Applied Intelligence (2023) 53:26781–26796
https://doi.org/10.1007/s10489-023-04927-4

Persistence Initialization: a novel adaptation of the Transformer architecture for time series forecasting

Espen Haugsdal[1] · **Erlend Aune**[1,2] · **Massimiliano Ruocco**[1,3]

Accepted: 29 July 2023 / Published online: 29 August 2023

Abstract
Time series forecasting is an important problem, with many real world applications. Transformer models have been successfully applied to natural language processing tasks, but have received relatively little attention for time series forecasting. Motivated by the differences between classification tasks and forecasting, we propose *PI-Transformer*, an adaptation of the Transformer architecture designed for time series forecasting, consisting of three parts: First, we propose a novel initialization method called *Persistence Initialization*, with the goal of increasing training stability of forecasting models by ensuring that the initial outputs of an untrained model are identical to the outputs of a simple baseline model. Second, we use ReZero normalization instead of Layer Normalization, in order to further tackle issues related to training stability. Third, we use Rotary positional encodings to provide a better inductive bias for forecasting. Multiple ablation studies show that the PI-Transformer is more accurate, learns faster, and scales better than regular Transformer models. Finally, PI-Transformer achieves competitive performance on the challenging M4 dataset, both when compared to the current state of the art, and to recently proposed Transformer models for time series forecasting.

Keywords Transformer · Time series forecasting · M4 competition · Deep neural networks

1 Introduction

The ability to forecast the future is a valuable tool across a wide range of applications, such as finance, energy, and industry. Forecasting allows for better decision-making in the present, and even small improvements in accuracy can often provide great benefits.

The Transformer [1] has recently become the dominant method for most Natural Language Processing (NLP) tasks [2, 3]. It has also been successfully applied to a diverse set of challenging problems outside of NLP, such as protein folding [4] and Reinforcement Learning [5]. However, relatively little attention has been given to the use of Transformers for time series forecasting. Most prior work in this direction has focused on the addressing the computational limitations of the Transformer, by proposing computationally efficient alternatives to regular attention [6–8]. In contrast, this work is primarily focused on improving the forecasting accuracy of the Transformer on time series with shorter forecasting horizons, by addressing differences between time series data and text data.

Time series forecasting and natural language modeling might at first glance appear to be highly similar; they both form ordered sequences, and forecasting the next step of a time series can be seen as analogous to predicting the next token in a language modeling task.

However, there are also important differences between time series data and text data. First, time series forecasting is a continuous regression problem, while language modeling is a discrete classification problem. Consequently, in order to use the Transformer to forecast, the final softmax activation layer must be removed. We argue that this makes the model more sensitive to how the weights are initialized, as the initial forecasts will now be proportional to the weights

✉ Espen Haugsdal
espen.haugsdal@ntnu.no

Erlend Aune
erlend.aune@ntnu.no

Massimiliano Ruocco
massimiliano.ruocco@ntnu.no

[1] Norwegian University of Science and Technology, Trondheim, Norway

[2] BI Norwegian Business School, Oslo, Norway

[3] Sintef Digital, Trondheim, Norway

of the final linear layer. Conversely, models including a final softmax layer are likely to be more robust to weight initialization. A random initialization of such a model likely has an approximately uniform output distribution, which is arguably a good starting point for a reasonably balanced classification task. Second, time series data often do not have any particular semantics associated with the beginning or conclusion of a sequence. In other words, for time series data, there is in general no reason to assume any meaning from the fact that the sequence started or ended at some particular point in time. As a consequence, time series sequences can often be subdivided into smaller sub-sequences, a technique which is commonly referred to as *windowing*. In contrast, for text sequences, both the start and the end of a sequence has semantic meaning, because the start and end of the sequence signifies the bounds of a connected body of text.

We propose three modifications of the Transformer architecture, directly motivated by these differences. First, we propose an adaptation called *Persistence Initialization* (PI), which aims to improve the Transformer's ability to forecast. It has long been known that initialization is an important component in the process of training deep neural networks [9–11]. Persistence Initialization works by implicitly initializing the model in such a way that the initial forecasts (before training) become equal to the forecasts of a *persistence model*. The persistence model, also known as a random walk method [12], is defined by letting the forecast $\hat{x}_{t+1}$ be equal to the previous value x_t. In order to implement PI, we add two components: a residual skip connection, and a scalar multiplicative gating parameter γ. The residual skip connection has the effect of adding the value at time t (i.e. x_t) to the forecast value for time $t+1$ (i.e. $\hat{x}_{t+1}$). The scalar multiplicative gating parameter γ is initialized to 0, and is multiplied with the outputs of the Transformer. As a consequence of this combination, only the skip connection contributes to the initial forecasts, which means that any complex model can be effectively initialized as a persistence model, regardless of the values of the randomly initialized parameters within the model.

Our second proposed adaption attempts to further improve training stability by replacing the commonly used Layer Normalization [13] layer with ReZero normalization [14]. ReZero is a technique designed to improve the training stability of deep networks, and was proposed as an alternative to normalization layers such as Layer Norm and Batch norm. Note that while the implementation of Persistence Initialization is almost identical to that of ReZero, these techniques are intended to solve different problems. The goal of ReZero normalization is to control the magnitude of gradients in deep networks, while The goal of Persistence Initialization is to improve the Transformer's forecasting accuracy by providing an inductive bias towards models with a significant autoregressive component.

Our third proposed adaptation is related to the difference in the semantics of the time series sequences, compared to natural language sequences. Instead of using the absolute sinusoidal encoding [1], we propose to use the relative Rotary Encoding [15], which has been shown to outperform the sinusoidal encoding in some NLP tasks [15]. In the context of time series, we argue that a relative positional encoding provides a better inductive bias for forecasting. Time series sequences are often "windowed", which means that the absolute position within the window has no semantic significance. Consequently, absolute positional encodings are ill-suited for forecasting, as they put undue emphasis on an arbitrary location in the sequence. In contrast, a relative encoding emphasizes the position of the outputs, i.e. the forecasts, which should result in a better inductive bias for forecasting.

In summary, our contributions are:

1. We propose Persistence Initialization, a novel and general adaptation autoregressive for time series forecasting with neural networks. This adaptation initializes the model such that it starts off as a persistence model, which provides a good starting point for further learning.
2. We propose the PI-Transformer architecture, a Transformer architecture with three main modifications: Persistence Initialization, ReZero normalization, and Rotary positional encodings. We perform two ablation studies to verify the importance of each modification. The first ablation study compares the effects of the components of Persistence Initialization, and the second compares the effect of positional encoding and normalization layers. Both studies show that the proposed modifications are necessary for good forecasting performance.
3. We evaluate PI-Transformer on the challenging M4 forecasting dataset, and show that PI-Transformer achieves competitive accuracy, outperforming the winner of the original M4 competition. Furthermore, PI-Transformer is highly accurate without the need for a large ensemble of models, in contrast to other state-of-the-art methods on the M4 dataset. To the best of our knowledge, this is the first time a Transformer model has been successfully used to forecast the complete M4 dataset to a high degree of accuracy. We also compare PI-Transformer with recent existing Transformer architectures for time series forecasting, and show that PI-Transformer outperforms these by a large margin on the M4-Hourly dataset.

In order to ensure reproducibility, all the code related to our work is publicly available[1]. The rest of the paper is orga-

[1] A public implementation is available at https://github.com/EspenHa/persistence_initialization.

nized as follows: Section 2 provides some background on the Transformer, Section 4 describes our proposed adaptation, Section 3 reviews existing related work, Section 5 describes the experiments, and Section 6 provides analysis and discussion of the results. Finally, Section 7 concludes with a summary.

2 Background

2.1 Decoder-only transformer

A decoder-only Transformer [1] consists of blocks of causal self-attention layers and feedforward layers, both followed by a residual skip connection and Layer-Normalization [13]. The model can be defined recursively by letting X_i be the output of the ith block, as follows:

$$X_i(X_{i-1}) = \mathrm{FF}_i(\mathrm{SA}_i(X_{i-1})) \quad (1)$$

$$\mathrm{SA}_i(X) = \mathrm{LayerNorm}(X + \mathrm{SelfAttention}_i(X)) \quad (2)$$

$$\mathrm{FF}_i(X) = \mathrm{LayerNorm}(X + \mathrm{FeedForward}_i(X)), \quad (3)$$

where X_i is a matrix of shape $L \times d_{\text{model}}$, with L representing the "sequence" or "time" dimension and d_{model} representing the feature dimension. X_0 is the base case of the recursion, and represents the initial input to the model. The number of blocks N is a hyperparameter which determines the final output of the model, X_N.

In order to define self-attention, we must first define multi-head attention. Multi-head attention combines multiple attention heads, by giving each head a separate set of learnable weights, ensuring that each head can perform a different operation. Self-attention is then defined as a special case of multi-head attention where the keys, queries, and values are all equal:

$$\mathrm{SelfAttention}(x) = \mathrm{MHA}(X, X, X) \quad (4)$$

$$\mathrm{MHA}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W_O \quad (5)$$

$$\mathrm{head}_j = \mathrm{Attention}\left(QW_Q^{(j)}, KW_K^{(j)}, VW_V^{(j)}\right) \quad (6)$$

$$\mathrm{Attention}(Q,\ K,\ V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_{\text{head}}}} + M\right)V, \quad (7)$$

where h is the number of attention heads, and $d_{\text{head}} = d_{\text{model}}/h$. The learnable weight matrices $W_Q^{(j)}$, $W_K^{(j)}$, and $W_V^{(j)}$ are of shape $d_{\text{model}} \times d_{\text{head}}$. W_O is a learnable weight matrix of shape $d_{\text{model}} \times d_{\text{model}}$, and has the effect of mixing the outputs of each head. M is an upper triangular masking matrix, which ensures that the model does not attend to "future" time steps.

The feed-forward layer is performed point-wise, i.e. it only considers information in the current time step, like a 1-D convolution. It is defined as two affine transformations with a ReLU non-linearity in between:

$$\mathrm{FeedForward}(X) = \mathrm{ReLU}(XW_1 + b_1)W_2 + b_2, \quad (8)$$

where W_1 and W_2 are learnable weight matrices of shape $d_{\text{model}} \times d_{\text{ff}}$ and $d_{\text{ff}} \times d_{\text{model}}$, and b_1 and b_2 are learnable bias vectors of shape d_{ff} and d_{model}.

2.2 Positional encoding

The Transformer cannot distinguish elements of a sequence based on their ordering, because the attention operation is permutation invariant. Consequently, it is necessary to provide explicit positional information to the model, which is the purpose of the positional encoding.

2.2.1 Absolute positional encoding

The sinusoidal positional encoding is one of the most commonly used absolute positional encodings. The encoding is applied by adding it to the inputs of the model, and can be implemented by creating a matrix E of size $L \times d_{\text{model}}$, where L is the sequence length. Each row contains $d_{\text{model}}/2$ pairs of sine and cosine functions with varying wavelengths, with each pair sharing the same wavelength. The wavelength is increased geometrically for each pair, which can be written as follows:

$$E_{i,2j} = \sin\left(\frac{i}{K^{j/d_{\text{model}}}}\right) \quad (9)$$

$$E_{i,2j+1} = \cos\left(\frac{i}{K^{j/d_{\text{model}}}}\right) \quad (10)$$

where $i \in [1, L]$ is the position in the sequence, and $j \in [1, d_{\text{model}}/2]$ is the index of the feature dimension. The value of K determines what the largest wavelength will be, and is commonly set to 10000.

2.2.2 Rotary encoding

The Rotary Encoding [15] is a relative positional encoding, which was introduced as an alternative to the absolute positional encoding and other previously proposed relative positional encodings. It encodes relative positional information in the angle between the key and query vectors. This is different to most other positional encodings, which are typically additive. The encoding function f is derived by considering a relation between $\mathbf{q}_m$, a query vector at position m, and $\mathbf{k}_n$, a key vector at position n. The dot product of the encoded vectors should be equal to the output of some function g, which only depends on the original vectors and

the relative distance between their positions:

$$f(\mathbf{q}_m, m) \cdot f(\mathbf{k}_n, n) = g(\mathbf{q}_m,\ \mathbf{k}_n,\ m - n) \tag{11}$$

We will consider only case of 2D vectors $\mathbf{q}$ and $\mathbf{k}$. (The general case of an arbitrarily sized vector is more cumbersome to state, but is a straightforward generalization of the 2D case.) The desired relation can be achieved by the following encoding:

$$f(\mathbf{x}, k) = \mathbf{x}e^{ik\theta}, \tag{12}$$

where the 2D vector $\mathbf{x}$ is considered as a number in the complex plane, and θ is a real non-zero constant. This encoding satisfies the desired relation with the following function:

$$g(\mathbf{q}_m, \mathbf{k}_n, m - n) = \mathrm{Re}[\mathbf{q}_m \mathbf{k}_n^* e^{i(m-n)\theta}], \tag{13}$$

where $\mathrm{Re}[\cdot]$ is the real part of the complex number and $\mathbf{k}_n^*$ is the complex conjugate of $\mathbf{k}_n$.

2.3 Normalization

Training deep neural networks can be a challenging, due to issues such as the vanishing gradient problem. Normalization layers [13, 16] have been proposed to speed up the training process, and to make training more robust to random weight initialization. The Transformer architecture also includes normalization layers, specifically Layer Normalization [13]. There are mainly two alternatives for the location of the normalization layer within the architecture; the first is *post-layer* normalization, and the second is *pre-layer* normalization. The original Transformer [1] used post-layer normalization, however pre-layer normalization has been found by some to lead to more effective training [17]. The two orderings can be formalized as follows. Let $\mathrm{Sublayer}(\cdot)$ refer to either the multi-head attention or the feedforward layers of the Transformer. Then we have:

$$\mathrm{PostLN}(x) = \mathrm{LayerNorm}(x + \mathrm{Sublayer}(x)) \tag{14}$$

$$\mathrm{PreLN}(x) = x + \mathrm{Sublayer}(\mathrm{LayerNorm}(x)) \tag{15}$$

ReZero [14] is an alternative to Layer Normalization for training deep networks. Instead of calculating statistics and using these to normalize the data, it simply uses a multiplicative gating parameter α, which is initially set to 0:

$$\mathrm{ReZero}(x) = x + \alpha \cdot \mathrm{Sublayer}(x) \tag{16}$$

The same α parameter is used within a single Transformer block, both for the multi-head attention layer and for the feedforward layer.

3 Related work

3.1 Transformers for time series forecasting

Following the introduction of the Transformer [1] in 2017, researchers also started to use Transformers for time series tasks. However, the topic of using Transformers for time series tasks have received relatively little research attention, compared to the use of Transformers for other kinds of data.

The main focus of the work on Transformers for time series has been on the quadratic computational complexity of the attention operation. In the context of NLP, there are numerous works attempting to address this issue, see for instance the recent survey by Tay et al. [18]. We will not attempt to summarize this line of work here, but instead focus on works that specifically target time series forecasting problems.

The first work in the direction of efficient Transformers for time series was by Li et al. [6], who introduced the LogSparse Transformer. The architecture improves the efficiency of the attention operation by removing queries that are far away from the current time step, by exponentially increasing the space between consecutive queries, resulting in a complexity of $O(N \log N)$ instead of $O(N^2)$. Moreover, a causal convolution layer was added before the attention layer, to allow the model to easily discover similarities between ranges of time series points.

The Informer [7] is a Transformer designed for the task of Long Sequence Time-Series Forecasting, which was defined by the authors as forecasting horizons of size 48 or longer. The authors propose an efficient attention operation, which first approximates the query-key similarity and then selects the most important queries, resulting in $O(N \log N)$ computational complexity. The model uses an encoder-decoder architecture that produces forecasts for the entire horizon in a single evaluation. Compared to the LogSparse Transformer, this leads to improved speed when forecasting long horizons, as the model does not need to iteratively generate forecasts.

The Autoformer [8] is another Transformer designed for Long Sequence Time-Series Forecasting, and similarly to the Informer, it also has an encoder-decoder architecture which forecasts the entire horizon in a single step. It proposes a different modification of the attention operation, replacing the dot-product attention mechanism with an auto-correlation based mechanism, again with $O(N \log N)$ complexity. Like the Informer, it can evaluate long horizons quickly and efficiently, and moreover, the authors show improved accuracy compared to the Informer on several datasets.

In contrast to these works, we are primarily interested in improving the forecasting accuracy of the Transformer, and do not attempt to improve the computational complexity of the model.

3.2 M4 competition methods

Time series forecasting is an increasingly relevant area of research. However, in our opinion, there has not been a clear consensus within the Machine Learning community on how to benchmark forecasting models. Earlier work, such as the LogSparse Transformer by Li et al. [6], frequently used the traffic[2] and electricity[3] datasets. However, these datasets have some issues that make them less suitable as a benchmark dataset. There are at least three different train/test split points used for each dataset, which makes comparing performance across different splits difficult [19]. Moreover, these datasets contain missing data, which complicates the training and evaluation setup. The traffic dataset also has missing data during some public holidays, but lacks documentation regarding which specific dates have been removed [19].

We suggest that the M4 dataset should be used as a standard benchmark dataset for research into time series forecasting. The M4 dataset was introduced in the M4 forecasting competition [20], the fourth competition in a series of highly influential forecasting competitions, known as the Makridakis competitions. The dataset contains 100,000 time series; compared to previously used datasets in Machine Learning forecasting studies, this is a very large dataset. These time series were collected from a wide range of domains, and exhibit a wide range of behaviors. The organizers of the competition provided forecasts of several well-known baseline methods, including various naïve methods, exponential smoothing methods, and the well-known ARIMA method. These baseline methods are able to capture linear relationships well, and are often difficult to beat in many real world problems which have strong auto-correlation. However, on the M4 dataset, the best methods outperform these baselines substantially, which indicates the necessity of being able to capture non-linear dynamics to achieve a high level of accuracy on this dataset.

The M4 dataset has several desirable properties compared to previously used forecasting datasets. It is quality controlled, and there are no missing data. The evaluation procedures are clearly defined, and there is no confusion regarding train/test split points. Furthermore, the M4 dataset is large, which arguably is a precondition for Deep Learning methods to be successful. (The small size of the previous M3 competition dataset is believed to be a major reason for why neural networks did not perform well in that competition [21].)

The M4 competition included 61 methods in total. These methods used a wide variety of techniques, the majority of which were not Deep Learning techniques. However, to the surprise of some, the winning method relied heavily on Deep Learning. This method was developed by Smyl [22], and was a hybrid model combining a recurrent neural network with multiple exponential smoothing models. The parameters of the exponential smoothing model were learned for a single time series, while the neural network parameters were shared across time series. The recurrent neural network was a dilated LSTM [23] with attention [24] and residual connections [25]. Moreover, several such hybrid models were combined in ensembles in order to improve forecasting accuracy.

After the competition was finished, Oreshkin et al. proposed a new method which outperformed even the winner of the competition, called *N-BEATS* [19]. The authors wanted to show that a deep neural network could perform well on the M4 dataset, without the need for classical time series forecasting techniques, such as those used in Smyl's hybrid model. N-BEATS consists of blocks of feed-forward networks, which are combined such that each block provides a partial forecast, and these are added together to produce the final forecast. Furthermore, instead of using a regular residual skip connection, the partial forecasts are subtracted from the input of the next block, which the authors call *double residual stacking*. The final model is an ensemble of 180 such networks, which are trained with 18 different configurations in order to ensure sufficient diversity in the ensemble.

None of the works on Transformers for time series from the previous section evaluate their method on the complete M4 dataset. While Li et al. [6] evaluate their LogSparse Transformer on the hourly portion of the M4 dataset, the authors do not report their performance in the metric used in the competition (OWA), making it difficult to compare their accuracy to the original M4 contestants.

To the best of our knowledge, our proposed method is the first to achieve competitive results on the complete M4 dataset using a Transformer model. Moreover, to the best of our knowledge, our method is also the first to achieve competitive results using a single neural network model, instead of using ensembles of Deep Learning models.

4 Method

4.1 The time series forecasting task

A time series is defined to be a sequence of fully observable measurements $\mathbf{x} = [x_1, \ldots, x_T] \in \mathbb{R}^T$, where x_t is the observation at time t, and T is the length of the series. The goal of forecasting is to predict $\mathbf{y}$ given $\mathbf{x}$, where $\mathbf{y} = [x_{T+1}, \ldots, x_{T+H}] \in \mathbb{R}^H$, and H is the forecasting horizon.

[2] https://archive.ics.uci.edu/dataset/204/pems+sf

[3] https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014

4.2 The PI-transformer

We will now introduce the PI-Transformer architecture. The architecture can be divided into four parts: normalization, linear projections, a decoder-only Transformer with Rotary positional encodings and ReZero normalization, and Persistence Initialization. A diagram of the complete architecture is shown in Fig. 1.

4.2.1 Normalization

The first step of our method is the normalization step. We first divide by the mean μ_H of the H most recent values of $\mathbf{x}$, and then perform a log transform:

$$\mathbf{z} = f(\mathbf{x}) = \ln \frac{\mathbf{x}}{\mu_H}, \tag{17}$$

By using only the H most recent values, instead of the entire sequence, we can better capture the trend of the series close to the forecasting window. To produce an output in the original data space, we perform the inverse transformation:

$$f^{-1}(\mathbf{z}) = \mu_H \cdot e^{\mathbf{z}} \tag{18}$$

During training, gradients are back-propagated through the inverse transformation. From this point on, we will focus on how to forecast the value of z_{t+1}, as this can be converted to a forecast in the original data space by using the inverse normalization function: $\hat{x}_{t+1} = f^{-1}(\hat{z}_{t+1})$.

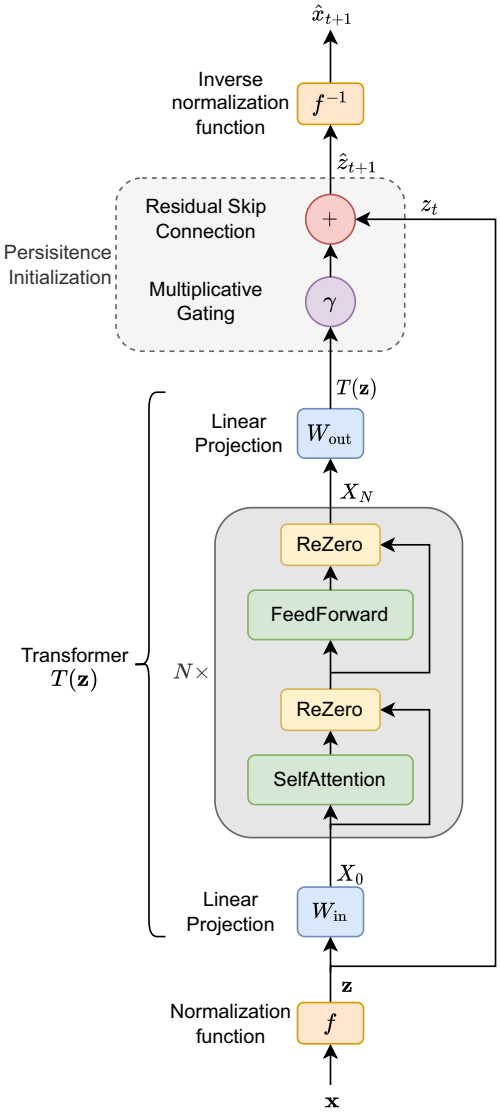

Fig. 1 The proposed adaptation consists of a skip connection and a scalar multiplicative gating mechanism initialized to 0. The initial model becomes the naïve persistence model, i.e. the model that predicts $\hat{x}_{t+1} = x_t$

4.2.2 Transformer model

Our method generates forecasts autoregressively by using a decoder-only Transformer architecture, similar to generative language models in NLP [3]. The architecture consists of blocks of causal self-attention layers and by feedforward layers. In order to improve the stability of training the Transformer for forecasting, we connect the layers using residual skip connections with ReZero gating [14].

In NLP, embedding layers are commonly used to transform text tokens into feature vectors with size d_{model}. We instead need to transform a univariate time series into a sequence of feature vectors, which we do with a linear projection. In other words, the initial input to the Transformer, X_0, is defined as follows:

$$X_0 = \mathbf{z}W_{\text{in}}\,, \tag{19}$$

where $W_{\text{in}} \in \mathbb{R}^{1\times d_{\text{model}}}$ is a learnable weight matrix and $\mathbf{z}$ is the normalized input vector. Now, if we let X_i be the output of the i-th Transformer block, the rest of the model can be defined recursively:

$$X_i(X_{i-1}) = \text{FF}_i(\text{SA}_i(X_{i-1})) \tag{20}$$

$$\text{SA}_i(X) = X + \alpha_i \cdot \text{SelfAttention}_i(X) \tag{21}$$

$$\text{FF}_i(X) = X + \alpha_i \cdot \text{FeedForward}_i(X), \tag{22}$$

where α_i is the learnable ReZero scalar parameter shared between the self-attention and feedforward layer within each block.

The feedforward layer is defined as in the original Transformer, i.e. (8). However, for the self-attention layer, we replace the standard absolute sinusoidal positional encoding with a relative positional encoding, more specifically

the Rotary encoding [15]. The effect of the Rotary encoding is to multiply each key and query with a rotation matrix, which causes positional information to be encoded in the angle between the vectors. The use of the Rotary encoding is motivated by the fact that there the absolute position within a time series window has no semantic significance, in contrast to text data, where the start of the sequence often has a semantic meaning (for instance as the start of a document or a sentence). Instead of adding the positional encoding to the input features of the model, as is done with a standard sinusoidal encoding, the Rotary encoding is implemented by modifying the definition of self-attention:

$$\text{Attention}(Q,\ K,\ V) = \text{softmax}\left(\frac{\widetilde{Q}_{\text{R}}\widetilde{K}_{\text{R}}^{T}}{\sqrt{d_{qk}}} + M\right)V, \quad (23)$$

where $\widetilde{Q}_{\text{R}}$ and $\widetilde{K}_{\text{R}}$ represents the Q and K matrices with Rotary positional encoding applied.

Finally, we perform a linear projection on the output of the final Transformer block in order to go back to a univariate sequence. We define the $T(\mathbf{z})$, the "Transformer function", to be the value after this projection:

$$T(\mathbf{z}) = X_N W_{\text{out}}, \quad (24)$$

where X_N is the output of the Nth block, and $W_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times 1}$ is a learnable weight matrix.

4.2.3 Persistence initialization

Persistence Initialization (PI) is a technique to implicitly initialize an autoregressive neural network for forecasting, in order to improve training stability and forecasting performance. Specifically, the neural network is initialized to become a persistence model, which is a model that simply uses the last known value at time t as the forecast for $t+1$, i.e. $\hat{z}_{t+1} = z_t$. One way of combining this persistence forecast with the forecast from the Transformer model defined above, would be to add the outputs of both models: $\hat{z}_{t+1} = z_t + T(\mathbf{z})$. However, this combined model will not become a persistence model at initialization, as the initial outputs will depend on the randomly initialized weights of network. In order to ensure that the neural network does not contribute to the initial forecasts, we introduce a new zero-initialized gating parameter γ:

$$\hat{z}_{t+1} = z_t + \gamma \cdot T(\mathbf{z}) \quad (25)$$

This results in a combined architecture with the property that the forecasts produced by the initial model are exactly equal to the persistence forecast. However, the combined architecture is still able to improve upon this simple forecast by changing the value of γ through learning.

5 Experimental settings

5.1 Dataset

Public datasets have played an important role for the development of both deep learning methods and forecasting methods. There are clear benefits to having a publicly available high quality dataset, of which the most important is that it allows researchers to measure progress in a standardized way. However, Machine Learning research focusing on time series forecasting has lacked a commonly agreed up benchmark dataset. We propose to use the M4 dataset [20] for this purpose.

The M4 dataset was introduced in the fourth Makridakis competition, held in 2018. The previous Makridakis competitions have been very influential for the development of forecasting methods, and are well regarded in the forecasting community. The M4 dataset consists of 100,000 time series from various domains, divided into six sub-sets based on their sampling frequency. Each frequency has a corresponding forecasting horizon H: Yearly ($H = 6$), Quarterly ($H = 8$), Monthly ($H = 18$), Weekly ($H = 13$), Daily ($H = 14$), and Hourly ($H = 48$). Table 1 contains some descriptive statistics for each frequency. We consider each data frequency as an independent learning problem, and consequently train separate models for each data frequency.

5.2 Metrics

Performance in on the M4 dataset is measured a metric called Overall Weighted Average (OWA) [20]. OWA is a combination metric, which combines the Mean Absolute Scaled Error (MASE) and symmetric Mean Absolute Percentage Error (sMAPE) metrics. MASE and sMAPE are both scale-independent error metrics which are commonly used in the time series forecasting literature [26]. The purpose of these metrics is to enable comparisons of forecasting accuracy across time series data with varying scales. In order to combine sMAPE and MASE metric into the single OWA metric, these metric scores are scaled by corresponding metric scores from a baseline model. In the M4 competition, the baseline model was the Naïve2 model, which is a persistence model that is seasonally adjusted by multiplicative decomposition [20, 27]. After scaling the metrics, their values are combined by taking the average of the two, as follows:

$$\text{OWA} = \frac{1}{2}\left[\frac{\text{sMAPE}}{\text{sMAPE}_{\text{Naïve2}}} + \frac{\text{MASE}}{\text{MASE}_{\text{Naïve2}}}\right] \quad (26)$$

5.2.1 MASE

MASE is a scaled version of Mean Absolute Error (MAE). The scaling factor for MASE is the MAE of a baseline model

Table 1 Descriptive statistics for each frequency in the M4 dataset

	Yearly	Quarterly	Monthly	Weekly	Daily	Hourly
Number of time series	23,000	24,000	48,000	359	4227	414
Forecasting horizon, H	6	8	18	13	14	48
Seasonality, S	1	4	12	1	1	24
Minimum length	19	24	60	93	107	748
25% percentile length	26	70	100	392	337	748
50% percentile length	35	96	220	947	2954	1008
75% percentile length	46	123	324	1616	4211	1008
Maximum length	841	874	2812	2610	9933	1008

on the training set. The baseline model used in the M4 competition is the seasonal naïve model, which always predicts the value S steps in the past (e.g. 24 for all hourly time series, 12 for all monthly time series, etc.). Let x be the training portion of the series, y the true continuation of the series, and $\hat{y}$ be the forecast. Then MASE can be defined as follows:

$$\text{MASE} = \frac{1}{N}\sum_{i=1}^{N}\frac{\frac{1}{H}\sum_{j=1}^{H}|y_j^{(i)} - \hat{y}_j^{(i)}|}{\frac{1}{T^{(i)}-S}\sum_{j=S+1}^{T^{(i)}}|x_j^{(i)} - x_{j-S}^{(i)}|}, \tag{27}$$

where N is the number of time series, $T^{(i)}$ is the length of the time series, H is the forecasting horizon, and S is the seasonality. The superscript (i) (as in $x^{(i)}$) denotes the time series with index i, with $1 \leq i \leq N$.

5.2.2 sMAPE

sMAPE calculates the symmetric percentage difference between the forecast and the actual values. sMAPE scales the absolute error at each time step by the average between the forecast and ground truth at that time step, and can be defined as follows, using the previously introduced notation:

$$\text{sMAPE} = 100 \cdot \frac{1}{N}\sum_{i=1}^{N}\frac{1}{H}\sum_{j=1}^{H}\frac{|y_j^{(i)} - \hat{y}_j^{(i)}|}{\left(|y_j^{(i)}| + |\hat{y}_j^{(i)}|\right)/2} \tag{28}$$

5.3 Training

We used a sliding window approach to train our models. In other words, instead of full length time series, fixed length sub-sequences (windows) were used to train. This was done to avoid the computational issues related to attention over long sequences. The size of the sliding window was defined to be nH, where H is the forecasting horizon and n is a hyperparameter determining the size of the window relative the forecasting horizon.

During training, *teacher forcing* was used to produce H predictions in parallel. Consequently, it is necessary to sample sub-sequences of length $L = nH + H$ to train the model, where the first nH elements are the sliding window inputs and the final H elements are targets. To construct a training mini-batch, we first a sampled a time series i with uniform probability, and then sampled from the sub-sequences within that time series with (conditional) uniform probability.

We created our validation set by combining all the rightmost sub-sequences of length L. However, in order to have a greater number of sub-sequences available in the training set, we excluded the rightmost sub-sequences belonging to the shortest sequences of the dataset. Using set notation, the procedure can be described as follows: Create an index set $\mathcal{I} = \{\, i \mid \forall_{i,1\leq i\leq N}\ T^{(i)} \geq P_{25} \,\}$, where $T^{(i)}$ is the length of time series i, and P_{25} is equal to the 25th percentile in the distribution of time series lengths. Then the validation set is $\mathbb{X}_{\text{val}} = \bigcup_i \mathbb{X}_{\text{val}}^{(i)} = \{x_{T^{(i)}-L<t\leq T^{(i)}}^{(i)} \mid i \in \mathcal{I}\}$, where the $x_{a\leq t\leq b}$ notation indicates the sub-sequence of x starting at a and ending at b, i.e.: $[x_a, x_{a+1}, \ldots, x_{b-1}, x_b]$. The training set is then created by enumerating all possible sub-sequences without overlapping targets in the validation set: $\mathbb{X}_{\text{train}} = \bigcup_i \mathbb{X}_{\text{train}}^{(i)} = \bigcup_i \{x_{j\leq t<j+L}^{(i)} \mid \forall_{j,1\leq j\leq T^{(i)}-L-H\mathbb{I}_{\mathcal{I}}(i)}\}$, where $\mathbb{I}_{\mathcal{I}}$ is the indicator function for $\mathcal{I}$. See Fig. 2 for a graphical representation.

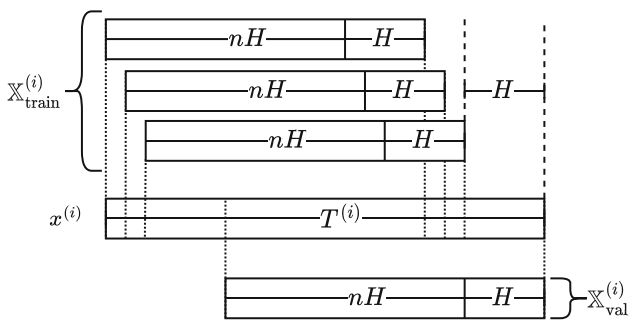

Fig. 2 The validation set was created by taking the rightmost sub-sequence of length $L = nH + H$. The training set was created by enumerating all sub-sequences that do not overlap with the forecasting horizon (i.e. the final H time steps) of the validation set sub-sequence

5.4 Hyperparameters

We performed manual hyperparameter tuning with the goal of finding a general setting which could work well across all data frequencies of the M4 dataset. However, most of the tuning focused on the Monthly frequency. We were largely successful in finding a general setting for all data frequencies, except for the value of n, which determines the size of the input window relative to the forecasting horizon.

For the Yearly, Quarterly, Monthly, and Daily frequencies we set $n = 3$; while for the Weekly and Hourly frequencies we set $n = 4$. A value of $n = 3$ resulted in to poor performance on the Weekly and Hourly frequencies, likely due to seasonal patterns that are only included with window sizes corresponding to $n = 4$. In the case of Weekly, $n = 4$ corresponds to 52 weeks, which indicates the presence of yearly seasonality. For the Hourly frequency, $n = 4$ corresponds to 192 hours, which is approximately 8 days, indicating a weekly seasonality.

The remaining hyperparameters were set to be identical for all data frequencies. The model has 4 layers, 4 attention heads, $d_{\text{model}} = 512$, and $d_{\text{ff}} = 2048$. We use the Lamb [28] optimizer with default hyperparameters, bias correction, and gradient clipping for gradients with norms greater than 10. Our loss function is defined to be identical to the MASE metric (27). We define a training epoch to consist of 128 mini-batches of size 1024. As our stopping criterion, we use early stopping with a patience value of 8, such that training was stopped after 8 epochs without improvement in the validation loss.

5.5 Ablation studies

In order to better understand the effects of the various components of our models, we perform two ablation studies. To reduce the complexity of these studies, we focus exclusively on the monthly portion of M4 dataset, which contains 48% of the series in the M4 dataset.

The first ablation study focuses on the effects of Persistence Initialization, and the second ablation study focuses on the effect of the positional encoding and the normalization layer. Moreover, in both studies we are also interested in the effect of the size of the Transformer model, and possible interactions between architectural components and model size. For this reason we also we vary the model size by setting the hyperparameter d_{model} to values in the set $\{32, 64, 128, 256, 512\}$, with the feedforward size d_{ff} set to $4 \cdot d_{\text{model}}$.

In order to ensure fair comparisons, we perform 9 repeated experiments for each model setting, such that each repeated experiment has different weight initialization and data sampling. This minimizes the effect of randomness due to weight initialization and data sampling, and ensures that we are not cherry-picking the best performing models after the fact.

5.5.1 First ablation study

The first ablation study investigates the effects of the skip connection and the multiplicative gating. We compare architectures with neither skip connections nor multiplicative gating (29), architectures with a skip connection but without multiplicative gating (30), and architectures with both a skip connection and multiplicative gating, i.e. Persistence Initialization (31):

$$\hat{z}_{t+1} = T(\mathbf{z}) \tag{29}$$

$$\hat{z}_{t+1} = z_t + T(\mathbf{z}) \tag{30}$$

$$\hat{z}_{t+1} = z_t + \gamma \cdot T(\mathbf{z}) \tag{31}$$

5.5.2 Second ablation study

The second ablation study compares the effect of the positional encoding and the normalization layers. We compare two positional encodings: the original sinusoidal encoding [1], and the Rotary encoding [15], both of which are described in Section 2.2. We compare three kinds of normalization: ReZero, post-activation Layer Norm, and pre-activation Layer Norm, as described in Section 2.3. This results in a total of six combinations of architecture settings for the second ablation study.

5.6 M4 comparison

In our second experiment we want to measure the performance of our PI-Transformer on the complete M4 dataset, in order to compare it to other state-of-the-art methods that have been evaluated on the complete M4 dataset. In particular, we compare against the top 10 methods of the M4 competition. We also compare against two versions of the *N-BEATS* [19] method, which was developed after the conclusion of the competition.

All the top performing methods on the M4 dataset are ensemble methods, and for this reason we are mainly interested in two issues: forecasting performance and ensemble size. This presents an obvious difficulty, as an ensemble model might have better forecasting accuracy compared to a single model, which would mean that neither model is strictly superior. To address this issue, we measure both the performance of a single PI-Transformer model, and the performance of an ensemble of PI-Transformers. First, we perform 9 repeated experiments for each subset of the M4 dataset. As in the previous ablation studies, each repeated experiment has different weight initialization and data sampling. We will consider the model with the median OWA

score within each data subset to estimate the expected performance of a single PI-Transformer on that subset. The total OWA score is computed by concatenating the predictions of these median-score models from each data frequency into a single prediction on the complete dataset. Second, in order to estimate the effect of using a PI-Transformer in an ensemble, we compute the mean of the 9 predictions for each data subset. These mean predictions are then concatenated into a single prediction for the complete M4 dataset.

5.7 Comparison to other transformer models for time series

For the sake of completeness, we perform a final experiment where we compare the performance of our architecture against three Transformer models which have been applied to time series forecasting: LogSparse Transformer [6], Informer [7], and Autoformer [8].

In this comparison, we only use data from the Hourly subset of the M4 dataset, for two reasons. First, the Informer and Autoformer architectures were designed to deal with very long forecasting horizons. The Hourly sub-set of the M4 for has the longest forecasting horizon ($H = 48$), so it is the part of the M4 dataset that most resembles the problems these architectures were designed for. Second, in the case of the LogSparse Transformer, the authors report the performance of their model on the Hourly sub-set of the M4 dataset. This allows us to refer to the authors' own reported performance, instead of re-implementing the architecture.

Similarly to previous experiments, we perform repeated experiments and report the median score. However, we only perform 5 repeats in this comparison instead of 9, as was done previously. (For the LogSparse Transformer, we use the authors' own reported performance, which was not the median of 5 repeated experiments.)

We used publicly available code[4,5] to implement the Informer and Autoformer. However, we found training the Autoformer to be challenging, as the loss values were generally high throughout training. This was especially the case as the number of parameters increased, and for this reason we decided to only consider the relatively small setting of $d_{\text{model}} = 32$. Similarly to the previous experiments, we set $d_{\text{ff}} = 4 \cdot d_{\text{model}}$. For the Autoformer and the Informer we used 2 encoder layers and 2 decoder layers, and for the Transformer we used 4 layers, as before. This setting results in a similar number of parameters for the three models. We use a context window of length H for the decoder of both the Informer and Autoformer.

We also found the previously used strategy of early stopping on the validation loss to be unreliable when training the Informer and Autoformer, as the validation loss would often have much larger variance than in the previous experiments. (We believe this difference comes from the one-shot forecasting approach taken by both methods. In contrast, our autoregressive model uses teacher forcing during training, leading to more stable loss values, as the model only has to perform 1-step predictions.) To provide a more fair comparison, we instead allocate a fixed amount of computation to each method by setting a limit of 100 epochs, and then selecting the weights with the lowest validation loss to compute the final test score.

The authors of LogSparse Transformer measured performance on M4-Hourly using a 0.5-quantile loss. To be comparable, we also report the 0.5-quantile loss, which can be defined as follows:

$$R_{0.5} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{H} |y_j^{(i)} - \hat{y}_j^{(i)}|}{\sum_{i=1}^{N} \sum_{j=1}^{H} |y_j^{(i)}|} \tag{32}$$

6 Results and discussion

6.1 First ablation study

The first ablation study focuses on the components of Persistence Initialization. We compare models using Persistence Initialization (PI) to two different ablation settings: models lacking both skip connection and multiplicative gating, and models with a skip connection but no multiplicative gating. Additional details regarding this experiment can be found in Section 5.5.

Figure 3 shows box plots of OWA test scores the three settings, and Fig. 4 shows the training and validation loss curves of the three settings. To give additional context about the accuracy of the settings relative to other methods, the box plots in Fig. 3 also includes striped lines representing the first and second place entries in the M4 competition. As a shorthand we will refer to the three settings by their ordering in the plots; i.e. setting 1 refers to models lacking both skip connections and gating, setting 2 refers to the models with skip connections but no gating, and setting 3 refers to models with Persistence Initialization.

The box plots in Fig. 3 shows that each setting has a different relationship between the size of the model and forecasting accuracy. The two ablation settings do not show improved accuracy as model size is increased. For setting 1, the smallest model performs worse than the largest model. In setting 2 all the model sizes perform at a similar level. Only models in setting 3 (i.e. with PI) improve in accuracy when model size is increased. Moreover, the largest model size has a lower median OWA model than the winner of the

[4] https://github.com/zhouhaoyi/Informer2020

[5] https://github.com/thuml/Autoformer

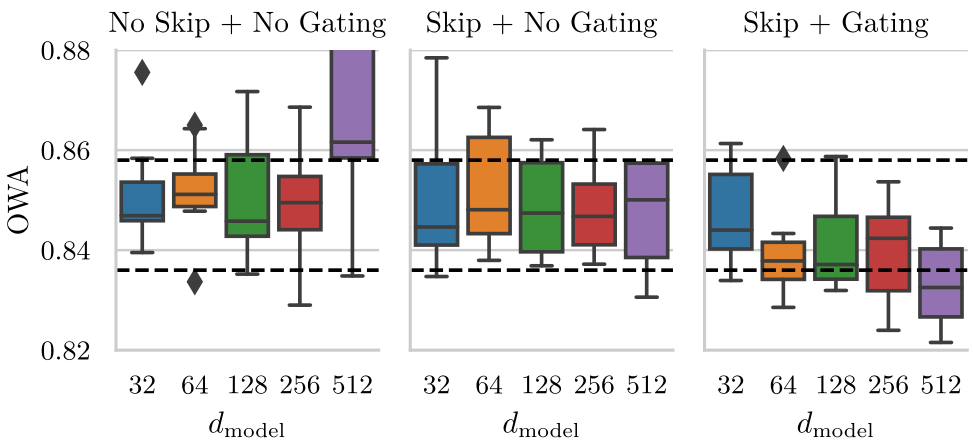

Fig. 3 Box plots of OWA test scores from the first ablation study. Each box represents 9 repeated runs. The striped lines correspond to the first and second place entries in the M4 competition

M4 competition, which shows that Transformer models with Persistence Initialization are able to achieve a high level of accuracy.

Looking at the loss curves for in Fig. 4, we see several indications as to why models with PI achieve better accuracy. Compared to the curves of the two ablation settings, the loss curves of setting 3 (i.e. PI) are shifted both down and to the left. In other words, models with PI start at a lower loss and end at a lower loss, and do so in fewer iterations. It is not surprising that these models start at lower initial loss values, as PI is an initialization technique, designed to improve the initial forecasts of a forecasting model. It might be more surprising that these models also achieve lower final loss values, as the Transformer is known to be a very powerful architecture. One might expect that a Transformer without PI would be able to achieve the same level of accuracy by quickly learning to select the previous time step in one of its attention heads. However, the fact that the models without PI train for more iterations, only to achieve worse results, suggests that learning this simple mapping is in fact non-trivial.

In conclusion, this ablation study clearly shows that Persistence Initialization has a large effect on the training process, improving both performance and training stability. Both the residual skip connection and the multiplicative gating parameter are necessary to see these effects.

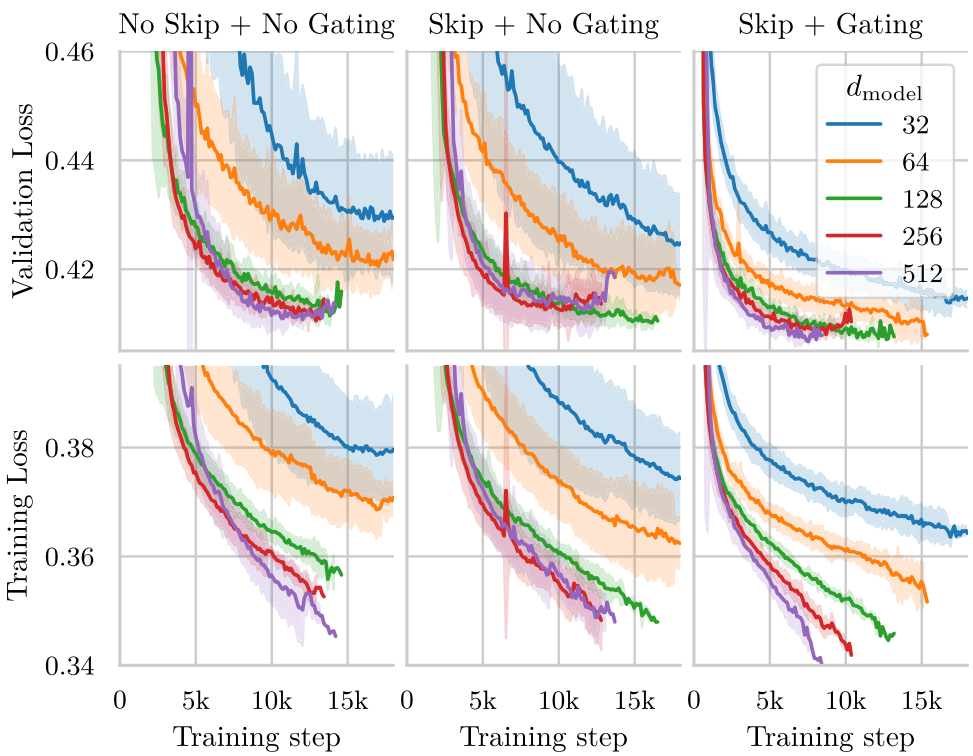

Fig. 4 Validation and training losses from the first ablation study. Each line represents the mean loss over 9 repeated runs, with the shaded area representing the standard deviation. Note that this plot contains a form of survival bias, as training is stopped once the validation loss flattens or increases

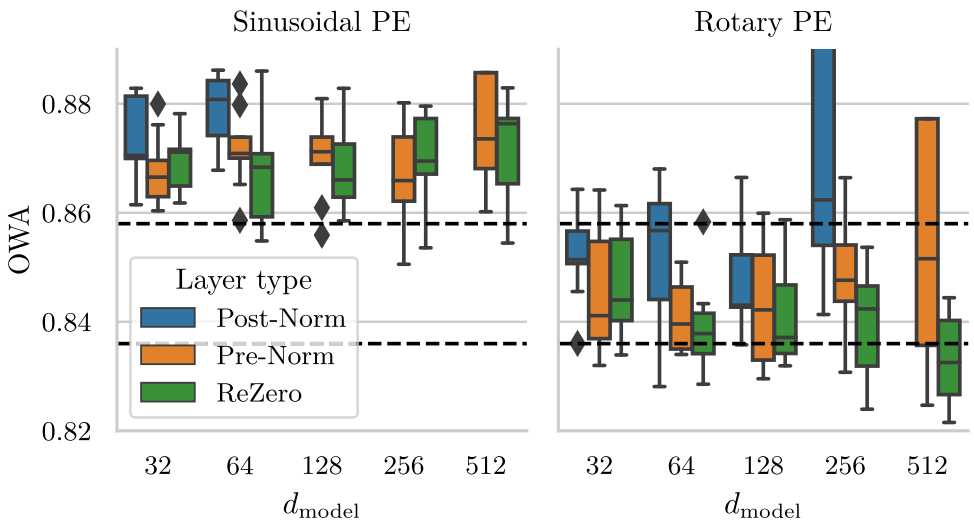

Fig. 5 Box plots of OWA test scores from the second ablation study. Each box represents 9 repeated runs. The striped lines correspond to the first and second place entries in the M4 competition. Note that some boxes are located entirely outside the bounds of the plot

6.2 Second ablation study

The second ablation study compares the effect of the positional encodings and normalization layers. We compare two kinds of positional encodings and three kinds of normalization layers. The two positional encodings are the standard sinusoidal positional encoding and the Rotary encoding. The three kinds of normalization layers are post-activation Layer Normalization, pre-activation Layer Normalization, and ReZero normalization. Additional details regarding this experiment can be found in Section 5.5.

Figure 5 shows box plots of test scores for the six ablation settings. As in the previous experiment, we include striped lines representing the first and second place entries in the M4 competition to give additional context about the level of accuracy.

By inspecting the box plots we immediately see that the Rotary encoding outperforms the sinusoidal encoding in every setting of the experiment. This suggests that the Rotary encoding is better suited for time series tasks than the sinusoidal encoding. Furthermore, the models with ReZero normalization show improved performance compared to the other two options, indicating that ReZero might be the better choice of normalization function for time series tasks.

6.3 M4 dataset performance

This experiment compares the PI-Transformer to other state-of-the-art methods on the complete M4 dataset. We are mainly interested in two issues: forecasting performance and ensemble size. Table 2 compares our method to the top 10 methods of the M4 competition. We also include two versions of the *N-BEATS* [19] method, which was proposed after the conclusion of the competition. See Section 5.6 for more details regarding this experiment.

The size of an ensemble of models is an important aspect of practical usefulness in real world settings. This is especially true when the ensemble members are themselves complex models, such as Deep Learning models. The current top performing method on the M4 dataset is the N-BEATS method by Oreshkin et al. [19], which consists of an ensemble of 180 feed-forward neural networks. To ensure diversity in the ensemble, the authors used three different loss functions and six different window sizes, resulting in a total of 18 different model configurations. The final ensemble was then formed by using 18 copies of 10 identical model configurations, resulting in 180 models in total. The authors also reported the performance of a smaller ensemble which only used one copy of each model configuration, which we have called *N-BEATS-18* in Table 2. The top performing method of the M4 competition was also an ensemble. The winner of the competition, Smyl [22], used a complex strategy which combined models at multiple conceptual levels. First, at the level of ensembles of models, the method combines forecasts from 6-9 independent training runs. Second, each of these training runs consisted of multiple models which were trained on subsets of the dataset. Finally, the forecasts of each such model were produced by taking the average of the predictions produced by the models in the final 4-5 training epochs.

Both of these methods are arguably highly complex, but they also substantially improved forecasting performance compared simpler methods. The significance of the improvements in accuracy can be most easily be seen by inspecting the total OWA of the methods from rank 2 to until rank 6. The median difference between these consecutive ranks is 0.001 OWA, and the difference between the best and the worst method in this group is 0.010 OWA. In contrast, the difference between the rank 1 method (i.e. the winner, Smyl) and the rank 2 method is 0.017 OWA. This observation led the organizers of the competition to characterize

Table 2 OWA test scores for each frequency of the M4 dataset. Bold font is used to indicate the best model, an underline indicates second best. N is the number of time series, and H is the forecasting horizon. N-BEATS-18 is a version of the N-BEATS model with 18 models in its ensemble instead of 180. We report the OWA score for this model based on Fig. 3 from Oreshkin et al. [19]

	Number of models	Yearly N=23000 H=6	Quarterly N=24000 H=8	Monthly N=48000 H=18	Weekly N=359 H=13	Daily N=4227 H=14	Hourly N=414 H=48	Total
M4 Rank 10	7	0.824	0.883	0.899	0.939	0.990	0.485	0.869
M4 Rank 9	2	0.836	0.878	0.881	0.782	1.002	0.410	0.865
M4 Rank 8	1	0.788	0.898	0.905	0.968	0.996	1.012	0.861
M4 Rank 7	2 to 3	0.801	0.908	0.882	0.957	1.060	0.653	0.860
M4 Rank 6	4	0.806	0.853	0.876	0.751	0.984	0.663	0.848
M4 Rank 5	4	0.802	0.855	0.868	0.897	0.977	0.674	0.843
M4 Rank 4	24	0.813	0.859	0.854	0.795	0.996	0.474	0.842
M4 Rank 3	4 to 15	0.820	0.855	0.867	0.766	**0.806**	0.444	0.841
M4 Rank 2	9	0.799	0.847	0.858	0.796	1.019	0.484	0.838
M4 Rank 1	6 to 9	0.778	0.847	0.836	0.851	1.046	0.440	0.821
N-BEATS-18[1]	18	-	-	-	-	-	-	0.802
N-BEATS[1]	180	**0.758**	**0.800**	0.819	-	-	-	**0.795**
PI-Transformer[2]	1	0.777	0.852	0.833	0.733	0.987	0.431	0.815
PI-Transformer[3]	9	0.769	0.836	**0.813**	**0.697**	0.987	**0.397**	0.800

[1] Authors do not report OWA scores for columns containing "-"
[2] Median OWA scores from 9 repeated experiments
[3] Mean ensemble of the predictions of the 9 repeated experiments

the difference between ranks 6 to 2 as "miniscule", while the difference between ranks 2 and 1 was characterized as "considerable" [29]. A similar argument can be made for the difference between N-BEATS and the rank 1 method, which is even greater: 0.026 OWA.

In this experiment, we trained 9 PI-Transformer models for each frequency of the M4 dataset. We measure the expected OWA of a single PI-Transformer by the median OWA of the 9 models. We measure the OWA of an ensemble of PI-Transformer models by using the mean of the 9 predictions, which is arguably a more fair comparison to the other methods of Table 2, as most of these are in fact also ensembles.

As can be seen from Table 2, the median-OWA PI-Transformer achieved a score between the winner of the M4 competition and the N-BEATS method. The difference to the winner of the competition is 0.006 OWA, which is relatively small, as we have discussed above. However, we would argue that the most important advantage of our method is that it is easier to use for Machine Learning practitioners.

The mean-ensemble PI-Transformer achieves a score between N-BEATS-18 and the full N-BEATS model. As before, the differences in OWA are small: 0.002 to 0.005 OWA. Considering that N-BEATS consists of an ensemble of 180 models, we believe that our approach represents a favorable trade-off between forecasting accuracy and ensemble size in this case.

6.4 Comparison with other transformer models for time series

In this experiment, we compare our proposed Transformer architecture against three other Transformer models recently proposed for time series forecasting: the LogSparse Transformer [6], the Informer [7], and the Autoformer [8]. The details regarding the experimental setup for this experiment can be found in Section 5.7.

Table 3 shows the results of the comparison. As can be seen from the table, our method outperforms the others, both in terms of OWA and in terms of $R_{0.5}$.

Table 3 Comparison of Transformer models on M4-Hourly. We include the $R_{0.5}$ score to be comparable with the LogSparse Transformer, as the authors do not report the OWA score

	OWA	$R_{0.5}$
LogSparse Transformer[1]	-	0.067
Informer[2]	0.670	0.056
Autoformer[2]	1.033	0.078
PI-Transformer[2]	**0.525**	**0.046**

For this experiment, we only used small models with $d_{\text{model}} = 32$.
[1] Score taken from [6]
[2] Median scores of 5 repeated experiments

This shows that our method is able to achieve better forecasting accuracy compared to recently proposed Transformer methods for time series. However, we would emphasize that these architectures are inherently different, and were designed for different purposes. The PI-Transformer is a decoder-only architecture, while the Informer and Autoformer are encoder-decoder architectures. We have attempted to make the comparison between these models as fair as possible by keeping the number of parameters in these models approximately equal. This required using two encoder layers and two decoder layers for the Informer and Autoformer, compared to the four decoder layers of our PI-Transformer. However, this results in models with different depth; a better comparison might be to use a depth of 4 for both the encoder layers and decoder layers, regardless of the total parameter count. Moreover, our model is an autoregressive architecture which needs to perform several model evaluations whenever the forecasting horizon is greater than 1, in contrast to the Autoformer and Informer which produce a full horizon of forecasts in a single evaluation. Consequently, these models are likely much faster to evaluate.

6.5 Interpretations of persistence initialization

It is perhaps surprising that adding a single parameter, as Persistence Initialization does, can have such a big impact on forecasting accuracy. Persistence Initialization is a relatively simple change, and the Transformer is a powerful model. One might expect that it would be able to "discover" this pattern by itself, by using the attention mechanism to select the previous time step in one of its attention heads. In this section we will discuss some interpretations of what Persistence Initialization is doing.

One interpretation, which is the one suggested by our naming choice, is that the model is initialized to become a persistence model. During training the model changes from the naïve persistence model to become a more complex model. In other words, Persistence Initialization can be seen as a kind of implicit weight initialization.

A related interpretation is that Persistence Initialization is a re-parametrization of the forecasting problem. Instead of directly forecasting the values of the time series, the model must instead predict the difference to the previous time step. Furthermore, the way the model is initialized corresponds to a prior belief that these differences are zero. This interpretation is somewhat related to the concept of *differencing*, which is a technique commonly used in statistical forecasting methods to make time series more stationary. However, Persistence Initialization is not the same as differencing, as only the outputs of the PI-Transformer are (implicitly) differenced, and not the inputs.

A third interpretation is that we are combining two models in a way that is somewhat similar to boosting. In boosting, a sequence of models are trained iteratively to predict the residual errors of the previous models. We combine the naïve persistence model and a Transformer, such that the Transformer predicts the residuals of the persistence model. The persistence model has a fixed weight of 1, and the weight of the Transformer is the gating parameter γ.

7 Conclusion

In this work, we have presented Persistence Initialization, a novel and general adaptation for autoregressive time series forecasting with neural networks. Furthermore, we introduced PI-Transformer, a Transformer model based on Persistence Initialization, Rotary positional encodings, and ReZero normalization. We perform two ablation studies, and show that the PI-Transformer learns faster, is more accurate, and scales better than Transformer models without our proposed modifications. Moreover, we measure the performance of our proposed PI-Transformer model on the complete M4 dataset, and find that it is able to achieve a high level of forecasting accuracy, similar to other state-of-the-art methods. Our method outperforms the original winner of the M4 competition, and achieves a comparable level of accuracy to N-BEATS, which is an ensemble of 180 deep neural networks, using a significantly smaller ensemble of only 9 PI-Transformers.

Acknowledgements This research was carried out with the support of the ML4ITS project (312062), funded by the Norwegian Research Council (NFR)

Funding Open access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital).

Data and Code Availability Source code and raw data (i.e. model predictions) are available at https://github.com/EspenHa/persistence_initialization

Declarations

Conflicts of Interest The authors have no conflicts of interest to declare

References

1. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008
2. Devlin J, Chang M-W, Lee K, Toutanova K (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186. https://aclanthology.org/N19-1423
3. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A et al (2020) Language models are few-shot learners. Adv Neural Inf Process Syst 33:1877–1901
4. Jumper J, Evans R, Pritzel A, Green T, Figurnov M, Ronneberger O, Tunyasuvunakool K, Bates R, Žídek A, Potapenko A et al (2021) Highly accurate protein structure prediction with AlphaFold. Nature 596(7873):583–589
5. Chen L, Lu K, Rajeswaran A, Lee K, Grover A, Laskin M, Abbeel P, Srinivas A, Mordatch I (2021) Decision Transformer: Reinforcement learning via sequence modeling. Adv Neural Inf Process Syst 34:5084–15097
6. 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. Adv Neural Inf Process Syst 32:5243–5253
7. 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. In: Proceedings of AAAI
8. Wu H, Xu J, Wang J, Long M (2021) Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Adv Neural Inf Process Syst 34:22419–22430
9. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
10. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feed-forward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256. JMLR Workshop and Conference Proceedings
11. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning
12. Hyndman RJ, Athanasopoulos G (2018) Forecasting: Principles and Practice
13. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. arXiv:1607.06450
14. Bachlechner T, Majumder BP, Mao H, Cottrell G, McAuley J (2021) Rezero is all you need: Fast convergence at large depth. In: Uncertainty in Artificial Intelligence, pp. 1352–1361. PMLR
15. Su J, Lu Y, Pan S, Murtadha A, Wen B, Liu Y (2021) Roformer: Enhanced transformer with rotary position embedding. arXiv:2104.09864
16. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456. PMLR
17. Baevski A, Auli M (2019) Adaptive Input Representations for Neural Language Modeling. In: International Conference on Learning Representations. https://openreview.net/forum?id=ByxZX20qFQ
18. Tay Y, Dehghani M, Bahri D, Metzler D (2022) Efficient transformers: A survey. ACM Comput Surv 55(6):1–28
19. Oreshkin BN, Carpov D, Chapados N, Bengio Y (2020) N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In: International Conference on Learning Representations. https://openreview.net/forum?id=r1ecqn4YwB
20. Makridakis S, Spiliotis E, Assimakopoulos V (2020) The M4 Competition: 100,000 time series and 61 forecasting methods. Int J Forecasting 36(1):54–74
21. Hyndman RJ (2020) A brief history of forecasting competitions. Int J Forecasting 36(1):7–14
22. Smyl S (2020) A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. Int J Forecasting 36(1):75–85
23. Chang S, Zhang Y, Han W, Yu M, Guo X, Tan W, Cui X, Witbrock M, Hasegawa-Johnson MA, Huang TS (2017) Dilated recurrent neural networks. Adv Neural Inf Process Syst 30
24. Qin Y, Song D, Cheng H, Cheng W, Jiang G, Cottrell GW (2017) A dual-stage attention-based recurrent neural network for time series prediction. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 2627–2633
25. Kim J, El-Khamy M, Lee J (2017) Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition. In: Proc. Inter-speech 2017, pp. 1591–1595. https://doi.org/10.21437/Interspeech.2017-477
26. Hyndman RJ, Koehler AB (2006) Another look at measures of forecast accuracy. Int J Forecasting 22(4):679–688
27. Makridakis S, Hibon M (2000) The M3-competition: Results. Conclusions and Implications. Intl J Forecasting 16(4):451–476
28. You Y, Li J, Reddi S, Hseu J, Kumar S, Bhojanapalli S, Song X, Demmel J, Keutzer K, Hsieh C-J (2020) Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. In: International Conference on Learning Representations. https://openreview.net/forum?id=Syx4wnEtvH
29. Makridakis S, Spiliotis E, Assimakopoulos V (2018) The M4 competition: Results, findings, conclusion and way forward. Int J Forecasting 34(4):802–808

Espen Haugsdal is currently pursuing a PhD in Computer Science and Artificial Intelligence at the Norwegian University of Science and Technology (NTNU). He completed his Master's degree in Computer Science at NTNU in 2021. His current research is focused on the use of Transformer models for time series tasks.

Erlend Aune is a quant researcher at Abelee, and Adjunct Associate Professor of Data Science at NTNU and BI. He has a PhD in statistics from NTNU. Research interests include: creative use of AI and data, machine learning for time series modeling, innovation using AI.

Massimiliano Ruocco is presently a Senior Researcher at SINTEF Digital. He received his PhD at the Norwegian University of Science and Technology (NTNU). Currently, he also holds the position of Adjunct Associate Professor at NTNU in the Data and Artificial Intelligence Group within the Computer Science Department. His research interests encompass a wide spectrum, focusing on Machine Learning and Deep Neural Networks with application across various domains, including Telecommunication, Healthcare, Transport, and Aviation.