Exploring the Effectiveness of Transformers for Long-Term Time Series Forecasting

Pranav Pativada

May 2024

1 INTRODUCTION

Time series forecasting (TSF) has been prevalent in many domains, with applications being found in stock market prediction, weather forecasting, energy management, and transportation [1, 2, 3, 4, 5]. Especially in today's data-driven world, we require effective models that can be used to forecast and solve a variety of domain independent problems [5, 6]. TSF models have seen significant advancements over the past few decades. Classical statistical models such as ARIMA, GARCH, and exponential smoothing were quite popular initially [7]. They provided robustness and simplicity, but failed to be flexible as they often required domain expertise and different parameters for distinct scenarios [1, 7]. This made them less applicable to domain independent TSF problems. **Machine learning (ML)**-based solutions improved on these drawbacks. Techniques such gradient boosting regression tree (GBRT) were introduced and offered improved flexibility and accuracy [1, 6]. ML based solutions could better capture inherent relationships in datasets, and were more adaptable to different domains. But, manual feature engineering and model design was required [1]. Given rich, multivariate time series data, inherent limitations also rose in capturing intricate patterns [1, 2, 3].

Recently, **deep learning (DL)** methods have surged in use for TSF, with many **state-of-the-art (SOTA)** models being built with DL [1]. DL methods offer improved accuracy, as they can capture the complex patterns that ML based solutions had trouble with. They can be used as pre-trained models for fine-tuning downstream tasks, and are also very effective in representation and ensemble learning [1, 3, 4]. This makes them extremely flexible and domain independent. Among the various DL methods that are used for TSF, we seek particular interest in a DL architecture that has been very popular recently - the **Transformer**.

The Transformer has been a SOTA architecture for a myriad of tasks in recent times, bringing significant success to areas such as **natural language processing (NLP)** and **computer vision (CV)** [1, 3, 8]. However, there has been much discussion about it's applicability to the **long-term time series forecasting (LTSF)** problem. It has been argued by many that the effectiveness of Transformers for LTSF should be re-visited, as simpler DL methods have rivalled that of Transformer-based solutions [1, 2, 3, 4]. In this review, we discuss the use of Transformers for the LTSF problem, and contrast Transformer-based solutions to non Transformer-based solutions to promptly identify their effectiveness.

1.1 Problem Definition

A TSF problem can be formulated as a prediction task. We denote the following terms.

- C: The number of variables that vary with time.
- L: The lookback period
- T: The number of timesteps to predict.

Thus, given a historical dataset $\mathcal{X} = \{X_1^t, ..., X_C^t\}_{t=1}^L$, we aim to predict the next T future timesteps $\hat{\mathcal{X}} = \{\hat{X}_1^t, ..., \hat{X}_C^t\}_{t=L+1}^{L+T}$ [1]. In the ideal scenario, we want to find an estimator F such that

$$\hat{\mathcal{X}} = F(\mathcal{X}; \theta) \quad \text{s.t} \quad \mathcal{L}(\hat{\mathcal{X}}, \mathcal{Y}) \to 0$$

where $\mathcal{Y}$ are the true values for the next T timesteps, θ are the parameters of the estimator, and the choice of F can vary between the aforementioned methods discussed. We define LTSF as a prediction task where specifically $T \gg 1$ [1, 3]. Specifically, focus in recent literature is on **mutlivariate LTSF**, in which we have $C > 1$. For $C = 1$, the prediction task is **univariate** [2].

Given the LTSF case, two plausible prediction methods are **iterated multi-step forecasting (IMS)** and **direct multi-step forecasting (DMS)** [1]. IMS methods learn a single step forecaster and iteratively apply it to get multi-step methods via an autoregressive estimation procedure [1]. DMS methods predicts the multi-step forecast at once, and no autoregressive method is applied [1]. While IMS has smaller variance, it of course suffers from error accumulation and gradient vanishing/explosion, as is the case with many DL-based autoregressive models.

For a small T and an accurate next step predictor, IMS is preferred [1, 4]. When T is large, directly optimising the objective via DMS is much better and yields better predictions [1, 9]. We note that recent DL methods, especially Transformer-based ones, mainly focus on DMS methods, though IMS methods are still implored.

1.2 The Transformer Architecture

Perhaps the primary reason for the Transformer's success is it's self-attention mechanism, which captures rich semantic information amongst a sequence [1, 10]. A Transformer works by feeding in a tokenised input sequence with positional encodings into a Transformer encoder, where multi-head self-attention occurs to encode the sequence as a learned representation [3, 10]. The learned representation through is then passed through a Transformer decoder, where masked self-attention occurs to generate the output [1, 3].

Sequence tokenisation can be done in many ways, one of which is **patching** [3, 8]. Patching is a technique used extensively in Transformer-based models. It helps retain semantic correlations by grouping up similar data, and reduces the size of the input tokens which helps with algorithmic complexity and improves performance. In **Vision Transformers (ViT)**, a 2D image is broken up into 16x16 patches [3, 11]. In the case of time series, this can be done by splitting the multivariate data into univariate patches and forecasting each variable separately, or grouping different timesteps of data together as patches. We see that the former is used in a Transformer-based model called **PatchTST** that has proven to be reasonably effective [3].

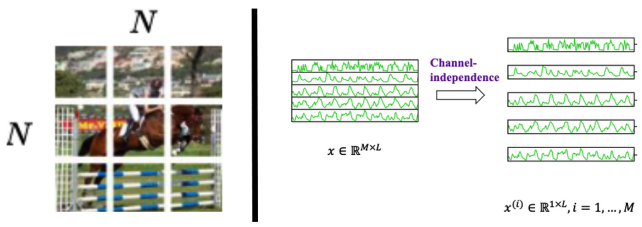
N
N
Channel-independence
$x \in \mathbb{R}^{M \times L}$
$x^{(i)} \in \mathbb{R}^{1 \times L}, i = 1, \ldots, M$

Figure 1: Left: Patching done in ViT's for semantic segmentation [11]. Right: Multivariate data being split into patches of univariate data for TSF [3].

The input tokens are mapped to a latent representation of dimension D via a trainable linear map, and positional encodings are employed to help preserve the ordering of the sequence [3]. Our observed signals are then encoded in the Transformer encoder with self-attention. Learnable parameters include KQV matrices, which transform the representations through the different heads in the Transformer [10]. This effectively extracts the complex semantic relationships in the data. Once extracted, a prediction is made by feeding our representation through the Transformer decoder which generates the output sequence via masked self-attention [10]. We then project the output representation down to the correct dimension and selecting the most likely option. We note that temperature parameters can be used to influence the output selected, incorporating a degree of randomness if needed.

1.3 Problems with the Transformer

Though Transformers retain excellent semantic correlations within sequences, we tackle two main problems with TSF data.

Lack of semantic correlation: There exists a lack of useful semantic information when observing a sequence of successive iterates in TSF data. Given we just have numerical data, semantic information is harder to extract. Temporal information is much richer in TSF, as a lot of data is seasonal and event based [1, 12]. We want to look at the change in temporal information over a continuous set of points, rather than look at numerical data and considering the semantic value of each iterate by itself. As TSF data consists of values at different time points, temporal order and numerical relationships are more important in comparison to any semantic context [1].

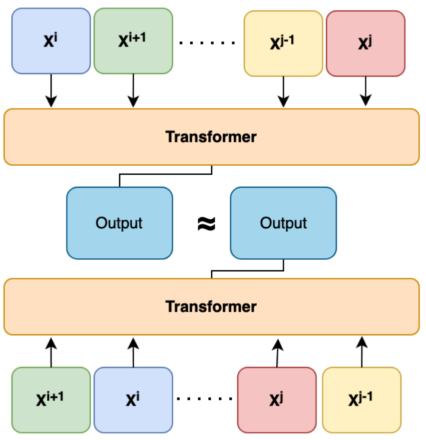

Figure 2: Permutation invariance: output prediction of Transformer is approximately similar regardless of change in order of input sequence.

Permutation-invariance: The self-attention mechanism treats inputs as a set rather than a sequence, meaning it is order invariant [1, 10]. Usually, this is great as it allows for parallel processing of the input, making the Transformer very efficient. However, this also means that re-ordering the data does not greatly impact the prediction. This is problematic, as the order of TSF data is crucial - changing the order should invoke a completely new set of information. This is unlike words in a sequence or patches in an image, as they themselves can carry rich semantic meanings. While positional encodings are employed, temporal information loss still occurs when self-attention is applied [1]. Of course, this is because the mechanism is not designed to preserve the strict temporal order that is essential for TSF analysis.

While these problems exist, unique solutions and problem re-formulations may result in the Transformer still being successful. In the following section, we cover recent literature in LTSF, with a primary focus on Transformer and non-Transformer based DL methods. We also briefly look at other related work in LTSF.

2 RECENT WORKS

2.1 Transformer-Based Methods

Transformer based LTSF designs mainly go through 4 pipeline stages: Pre-processing, Embedding, Encoding and Decoding [1]. We analyse recent works with respect to these stages.

Pre-processing: TSF pre-processing usually involves normalisation to zero-mean [4]. However, further methods are also used. **Autoformer**, a transformer-based solution, uses seasonal-trend decomposition in each neural block. This enhances predictability [9]. In specific, a **moving average (MA)** kernel is applied to the input to extract out the trend of the time series. Autoformer defines the seasonal component as the difference between the original input and the trend component [1, 9]. As such, it aims to alleviate the problems discussed in Section 1.3, since it extracts the seasonal trends out. Another work, **FedFormer**, builds on this [13]. It introduces a mixture of experts and mixes the trend components extracted using different sizes of MA kernels. This allows for the trend to be captured at multiple scales. Different kernel sizes can smooth out variations over different time spans, allowing it to better identify short and long-term trends in the data.

Meanwhile, PatchTST pre-processes the input by splitting multivariate data with C variates into C separate univariate streams, as seen in Section 1.2 [3]. This makes them channel-independent. That is, each token of the transformer only contains information from a single channel. We refer to Figure 1 for an example. Other works such as **Triformer** also uses patch-based techniques for pre-processing [14].

Embedding: Besides positional encodings that preserve local order, global temporal information like timestamps and re-curring events (holidays) are used as well [1, 15]. This is because they help inform the model of potential trends. Many of the SOTA Transformer models use a combination of embeddings to help with identifying and learning trends. Examples of these include temporal embeddings with learnable timestamps, and channel embeddings so the model can identify trends specific to certain channels [1, 3]. While PatchTST uses just positional encodings, **Triformer** uses a learnable pseudo-timestamp embedding to act as a query for each patch [3, 14]. Most recently in ICLR 2024, the **Data-dependant Approximate Analytical Model (DAM)** model was introduced, which combines the Transformer with classical statistical techniques [4]. It combines the timestamp technique with basis co-efficient embeddings that correspond to frequency information [4].

Encoding: Self-attention is used to encode the input sequence and extract semantic dependencies between pairwise elements. However, many Transformer methods have an $O(L^2)$ complexity if they just use the vanilla Transformer encoder [1]. This is the case with PatchTST [3]. **LogTrans**, a baseline solution, introduces a bias into the self-attention self-attention scheme to improve efficiency [16]. This reduces the complexity to $O(L \log L)$. Another model, **Informer** also achieves this. **Pyraformer** reduces it further to $O(L)$ as it uses a pyramidal attention scheme [17]. FEDFormer also obtains $O(L)$ complexity by using a Fourier transform technique. More information can be found in 2.3 [13]. These self-attention mechanisms help extract the semantic, and now with pre-processing and extra embeddings, the temporal information for better prediction.

Decoding: The vanilla Transformer decodes in an auto-regressive manner. This results in slow inference and error accumulation. This is especially the case if T is large. Many Transformer variants use DMS strategies such a generative-style decoder, as is the case for Informer [1, 15]. Pyraformer uses a fully-connected layer as the decoder, which concatenates spatial and temporal information to do prediction. FEDFormer and Autoformer use a decomposition scheme. They implement an auto-correlation mechanism which decomposes the trend components and uses it for prediction [9, 13].

A unique way of handling this problem is PatchTST. PatchTST handles each patch separately and hence does not use a decoder [3]. As such, it can predict the output directly in a non auto-regressive manner. DAM also does something similar. It leverages the basis embeddings and the frequency specific information to directly predict values, not being auto-regressive [4].

2.2 Non-Transformer Based DL Methods

LTSF-Linear was the first successful Non-Transformer Based DL method that rose to fame, as it pointed out to the community to re-visit Transformer for LTSF [1]. LTSF-Linear, a collection of linear models, stood out by introducing a simple one layer model [1]. Simply put, the basic formulation used a weighted sum of the of the historical time series data $\mathcal{X}$ and learned a direct mapping to predict $\hat{\mathcal{X}}$. Within LTSF-Linear, DLinear was introduced, which used MA kernel scheme from Auto and FEDFormer, where the single linear layers were applied to each component and used to make the final prediction [1]. NLinear was also introduced, which normalised the input sequence by disgarding the last value and subtracting it from the sequence, then adding it back in like a residual connection after. Both these models proved to be very performant since their introduction [1, 6]. We specifically focus on DLinear in this review as the most recent literature uses it as the baseline non-Transformer DL solution.

Building on the idea that simplicity can be powerful, the Time-series Dense Encoder (TiDE) leverages **multi-layer perceptrons (MLPs)** [2]. It does this by using a dense encoder to create an embedding of the input sequence. It then decodes the hidden representation from the embedding via a dense decoder for future predictions [2]. These dense encoders and decoders are just MLP-based residual blocks. There is no self-attention done here. Prediction is done through a temporal decode - a residual block which is what actually generates the output [2]. This has proven to be successful and has achieved multiple SOTA instances on many datasets. TiDE also shows that a near optimal error rate can be achieved if the dataset were under linear dynamical assumptions.

Another MLP-based solution introduced in NeurIPS 2023 is ***Fre*quency-domain MLPs for *T*ime *S*eries (FreTS)** [6]. FreTS uses a new architecture of frquency-domaion MLPs for TSF. It operates by transforming time-domain signals into the frequency by using a **Discrete Fourier Transform (DFT)** to capture global dependencies more effectively [6, 13]. This frequency-domain representation allows FreTS to focus on the most relevant parts of the signal, improving it's ability to learna and predict long-term trends [6]. It leverages the compact nature of the frequency domain and with the DFT, can mitigate noise, achieving SOTA on many benchmarks. We talk about some more frequency based methods and their uses in the following section.

2.3 Other Related Work

Several LTSF solutions have focused on extracting knowledge about the frequency domain. This is because it is argued that frequency domain information enables a more global perspective of time series [15]. The main technique here is to use a Fourier transform. We note that this technique is already used in both Transformer and non-Transformer based solutions. Apart from the already mentioned FreTS model, Autoformer uses a Fast Fourier Transformer (FFT) that implements the auto-correlation mechanism [9]. FEDformer also uses a DFT to achieve the $O(L)$ complexity [13]. It does this by using Fourier enhanced block [13].

StemGNN perform graph convolutions based on Graph Fourier Transform as a way to compute series correlation [18]. CoST implements an ML-based approach. It uses contrastive learning and a DFT to map intermediate features to the frequency domain [2]. This allows it to enable interactions with the representation. Recent classical approaches also exist. FilM uses Legendre polynomials and Fourier projections to preserve historical information and remove noisy signals [12].

Unique problem re-formulations have also been used for LTSF. In NeurIPS 2023, a novel method was introduced where TSF data was converted into line graph images [8]. It then utilises a pre-trained ViT model for time series prediction. Though currently it is used for time-series classification, portability may soon come for LTSF. This method simplifies specialised algorithms and servers as a framework for future time series modelling tasks.

3 RESULTS AND ANALYSIS

To contrast the use of Transformers for LTSF, we combine results for non-Transformer and Transformer-based models. We compare the staple non-Transformer based DL solutions, DLinear and TiDe, to a variety of Transformer based solutions we discussed in Section 2.1. We detail the combined results in Figure 3.

3.1 Choice of Datasets for Comparison

For our comparison, we use the following datasets [2]. We choose these are they were the most tested across both Transformer and non Transformer-based solutions.

Dataset	#Time-Series	#Time-Points	Frequency
Electricity	321	26304	1 Hour
Traffic	862	17544	1 Hour
Weather	21	52696	10 Minutes
ETTh1	7	17420	1 Hour
ETTm1	7	69680	15 Minutes

Table 1: Datasets used for review

Traffic and electricity are the biggest datasets, with > 800 and > 300 time series each of which contain thousands of points. This is followed by Electricity Transformer Temperature (ETT) datasets, who are separated by different frequencies - h for 1-hour intervals and m for 15-minute intervals [15]. These datasets are used to predict $T = 96, 192, 336$ and 720 steps into the future. In Figure 3, we denote the future time steps as the horizon.

Mean Squared Error (MSE) and Mean Absolute eEror (MAE) are used for comparison, as they are standard metrics of choice for many works.

*3.2 **Analysis***

Primary LTSF results from Figure 3 show that Transformers perform reasonably well on ETTh1, ETTm1 and weather datasets. For ETTh1, the most recent ICLR 2024 HSR-tuned DAM model achieves the best MSE and MAE for Etth1. ETTm1 is followed by FEDFormer and AutoFormer achieving the net best, but with DAM, TiDE and DLinear following closely after. We do note that we have only compared 2 non-Transformer based solutions to 7 transformer based solutions - 2 non auto-regressive and 5 auto-regressive. Still, non-Transformer based DL solutions outperform Transformers on the Traffic and Electricity datasets. Specifically, TiDe achieves the best MSE and MAE scores for Traffic, and achieves the best MSE for Electricity. For both datasets, DLinear follows closley behind. This is surprising, as these datasets had much more data than the others and so it makes sense for a Transformer-based solution to capitalise and perform better.

Between the Transformer-based solutions, PatchTST and DAM outperform the rest on all datasets except for ETTm1, where FEDFormer and Autoformer employ much better MAE and MSE scores than the rest. This is likely because these methods bring out the mentioned frequency specific knowledge in Section 2.3. This allows them to benefit with temporal feature extraction, especially given the consistency of the dataset in comparison to something like Weather, and the lowered frequency of 15 minutes. Furthermore, when we include TiDE and DLinear, we see that they consistently outperform Crossformer, Pyraformer, and Informer in all instances. They also outperform FEDFormer and Autoformer on everything but the ETTm1 dataset. Similarly, PatchTST and DAM, also outperform the rest of the Transformer solutions. This brings us to the conclusion that autoregressive transformer solutions are not that effective for LTSF. As such, the comparison for SOTA is really between both of these methods and DAM and PatchTST.

The decision to choose which is better for LTSF is an interesting one. DAM is a brand new architecture. It uses a combination of statistical methods and a **history sampling regime (HSR)** to ingest context data into the transformer [4]. PatchTST is a simpler Transformer solution, which uses a vanilla Transformer encoder and a standard Transformer backbone for prediction. Meanwhile, DLinear is extremely simple - a one layer linear model, and TiDE is an encoder-decoder model [1, 2]. To answer this question, as is the point of this this review, we implore performance not only LTSF, but different TSF tasks and also consider memory footprint.

*3.3 **TSF Learning Tasks***

If the task were zero-shot learning, transformer solutions, specifically DAM, win by a large margin.

	DAM		PatchTST		DLinear	
	MSE	MAE	MSE	MAE	MSE	MAE
Illness	**1.964**	**0.920**	4.266	1.497	4.173	1.460
Weekdays	**0.942**	**0.573**	1.502	0.736	1.135	0.672
UCIPower	1.595	**0.514**	**1.590**	0.545	1.615	0.570
Azure	1.531	**0.640**	1.630	0.667	**1.444**	0.649
MTemp	**0.950**	**0.639**	1.030	0.713	0.975	0.699
MWeb	**12.633**	**0.586**	12.656	0.700	13.321	0.732
MWeather	**0.479**	**0.506**	0.907	0.792	0.862	0.769
MMeters	**0.836**	**0.499**	0.895	0.546	0.940	0.566

Figure 4: Zero-shot fine-tuning learning performance between DAM, PatchTST and DLinear. Results are averaged out over horizons 96, 192, 336, and 720 [4].

Figure 4 shows that DAM achieves SOTA zero shot learning performance on 14/16 instances. DAM generalises very well to new datasets, whereas PatchTST and DLinear perform similarly, but much worse compared to that of DAM. Moreover, though not explicitly shown here, Transformers are also better when training from scratch. PatchTST and DAM combined achieve 9/16 SOTA benchmarks for this case, compared to DLinear which is only the best in 1/16 [4].

Models		PatchTST						DLinear	
		Fine-tuning		Lin. Prob.		Sup.			
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
Weather	96	**0.145**	**0.195**	0.163	0.216	0.152	0.199	0.176	0.237
	192	**0.193**	**0.243**	0.205	0.252	0.197	**0.243**	0.220	0.282
	336	**0.244**	**0.280**	0.253	0.289	0.249	0.283	0.265	0.319
	720	0.321	0.337	**0.320**	0.336	**0.320**	**0.335**	0.323	0.362
Traffic	96	0.388	0.273	0.400	0.288	**0.367**	**0.251**	0.410	0.282
	192	0.400	0.277	0.412	0.293	**0.385**	**0.259**	0.423	0.287
	336	0.408	0.280	0.425	0.307	**0.398**	**0.265**	0.436	0.296
	720	0.447	0.310	0.457	0.317	**0.434**	**0.287**	0.466	0.315

Figure 5: PatchTST transfer learning results. Models are pre-trained on the Electricity dataset [3].

Transfer learning is also favoured, as PatchTST outperforms DLinear [3]. It achieves better results especially when being fine-tuned for Weather, and is also better when supervised for Traffic. Though not shown here, we also note that DLinear outperforms FedFormer, Autoformer and Informer [1, 3]. As such, we once again come to a conclusion where autoregressive transformer models fail in comparison to DLinear and PatchTST.

Horizon	TiDE		DLinear		DAM		DAM$_{HSR-tuned}$		PatchTST64		Crossformer		Pyraformer		FEDFormer		Autoformer		Informer	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1																				
96	0.375	**0.398**	0.375	0.399	0.373	0.404	**0.367**	0.401	0.371	0.401	0.427	0.431	0.397	0.422	0.420	0.410	0.500	0.482	0.846	0.795
192	0.412	0.422	0.412	0.420	0.409	0.439	**0.390**	**0.415**	0.401	0.433	0.510	0.501	0.519	0.591	0.459	0.445	0.590	0.524	1.238	0.932
336	0.435	0.433	0.439	0.443	0.409	0.427	**0.367**	0.419	0.432	**0.414**	0.555	0.506	0.661	1.019	0.506	0.459	0.621	0.537	1.181	0.809
720	0.454	0.465	0.501	0.490	0.438	0.462	**0.421**	**0.430**	0.452	0.447	0.592	0.540	0.812	0.688	0.543	0.499	0.671	0.561	1.166	0.823
ETTm1																				
96	0.306	0.349	0.299	0.343	0.309	0.358	0.306	0.354	0.303	0.301	0.374	0.412	**0.162**	0.529	**0.162**	0.300	0.317	**0.274**	0.386	0.368
192	0.335	0.366	0.353	0.365	0.343	0.379	0.343	0.357	0.330	0.366	0.452	0.457	0.617	0.519	**0.216**	0.325	0.334	**0.296**	0.386	0.368
336	0.364	0.384	0.369	0.386	0.351	0.348	0.351	0.348	0.364	0.370	0.494	0.443	0.685	0.749	**0.246**	0.339	0.338	**0.300**	0.394	0.373
720	0.413	0.413	0.425	0.421	0.403	0.410	0.407	0.418	0.410	0.420	0.542	0.507	1.029	0.975	**0.300**	0.370	0.378	**0.317**	0.408	0.391
Electricity																				
96	0.132	0.229	0.140	0.237	0.159	0.165	0.154	**0.159**	**0.129**	0.224	0.147	0.248	0.285	0.378	0.193	0.308	0.317	0.274	0.368	0.386
192	**0.147**	0.243	0.153	0.249	0.176	0.188	0.177	**0.186**	0.162	0.241	0.168	0.262	0.298	0.403	0.246	0.325	0.334	0.296	0.386	0.386
336	**0.161**	0.261	0.169	0.267	0.181	**0.187**	0.178	0.189	0.164	0.260	0.205	0.302	0.306	0.390	0.305	0.414	0.524	0.338	1.012	0.785
720	**0.196**	0.294	0.203	0.301	0.242	**0.237**	0.237	0.238	0.202	0.259	0.237	0.303	0.338	0.306	0.305	0.415	0.524	0.338	1.012	0.785
Traffic																				
96	**0.336**	0.253	0.410	0.282	0.468	0.335	0.460	0.332	0.360	**0.249**	0.544	0.328	0.518	0.687	0.587	0.366	0.660	0.408	0.739	0.467
192	**0.346**	**0.257**	0.423	0.287	0.481	0.347	0.474	0.339	0.408	0.274	0.544	0.365	0.558	0.692	0.696	0.382	0.696	0.408	0.739	0.467
336	**0.355**	**0.260**	0.436	0.296	0.486	0.354	0.479	0.341	0.389	**0.260**	0.571	0.343	0.579	0.673	0.696	0.382	0.696	0.408	0.739	0.467
720	**0.386**	**0.273**	0.466	0.315	0.547	0.391	0.538	0.377	0.492	0.308	0.619	0.358	0.707	0.754	0.696	0.382	0.696	0.408	0.739	0.467
Weather																				
96	0.166	0.222	0.176	0.237	0.195	0.238	0.194	0.233	**0.148**	0.174	0.157	**0.172**	**0.203**	0.294	0.266	0.306	0.336	0.307	0.336	0.306
192	0.209	0.263	0.220	0.282	0.205	0.283	0.203	0.247	0.244	**0.233**	0.267	0.249	0.294	0.292	0.294	0.296	0.336	0.307	0.336	0.306
336	0.254	0.301	0.300	0.319	0.244	0.327	**0.241**	0.280	0.283	**0.261**	0.365	0.337	0.409	0.294	0.298	0.297	0.358	0.325	0.428	0.350
720	**0.313**	0.340	0.323	0.362	0.325	0.365	0.321	0.334	0.351	**0.308**	0.492	0.430	0.494	0.389	0.428	0.413	0.551	0.478	0.698	0.596

Figure 3: Combined results of Transformer and non-Transformer DL solutions. TiDE and DLinear are the non-Transformer DL methods. The rest are various Transformer based solutions. We note that DAM and PatchTST are **not autoregressive**. The DAM baseline is denoted as DAM, whereas the DAM-HSR implies that the model was tuned using HSR samples. We combine the results with horizons 96, 192, 336 and 720. Highlighted in blue are the best MAE scores for each dataset w.r.t to it's horizon. Highlighted in red is the best MSE score. These results were taken from the following sources: [1, 2, 3, 4]

3.4 Memory Footprints

When contrasting both choices, we take into account memory footprint and complexity analysis. As mentioned in 2.1, many transformers have complexity $O(L^2)$ [15]. While self attention approximations exist and theoretically the complexity can be brought down to to $O(L)$, we find that inference time and memory are still a cost [1, 2]. Figure 6 shows the comparison of DLinear to a few other Transformer solutions.

Method	MACs	Parameter	Time	Memory
DLinear	**0.04G**	**139.7K**	**0.4ms**	**687MiB**
Transformer×	4.03G	13.61M	26.8ms	6091MiB
Informer	3.93G	14.39M	49.3ms	3869MiB
Autoformer	4.41G	14.91M	164.1ms	7607MiB
Pyraformer	0.80G	241.4M*	3.4ms	7017MiB
FEDformer	4.41G	20.68M	40.5ms	4143MiB

Figure 6: Memory footprint of DLinear compared to other Transformer solutions. MAC is the number of multiply-accumulate operations. The time is the time taken to do inference. For Pyformer, the asterisk signifies that majority of it's parameters come from it's decoder [1, 17].

We find that for the above transformer models, the memory can be up to tenfold that of DLinear. Inference time also is slower. This is to be expected, as the parameters for DLinear are much less than that of the rest. Though quantitative results are not explicitly shown, TiDE follows suit - as it is 5 times faster than the above Transformer based solutions and has less parameters [2]. This trend in inference time and memory would likely follow suit with DAM and PatchTST, though one could argue that they would have less parameters and could be more efficient than the other Transformer solutions given they're not autoregressive.

4 DISCUSSION

Our review of Transformers for LTSF brings us to a nuanced understanding. While Transformers have achieved SOTA results in other domains, their application to LTSF reveals both limitations and advantages.

We first note that auto-regressive Transformer solutions perform worse than other solutions, not only for LTSF but also for zero-shot and transfer learning. DAM, PatchTST, and TiDE all achieve the best results on many instances of the 4/5 datasets for LTSF. This was closely followed up by DLinear, which while did not achieve the best results, performed well given it's extreme simplicity. DLinear also outperforms these on zero-shot learning and transfer learning. Thus, we arrive at the conclusion that auto-regressive Transformer solutions aren't as effective for LTSF and other TSF tasks.

However, we do note that the performance of auto-regressive Transformers on ETTm1 showcase their potential when enhanced with techniques like Fourier transforms. This is seen in FEDFormer and Autoformer who did achieve the best MSE and MAE for ETTm1. While auto-regressive models may not be the best for LTSF, we implore the use of frequency specific methods such as Fourier analysis in the future.

Non auto-regressive Transformers such as DAM and PatchTST have demonstrated great performance in not only the LTSF problem, but also other TSF tasks. In particular, zero-shot learning and transfer learning work very well with these solutions. The unique architecture of DAM, which combines statistical methods and does HSR sampling, allows it to achieve SOTA performance on a significant number of instances for zero-shot learning. Similarly, PatchTST's use of a vanilla Transformer encoder without an auto-regressive decoder helps in achieving strong results by directly predicting the output sequence. As a result, we arrive at the conclusion that non auto-regressive Transformers can still be effective.

Ablation studies also support the above claim, as PatchTST without it's temporal decoder performs much worse especially for large T [2]. This can be seen in Figure 7.

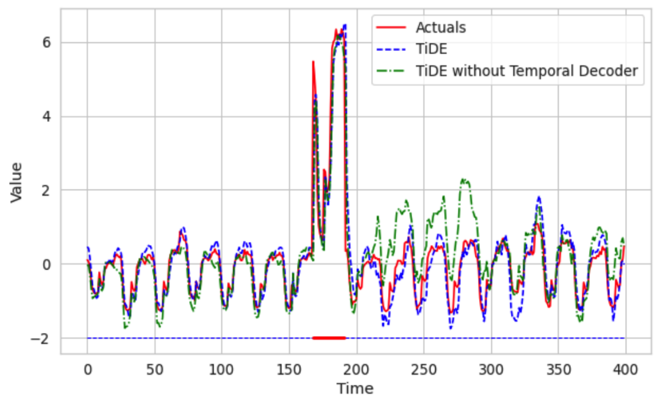

Figure 7: TiDE performance with and without the temporal decoder [2].

When comparing these models to non-Transformer-based DL solutions like DLinear and TiDE, we observe that simpler models often rival and even surpass Transformers. This is the case even when the dataset contains a lot of data, which should be advantageous for Transformers - but non-Transformer solutions like TiDE come out on top. Though, we note that these simpler solutions do not outperform the non auto-regressive transformers on downstream TSF tasks like zero-shot or transfer learning. As such, we conclude that they are not as generalisable to different datasets and different domains as Transformer solutions like DAM and PatchTST. Thus, we note that Transformer can be better used for more domain independent TSF tasks.

While this is an upside of Transformers, the memory footprint and inference time as highlighted in Section 6 makes it a drawback. Though we note that most of the long inference times is due to the auto-regressive solutions, these simpler models are still likely to be more efficient than DAM and PatchTST. This is especially the case with DLinear - which is ten times as fast as the auto-regressive Transformers.

5 CONCLUSION

While many gauge the effectiveness of Transformer-based models for LTSF, we find that Transformers can excel, especially if they are non auto-regressive. They have many SOTA instances in datasets such as ETTh1 and ETTm1. But, their advantage is overtaken by their memory complexity and inference time, as they can be up ten times worse in these metrics as discussed in Section 3.4 in comparison. Non-Transformer based DL models like DLinear and TiDE have greater efficiency and rival their accuracy. This is all while achieving the SOTA or rivalling them in many instances. These simpler models are also better in scenarios where time or computational budget is restricted. Given all of the above, we consider that non-Transformer based DL methods like DLinear and TiDE are more effective for the LTSF problem.

However, we note that Transformers are very effective for TSF tasks such as zero-shot and transfer learning. They generalise better to other datasets and can be used to extract good representations in the case of DAM and PatchTST. Because of this, Transformers can still be effective depending on the TSF problem instance. Thus, while LTSF might prove to prefer non-Transformer DL models, we have seen that unique problem reformulations such as using ViT's (as discussed in 2.3 could level the playing field in the future. Future work should explore hybrid approaches that integrate the strengths of both model types to advance LTSF capabilities.

References

[1] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting?, 2022.

[2] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with tide: Time-series dense encoder, 2024.

[3] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023.

[4] Luke Nicholas Darlow, Qiwen Deng, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Artjom Joosen, Adam Barker, and Amos Storkey. DAM: Towards a foundation model for forecasting. In *The Twelfth International Conference on Learning Representations*, 2024.

[5] Jing Su, Dirui Xie, Yuanzhi Duan, Yue Zhou, Xiaofang Hu, and Shukai Duan. Mdcnet: Long-term time series forecasting with mode decomposition and 2d convolution. *Knowledge-Based Systems*, page 111986, 2024.

[6] Kun Yi, Qi Zhang, Wei Fan, Shoujin Wang, Pengyang Wang, Hui He, Defu Lian, Ning An, Longbing Cao, and Zhendong Niu. Frequency-domain mlps are more effective learners in time series forecasting, 2023.

[7] Xiaoqian Wang, Yanfei Kang, Rob J. Hyndman, and Feng Li. Distributed arima models for ultra-long time series. *International Journal of Forecasting*, 39(3):1163–1184, July 2023.

[8] Zekun Li, Shiyang Li, and Xifeng Yan. Time series as images: Vision transformer for irregularly sampled time series, 2023.

[9] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2022.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[11] Lian Xu, Wanli Ouyang, Mohammed Bennamoun, Farid Boussaid, and Dan Xu. Multi-class token transformer for weakly supervised semantic segmentation, 2022.

[12] Tian Zhou, Ziqing Ma, Xue wang, Qingsong Wen, Liang Sun, Tao Yao, Wotao Yin, and Rong Jin. Film: Frequency improved legendre memory model for long-term time series forecasting, 2022.

[13] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting, 2022.

[14] Razvan-Gabriel Cirstea, Chenjuan Guo, B. Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. Triformer: Triangular, variable-specific attentions for long sequence multivariate time series forecasting-full version. *ArXiv*, abs/2204.13767, 2022.

[15] 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, 2021.

[16] Xingqing Nie, Xiaogen Zhou, Zhiqiang Li, Luoyan Wang, Xingtao Lin, and Tong Tong. Logtrans: Providing efficient local-global fusion with transformer and cnn parallel network for biomedical image segmentation. In *2022 IEEE 24th Int Conf on High Performance Computing Communications; 8th Int Conf on Data Science Systems; 20th Int Conf on Smart City; 8th Int Conf on Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*, pages 769–776, 2022.

[17] 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. In *International Conference on Learning Representations*, 2022.

[18] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Conguri Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Spectral temporal graph neural network for multivariate time-series forecasting, 2021.