected from $\mathbb{R}^{t\times(d_x+d_s+1)}$ to $\mathbb{R}^{t\times d_{model}}$. Notice in Figure 2 (left-block) that the conditioning range passed as input to this model is split into an encoder input with dimension e and a decoder input with dimension d, where the value of t from (2) is given as $t = e + d + 1$ (the last value of the encoder input overlaps the first value of the decoder input, the decoder output is the decoder input shifted one time step to the right, and the only predicted value is the last time step in the decoder output).

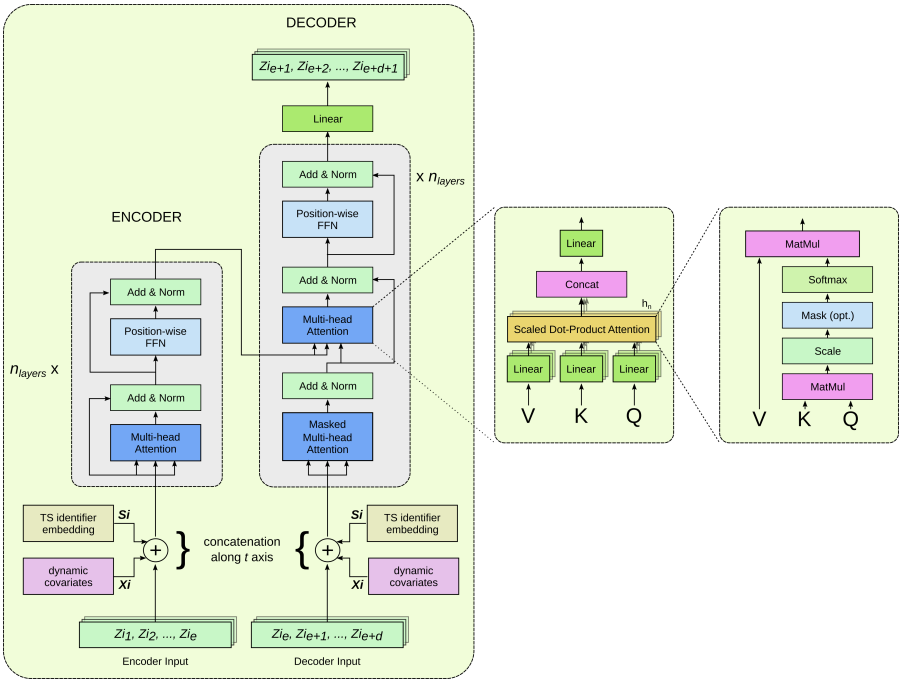

Fig. 2 Architecture of the transformer-based model. Left-block: Main architecture. Center-block: Multi-head attention. Right-block: Scaled Dot-product attention

Then a Multi-head Attention (MHA) layer (Figure 2, center-block) is used to transform $\mathbf{Y}$ into H distinct query matrices $\mathbf{Q}_h = \mathbf{Y}\mathbf{W}_h^Q$, key matrices $\mathbf{K}_h = \mathbf{Y}\mathbf{W}_h^K$, and value matrices $\mathbf{V}_h = \mathbf{Y}\mathbf{W}_h^V$, with $h = 1, \cdots, H$. The weight matrices $\mathbf{W}_h^Q, \mathbf{W}_h^K \in \mathbb{R}^{d_{model}\times d_k}$ and $\mathbf{W}_h^V \in \mathbb{R}^{d_{model}\times d_v}$ are learnable parameters. The Scaled Dot-product attention (Figure 2, right-block) computes a sequence of vector outputs:

$$\mathbf{O}_h = \text{Attention}(\mathbf{Q}_h, \mathbf{K}_h, \mathbf{V}_h) = \text{softmax}(\frac{\mathbf{Q}_h\mathbf{K}_h^T}{\sqrt{d_k}})\mathbf{V}_h \tag{3}$$

The output of the MHA layer is calculated as follows

$$\text{MultiHeadAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\mathbf{O}_1, \cdots, \mathbf{O}_\text{h}]\mathbf{W}^O \tag{4}$$

where $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$.

Finally, the MHA output is passed to a Position-wise Feed-forward Network (FFN) with two dense layers (fully-connected). The first dense projects d_{model} to a given d_{ff} using a ReLU activation, while the second dense returns the projection to d_{model}. Additional *Add and Layer-normalization* layers provide residual connections that allow the gradients to skip the MHA and/or FFN layers if they do not improve the model.

3.3 Tensor Processing Units

A TPU is an application-specific integrated circuit (ASIC) developed by Google to accelerate large-scale ML. Google Cloud (GC) first released TPU v1 in 2018, and at the time of this writing, its highest publicly available version is TPU v3. Each one of the 8 cores in a TPU v3 board provides a Matrix Multiplier Unit (MXU) with 65,536 8-bit multiply-and-add units, a Unified Buffer (UB) with 24MB of SRAM, and an Activation Unit (AU) with hardwired activation functions. These computational resources are designed to process basic neural network-related operations at a very low level: the MXU performs matrix operations, while the UB holds intermediate results, and the AU performs the nonlinear operations over the MXU output [28]. A Complex Instruction Set Computer (CISC) controls the TPU resources with an instruction set also designed for neural network-specific operations. The core of TPU logic is a *systolic array* [29]. This chip architecture accelerates matrix multiplications on the MXU by making input values flow through fixed patterns of multiply-and-add units, substantially reducing read-write operations to the UB (see Figure 3). TPU ML-accelerators are only available via Cloud TPU, a high-performance computing service of GC.

Cloud TPU provides a TPU board connected via PCI to a host virtual machine (VM). Inside this combination, the TPU runs the consuming model tranining and evaluation stages in parallel. TPU computing performance is extremely high (up to 420 TFLOPS) and data infeed usually becomes an operation bottleneck. For this reason, the host VM is used to execute code that feeds data to the TPU as fast as possible. TPU hardware is language-agnostic, so an Accelerated Linear Algebra (XLA) compiler maps the ML-code representations from a specific programming language to TPU machine code. The XLA compiler is designed for linear algebra and vector computations; any programming construct that relies on different operations has to be directed to the host VM's CPU. The MXUs of a TPU can perform at *mixed precision*, combining different numerical formats in the same computational workload. TPU cores can use `bfloat16` format for multiplications and `float32` format for results accumulation. As most deep learning applications do not need a high precision to get the target application accuracy [17], floating point operations, and consequently the training process, are accelerated if weights, gradients,

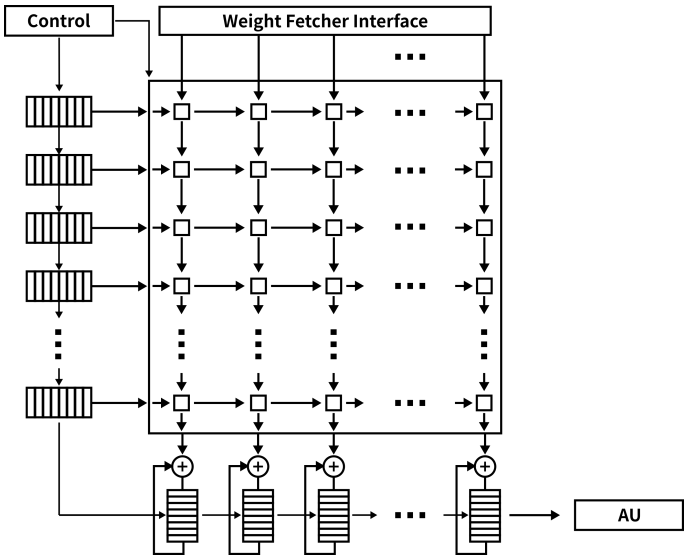

Fig. 3 Systolic data flow of the MXU. Weights are pre-loaded from the Weight Fetcher Interface into the computation cells inside the MXU. Data flows in at regular intervals from the vertical stack of systolic data setup blocks (left). A given 256-element multiply-accumulate operation moves through the matrix as a diagonal wavefront. Partial sums are collected on the horizontal stack of accumulators (bottom). Results flow out the MXU at the bottom-right corner into the Activation Unit [28].

data, and activations are represented in `bfloat16` format. Figure 4 compares float32, bfloat16, and float16 numerical formats.

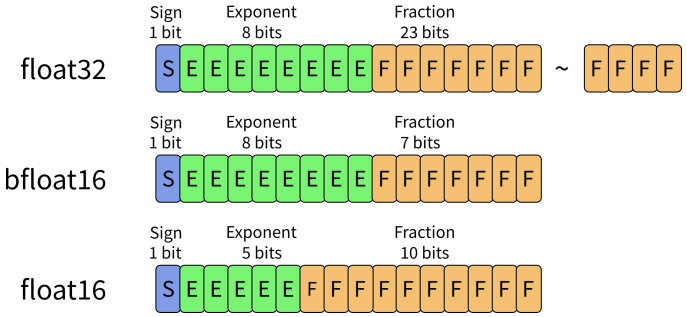

Fig. 4 Comparison of the float32, bfloat16, and float16 numerical formats. The bfloat16 format implements the same range as the float32 format but with a lower precision. TPUs build mixed precision as a combination of bfloat16 and float32, while GPUs commonly used for ML workloads use a combination of float16 and float32.

3.4 Implementation

Figure 5 shows the cloud-based TPU computing pattern implemented for training the transformer. We programmed our transformer with TensorFlow [30] as

front end, based on the Multi-head Attention (MHA) layer from TensorFlow AddOns. This layer is a subclass of the Keras Layer class that expresses the query, key, and value tensors as linear transformations based on the `tf.einsum` tensor contraction. The encoder and decoder layers that integrate the transformer are also defined as subclasses of Keras Layer. To improve the TPU performance and to ensure the model scalability, the transformer was coded using `TPUEstimator`, a high-level Application Programming Interface (API) that simplifies TPU usage by handling numerous low-level, hardware-specific details.

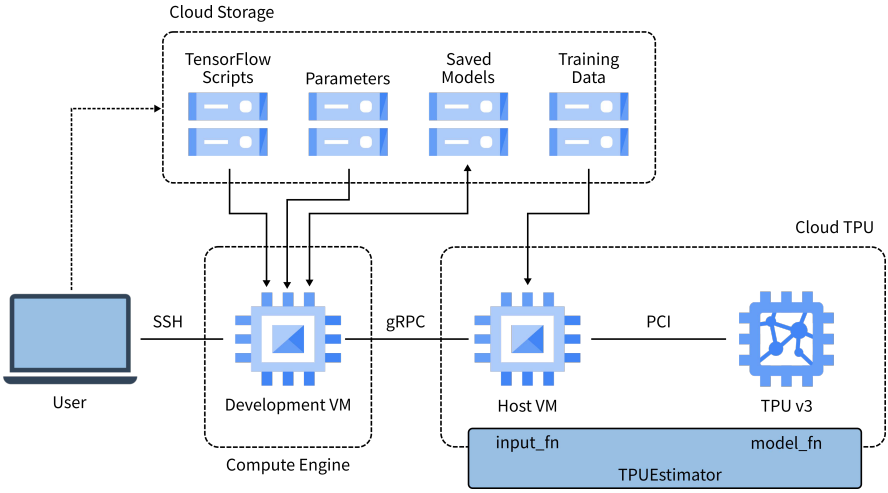

Fig. 5 Cloud-based TPU computing pattern for training the transformer. The TensorFlow TPUEstimator API allows decoupling the `input_fn` from the `model_fn` for CPU-based execution of the former and TPU-based execution of the latter. Dotted boxes represent cloud computing services.

`TPUEstimator` is built on the basis of `Estimator`, a high-level API for TensorFlow. `TPUEstimator` presents an interface similar to `Scikit-learn`, and some adaptations to simplify its deployment to production stages [31]. The `Estimator` class is the base for executing the different stages of the ML model (training, evaluation, and prediction) from the same base-code expressed in a `model_fn` function. A user-defined function called `input_fn` produces the input for the `Estimator`, decoupled from the `model_fn`. Based on this decoupling, the TPU runs the `model_fn`, which encloses anything involving trainable variables or gradients. The host VM's CPU runs the `input_fn`, which focuses on data preparation and infeed. The `TPUEstimator` API implements *all-reduce* syncrhronous data parallelism, which means it evenly splits the training batch among the workers (cores) of the TPU board. Conversely, the ML model is not distributed and every worker keeps a replication of the TensorFlow computation graph. Workers operate synchronously, with each worker performing the same step simultaneously. A reduction operation is performed across all the

workers once the training step is completed. For this cloud-based implementation of the transformer, we built an advanced architecture by leveraging several services of GC. A development VM retrieves TensorFlow scripts and parameters from cloud storage, spins-up a Cloud TPU as a training co-processor, sends the TensorFlow computation graph to the TPU's host VM and, during training, writes the resulting serialized model back to cloud storage. TPU's host VM pulls data directly from cloud storage and distributes it to the TPU cores.

3.5 Experimental Study

This subsection covers the datasets, data preprocessing, model parametrization, and testing metrics used in our experimental study.

3.5.1 Datasets and Data Preprocessing

We trained our transformer on two widely-known datasets: `electricity`[3] that contains hourly time series of the electricity consumption of 370 customers, and `traffic`[4] which is composed of 963 time series with hourly readings of car lane occupancy of San Francisco bay area freeways. Figure 6 illustrates the data setup of the transformer. Dynamic inputs to the model include all of the univariate time series z_t, and, for each time series, two positional encoding functions $f_1(x_{k,t})$ and $f_2(x_{k,t}), k \in [1, j]$ of j time-dependent covariates, where f_1 and f_2 define sine- and cosine-based cycles over the period length of the covariate, e.g. 24 for the hour of day. The part of the time series to the left of the green vertical line is the data that is presented to the model for training. The part to the right of the green line is the data used to test the model once it is trained. The bisected red horizontal lines represent time windows used as data examples: the left section of the red line is the data passed as input to the model (*conditioning range*), while the right section is the data used as target or output from the model (*prediction range*). (a) During training, time windows are required to completely lie to the left of the green line, as only *seen data* can be used for training or evaluation purposes. (b) During forecasting, the prediction range of the data examples passed to test the model must entirely lie to the right of the green line (*unseen data*).

Data setup is completed with two additional components (not shown in the figure): a time-dependent covariate with the consecutive value of hour on the observation timestamp measured from the time series starting point (*age-covariate*), and a scalar time series identifier which is passed as a static covariate for the model, then conformed into a learnable embedding which provides a finite-dimensionality space for time series with similar behavior to group. Datasets were split for training and test stages using date ranges as in [32]. Min-max normalization to $[0, 1]$ was applied to both datasets, in order to compensate for the large difference in time series values in `electricity`, and to

[3] http://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014
[4] http://archive.ics.uci.edu/ml/datasets/PEMS-SF

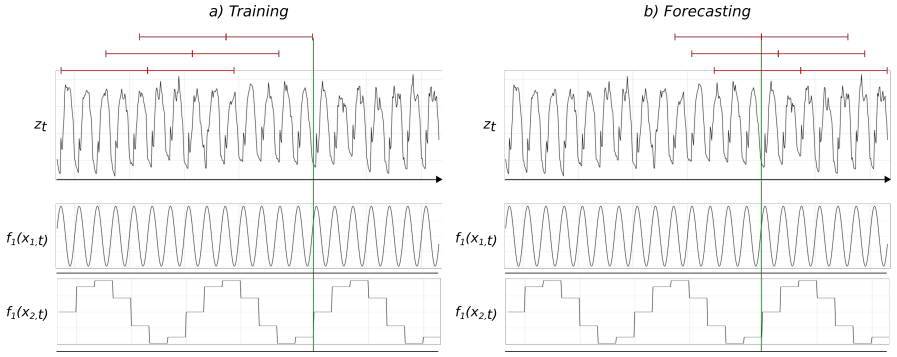

Fig. 6 Data setup for the transformer model. In the first row, a single univariate time series to forecast z_t. In the second and third rows, two exemplary sine-based positional encodings based on the hour of the day $x_{1,t}$ and the day of the week $x_{2,t}$ of the observation timestamp. Figure based on [32]

speed up the model convergence in `traffic`. Proper normalization scalers were acquired only from training data to avoid leaking information from the future into the model. Once preprocessed and separated, data was persisted to cloud storage as sequences of binary strings (`TFRecords`) in accordance with best-TPU-performance guidelines. A data ingestion pipeline for training the model was defined on the basis of the `tf.data.dataset` TensorFlow API. Prefetchig was also used for parallel preprocessing, that means to start preparing the next batch while still training the current batch. To ensure good throughput during ingestion from cloud storage, individual `TFRecord` files were prepared for each time series identifier, resulting in 370 50-MB files for `electricity` and 963 33-MB files for `traffic`.

3.5.2 Model Parametrization

The transformer was trained using the Adam optimizer [33] with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 10^{-8}$, and a custom learning rate $lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$ which results in increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number [2]. The Mean Squared Error (MSE) was used as the training loss function, defined as $MSE = \frac{1}{|\Omega_{train}|} \sum_{(i,t)\in\Omega_{train}} |\hat{z}_{i,t} - z_{i,t}|^2$ where Ω_{train} is the collection of readings for all t timesteps in all i time series in the training dataset, and $|\Omega_{train}|$ is the number of readings in that collection. The dynamics of the used learning rate schedule is shown in Figure 7.

3.5.3 Inference

After training the transformer, we tested it using 24 hours (one day) and 168 hours (one week) as forecasting horizons. To exactly match the specific one-week inference interval reported by the reference models, 7 rolling-day

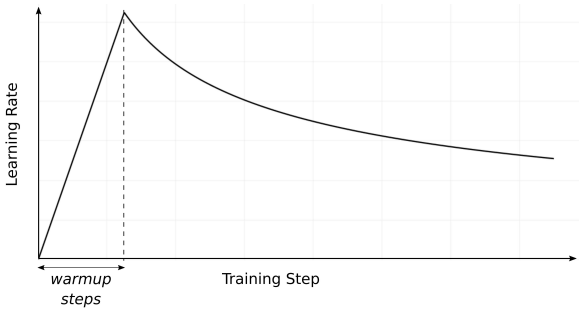

Fig. 7 Learning rate schedule for the Transformer model.

predictions of one day ahead were produced on each dataset. Following the convention in literature we labeled these inference results as $\texttt{electricity}_{1d}$ and $\texttt{traffic}_{1d}$. In order to study the predictive performance of our model on a longer horizon, we also produced one-time predictions of seven days ahead for each dataset, and labeled them as $\texttt{electricity}_{7d}$ and $\texttt{traffic}_{7d}$. Normalized Deviation (ND) and Normalized Root Mean Squared Error (NRMSE) were used as metrics for testing the transformer model, and are defined as follows:

$$ND = \frac{\sum_{(i,t)\in\Omega_{test}} |\hat{z}_{i,t} - z_{i,t}|}{\sum_{(i,t)\in\Omega_{test}} |z_{i,t}|} \qquad NRMSE = \frac{\sqrt{\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}} |\hat{z}_{i,t} - z_{i,t}|^2}}{\frac{1}{|\Omega_{test}|}\sum_{(i,t)\in\Omega_{test}} |z_{i,t}|}$$

where Ω_{test} is the collection of readings for all t timesteps in all i time series in the test dataset, and $|\Omega_{test}|$ is the number of readings in that collection. Table 1 shows the complete dataset statistics as well as the architecture, training, and testing parameters used in the transformer.

4 Results and Discusion

This section presents the results of our experiments with the transformer. First, two well-renowned TSF models are presented as the baseline for evaluation. Next, forecasting accuracy and computation times obtained are compared with the baseline.

4.1 Baseline

Two robust DL-based models were used as baseline to evaluate the performance of the transformer: (a) `DeepAR` [32] is an autoregressive model for probabilistic forecasting with a RNN-based architecture, and (b) `TRMF` [34] is a temporal regularized matrix factorization framework for high-dimensional time series problems (including TSF) with missing values. `DeepAR` represents

Table 1 Dataset statistics and Transformer parameters.

	`electricity`	`traffic`
dataset statistics		
# of time series	370	963
time granularity	hour	hour
domain	$\mathbb{R}^+$	$[0, 1]$
architecture parameters		
encoder length	168	168
decoder length	168	168
input embedding dimension	370	963
ouput embedding dimension	24	24
# of layers	2	2
# of heads	4	4
model dimension	256	256
feed-forward dimension	512	512
training parameters		
# of training samples	1.8 M	3.1 M
batch size	[128, 256, 512, 1024]	[128, 256, 512, 1024]
warmup steps	5000	5000
# of epochs	1.07	0.82
testing parameters		
forecasting horizon	[24, 168]	[24, 168]

a very useful baseline for our transformer because it not only provides a reference for point-forecasting accuracy but also for computation times on the selected datasets.

4.2 Forecasting Accuracy

Our transformer consistently reaches convergence at a very fast pace, after training it for roughly one epoch on the complete dataset, that is 1.8 million samples for `electricity` and 3.1 million samples for `traffic`. Our model outperforms `TRMF` and `DeepAR` on $\texttt{electricity}_{1d}$ (Table 2) after 1.07 training epochs.

Table 2 Forecasting accuracy comparison for $\texttt{electricity}_{1d}$ (rolling-day prediction of 7 days).

Metric	`TRMF`	`DeepAR`	Ours
ND ↓	0.16[1]	**0.07**[1]	**0.070** ± 0.004
NRMSE ↓	1.15[1]	1.00[1]	**0.546** ± 0.049

[1]Results from [32].

Concerning $\texttt{traffic}_{1d}$, our model outperforms TRMF and ties DeepAR (Table 3) after 0.82 training epochs. The fact that our model achieves a good predictive performance on traffic even though the dataset was not entirely passed to the model during training, supports the assumption that this transformer is able to learn patterns across time series that exhibit similar behavior.

Table 3 Forecasting accuracy comparison for $\texttt{traffic}_{1d}$ (rolling-day prediction of 7 days).

Metric	TRMF	DeepAR	Ours
ND ↓	0.20[1]	**0.17**[1]	**0.167** ± 0.019
NRMSE ↓	0.43[1]	**0.42**[1]	**0.422** ± 0.015

[1]Results from [32].

Figures 8 and 9 show the test metrics for different training batch sizes of our transformer on $\texttt{electricity}_{1d}$ and $\texttt{traffic}_{1d}$, respectively. On both datasets and for the two selected inference metrics, the transformer produces the best results when trained with a batch size of 256.

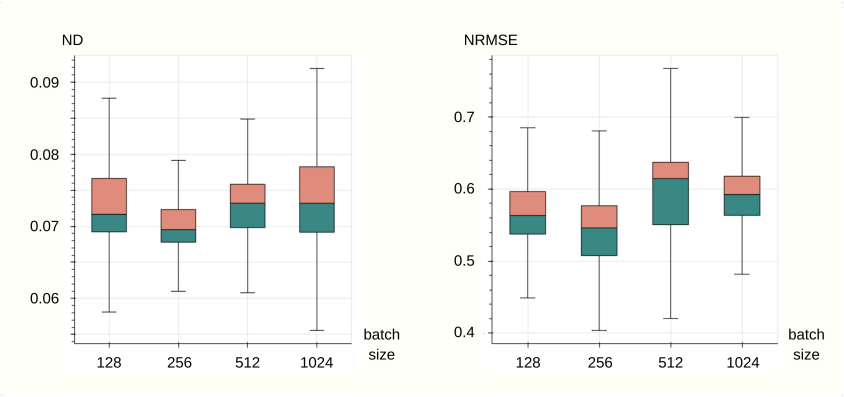

Fig. 8 Test metrics for $\texttt{electricity}_{1d}$ (rolling-day prediction of 7 days) at different training batch sizes. Best values for both ND and NRMSE are obtained with a batch size of 256.

Regarding the seven-day forecasting horizon, our transformer outperforms TRMF and ties DeepAR on $\texttt{electricity}_{7d}$, as shown in Table 4. Our model exhibits predictive performance degradation due to error accumulation on the iterative inference process for $\texttt{traffic}_{7d}$, which suggests the need for training for more epochs if longer forecasting horizons are planned for this dataset. Figure 10 shows that the best inference metrics for a seven-day forecasting horizon are also produced by the transformer trained with a batch size of 256.

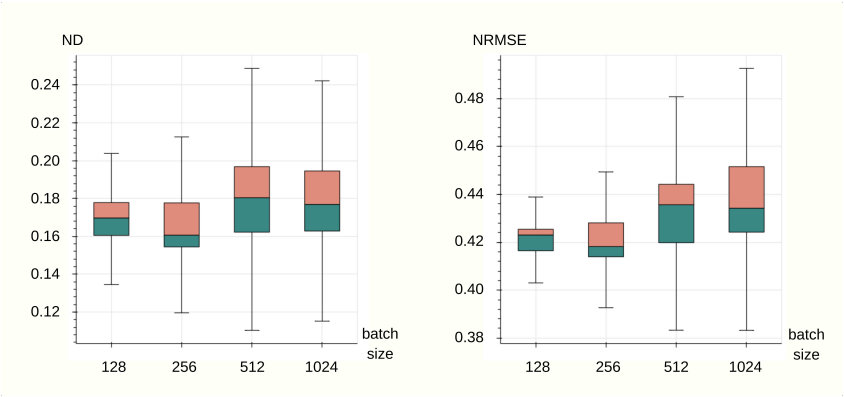

Fig. 9 Test metrics for $\texttt{traffic}_{1d}$ (rolling-day prediction of 7 days) at different training batch sizes. Best values for both ND and NRMSE are obtained with a batch size of 256.

Table 4 Forecasting accuracy comparison (ND values only) for $\texttt{electricity}_{7d}$ and $\texttt{traffic}_{7d}$ (one-time prediction of 7 days ahead).

Dataset	TRMF	DeepAR	Ours
$\texttt{electricity}_{7d}$	0.087[1]	**0.082**[1]	**0.082** $\pm$ 0.006
$\texttt{traffic}_{7d}$	0.202[1]	**0.179**[1]	0.203 $\pm$ 0.032

[1]Results from [8].

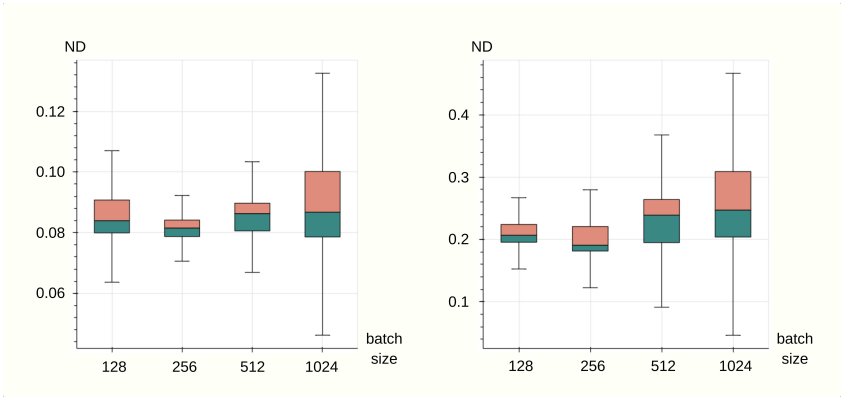

Fig. 10 Test metrics (ND only) for $\texttt{electricity}_{7d}$ (left) and $\texttt{traffic}_{7d}$ (right), one-time prediction of 7 days ahead at different training batch sizes. Best ND values for both datasets are obtained with a batch size of 256.

4.3 Computation Time

Concerning computation time, our transformer clearly outperforms the models in the baseline, as shown in Table 5. Our model takes only 89 seconds of training wall-time[5] on `electricity` and 119 seconds on `traffic` to consistently reach the accuracy metrics reported above. This represents an outstanding improvement over the 7 hours and 3 hours of total running time reported by `DeepAR`, respectively. The extreme reduction on training computation time achieved by our model does not exclusively apply to `DeepAR`, a DL architecture that exhibits a very close predictive performance to ours. The `Temporal Fusion Transformer` (TFT) [7] reported a training computation time, on GPU-accelerated hardware, that is slightly over 360 minutes, for the `electricity` dataset. In the same line, training times close to 480 minutes are reported for both `LogSparse Transformer` and `Informer` in [9], when these models are trained with the encoder length set to 168 (as in our main experiment) on `electricity` and three other similar datasets, using GPU-accelerated hardware. It is important to emphazise that we did not attempt a direct comparison with the `TFT` or `LogSparse` models, as their functionalities are very different, and their reported forecasting accuracies are superior than the ones achieved by our transformer, so far. Instead we use their reported results to illustrate the huge training times usually required by most DL-based, hardware-accelerated TSF models to deal with a problem we solved in under 2 minutes wall-time. Inference computation times for our transformer are highly dependent on the infrastructure where prediction is served and, due to the iterative nature of our model, this inference process is not suitable for TPU-based implementation. Instead, a parallel execution based on separated time series processing was prototyped to obtain inference computation times that are comparable in magnitude to those required by training.

Table 5 Computation time comparison.

Dataset	`TRMF`	`DeepAR`	Ours
`electricity`[2][3]	-	420 min[1]	**1.5 min**
`traffic`	-	180 min[1]	**2.0 min**

[1]Results from [32]. Reported as *total running time*, it is assumed it includes training and inference stages. TRMF computation times not available.

[2]A training computation time (GPU-accelerated) over 360 minutes is reported for `TFT` in [7] ror this dataset.

[3]Training times close to 480 minutes (GPU-accelerated) are reported for `LogSparse Transformer` and `Informer` in [9] for this dataset and three other similar datasets.

[5]Elapsed real time or wall-clock time, is the actual time taken to complete the training process. Training wall time is reported by Tensorboard and does not include the time spent on evaluation or checkpoint-related operations.

5 Conclusions and Future Work

Real-world TSF applications demand ML-based models to be not only accurate, but also fast. Low-complexity, sparse variants of the VT have consistently advanced the state-of-the-art on TSF but do not consistently report their speed metrics. This situation hinders the analysis on the accuracy-speed trade-off of ML-based models, which is an invaluable tool for operating TSF processes on modern production environments. In this work, we present in detail the accelerated training of a transformer for global, multi-horizon TSF. We implemented our transformer on a cloud computing ecosystem that follows the best practices for TPU operation, such as decoupled input- and model-function, model training with mixed-precision arithmetics, cloud-based artifact storage, and efficient serialization (TFRecord file-based) for data infeed. Experiments on two widely-known, large-scale datasets show that our dense-matrix, full self-attention, multi-layer, encoder-decoder transformer outperforms two close to the state-of-the-art reference models in accuracy, while reducing training computation times from several hours to under 2 minutes wall-time. As the training computation times of the models used as baseline in this paper are comparable to most of the DL models currently proposed to solve the global, multi-horizon TSF problem, our work provides a basis for further research aimed to produce a more balanced comparison, in terms of both predictive performance and speed, of the original VT and its sparse variants. Future work in this direction includes the following two research lines: adding sparsity and enhanced locality to our transformer's self-attention mechanism via the design of convolutional filters (as in [8]) and activation functions (as in [14]); and downsampling the input sequence as it passes through the model's self-attention layers (as in [9]). These contributions have already proved to increase the transformer accuracy on TSF problems, therefore they constitute a good benchmark to evaluate the impact that moving the computation process outside the dense-matrix domain will have in the speed of our TPU-accelerated transformer.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

The datasets analysed during the current study are available in the UCI Machine Learning Repository, at http://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

(`electricity`), and http://archive.ics.uci.edu/ml/datasets/PEMS-SF (`traffic`). The datasets generated during the current study are available on request from the corresponding author.

Competing interests

All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

Funding

J. Luis García-Nava's Sc.D. program is supported by a National Scholarship granted by the Consejo Nacional de Ciencia y Tecnología (National Council of Science and Technology) under the CVU No. 737505. Victor M. Tellez' Sc.D. program is supported by a National Scholarship granted by the Consejo Nacional de Ciencia y Tecnología (National Council of Science and Technology) under the CVU No. 816803.

Authors' contributions

All authors contributed to the research conceptualization and design. Data were collected, reviewed and prepared by J. Luis García-Nava, Juan J. Flores, and Victor M. Tellez. Software was developed, deployed in the cloud, and executed by J. Luis García-Nava. Results analysis was performed by J. Luis García-Nava, Juan J. Flores, and Felix Calderon. The first draft of the manuscript was written by J. Luis García-Nava and Juan J. Flores. All authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Acknowledgments

This research was supported by the Consejo Nacional de Ciencia y Tecnología (National Council of Science and Technology) of Mexico.

References

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

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. Advances in Neural Information Processing Systems **30** (2017)

[3] Khan, S., Naseer, M., Hayat, M., Zamir, S.W., Khan, F.S., Shah, M.: Transformers in vision: A survey. ACM Computing Surveys (CSUR) (2021)

[4] Lin, T., Wang, Y., Liu, X., Qiu, X.: A Survey of Transformers. arXiv (2021). https://doi.org/10.48550/ARXIV.2106.04554. https://arxiv.org/abs/2106.04554

[5] Tay, Y., Dehghani, M., Bahri, D., Metzler, D.: Efficient transformers: A survey. ACM Computing Surveys (CSUR) (2020)

[6] Hewamalage, H., Bergmeir, C., Bandara, K.: Global models for time series forecasting: A simulation study. Pattern Recognition, 108441 (2021)

[7] Lim, B., Arık, S.Ö., Loeff, N., Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting (2021)

[8] Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., Yan, X.: Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. Advances in Neural Information Processing Systems **32**, 5243–5253 (2019)

[9] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W.: Informer: Beyond efficient transformer for long sequence time-series forecasting. In: Proceedings of AAAI (2021)

[10] Taieb, S.B., Bontempi, G., Atiya, A.F., Sorjamaa, A.: A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. Expert systems with applications **39**(8), 7067–7083 (2012)

[11] Fournier, Q., Caron, G.M., Aloise, D.: A Practical Survey on Faster and Lighter Transformers. arXiv (2021). https://doi.org/10.48550/ARXIV.2103.14636. https://arxiv.org/abs/2103.14636

[12] He, X., Pal, S., Amarnath, A., Feng, S., Park, D.-H., Rovinski, A., Ye, H., Chen, Y., Dreslinski, R., Mudge, T.: Sparse-TPU: Adapting systolic arrays for sparse matrices. In: Proceedings of the 34th ACM International Conference on Supercomputing, pp. 1–12 (2020)

[13] Wu, N., Green, B., Ben, X., O'Banion, S.: Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case. arXiv (2020). https://doi.org/10.48550/ARXIV.2001.08317. https://arxiv.org/abs/2001.08317

[14] Wu, S., Xiao, X., Ding, Q., Zhao, P., Wei, Y., Huang, J.: Adversarial sparse transformer for time series forecasting. Advances in Neural Information Processing Systems **33** (2020)

[15] Xu, J., Wang, J., Long, M., et al.: Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Advances in Neural Information Processing Systems **34** (2021)

[16] Woo, G., Liu, C., Sahoo, D., Kumar, A., Hoi, S.: ETSformer: Exponential Smoothing Transformers for Time-series Forecasting. arXiv (2022). https://doi.org/10.48550/ARXIV.2202.01381. https://arxiv.org/abs/2202.01381

[17] You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., Keutzer, K.: Fast deep neural network training on distributed systems and Cloud TPUs. IEEE Transactions on Parallel and Distributed Systems **30**(11), 2449–2462 (2019)

[18] Wongpanich, A., Pham, H., Demmel, J., Tan, M., Le, Q., You, Y., Kumar, S.: Training EfficientNets at supercomputer scale: 83% ImageNet top-1 accuracy in one hour. In: 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 947–950 (2021). IEEE

[19] Sheng, A., He, J.Y.D.: Distributed evolution strategies using tpus for meta-learning. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 721–728 (2020). IEEE

[20] Pan, Z., Mishra, P.: Hardware acceleration of explainable machine learning. In: 2022 Design, Automation And Test in Europe Conference And Exhibition DATE, pp. 1127–1130 (2022). https://doi.org/10.23919/DATE54114.2022.9774739

[21] Hauru, M., Morningstar, A., Beall, J., Ganahl, M., Lewis, A., Vidal, G.: Simulation of quantum physics with Tensor Processing Units: brute-force computation of ground states and time evolution. arXiv (2021). https://doi.org/10.48550/ARXIV.2111.10466. https://arxiv.org/abs/2111.10466

[22] Xu, H., Van Durme, B., Murray, K.: BERT, mBERT, or BiBERT? a study on contextualized embeddings for neural machine translation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6663–6675 (2021)

[23] Alrowili, S., Vijay-Shanker, K.: BioM-transformers: building large biomedical language models with BERT, ALBERT and ELECTRA. In: Proceedings of the 20th Workshop on Biomedical Language Processing, pp. 221–227 (2021)

[24] Zhao, L., Zhang, Z., Chen, T., Metaxas, D., Zhang, H.: Improved transformer for high-resolution GANs. Advances in Neural Information Processing Systems **34**, 18367–18380 (2021)

[25] Srinivas, A., Lin, T.-Y., Parmar, N., Shlens, J., Abbeel, P., Vaswani, A.: Bottleneck transformers for visual recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 16519–16529 (2021)

[26] Abdellaoui, I.A., Mehrkanoon, S.: Deep multi-stations weather forecasting: explainable recurrent convolutional neural networks. arXiv (2020). https://doi.org/10.48550/ARXIV.2009.11239. https://arxiv.org/abs/2009.11239

[27] Abdellaoui, I.A., Mehrkanoon, S.: Symbolic regression for scientific discovery: an application to wind speed forecasting. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 01–08 (2021). IEEE

[28] Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., *et al.*: In-datacenter performance analysis of a Tensor Processing Unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 1–12 (2017)

[29] Kung, H., Leiserson, C.E.: Systolic arrays (for VLSI). In: Sparse Matrix Proceedings 1978, vol. 1, pp. 256–282 (1979). Society for industrial and applied mathematics

[30] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.*: TensorFlow: A system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283 (2016)

[31] Cheng, H.-T., Haque, Z., Hong, L., Ispir, M., Mewald, C., Polosukhin, I., Roumpos, G., Sculley, D., Smith, J., Soergel, D., *et al.*: TensorFlow estimators: Managing simplicity vs. flexibility in high-level machine learning frameworks. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1763–1771 (2017)

[32] Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181–1191 (2020)

[33] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA,May 7-9, 2015, Conference Track Proceedings (2015)

[34] Yu, H.-F., Rao, N., Dhillon, I.S.: Temporal regularized matrix factorization for high-dimensional time series prediction. Advances in Neural Information Processing Systems **29**, 847–855 (2016)