Data and Web Science Seminar FSS 24

Kai Reffert
1980476
kai.reffert@students.uni-mannheim.de

Abstract

Long-term time series forecasting (LTSF) is an important task for various domains. Transformer have surpassed previous state-of-the-art models on this task, however a simple linear model was able to outperform them once again. Therefore, the question arose if Transformer are effective for time series forecasting. Subsequently, the PatchTST model was released by Nie et al. [21], which recovered some ground for Transformer-based models by exceeding previous state-of-the-art models on LTSF problems. In this report, I take a closer look at the PatchTST model, its contributions and related literature to understand and present the relationship and current status of the field.

1 Introduction

Time series analysis comprises multiple downstream tasks, such as forecasting, classification or anomaly detection. Among those, time series forecasting (TSF) is one of the most important tasks, for example including epidemiological [32], traffic [22], energy demand [28] or stock price prediction [20]. Traditional statistical time series forecasting models, such as ARIMA [4] or Prophet [24], are still popular to this day. However, they are fit separately to each time series, often come with many prior assumptions and their performances may deteriorate for long-range forecasting. Therefore, similar to other domains, time series forecasting research showed an increasingly large interest towards deep learning based approaches [3, 13, 16]. Convolutional neural networks (CNNs) have been implemented successfully to TSF tasks, e.g. TCN [2] or TimesNet [30], however their receptive field size poses difficulties, as it mainly captures local information. Although the receptive field of CNNs can be extended to longer windows by stacking several convolutional layers, it still exhibits difficulties of capturing long-term time series relations effectively. Furthermore, recurrent neural networks (RNNs), which are specifically designed to work with sequential data, were employed succesfully in many time series forecasting applications [23]. However, RNNs exhibit vanishing and exploding gradient problems, in which long short-term memory and gated recurrent units have been specifically designed to tackle these problems. However, they were not able to solve these issues fully. On top of that, since RNNs process the data sequentially, they are generally difficult to parallelize. Overall, these limitations are usually not problematic for short time series, nevertheless they are regularly prevalent for longer time series. In addition, time series often contain short- and long-term correlations, which are relevant during time series prediction [15].

Transformer [26] have displayed state-of-the-art performances in various domains, e.g. natural language processing, speech and computer vision. Due to the large influence of the self-attention mechanism, the Transformer architecture resolves some issues of RNNs, as it enables global access to all of the past history and is parallelizable. Hence, Li et al. [18] propose Transformer is possibly better suited to capture long-term dependencies within time series as well. Therefore, several Transformer-based models have been proposed for time series [29] and in particular for the long-term TSF (LTSF) task. However, the memory and time complexity of basic Transformers grows quadratically $O(L^2)$ with the input length L. Hence, many of the first Transformer-based models for time series focused on creating novel architectures to reduce this bottleneck.

- Li et al. [18] proposed the LogSparse model, which reduces the memory cost to $O(L\,(\log L)^2)$. In their experiments on synthetic and real-world data, they were able to show that LogSparse outperforms state-of-the-art time series models, e.g. DeepAR, ARIMA or DeepState, on short- and long-term TSF tasks.
- Informer [40] used ProbSparse self-attention to reduce the time and memory complexity to $O(L \log L)$. The authors were able to outperform LogSparse, ARIMA and Prophet amongst other models.
- Liu et al. [19] introduce Pyraformer, which makes use of pyramidal attention to reduce the memory and time complexity to $O(L)$. Pyraformer further outperformed Informer and LogSparse models on LTSF tasks.
- Autoformer [31] implements an auto-correlation mechanism based on the series periodicity to achieve $O(L \log L)$ and outperform rivalling models, such as Informer and LogTrans.
- Similarly, FEDformer [41] utilizes a frequency enhanced Transformer and reduces the complexity to $O(L)$. It outperforms Informer, Autoformer and LogSparse on LTSF data sets.

Although these Transformer-based time series models outperform state-of-the-art models, Zeng et al. [36] criticize that those methods were solely compared to autoregressive or iterated multi-step (IMS) forecasting methods, i.e. where a single-step predictor is sequentially applied to produce multi-step forecasts. IMS methods are known to exhibit error accumulation problems, which is especially prevalent for the LTSF task. Therefore, they hypothesized that the improvement of those novel Transformer-based models can be mainly attributed to their direct multi-step (DMS) forecasting typology, in which the multi-step prediction task is optimized directly. To investigate this, Zeng et al. [36] introduce a simple linear DMS model, which was able to outperform the Transformer-based methods on multiple different benchmarks. Thus, the effectiveness of Transformers on LTSF, and subsequently TSF, tasks was heavily challenged. In succession to this, Nie et al. [21] introduced the PatchTST model, which was built based on the TST model [37], i.e. a vanilla Transformer Encoder [26] approach, by adding channel independence, which was previously also employed by Zeng et al. [36], and patching. Channel independent approaches consider all channels[1] as multiple univariate time series and fit one combined model on the individual series. Therefore, they ignore the correlation between channels and during prediction of a time series solely the historical values of this particular time series are considered. In contrast, channel dependent methods incorporate information from all channels as input and use the past of all time series to jointly forecast all series. Besides channel independence, patching is also a key concept of PatchTST, which was already successfully implemented in Computer Vision by the Vision Transformer [6] prior to PatchTST. Since PatchTST outperformed all previously mentioned models, it sparked further research to modify and analyze the proposed channel independence [11, 39] or patching [10] processes and their benefits.

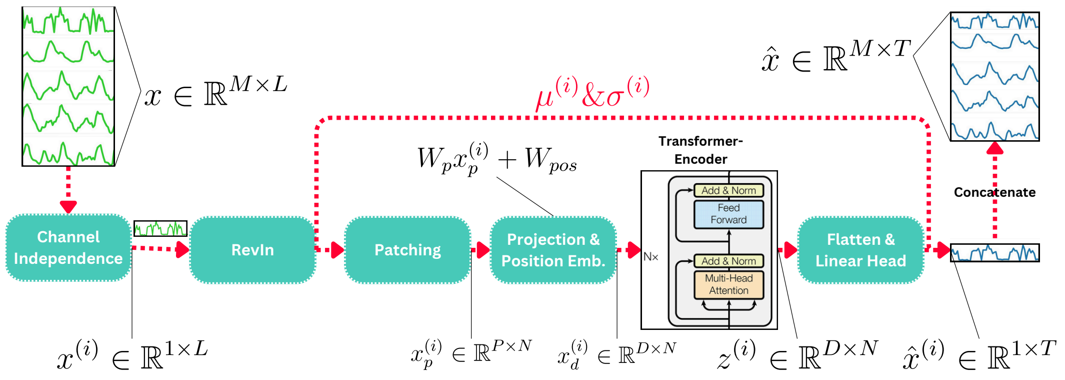
In this seminar paper, I first theoretically explain the PatchTST model in Section 2. After this, I take a closer look at the contributions of PatchTST and compare them to related literature in Section 3. In Section 4, I evaluate the performances of previously seen related approaches. In the end, I conclude this report with a short summary and some personal ideas and improvements in Section 5.

2 Model

Nie et al. [21] regard the task of multivariate time-series forecasting, in which the input is a group of time series $(\mathbf{x}_1, \ldots, \mathbf{x}_L)$. Moreover, L represents the look-back window length, while each vector $\mathbf{x}_t$ corresponds to the values of the M time series at a time point t, hence $\mathbf{x}_t \in \mathbb{R}^M$. In general, the goal is to forecast T future values for each of the M time series $(\mathbf{x}_{L+1}, \ldots, \mathbf{x}_{L+T})$.

An overview of PatchTST is depicted in Figure 1. On the left, the input $(\mathbf{x}_1, \ldots, \mathbf{x}_L)$ is firstly divided into M univariate time series $(x_1^{(i)}, \ldots, x_L^{(i)})$, such that $x_t^{(i)}$ corresponds to the scalar value of time series i at time point t. Then, each of the univariate time series is fed to the remaining modules of PatchTST independently, aiming to produce univariate predictions of form $(\hat{x}_{L+1}^{(i)}, \ldots, \hat{x}_{L+T}^{(i)})$ in the end. Because time series often suffer from a distribution shift problem, i.e. their mean and standard deviation change over time, Nie et al.[21] adopt a reversible instance normalization approach

[1]The term channel is often used interchangeably with time series, where one channel corresponds to a single univariate time series.



$x \in \mathbb{R}^{M\times L}$
$\mu^{(i)} \& \sigma^{(i)}$
$\hat{x} \in \mathbb{R}^{M\times T}$
$W_p x_p^{(i)} + W_{pos}$
Transformer-Encoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Concatenate
Channel Independence
RevIn
Patching
Projection & Position Emb.
Flatten & Linear Head
$x^{(i)} \in \mathbb{R}^{1\times L}$
$x_p^{(i)} \in \mathbb{R}^{P\times N}$
$x_d^{(i)} \in \mathbb{R}^{D\times N}$
$z^{(i)} \in \mathbb{R}^{D\times N}$
$\hat{x}^{(i)} \in \mathbb{R}^{1\times T}$

Figure 1: Illustration of the PatchTST model.

[14], see *RevIn* block in Figure 1, to each univariate time series. During this step, each series $x^{(i)}$ is standardized to a mean of zero and variance of one. Later on, when the predictions are determined, they are then summed back together with the original mean and standard deviation.

In the Transformer backbone, the univariate time series is first divided into overlapping or non-overlapping patches. To be precise, the patching process, which generates patches $\mathbf{x}_p^{(i)} \in \mathbb{R}^{P\times N}$ out of a univariate time series $\mathbf{x}_{1:L}^{(i)}$, can be described as follows:

$$\mathbf{x}_p^{(i)} = \mathbf{x}_{(1+(p-1)\cdot S):(P+(p-1)\cdot S)}^{(i)} \qquad \forall p : 1 \leq p \leq N \tag{1}$$

where P denotes the patch length, while the non-overlapping shift between successive patches is represented by the stride S. $N = \lfloor \frac{L-P}{S} \rfloor + 2$ represents the total number of patches per univariate time series. If the patch length P is equivalent to the stride S, the patches are strictly non-overlapping. Note that, to ensure equal length among the patches, the last element $x_L^{(i)}$ is padded S times to the end of the time series prior to the patching process. Let us consider an example to illustrate the process more intuitively. Propose we are given the following time series of length $L = 9$: [1, 3, 2, 4, 1, 2, 5, 5, 3]. If we apply patching with $P = 4$ and $S = 2$, we first have to pad the time series with the last value: [1, 3, 2, 4, 1, 2, 5, 5, 3, 3, 3]. Then, we can determine the $N = \lfloor \frac{L-P}{S} \rfloor + 2 = \lfloor \frac{9-4}{2} \rfloor + 2 = \lfloor 2.5 \rfloor + 2 = 4$ patches: [1, 3, 2, 4], [2, 4, 1, 2], [1, 2, 5, 5] and [5, 5, 3, 3].

After the patching process, the patches $\mathbf{x}_p^{(i)} \in \mathbb{R}^{P\times N}$ are projected to the embedding dimension D of the Transformer through a trainable weight matrix $\mathbf{W}_p \in \mathbb{R}^{D\times P}$. Furthermore, a learnable position encoding matrix $\mathbf{W}_{pos} \in \mathbb{R}^{D\times N}$ is added as well, which produces the Transformer's input $\mathbf{x}_d^{(i)} \in \mathbb{R}^{D\times N}$ with the following equation:

$$\mathbf{x}_d^{(i)} = \mathbf{W}_p \mathbf{x}_p^{(i)} + \mathbf{W}_{pos} \tag{2}$$

Equivalent to Zerveas et al. [37], the upcoming Transformer architecture implemented within PatchTST is the vanilla Transformer encoder introduced by Vaswani et al.[26]. In fact, PatchTST without patching nor channel-independence is identical to the TST model of Zerveas et al. [37]. They further explain that the decoder Transformer block is primarily important for generative tasks, such as time series forecasting. However, the Transformer decoder receives the (masked) "ground truth" output sequence as an input, which makes it inappropriate for (extrinsic) regression or classification tasks. Hence, to assure task versatility and reduce the total amount of parameters by approximately half, solely the Transformer encoder is used.

During multi-head attention with H heads, a single head $h \in (1, \ldots, H)$ within the Transformer encoder determines the query matrix $\mathbf{Q}_h^{(i)} \in \mathbb{R}^{N\times d_k}$ with $\mathbf{Q}_h^{(i)} = (\mathbf{x}_d^{(i)})^\top \mathbf{W}_h^Q$, by using a learnable weight matrix $\mathbf{W}_h^Q \in \mathbb{R}^{D\times d_k}$. Similarly, the key matrix $\mathbf{K}_h^{(i)} \in \mathbb{R}^{N\times d_k}$ is defined as $\mathbf{K}_h^{(i)} = (\mathbf{x}_d^{(i)})^\top \mathbf{W}_h^K$, where $\mathbf{W}_h^K \in \mathbb{R}^{D\times d_k}$ is a trainable matrix. In addition, the value matrix $\mathbf{V}_h^{(i)} \in \mathbb{R}^{N\times D}$ is determined through $\mathbf{V}_h^{(i)} = (\mathbf{x}_d^{(i)})^\top \mathbf{W}_h^V$, where $\mathbf{W}_h^V \in \mathbb{R}^{D\times D}$ is again trainable. Lastly, the following Equation describes the computation of the output of a single attention head $\mathbf{O}_h^{(i)} \in \mathbb{R}^{D \times N}$:

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

After the outputs of all attention heads have been determined, the output of the multi-head attention block $\mathbf{y}^{(i)} \in \mathbb{R}^{D \times N}$ is determined as follows:

$$\mathbf{y}^{(i)} = \mathbf{W}^O \mathbf{O}^{(i)} \tag{4}$$

where $\mathbf{O}^{(i)} = [\mathbf{O}_1^{(i)^\top}, \ldots, \mathbf{O}_H^{(i)^\top}]^\top \in \mathbb{R}^{(H \cdot D) \times N}$ represents the concatenated output of all individual attention heads, while $\mathbf{W}^O \in \mathbb{R}^{D \times (H \cdot D)}$ is a trainable weight matrix [1]. Before the output $\mathbf{y}^{(i)}$ is fed into the FNN, batch normalization as well as a residual connection have been implemented. In contrast to the NLP domain, Zerveas et al. [37] propose that layer normalization is inferior to batch normalization in time series applications, as batch normalization can reduce the extent of outlier values, which is a specific problem of time series. In addition, they state that the sequence length of NLP tasks exhibits larger variations, which is preferable for layer normalization, however this is not as prevalent for the time series data they assessed. Afterwards a FNN layer is applied, once again surrounded by a residual connection and batch normalization, which produces $\mathbf{z}^{(i)} \in \mathbb{R}^{D \times N}$. Lastly, a flatten layer with linear head receives $\mathbf{z}^{(i)}$ and returns univariate predictions $(\hat{x}_{L+1}^{(i)}, \ldots, \hat{x}_{L+T}^{(i)})$. In the next section, I present the three major contributions of PatchTST in more detail and discuss important concept that relate to them.

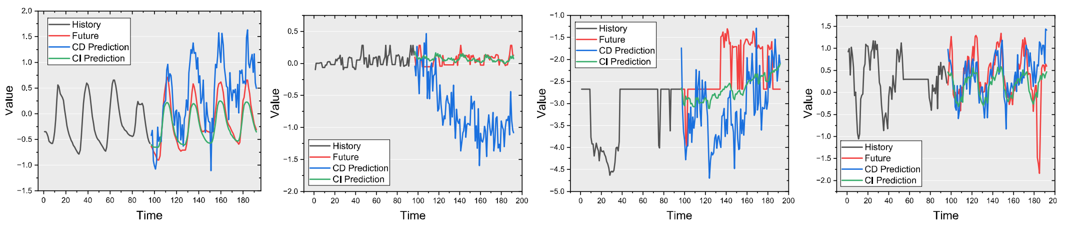
3 Discussion

In this section, I revisit the three primary contributions of PatchTST. First, I will explain how channel independence relates to channel dependence in more detail. Then, I will go over the advantages of the patching process and potential alternatives. Finally, I explain how PatchTST can be implemented for representation learning.

Channel independence. Better performances of channel independent methods over channel dependent models seem unintuitive, as only information along the time axis is considered while cross-channel correlations are disregarded. Yet, there are several works in which CI models outperform CD models [36, 21, 11, 10]. Nie et al. [21] propose several potential reasons for this. First, they state that Transformer-based CI methods have higher adaptability, as different attention maps are learned for different series. Furthermore, with comparative CD methods, the same attention pattern is assigned to all series. Second, they propose that CD models may require more data to jointly learn cross-channel and temporal correlations. Lastly, they suggest that CD models are more likely to overfit to the training data. Afterwards, Nie et al. [21] highlight three technical advantages of CI methods: 1. Possibility of additionally learning cross-channel information through methods such as GNNs, 2. Ability to apply different losses for different time series and 3. Robustness to noise, as noise of time series is conserved within them and not propagated further to other time series.

Following the success of CI methods, Han et al. [11] investigate the relation between CI and CD methods more in-depth. By comparing a linear CI model to its CD counterpart, they propose that the optimal coefficients of the CD approach are dependent on the correlation of each channel. In contrast, the CI model's optimal parameters are determined by the sum (mean) of the correlation of all channels. Hence, the CI approach exhibits less distribution shift, because the sum of correlation differences between train and test data has lower variation than the correlation differences of individual channels. Subsequently, Han et al. [11] propose that CD methods have high capacity and low robustness, whereas CI approaches have low capacity and high robustness. Since robustness is often more important in real-world non-stationary time series with distribution shifts, CI methods often perform better.

Furthermore, in Figure 2 Han et al. [11] show representative examples of this idea, in which they compare the linear CI and CD models. Overall the CI strategy produces smoother forecasts than the CD approach, which can be explained by the previously mentioned fashion in which the optimal parameters are determined. In Figure 2(a), both methods capture trend and seasonality relatively well. In contrast, Figure 2(b) represents an example, where the CD strategy predicts a false trend



(a) (b) (c) (d)

Figure 2: Four representative example to illustrate CI and CD differences. The model is the linear model of Zeng et al. [36] and the data set is ETTh2. (a) Ci predictions smoother, whereas CD forecasts are sharper and exhibit sudden variations. (b) CD predicts wrong trends, which is less likely for CI. (c) Anomalous series, where CI is more robust. (d) Series with predictable pattern, CD model potentially advantageous due to higher complexity. Adapted from Han et al. [11].

while the CI approach does not make this error. Similarly, Figure 2(c) shows that the robustness of the CI strategy is favorable for anomalous series, as it exhibits less oscillation. Finally, Figure 2(d) corresponds to an example where the CD strategy is favorable, as the method's high capacity is able to capture the underlying trend and seasonality well. Ultimately, they conclude that the training strategy, e.g. CI/CD, and not their design may often be the reason why some sophisticated models perform better or worse. Thus, they suggest decomposing the training strategy and algorithm to ensure a fair comparison.

Due to the inherently unintuitive restriction to disregard the channel correlations, the CI strategy is often challenged and many approaches exist to handle downsides of CD and CI approaches. Therefore, I shortly summarize some research works :

Han et al. [11] propose to use the L1 loss during training of CD methods, as it increases the robustness. In addition, they compared the effects of the look-back window length L on CI and CD models, where they found that an increase of L results in more available information and a higher capacity of the model. Hence, their experiments suggest that CI models always benefit from longer window lengths, while CD models often perform worse following an increasing L.

Xue et al. [33] introduced Channel Aligned Robust Blend Transformer (CARD), which handles some disadvantages of CI Transformers and manages to outperform state-of-the-art models, such as PatchTST, TimesNet, Crossformer, FEDformer or Dlinear [21, 36], on several benchmark data sets. It introduces a channel-aligned attention mechanism, a token blend module and a robust loss function to efficiently leverage cross-channel correlations.

Zhao and Shen [39] argue that different time series are connected via locally stationary lead-lag relations, in which a leading indicator influences other series, e.g. the body temperature may decrease after one hour when certain medicine is taken. Hence, they introduce LIFT, which is a method to estimate and incorporate leading indicators, as a type of plugin to arbitrary forecasting models. Furthermore, a CI backbone in combination with LIFT, separates the modeling of time and channel dependence into two parts, which potentially reduces the optimization challenge of original CD methods. They showed that LIFT was able to improve the performances of PatchTST, DLinear and Crossformer[21, 36].

Wang et al. [27] present CSformer, which is a CD method that utilizes a two-stage self-attention strategy to separately extract time and channel information efficiently. CSformer outperformed iTransformer, PatchTST and Crossformer [21] among many more on several benchmark datasets.

Gao et al. [8] introduced Client, which similarly outperforms TimesNet, DLinear and FEDformer [41], by utilizing linear and attention modules to capture time and channel information respectively.

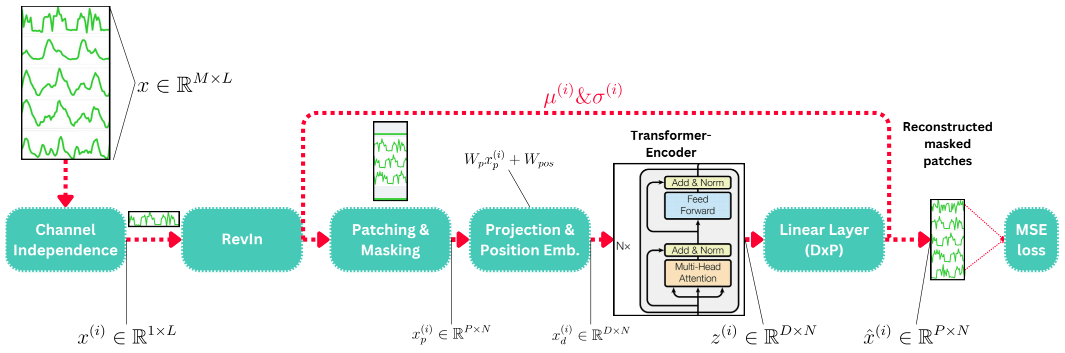
Patching. In contrast to channel independence, the patching process of PatchTST is motivated more naturally through three major benefits. First, it improves the preservation of local semantic information, which is especially important in time series prediction, where, unlike a word in a sentence, a single time point does not carry any semantic interpretation. Furthermore, Nie et al.[21] state that the combination of time series values into patches better grasps semantic relations in comparison to the widely implemented single point-wise tokens. Second, the time and space complexity of the original Transformer scale quadratically $O(N^2)$ with the size of the input tokens N, which poses as a major bottleneck. However, the size of the input tokens N, which is equal to L if no pre-processing is performed, can be approximately decreased towards $\frac{L}{S}$ through the patching process. Hence, resulting in a quadratic reduction of both time and space complexity by a factor of S assuming an equivalent look-back window L. Third, as a consequence of the reduced complexity, PatchTST can be fed inputs with longer look-back window sizes while working with the same memory and time resources as comparative Transformer-based methods. In theory, a longer look-back window should potentially increase the performance of TSF models. However, several authors [36, 21, 11] find that this is not the case for most channel dependent Transformer models. In fact, Han et al. [11] conclude that a longer look-back window length does provide more information to the model, however it also leads to an increase of capacity. Hence, while channel independent models always benefit from an increasing look-back window length, channel dependent approaches often perform worse, assuming that the look-back window length before the increase is not too small. Overall, PatchTST is thus capable of efficiently implementing longer look-back windows, while also benefiting from the increase. Furthermore, this was also supported by Nie et al. [21], who showed that PatchTST continuously improves performances following an increase of the look-back window length in several experiments.

Prior to PatchTST, patching was already implemented in other domains, e.g. BERT in NLP with subword-based tokenization [5] or Vision Transformer in CV [6]. However, in the time series domain, other models still primarily considered vectors per time step $\mathbf{x}_t \in \mathbb{R}^M$, i.e. the values of all M time series for the same time step, as input. Furthermore, Zhang and Yan [38] propose that these models primarily capture cross-time relations, whereas cross-channel relationships are not represented in the embedding. Therefore, they introduced dimension-segment-wise embedding, which is closely related to patching. In addition, they also employ linear projection with positional encoding for each patch, however, in contrast to PatchTST, the input to the transformer layer consists of the 2D vector array comprising all channels. Altogether, their resulting model, Crossformer, is a hierarchical encoder-decoder, which, similar to already mentioned approaches in the channel independence paragraph, utilizes two-stage attention to capture cross-channel and cross-time information. However, their approach was later on also outperformed by the simple linear model of Zeng et al. [36]. Wu et al. [30] introduced another patching process alternative. They propose to use the Fast Fourier Transform algorithm, transforming the 1D univariate time series into a 2D representation. On top of that, their model TimesNet outperforms many previously established state-of-the-art methods, e.g. DLinear, Autoformer, FEDformer and Informer [36, 41, 31, 40] amongst many more, on the LTSF task.

Lastly, there are also some works that follow up on the patching process of PatchTST, for example the aforementioned CARD model [33], which incorporates the patching process of PatchTST. Furthermore, Gong et al. [10] introduce PatchMixer, a channel independent CNN-based model, which utilizes PatchTST's patching process. PatchMixer was able to outperform PatchTST, DLinear and more on the LTSF task. In addition, they show that PatchMixer has 2x and 3x faster training and inference speed respectively compared to PatchTST, the authors suggest this is due to omitting the computational intensive self-attention module used in Transformers.

Representation Learning. Transformer models have been applied successfully to self-supervised representation learning tasks in other domains, e.g. NLP [5] or CV [6]. For representation learning PatchTST can be transformed to a masked autoencoder, in which certain parts of the input are reconstructed after they have been purposefully removed. This method was also implemented on time series tasks by Zerveas et al. [37]. However, in their approach multiple randomly selected time steps were masked out, which comes with two major limitations [21]. First, by having access to time steps directly before and after the masked time step, interpolation can be used to easily argue about the masked values. Second, in forecasting applications the output layer applied to these learned representations $\mathbf{z}_t \in \mathbb{R}^D$, representing all L time steps, to produce multivariate forecasts for the M series with prediction length T has the size $(L \cdot D) \times (M \cdot T)$, which can already be a problem if one dimension is large, whereas it is even more problematic if all four dimensions are large.

These two issues are handled by the PatchTST model, which has to be slightly modified for representation learning, see Figure 3. The first modification in Figure 3 is that patches are constructed so that they are strictly non-overlapping, this ensures that the information we want to predict is not already present in other patches. Then, some patches are selected uniformly at random to be masked, i.e. all values within this patch are set to zero. In Figure 3, this corresponds to the first and last patch. Lastly, once the Transformer Encoder produces the representations $\mathbf{z}_t^{(i)} \in \mathbb{R}^{D \times N}$, a $(D \times P)$-linear layer reconstructs the patches, while the MSE loss is solely computed for patches reconstructed from masked patches, e.g. the first and last patch in Figure 3.



$x \in \mathbb{R}^{M\times L}$
$\mu^{(i)} \& \sigma^{(i)}$
Channel Independence
RevIn
Patching & Masking
Projection & Position Emb.
$W_p x_p^{(i)} + W_{pos}$
Transformer-Encoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Linear Layer (DxP)
Reconstructed masked patches
MSE loss
$x^{(i)} \in \mathbb{R}^{1\times L}$
$x_p^{(i)} \in \mathbb{R}^{P\times N}$
$x_d^{(i)} \in \mathbb{R}^{D\times N}$
$z^{(i)} \in \mathbb{R}^{D\times N}$
$\hat{x}^{(i)} \in \mathbb{R}^{P\times N}$

Figure 3: Overview of the self-supervised PatchTST model.

To assess the representation learning capabilities of PatchTST, several experiments were performed by Nie et al. [21], in which the PatchTST variant of Figure 3 is firstly pre-trained on all available datasets for 100 epochs. Then, they propose two variants, linear probing and end-to-end fine-tuning. In linear probing, only the prediction head is trained for 20 epochs. In end-to-end fine-tuning, they first only train the prediction head for 10 epochs, then they fine-tune the entire network for 20 epochs. Nie et al. [21] come to the conclusion that the end-to-end fine-tuning model is outperforming all other supervised approaches, including the supervised PatchTST model.

On a different note, Nie et al. [21] also evaluated the transfer learning ability of PatchTST by pre-training the model on one data set and fine-tuning on the target data set. On this task, the end-to-end fine-tuning model again outperforms all other models, however except for the supervised PatchTST model.

Lastly, Nie et al. [21] compared PatchTST against state-of-the-art contrastive learning time series approaches, such as BTSF [34], TS2Vec [35], TNC [25], and TS-TCC [7]. In this experiment, both the transfer-learning PatchTST, which was pre-trained on a different data set and then fine-tuned with linear probing on the target data set, and original PatchTST, pre-trained and fine-tuned with linear probing on the target data set, outperform the state-of-the-art approaches.

In work that followed up on PatchTST, Lee et al. [17] criticize the representation learning approach of Nie et al. [21]. They propose that learning to predict masked patches from unmasked patches is suboptimal. Instead, they suggest to embed patches independently by considering the simple patch reconstruction task, where each patch is auto-encoded without interaction with other patches. Additionally, they use a patch-wise MLP to encode each patch independently. In their experiments, their model, PITS, outperformed PatchTST, DLinear, FEDformer and more in a supervised setting. In the self-supervised setting, PITS also managed to outperform PatchTST. In the next section, I compare the results of several models, released after PatchTST, in more detail.

4 Model comparison

In this section, I compare models that improved upon PatchTST by looking at Table 2, in which I gathered the published results of some of the models mentioned in Section 3.

Table 2 displays the MAE and MSE performances of PatchMixer, LIFT, MCformer, Client CAR, PITS, DLinear, FEDformer and PatchTST [10, 39, 12, 8, 33, 17] on several LTSF benchmark data sets. Statistics of these data sets are displayed in Table 1.

Table 1: Descriptive Statistics for popular LTSF benchmark data sets.

Datasets	Weather	Traffic	Electricity	ILI	ETTh1	ETTh2	ETTm1	ETTm2
Features	21	862	321	7	7	7	7	7
Time steps	52696	17544	26304	966	17420	17420	69680	69680

Table 2: Performances of selected models for several prediction horizons on LTSF benchmark data sets. The best model of each row is selected in bold, while the second best is underlined.

	PatchTST/64 (2023,[21])		**PatchTST/42** (2023, [21])		**DLinear** (2022, [36])		**FEDformer** (2022, [41])		**PatchMixer** (2023, [10])		**LIFT** (2024, [39])		**MCformer** (2024, [12])		**Client** (2023, [8])		**CARD** (2024, [33])		**PITS** (2024, [17])	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather																				
96	0.149	0.198	0.152	0.199	0.176	0.237	0.238	0.314	0.151	0.193	**0.146**	0.196	0.167	0.213	0.163	0.207	0.150	**0.188**	0.154	0.202
192	0.194	0.241	0.197	0.243	0.220	0.282	0.275	0.329	0.194	**0.236**	**0.190**	0.238	0.215	0.257	0.214	0.253	0.202	0.238	0.191	0.242
336	0.245	0.282	0.249	0.283	0.265	0.319	0.339	0.377	**0.225**	**0.267**	0.243	0.281	0.270	0.297	0.271	0.294	0.260	0.282	0.245	0.280
720	0.314	0.334	0.320	0.335	0.323	0.362	0.389	0.409	**0.305**	**0.323**	0.315	0.333	0.348	0.348	0.360	0.346	0.343	0.353	0.309	0.330
Traffic																				
96	0.360	0.249	0.367	0.251	0.410	0.282	0.576	0.359	0.363	0.245	**0.352**	**0.242**	0.433	0.288	0.438	0.292	0.419	0.269	0.375	0.264
192	0.379	0.256	0.385	0.259	0.423	0.287	0.610	0.380	0.384	0.254	**0.373**	**0.251**	0.440	0.286	0.451	0.298	0.443	0.276	0.389	0.270
336	0.392	0.264	0.398	0.265	0.436	0.296	0.608	0.375	0.393	**0.258**	**0.389**	0.262	0.454	0.292	0.472	0.305	0.460	0.283	0.401	0.277
720	0.432	0.286	0.434	0.287	0.466	0.315	0.621	0.375	**0.429**	**0.283**	**0.429**	0.286	0.489	0.312	0.499	0.321	0.490	0.299	0.437	0.294
Electricity																				
96	0.129	0.222	0.130	0.222	0.140	0.237	0.186	0.302	0.129	**0.221**	**0.128**	0.222	0.163	0.255	0.141	0.236	0.141	0.233	0.132	0.228
192	0.147	0.240	0.148	0.240	0.153	0.249	0.197	0.311	**0.144**	**0.237**	0.147	0.239	0.172	0.262	0.161	0.254	0.160	0.250	0.147	0.242
336	0.163	0.259	0.167	0.261	0.169	0.267	0.213	0.328	0.164	**0.257**	0.163	**0.257**	0.190	0.279	0.173	0.267	0.173	0.263	**0.162**	0.261
720	0.197	0.290	0.202	0.291	0.203	0.301	0.233	0.344	0.200	0.289	**0.195**	0.289	0.229	0.311	0.209	0.299	0.197	**0.284**	0.199	0.290
ILI																				
24	**1.319**	**0.754**	1.522	0.814	2.215	1.081	2.624	1.095	-	-	-	-	-	-	2.033	0.870	-	-	-	-
36	1.579	0.870	**1.430**	**0.834**	1.963	0.963	2.516	1.021	-	-	-	-	-	-	1.909	0.868	-	-	-	-
48	**1.553**	**0.815**	1.673	0.854	2.130	1.024	2.505	1.041	-	-	-	-	-	-	2.126	0.929	-	-	-	-
60	**1.470**	**0.788**	1.529	0.862	2.368	1.096	2.742	1.122	-	-	-	-	-	-	2.039	0.914	-	-	-	-
ETTh1																				
96	0.370	0.400	0.375	0.399	0.375	0.399	0.376	0.415	**0.353**	**0.381**	-	-	-	-	0.392	0.409	0.383	0.391	0.369	0.397
192	0.413	0.429	0.414	0.421	0.405	0.416	0.423	0.446	**0.373**	**0.394**	-	-	-	-	0.445	0.436	0.435	0.420	0.403	0.416
336	0.422	0.440	0.431	0.436	0.439	0.443	0.444	0.462	**0.392**	**0.414**	-	-	-	-	0.482	0.456	0.479	0.442	0.409	0.426
720	0.447	0.468	0.449	0.466	0.472	0.490	0.469	0.492	**0.445**	0.463	-	-	-	-	0.489	0.480	0.471	**0.461**	0.456	0.465
ETTh2																				
96	0.274	0.337	0.274	0.336	0.289	0.353	0.332	0.374	**0.225**	**0.300**	-	-	-	-	0.305	0.353	0.281	0.330	0.281	0.343
192	0.341	0.382	0.339	0.379	0.383	0.418	0.407	0.446	**0.274**	**0.334**	-	-	-	-	0.382	0.401	0.363	0.381	0.345	0.383
336	0.329	0.384	0.331	0.380	0.448	0.465	0.400	0.447	**0.317**	**0.368**	-	-	-	-	0.434	0.445	0.411	0.418	0.334	0.389
720	**0.379**	**0.422**	**0.379**	**0.422**	0.605	0.551	0.412	0.469	0.393	0.426	-	-	-	-	0.424	0.444	0.416	0.431	0.389	0.430
ETTm1																				
96	0.293	0.346	**0.290**	0.342	0.299	0.343	0.326	0.390	0.291	**0.340**	-	-	-	-	0.336	0.369	0.316	0.347	0.295	0.346
192	0.333	0.370	0.332	0.369	0.335	0.365	0.365	0.415	**0.325**	**0.362**	-	-	-	-	0.374	0.387	0.363	0.370	0.330	0.369
336	0.369	0.392	0.366	0.392	0.369	0.386	0.392	0.425	**0.353**	**0.382**	-	-	-	-	0.408	0.407	0.392	0.390	0.360	0.388
720	0.416	0.420	0.420	0.424	0.425	0.421	0.446	0.458	**0.413**	**0.413**	-	-	-	-	0.477	0.442	0.458	0.425	0.416	0.421
ETTm2																				
96	0.166	0.256	0.165	0.255	0.167	0.260	0.180	0.271	0.174	0.256	-	-	-	-	0.184	0.267	0.169	**0.248**	**0.163**	0.255
192	0.223	0.296	0.220	**0.292**	0.224	0.303	0.252	0.318	0.227	0.295	-	-	-	-	0.252	0.307	0.234	**0.292**	**0.215**	0.293
336	0.274	0.329	0.278	0.329	0.281	0.342	0.324	0.364	**0.266**	**0.323**	-	-	-	-	0.314	0.345	0.294	0.339	**0.266**	0.329
720	0.362	0.385	0.367	0.385	0.397	0.421	0.410	0.420	0.344	**0.372**	-	-	-	-	0.412	0.402	0.390	0.388	**0.342**	0.380

The prediction horizons for each data set are $T \in \{96, 192, 336, 720\}$, except for the ILI data set, as it comprises the prediction horizons $T \in \{24, 36, 48, 60\}$ due to its smaller size. The performances of the first four models, presented in Table 2, are taken from the original PatchTST paper [21]. The remaining columns are all taken from the respective work. Therefore, the results may not be directly comparable, since some models, that were implemented by succeeding papers e.g. PatchTST, sometimes produced different results than presented in the original work. Nie et al. [21] explain that the patches of PatchTST/64 and PatchTST/42 have a patch length P of 16 and a stride S of 8, such that they work with 64 and 42 patches, i.e. look-back window length L of 512 and 336, respectively. The look-back window length of the best performing FEDformer was chosen out of the values $L \in \{24, 48, 96, 192, 336, 720\}$. Lastly, DLinear has a look-back window length of 336. Out of the remaining models, CARD, Client and MCformer have a look-back length of 96, whereas PatchMixer and LIFT, which is the LIFT-plugin applied to PatchTST/42, have a look-back length of 336. The look-back window length of PITS was optimized in hyperparameter-optimization out of the values $L \in \{336, 512, 768\}$, however Lee et al. [17] did not clearly state which value was eventually chosen. Entries that are marked with "-" were not reported in the respective paper.

Overall, Table 2 illustrates that PatchMixer is often the best model, however the PatchTST/64 and LIFT-PatchTST/42 models are also often amongst the two best models. Nevertheless, a study that compares all approaches fairly would definitely be more insightful.

5 Conclusion

At last, I end this report with a short summary, in which I include some personal opinions and ideas.

In this report, I considered the paper "A Time Series is Worth 64 Words: Long-term Forecasting with Transformers" by Nie et al. [21]. While this title is justifiably similar to the influential paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" due to the adoption of a related patching process, the patching windows within PatchTST are not comprising 64 values. In fact, the patches only have a length of 8 to 16, hence the title can be considered unfitting. In the beginning, we have seen that PatchTST was motivated by the good performances of the linear model introduced by Zeng et al. [36] compared to previously established Transformers. Furthermore, Nie et al. [21] adopted their channel independent approach to Transformers. Additionally, they combined channel independence with patching to outperform the linear models on LTSF tasks. However, I believe that an adoption of the patching process to the linear model, would have further enlightened the discussion and comparison. Next, we looked at the model's components in detail. Furthermore, the work of Han et al. [11] helped tremendously to understand why channel independence performs better on many real-world data sets. Nevertheless, in my opinion it is important to utilize all available information wisely, especially in the time series domain where data sets are not as rich as in other domains, e.g. NLP or CV. Therefore, I think it is important to further improve channel dependence approaches. In Section 4, we compared the performances of methods that came after PatchTST. However, similar to Han et al. [11], I believe that in order to ensure fair comparisons it is important to analyze model typologies, e.g. IMS, DMS, CI or CD forecasting strategies, in more detail, whereas many authors simply compare architectures without discussing or comparing important typological changes. On top of that, although it is common to use point-wise forecasts, I believe the discussion of performances could be more insightful by producing and comparing probabilistic forecasts instead. Probabilistic time series forecasts are regarded superior for decision-making, as they quantify the uncertainty of time series forecasts [9]. Finally, as far as I am aware, additional exogenous variables, f.e. the day of the week, have not been considered yet and could provide models with beneficial information. However, it is difficult to find suitable exogenous variables for the LTSF task, as they would have to be available for the full prediction horizon.

References

[1] Sabeen Ahmed, Ian E. Nielsen, Aakash Tripathi, Shamoon Siddiqui, Ravi P. Ramachandran, and Ghulam Rasool. Transformers in Time-Series Analysis: A Tutorial. *Circuits, Systems, and Signal Processing*, 42(12):7433–7466, December 2023.

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, April 2018. arXiv:1803.01271 [cs].

[3] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, François-Xavier Aubet, Laurent Callot, and Tim Januschowski. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Computing Surveys*, 55(6):121:1–121:36, December 2022.

[4] G. E. P. Box and David A. Pierce. Distribution of Residual Autocorrelations in Autoregressive-Integrated Moving Average Time Series Models. *Journal of the American Statistical Association*, 65(332):1509–1526, December 1970. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1970.10481180.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. arXiv:1810.04805.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929.

[7] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, Min Wu, Chee Keong Kwoh, Xiaoli Li, and Cuntai Guan. Time-Series Representation Learning via Temporal and Contextual Contrasting, June 2021. arXiv:2106.14112.

[8] Jiaxin Gao, Wenbo Hu, and Yuntian Chen. Client: Cross-variable Linear Integrated Enhanced Transformer for Multivariate Long-Term Time Series Forecasting, May 2023. arXiv:2305.18838.

[9] Tilmann Gneiting and Matthias Katzfuss. Probabilistic Forecasting. *Annual Review of Statistics and Its Application*, 1(1):125–151, January 2014.

[10] Zeying Gong, Yujin Tang, and Junwei Liang. PatchMixer: A Patch-Mixing Architecture for Long-Term Time Series Forecasting, October 2023. arXiv:2310.00655.

[11] Lu Han, Han-Jia Ye, and De-Chuan Zhan. The Capacity and Robustness Trade-off: Revisiting the Channel Independent Strategy for Multivariate Time Series Forecasting, April 2023. arXiv:2304.05206.

[12] Wenyong Han, Tao Zhu Member, Liming Chen, Huansheng Ning, Yang Luo, and Yaping Wan. MCformer: Multivariate Time Series Forecasting with Mixed-Channels Transformer, March 2024. arXiv:2403.09223.

[13] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions. *International Journal of Forecasting*, 37(1):388–427, January 2021. arXiv:1909.00590.

[14] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible Instance Normalization for Accurate Time-Series Forecasting against Distribution Shift. October 2021.

[15] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 95–104, New York, NY, USA, June 2018. Association for Computing Machinery.

[16] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. An Experimental Review on Deep Learning Architectures for Time Series Forecasting. *International Journal of Neural Systems*, 31(03):2130001, March 2021.

[17] Seunghan Lee, Taeyoung Park, and Kibok Lee. Learning to Embed Time Series Patches Independently, February 2024. arXiv:2312.16427.

[18] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[19] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-Complexity Pyramidal Attention for Long-Range Time Series Modeling and Forecasting. October 2021.

[20] Sidra Mehtab and Jaydip Sen. A Time Series Analysis-Based Stock Price Prediction Using Machine Learning and Deep Learning Models. *International Journal of Business Forecasting and Marketing Intelligence*, 6(4):272, 2020. arXiv:2004.11697 [cs, q-fin, stat].

[21] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers, March 2023. arXiv:2211.14730.

[22] M. Raeesi, M. S. Mesgari, and P. Mahmoudi. Traffic time series forecasting by feedforward neural network: a case study based on traffic data of monroe. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-2/W3:219–223, October 2014.

[23] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, July 2020.

[24] Sean J. Taylor and Benjamin Letham. Forecasting at Scale. *The American Statistician*, 72(1):37–45, January 2018. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00031305.2017.1380080.

[25] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised Representation Learning for Time Series with Temporal Neighborhood Coding, June 2021. arXiv:2106.00750 [cs, stat].

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[27] Haoxin Wang, Yipeng Mo, Nan Yin, Honghe Dai, Bixiong Li, Songhai Fan, and Site Mo. Dance of Channel and Sequence: An Efficient Attention-Based Approach for Multivariate Time Series Forecasting, December 2023. arXiv:2312.06220 [cs].

[28] Qiang Wang, Shuyu Li, and Rongrong Li. Forecasting energy demand in China and India: Using single-linear, hybrid-linear, and non-linear time series forecast techniques. *Energy*, 161:821–831, October 2018.

[29] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in Time Series: A Survey, May 2023. arXiv:2202.07125 [cs, eess, stat].

[30] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. September 2022.

[31] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting, January 2022. arXiv:2106.13008 [cs].

[32] Yuexin Wu, Yiming Yang, Hiroshi Nishiura, and Masaya Saitoh. Deep Learning for Epidemiological Predictions. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1085–1088, Ann Arbor MI USA, June 2018. ACM.

[33] Wang Xue, Tian Zhou, Qingsong Wen, Jinyang Gao, Bolin Ding, and Rong Jin. CARD: Channel Aligned Robust Blend Transformer for Time Series Forecasting, February 2024. arXiv:2305.12095 [cs].

[34] Ling Yang and Shenda Hong. Unsupervised Time-Series Representation Learning with Iterative Bilinear Temporal-Spectral Fusion. In *Proceedings of the 39th International Conference on Machine Learning*, pages 25038–25054. PMLR, June 2022. ISSN: 2640-3498.

[35] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. TS2Vec: Towards Universal Representation of Time Series. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8980–8987, June 2022. Number: 8.

[36] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are Transformers Effective for Time Series Forecasting?, August 2022. arXiv:2205.13504.

[37] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A Transformer-based Framework for Multivariate Time Series Representation Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 2114–2124, New York, NY, USA, August 2021. Association for Computing Machinery.

[38] Yunhao Zhang and Junchi Yan. Crossformer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting. September 2022.

[39] Lifan Zhao and Yanyan Shen. Rethinking Channel Dependence for Multivariate Time Series Forecasting: Learning from Leading Indicators, April 2024. arXiv:2401.17548.

[40] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting, March 2021. arXiv:2012.07436.

[41] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting, June 2022. arXiv:2201.12740.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Bachelor-, Master-, Seminar-, oder Projektarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und in der untenstehenden Tabelle angegebenen Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Table 3: Tabelle der Hilfsmittel (z.B. generative AI).

Hilfsmittel	Verwendungszweck
ConnectedPapers	Literaturrecherche
GoogleScholar	Literaturrecherche
ResearchRabbit	Literaturrecherche

January 14, 2025

(Unterschrift)