scientific reports

Check for updates

OPEN

A multiscale model for multivariate time series forecasting

Vahid Naghashi[1,2], Mounir Boukadoum[1,2] & Abdoulaye Banire Diallo[1,2]✉

Transformer based models for time-series forecasting have shown promising performance and during the past few years different Transformer variants have been proposed in time-series forecasting domain. However, most of the existing methods, mainly represent the time-series from a single scale, making it challenging to capture various time granularities or ignore inter-series correlations between the series which might lead to inaccurate forecasts. In this paper, we address the above mentioned shortcomings and propose a Transformer based model which integrates multi-scale patch-wise temporal modeling and channel-wise representation. In the multi-scale temporal part, the input time-series is divided into patches of different resolutions to capture temporal correlations associated with various scales. The channel-wise encoder which comes after the temporal encoder, models the relations among the input series to capture the intricate interactions between them. In our framework, we further design a multi-step linear decoder to generate the final predictions for the purpose of reducing over-fitting and noise effects. Extensive experiments on seven real world datasets indicate that our model (MultiPatchFormer) achieves state-of-the-art results by surpassing other current baseline models in terms of error metrics and shows stronger generalizability.

Time series forecasting plays an essential role in many fields, including finance, agriculture, meteorology and energy consumption domains. Motivated by its widespread application in different fields of Natural Language Processing and Computer Vision[1,2], Transformer[3] is exploited in various time series applications (forecasting, imputation and classification) with some modifications in its architecture[4,5]. While recent studies have raised doubts about the performance of Transformers, specifically when employing linear models with superior performance[6], the inherent capabilities of Transformers remain promising[7,8], particularly in representing complex and non-linear datasets. This underscores the need for further modifications to fully harness their capability in the domain of time series forecasting.

Real-world multivariate time series exhibit high correlations between different variates and fluctuations at various temporal scales. For example, electricity consumption shows specific temporal variations spanning seasonal, daily and hourly granularities. Figure 1, illustrates a time series of a stock over one year, in which relations between patches of different scales are critical to capture more information regarding the local and global temporal dependencies from various perspectives. This calls for multi-scale modeling of time series[9] and representation of inter series correlations[10].

Most of the existing time series forecasting models lack an efficient mechanism for multi-scale representation and they totally rely on a single scale or time resolution. Although some of the baseline models consider multi-scale representation in their modeling process[8,11], however, these models employ separate sets of parameters for capturing temporal dependencies at each scale, which significantly increases time complexity and the risk of overfitting. Furthermore, most of the aforementioned methods, ignore the cross-channel relationships between time series channels which has been proved to be critical in time series analysis task[10]. Another limitation of the current research is the single step decoding of the encoded representation, particularly in dealing with long horizon prediction, which raises the risk of overfitting and makes the model more susceptible to be affected by noise.

To leverage the capabilities of Transformers for addressing the above mentioned issues, we aim to enhance the capture of both multi-scale and cross-channel dependencies, thereby aggregating this information for time series representation. We further divide input series into local patches to process the input time series and since the variable independence has been proved to be more efficient[7], we feed the multivariate series to the Transformer blocks by keeping the variate (channel) dimension intact. We further divide the series into patches of different size and map patch length to the model dimension by using 1-dimensional convolution in order to leverage the local information inside a patch (intra-patch relations). Instead of using several Transformer blocks, we first embed the time series using different scales and aggregate them into the same model dimension (feature space). Then, we process the mapped series using a Transformer block, followed by an channel-wise

[1]Computer Science, Université du Québec à Montréal, Montreal, Canada. [2] Vahid Naghashi, Mounir Boukadoum and Abdoulaye Banire Diallo contributed equally to this work. ✉email: diallo.abdoulaye@uqam.ca

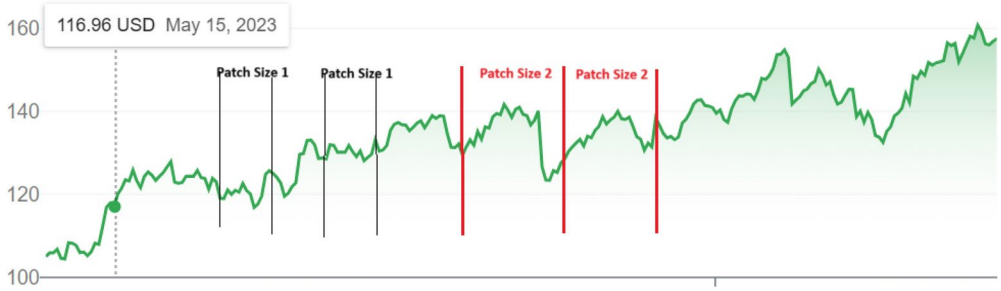

Fig. 1. Multi-scale dependencies in a real-world time series instance. The temporal patterns within patches of lengths Patch Size 1 and Patch Size 2 show similar trends and seasonality. Capturing these relations across different scales is crucial for analyzing time series data effectively.

Transformer component to capture cross-series dependencies. To further improve the Transformer for time series forecasting task, we introduce a channel dimensionality reduction method inside the multi-head self attention module when applied on the channel dimension, which helps to reduce over-fitting noise, specifically when dealing with the time series composed of many variates (channels). We additionally, devise a multiple step and pseudo auto-regressive decoding of prediction window by generating the predictions segment by segment using the encoded series and previously generated segment, in order to reduce the noise impact, particularly in long-term forecasting scenarios. The effectiveness of the proposed model is verified through different real-world benchmarks.

Here is the key summarization of our contributions:

1. We propose a time series forecasting model which effectively utilizes the multi-scale information and captures the inter-series dependencies among different channels.
2. Our model demonstrates superior performance on various time series datasets and shows robust performance in time series forecasting task with long input length.
3. We design a multi-scale time series embedding method which captures the local temporal information from various scales and divides time series to patches of different lengths, preparing it to represent long term temporal correlations from multiple temporal aspects.
4. We propose a pseudo auto-regressive (multi-step) decoding scheme to forecast the time series in multiple sequential steps rather than decoding the whole prediction length using a single linear layer in order to reduce the noise effect.

Related work

Time series forecasting involves predicting future values of one or multiple series based on the historical information. Time series forecasting methods are mainly categorized into classical and deep learning models. Among the statistical models, ARIMA[12] and methods based on exponential smoothing[13] are well-known baselines for time series forecasting. The deep learning methods are based on Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN) and Transformer models. Among the RNN based model, DeepAR exploits RNN and auto-regressive decoding to predict the time series[14]. CNN models use convolutions to extract temporal and sub-series dependencies[15,16]. For example, SCINet[16] utilizes multiple convolutions to extract temporal information from different down sampled versions of the series. TimesNet[15] on the other hand, converts the original one-dimensional time series into two-dimensional series and uses convolutions to capture inter-period and intra-period information. Some other linear models based on Multi-Layer Perceptron (MLP) have been proposed in[6,17] which exhibit an effective performance in time series domain. Several works have leveraged Graph Neural Networks (GNN) for time series forecasting, specifically for traffic forecasting. For example, MTGNN[18] utilizes temporal and graph convolutional layers to capture temporal and spatial (cross-channel) dependencies. An attention based spatial-temporal graph neural network is proposed in Ref.[19] which captures the temporal dynamics of traffic series by using the local context. In Ref.[20], a feature correlation-aware spatio-temporal graph convolutional network is designated for traffic prediction, which captures multi-scale spatial and temporal relations effectively, considering cross-scales dependencies. Dynamic graph structure learning for multivariate time series forecasting[21] exploits graph learning networks to capture hidden dependencies between variables, enhancing the accuracy of forecasting by effectively capturing complex interrelationships within the data. Another work[22] addresses the problem of capturing dynamic correlations by learning historical relation graphs and predicting future relation graphs. They also design a causal GNN for feature extraction and reasoning network to capture the relations between historical time steps and forecasting horizon. Recent studies in time series analysis have also focused on reducing the computational and memory requirements related to processing large datasets. For instance, Ref.[23] introduced TimeDC, a time series dataset condensation framework to preserve complex temporal relations while significantly reducing dataset size.

With the outstanding breakthrough in Natural Language Processing (NLP) and Computer Vision (CV) fields, Transformer models have recently shown superior performance in time series forecasting task and they have been continuously evolving. Among them, Informer[5] develops a Transformer model based on prob-sparse self-attention to select important keys and reduce time complexity of self-attention. In Autoformer[24],

the self-attention is replaced with auto-correlation to capture temporal dynamics. FEDformer[4] utilizes Fourier transformer to deal with time series data given the fact that time series tend to have a sparse representation in Fourier basis. Recently, some linear models have been developed which outperform Transformer models in time series domain[6] and raised the concern about the efficiency of Transformer for time series forecasting. However, the PatchTST model[7] proved the efficiency of Transformer models in time series analysis, which exploits patching and channel independence to model time series data, pinpointing that Transformer architecture is still a powerful model with some adaptation and architectural adjustments. Following PatchTST, other Transformer models have been developed for time series and proved high capability in dealing with high-dimensional time series[10]. Multi-scale representation proved to be necessary for time series analysis[11,25]. Triformer[25] designs a patch attention with linear complexity and variable specific parameters to enhance accuracy. Reference[11] develops a multi-scale framework to model time series using different resolutions and they utilize separate predictive models for each temporal scale which leads to high computational complexity. After the rise of generative pre-trained models, Large Language Models (LLMs) have been utilized for time series forecasting by fine-tuning and exploiting prompt engineering, e.g., Time-LLM[26] which keeps the backbone LLM intact and input time series is reprogrammed with text prototypes before feeding it to the frozen LLM, aligning two modalities. GPT4TS[27] employs GPT-2[28] model for time series forecasting by feeding the time series patches to the model in a Channel-Independent manner. A spatial-temporal large language model is proposed for traffic forecasting[29] by defining spatial-temporal embedding to learn the spatial locations and global temporal dependencies of time steps at each location. In traffic prediction often capturing spatial-temporal dependencies at multiple scales is required. To address the mentioned requirement, MT-STNets is designed in[30], for prediction of both fine-grained traffic conditions on individual roads and coarse-grained traffic flows across urban areas. More recently, Pathformer[8] is proposed which exploits adaptive pathways to capture multi-scale temporal relations in an adaptive manner by automatically selecting patches of different resolutions, which uses separate set of parameters for each temporal granularity in its design. Different from the above mentioned models, which either disregard multi-scale modeling entirely or represent multi-scale information inside their model architecture, we devise a way to project multiple-scale temporal representations to the model dimension in the time series embedding phase and utilize the same model parameters to capture multi-scale and cross-scale information efficiently.

We expect improvements by using multi-scale embedding, since real-world time series often exhibit multiple seasonal patterns and modeling all the scales would improve the model performance. We employed Fourier analysis to calculate the high frequency components and dominant periods of time series samples from various benchmark datasets in order to show that real-world datasets usually rely on more than one seasonality pattern (scale). For example, electricity dataset typically exhibits daily (e.g., 24-h or 48-h) seasonality, together with long-term periods, such as monthly consumption or seasonal peaks (with periods of 90, 102). Multi-scale embedding and sharing the same model space among different scales is the main difference between our proposed method and the existing literature which exploits the interactions among different scales. We also use channel dimensionality reduction through 1-dimensional convolution over the channel dimension in the cross-channel encoder layer, which helps to reduce the noise effect in the highly correlated time series by mitigating the over-fitting chance. The previous methods usually utilize a single linear layer to map the model dimension to the prediction window, which would lead to over-fitting in longer horizon forecasting scenarios. Therefore, we design a simple but effective way to avoid this effect, by decoding the extracted information through linear layers over consecutive steps.

Methods

In this paper, we deal with the problem of time series forecasting which can be stated as following: Given a set of historical time series of C variates over L timestamps $(X_1, X_2, ..., X_L)$, with X_i indicating the observation at timestamp i of dimension C, we aim to predict F steps of the future time steps: $(X_{L+1}, ..., X_{L+F})$. To effectively capture multi-scale information and inter-series correlations, we propose MultiPatchFormer, which utilizes a novel predictor composed of a sequence of linear layers to simulate auto-regressive decoding in patch level. The overall architecture is illustrated in Fig. 2. The whole framework consists of instance normalization[31], multi-scale embedding, temporal encoder, inter-series encoder module and predictor. Instance normalization[31] is a technique employed by many models to avoid distribution shifts between training and test sets. The multi-scale embedding first divides the input series into patches of various sizes and then embeds all of them to the same model space. In other words, given 4 temporal scales, time series is reshaped to $((B, C), L, 1)$, then is convolved with filters of different sizes and strides, corresponding to the patch sizes and strides (overlapping region between to consecutive patches). B, C and L indicate batch size, channels and series length, respectively. We deliberately decided to exploit one-dimensional convolution to split series for the purpose of capturing local temporal dynamics inside a patch (intra-patch relations). After splitting the input time series by using patches of four various lengths, they are merged together into the model dimension (d_{model}). To be more specific, the output channel of the individual convolutions are set to $\frac{d_{model}}{4}$, mapping each patched series into its own smaller sub-space and then merging them to the main model dimension. This strategy employs separate 1-dimensional convolutions with input channel of 1 and output channel of $\frac{d_{model}}{4}$ to project multi-scale information into the embedded space and each channel of time series is embedded independently in this manner, as we reshape the series into $((B, C), L, 1)$ before applying the convolutions. In order to reduce the over-fitting effect in channel-wise multi-head self-attention, we devise a simple but yet effective solution for reducing noise and time complexity of time series with large number of channels. The regular linear predictor which is usually used as the final layer of time-series forecasting models, is susceptible to over-fitting, specifically when the forecast horizon is relatively long. Rather than using a simple linear layer to map from model dimension to the prediction length, we break the final linear layer to multiple linear layers to decode the predictions in multiple steps by generating part of

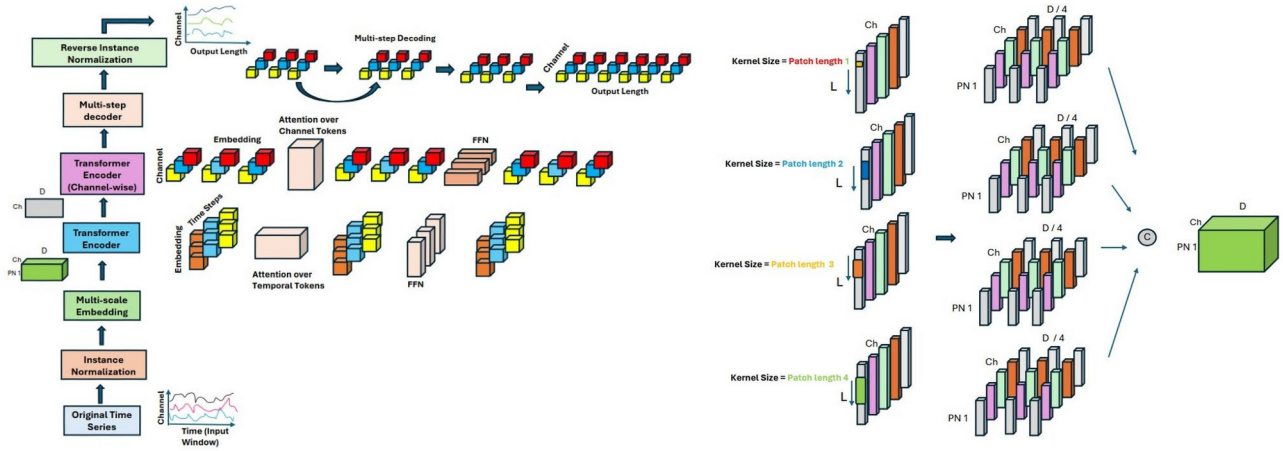

Fig. 2. General overview of the proposed framework (MultiPatchFormer). Left: the main steps include normalization, multi-scale embedding using different patch sizes, temporal encoder, reshaping the output of the temporal encoder using a 1-dimensional convolution layer and sending the result to the channel-wise encoder. The output of the channel-wise encoder is passed through the multi-step decoder to generate the final predictions followed by the de-normalization step. Right: Multi-scale embedding is illustrated using 4 different patch sizes (scales), where *D* refers to the model dimension (d_{model}), *Ch* indicates time series channels (variates), and *PN*1 refers to the number of patches fixed across four scales by choosing appropriate strides.

the prediction horizon at each step. In the following section, each component of our model is described in more detail.

Multi-scale embedding

In Computer Vision, prior to the main backbone, RGB channels are embedded into a D-dimensional vector at every pixel. This embedding process serves to mix information from channels through the embedding layer. However, employing a similar variable-mixing embedding, such as embedding *M* variables into a D-dimensional vector per time step, proved to be unsuitable for time series data because of two primary reasons. Firstly, the difference between variables within a time series is far greater than that among RGB channels within an image[25]. A mere embedding layer is insufficient for capturing the intricate correlations among variables, which might lead to the loss of their individual behavior. We refer X_{in} as the *C* variables input time series of length *L* which is further divided into *PN* patches of patch length *PL* after proper padding (padding by repeating the last time element stride times). The stride *S* is specified as the length of non overlapping region between two consecutive patches. Then, the patches will be embedded into embedding vectors of dimension $\frac{d_{model}}{num_{scales}}$:

$$X_{emb} = Embed_s(X_{in}) \tag{1}$$

In which, $X_{emb} \in C \times PN \times \frac{d_{model}}{num_{scales}}$ is the input embedding for one temporal scale and $Embed_s$ denotes embedding with scale or patch length of *s*. As we mentioned earlier we conduct this patchify embedding in a fully-convolution way to simplify computation and capture local dynamics in various scales. First, we extend (unsqueeze) the shape to $X_{emb} \in C \times L \times 1$, and then we feed the padded X_{embed} into a 1-dimensional convolution layer with kernel size *PL* and stride *S*, which maps the input with one channel into $\frac{d_{model}}{num_{scales}}$ output channels. It should be mentioned that in above process, each of the *C* univariate time series is embedded independently which keeps the variate dimension intact and helps to model inter-variable dependencies in the following blocks. We repeat the above embedding process for num_{scales} times, each time with different patch size and stride to capture and embed different scales of the input series. We utilize appropriate stride sizes with each patch length to obtain the same number of patches (patch numbers), because the resultant embeddings are concatenated as the final representation. Therefore, The multi-scale embedding is the result of concatenating embeddings with various patch sizes which are combined along their model dimension:

$$X_{emb} = Concatenate(Embed_1(X_{in}), ..., Embed_{num_{scale}}(X_{in})) \tag{2}$$

In our model, the embeddings of multiple scales are projected into a shared feature space. For example, given 4 different patch sizes, we project the time series into the same feature space by dividing the model dimension into 4 segments and assigning the embedding result of each scale to one segment of the model dimension. In other words, each scale is first projected into $model_{dim}/4$ space and then these embeddings are concatenated along the model dimension. This allows for interaction between different temporal resolutions within the same representation space and capturing multi-scale and cross-scale information through the self-attention component.

Temporal encoder

We define a set of different patch sizes $(P_1, P_2, ..., P_M)$ to represent various views corresponding to temporal resolution of input series. Each divided patch is embedded to a lower dimension than the main model dimension (to $\frac{d_{model}}{4}$ in case of using 4 different scales), which are then concatenated to produce the final embedding ready to be fed into the temporal and channel-wise encoders, respectively. The embedded series is first passed through the temporal encoder, which consists of a multi-head self attention followed by layer normalization and feed-forward layers to capture long-term temporal dependencies across multiple scales, simultaneously. The multi-head self-attention in temporal encoder, extracts temporal dependencies over the patches. Consider a set of divided patches which are the result of multiple scale division: $(P_1, P_2, ..., P_{PN})$, with *PN* indicating number of patches fixed for all scales and $P_i \in R^{d_{model}}$. Then, we employ trainable linear transformations to obtain query, key and value matrices, $Q, K, V \in R^{PN \times d}$ and perform attention score calculation as:

$$AttentionScore = softmax\left(\frac{Q \cdot K^T}{\sqrt{d}}\right) \tag{3}$$

In the above formula, *d* indicates the head dimension $\frac{d_{model}}{h}$, with *h* showing the number of heads. The attention scores are multiplied by the value matrix to obtain the result of multi-head self-attention over divided patches which demonstrates temporal dependencies along the patches.

Channel-wise attention

The output of the temporal encoder is passed to the channel-wise encoder, which employs a cross-channel multi-head attention module to capture the correlations among the channels (variates). However, the shape of resultant tensor of temporal encoder is $(B \times C) \times PN \times d_{model}$, which is reshaped into $B \times C \times (PN \times d_{model})$. Then we exploit a 1-dimensional convolution to reduce the last dimension into d_{model} (model dimension), preparing it to be fed into the channel-wise encoder layer. Therefore, the input of the channel-wise multi-head attention is of shape $B \times C \times d_{model}$. When performing attention over channel dimension, the vanilla self-attention may suffer from high computation and over-fitting, specially when dealing with a high-dimensional dataset with many variates. To reduce the noise over-fitting issue and computational complexity, we devise a solution to summarize the key and value matrices in channel-wise attention step. In other words, query matrix remains intact of shape $B \times C \times d$ and we apply a 1-dimensional convolution along the channel dimension with the same kernel size and stride (with proper padding) to reduce the number of channels in key and value matrices, resulting in tensors of size $B \times r \times d$ with $r < C$ which indicates the summarized channels. We deploy kernel and stride size according to the dataset in out experiments, e.g. kernel size and stride of 21 with padding 10 on each side of the input series are utilized for Traffic dataset. Therefore, it leads to decrease in time complexity and noise effect when calculating attention scores over the channel dimension:

$$AttentionScore = softmax\left(\frac{Q_{channel} \cdot K^T_{channel}}{\sqrt{d}}\right) \tag{4}$$

In which $Q_{channel} \cdot K^T_{channel}$ requires $C \times d \times r$ multiplications rather than $C^2 \times d$. The resultant attention output is further passed through layer normalization and feed-forward layers to extract meaningful features considering the dependencies over the channel dimension.

Multi-step decoder

Both long-term temporal and channel-wise dependencies are captured through the employed encoders in previous steps. The final output which comes from the encoder over channel dimension is of dimension $B \times C \times d_{model}$. Typically, a linear layer is employed as the final decoder or predictor to map from model dimension to prediction length (forecast window). But, the number of parameters in the output linear layer increases when dealing with long prediction horizons which might lead to over-fitting and affecting the far-future predictions by noise. To mitigate this issue, we propose a multi-step decoding process in which a sequence of linear layers are applied to the encoded feature map from the channel-wise encoder and current prediction results. In other words, we first divide the prediction length into different parts (length of 4 or 8) and generate the prediction part by part, by applying the combination of the final feature map and previously predicted parts to a linear layer. The multi-step decoding process is illustrated in Fig. 3.

Results

Datasets and baselines

Datasets We conduct different experiments using 7 real world datasets including electricity Transformer Temperature[24], weather forecasting[24], traffic[24] and electricity consumption[24]. These datasets consists of the well-known ETT (ETTh1, ETTh2, ETTm1, ETTm2), Weather, Electricity and Traffic. More details about the datasets are reported in Table 1. **Baseline and metrics** 14 well-known time series forecasting models are selected as baselines, including Pathformer[8], PatchTST[7], DLinear[6], NLinear[6], Crossformer[32], Scaleformer[11], TIDE[17], FEDformer[4], Pyraformer[33], Autoformer[24], Time-LLM[26], GPT4TS[27], MTGNN[18] and SDGL[21]. To ensure fairness in our experiments, the input length is fixed for all models (L = 96) and prediction length consists of (F = 96, 192, 336, 720). Since the authors of Time-LLM used input length of 512 in their paper[26], we set the context window to 512 in LLM comparison experiments and trained our model on different datasets with input length of 512 and prediction length in {96, 192, 336, 512}. We compare our model with graph models based on input window of length 96. Two well-known error metrics in time series forecasting are selected: Mean Absolute Error (MAE)

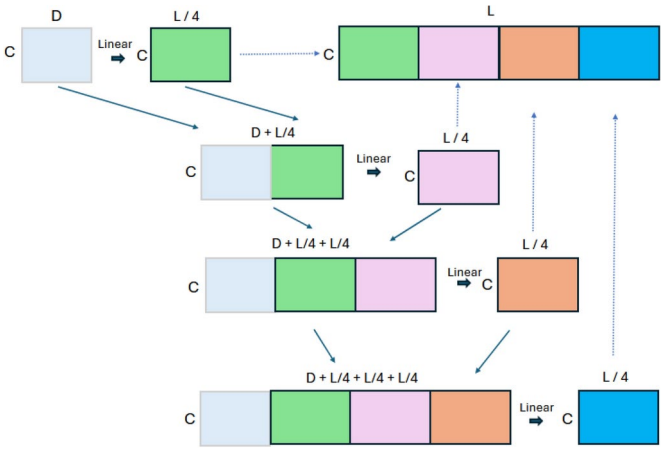

Fig. 3. Multi-step decoder is proposed to reduce noise effect in long prediction length. *C*, *D* and *L* refer to channels, model dimension and prediction length, respectively. At each step the feature map from the channel-wise encoder is concatenated with previously generated prediction segments and passed through a linear layer to generate the next prediction segment (part). The final prediction is the concatenation result of the individual predicted segments.

Dataset	Dim	Pred length	Dataset size	Mean corr	Max corr	Trend	Freq	Field
Electricity	321	96, 192, 336, 720	(18,317, 2633, 5261)	0.981	0.999	0.315	Hourly	Electricity
ETTh1, ETTh2	7	96, 192, 336, 720	(8545, 2881, 2881)	0.551 (0.551)	0.897 (0.946)	0.797 (0.761)	Hourly	Electricity
ETTm1, ETTm2	7	96, 192, 336, 720	(34,465, 11,521, 11,521)	0.551 (0.551)	0.897 (0.945)	0.896 (0.822)	15 min	Electricity
Weather	21	96, 192, 336, 720	(36,792, 5271, 10,540)	0.704	0.999	0.039	10 min	Weather
Traffic	862	96, 192, 336, 720	(12,185, 1757, 3509)	0.751	0.985	0.031	Hourly	Transportation

Table 1. Detailed dataset descriptions. Dim indicates the variate number of each dataset. Dataset Size denotes the total number of time points in (Train, Validation, Test) splits, respectively. Mean and maximum cross-correlations (Dynamic Time Warping based) between the channels of each dataset are also reported in Mean Corr and Max Corr columns. Trend column indicates strength of trend for each dataset.

and Mean Square Error (MSE) to compare the models. **Implementation details** We utilize Adam optimizer with learning rate in $\{10^{-3}, 10^{-4}\}$ and L1 loss function. The models are trained for 10 epochs with early stopping (patience of 3) based on the validation loss. We implemented our model using Pytorch framework[34] and the experiments are executed on an NVIDIA A100 40GB GPU. MultiPatchFormer utilizes 4 different scales to patchify and embed input time series (common scales in the time series forecasting domain, e.g., 8, 16, 24, 48 with proper strides, 8, 8, 7, 6 to create the same number of patches).

Main results

The main results are summarized in Tables 2, 3 and 4. The results related to the baseline models are taken from Ref.[8] and we follow the same settings to train and evaluate our model. All experiments are repeated 5 times and the average results are reported. According to Table 2, our MultiPatchFormer, shows better performance across four different prediction horizons. Regarding error measures across four prediction lengths, our model achieves the best or second best performance among the state-of-the-art models in 82% of cases in MSE metric and 86% of the conducted experiments in terms of MAE measure. Our MultiPatchFormer outperforms the baseline models on benchmarks with a high number of variates and complex structure. For instance, in Traffic dataset (862 covariates), MultiPatchFormer persistently outperforms the second best baseline by more than 5% on average MSE and 7.7% on average MAE across four prediction windows, while consuming less training time and parameters. By exploiting 321 variates in ECL (Electricity) dataset, we achieved average error reduction of 3.7% compared to Pathformer[8] and 10.7% improvement over the PatchTST model. This highlights that MultiPatchFormer is capable of utilizing extensive covariate dependencies for high accuracy prediction. Remarkably, our model makes predictions with lower error in complex problems, including Electricity (ECL) and Traffic, which underscores the efficiency of the channel-wise attention and multi-scale embedding to capture the inter-series dependencies across various scales. It worth noting that while NLinear[6] achieves promising performance in some datasets, MultiPatchFormer significantly outperforms the MLP-based models (NLinear and DLinear) by over 26.8% error reduction on Traffic and more than 10% improvement on Electricity dataset. This indicates that Transformer models are still powerful in forecasting long-term time series tasks. In terms of cross-variate modeling, Crossformer[32], utilizes a variant of attention to capture those type of correlations, but our model gains impressive improvement of over 28% and 18% compared to the Crossformer model on

Models	Ours		Pathformer		PatchTST		DLinear		NLinear		Crossformer		Scaleformer		TIDE		FEDformer		Autoformer	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1																				
96	**0.315**	**0.342**	*0.316*	*0.346*	0.324	0.361	0.392	0.405	0.386	0.392	0.429	0.440	0.396	0.440	0.356	0.381	0.379	0.419	0.505	0.475
192	0.367	**0.369**	*0.366*	*0.370*	**0.362**	0.383	0.441	0.436	0.440	0.430	0.494	0.482	0.434	0.460	0.391	0.399	0.426	0.441	0.553	0.496
336	0.399	**0.394**	**0.386**	**0.394**	*0.390*	*0.402*	0.501	0.478	0.480	0.443	0.706	0.625	0.462	0.476	0.424	0.423	0.445	0.459	0.621	0.537
720	0.464	**0.432**	**0.460**	**0.432**	*0.461*	*0.438*	0.538	0.526	0.486	0.472	0.750	0.689	0.494	0.500	0.480	0.465	0.543	0.490	0.671	0.561
Avg	0.386	**0.384**	**0.383**	*0.386*	*0.384*	0.396	0.468	0.461	0.448	0.434	0.595	0.559	0.447	0.469	0.413	0.417	0.448	0.452	0.588	0.517
ETTm2																				
96	*0.171*	*0.250*	**0.170**	**0.248**	0.177	0.260	0.186	0.279	0.177	0.257	0.197	0.321	0.182	0.275	0.182	0.264	0.203	0.287	0.255	0.339
192	**0.236**	**0.294**	*0.238*	*0.295*	0.248	0.306	0.266	0.339	0.241	0.297	0.326	0.375	0.251	0.318	0.256	0.323	0.269	0.328	0.281	0.340
336	*0.303*	*0.337*	**0.293**	**0.331**	0.304	0.342	0.332	0.376	0.302	0.337	0.372	0.421	0.340	0.375	0.313	0.354	0.325	0.366	0.339	0.372
720	*0.397*	0.393	**0.390**	**0.389**	0.403	0.397	0.462	0.455	0.405	*0.396*	0.410	0.448	0.435	0.433	0.419	0.410	0.421	0.415	0.433	0.432
Avg	*0.277*	*0.319*	**0.273**	**0.316**	0.283	0.326	0.312	0.362	0.281	0.322	0.326	0.391	0.302	0.350	0.293	0.338	0.305	0.349	0.327	0.371
ETTh1																				
96	*0.378*	**0.389**	0.382	0.400	0.394	0.408	0.392	0.405	0.386	0.392	0.429	0.440	0.396	0.440	0.427	0.450	**0.376**	0.419	0.449	0.459
192	**0.430**	**0.420**	0.440	0.427	0.446	0.438	0.441	0.436	0.440	0.430	0.494	0.482	0.434	0.460	0.472	0.486	0.420	0.448	0.500	0.482
336	*0.473*	0.442	**0.454**	**0.432**	0.485	0.455	0.501	0.478	0.480	*0.443*	0.706	0.689	0.462	0.476	0.527	0.527	0.459	0.465	0.521	0.496
720	**0.475**	*0.466*	*0.479*	**0.461**	0.495	0.474	0.538	0.526	0.486	0.472	0.750	0.689	0.494	0.500	0.644	0.605	0.506	0.507	0.514	0.512
Avg	**0.439**	**0.429**	**0.439**	*0.430*	0.455	0.444	0.468	0.461	0.448	0.434	0.595	0.575	*0.447*	0.469	0.518	0.517	0.440	0.460	0.496	0.487
ETTh2																				
96	*0.287*	*0.333*	**0.279**	**0.331**	0.294	0.343	0.331	0.381	0.290	0.339	0.632	0.547	0.364	0.407	0.304	0.359	0.346	0.388	0.358	0.397
192	*0.366*	*0.383*	**0.349**	**0.380**	0.378	0.394	0.432	0.435	0.379	0.395	0.876	0.663	0.466	0.458	0.394	0.422	0.429	0.439	0.456	0.452
336	0.413	0.420	**0.348**	**0.382**	*0.382*	*0.410*	0.441	0.451	0.421	0.431	0.924	0.702	0.479	0.476	0.385	0.421	0.496	0.487	0.482	0.486
720	0.417	0.435	**0.398**	**0.424**	*0.412*	*0.433*	0.564	0.578	0.436	0.453	1.390	0.863	0.487	0.492	0.463	0.475	0.463	0.474	0.515	0.511
Avg	0.372	*0.394*	**0.344**	**0.379**	*0.367*	0.395	0.442	0.461	0.382	0.405	0.956	0.694	0.449	0.458	0.387	0.419	0.434	0.447	0.453	0.462
Weather																				
96	*0.157*	*0.197*	**0.156**	**0.192**	0.177	0.218	0.195	0.253	0.168	0.208	0.181	0.231	0.288	0.365	0.202	0.261	0.238	0.314	0.249	0.329
192	*0.207*	*0.242*	**0.206**	**0.240**	0.224	0.258	0.239	0.299	0.217	0.255	0.219	0.275	0.368	0.425	0.242	0.298	0.275	0.329	0.325	0.370
336	*0.267*	*0.286*	**0.254**	**0.282**	0.277	0.297	0.282	0.333	0.267	0.292	0.274	0.332	0.447	0.469	0.287	0.335	0.339	0.377	0.351	0.391
720	*0.345*	*0.339*	**0.340**	**0.336**	0.350	0.345	0.352	0.390	0.351	0.346	0.356	0.387	0.640	0.574	0.351	0.386	0.389	0.409	0.415	0.426
Avg	*0.244*	*0.266*	**0.239**	**0.263**	0.257	0.280	0.267	0.319	0.251	0.275	0.258	0.306	0.436	0.458	0.271	0.320	0.310	0.357	0.335	0.379
Electricity																				
96	*0.146*	**0.233**	**0.145**	*0.236*	0.180	0.264	0.194	0.276	0.185	0.266	0.254	0.347	0.182	0.297	0.194	0.277	0.186	0.302	0.196	0.313
192	**0.163**	**0.247**	*0.167*	*0.256*	0.188	0.275	0.193	0.279	0.189	0.276	0.261	0.353	0.188	0.300	0.193	0.280	0.197	0.311	0.211	0.324
336	**0.178**	**0.263**	*0.186*	*0.275*	0.206	0.291	0.206	0.294	0.204	0.289	0.273	0.364	0.210	0.324	0.206	0.296	0.213	0.328	0.214	0.327
720	**0.213**	**0.293**	*0.231*	*0.309*	0.247	0.328	0.241	0.328	0.245	0.319	0.303	0.388	0.232	0.339	0.242	0.328	0.233	0.344	0.236	0.342
Avg	**0.175**	**0.259**	*0.182*	*0.269*	0.205	0.290	0.209	0.294	0.206	0.288	0.273	0.363	0.203	0.315	0.209	0.295	0.207	0.321	0.214	0.327
Traffic																				
96	**0.438**	**0.260**	*0.479*	*0.283*	0.492	0.324	0.648	0.396	0.645	0.388	0.558	0.320	2.678	1.071	0.568	0.352	0.576	0.359	0.597	0.371
192	**0.456**	**0.268**	*0.484*	*0.292*	0.487	0.303	0.613	0.386	0.599	0.365	0.572	0.331	0.564	0.351	0.612	0.371	0.610	0.380	0.607	0.382
336	**0.475**	**0.276**	*0.503*	*0.299*	0.505	0.317	0.614	0.383	0.606	0.367	0.587	0.342	0.570	0.349	0.605	0.374	0.608	0.375	0.623	0.387
512	**0.514**	**0.295**	*0.537*	*0.322*	0.542	0.337	0.655	0.405	0.645	0.388	0.652	0.359	0.576	0.349	0.647	0.410	0.621	0.375	0.639	0.395
Avg	**0.470**	**0.275**	*0.501*	*0.299*	0.507	0.320	0.632	0.393	0.624	0.377	0.592	0.338	1.097	0.530	0.608	0.377	0.604	0.372	0.617	0.384

Table 2. Multivariate time series forecasting results. The input length (lookback window) is fixed to 96 and prediction length is in {96, 192, 336, 720}. Bold and italics indicate the significant andsecond best values.

the Electricity and Traffic datasets, respectively. According to Table 3, our model consistently outperforms or achieves similar performance to LLM-based methods while using significantly fewer parameters. Notably, compared to Time-LLM, which utilizes 7 billion parameters, our model maintains competitive performance across different datasets. For example, when time series forecasting of the ETTh1 dataset at a prediction window of 720, MultiPatchFormer obtains an MSE of 0.434, lower than Time-LLM's 0.442 while consuming far fewer computational resources. This underscores the effectiveness of our approach relative to larger models. Graph learning models have been proved to be promising in forecasting traffic flow by modeling the temporal correlations and the spatial dependencies between the variables through the graph learning strategy[18,21]. We compare our MultiPatchFormer with the spatio-temporal graph models on various benchmarks, specially on Traffic dataset. As illustrated by Table 4, our model outperforms SDGL and MTGNN with a large margin, particularly on Traffic forecasting task, with average MSE improvement of 23% and 28%, respectively.

Methods	Ours		Time-LLM		GPT4TS	
Metric	MSE	MAE	MSE	MAE	MSE	MAE
Traffic						
96	0.370	**0.236**	**0.362**	0.248	0.388	0.282
192	0.383	**0.242**	**0.374**	0.247	0.407	0.290
336	0.398	**0.250**	**0.385**	0.271	0.412	0.294
720	0.433	**0.270**	**0.430**	0.288	0.450	0.312
Avg	0.396	**0.250**	**0.388**	0.264	0.414	0.294
Electricity						
96	**0.131**	**0.222**	0.131	0.224	0.139	0.238
192	**0.149**	**0.239**	0.152	0.241	0.153	0.251
336	0.162	0.253	**0.160**	**0.248**	0.169	0.266
720	**0.191**	**0.277**	0.192	0.298	0.206	0.297
Avg	**0.158**	**0.248**	**0.158**	0.252	0.167	0.263
ETTh1						
96	0.366	**0.392**	**0.362**	**0.392**	0.376	0.397
192	0.400	**0.418**	**0.398**	**0.418**	0.416	0.418
336	**0.423**	0.437	0.430	**0.427**	0.442	0.433
720	**0.434**	0.459	0.442	**0.457**	0.477	0.456
Avg	**0.406**	0.427	0.408	**0.423**	0.465	0.455
Weather						
96	**0.144**	**0.185**	**0.147**	0.201	0.162	0.212
192	0.190	**0.231**	**0.189**	0.234	0.204	0.248
336	**0.242**	**0.270**	**0.262**	0.279	0.254	0.286
720	0.313	0.323	**0.304**	**0.316**	0.326	0.337
Avg	**0.222**	**0.252**	0.225	0.257	0.237	0.270

Table 3. Comparison result of our model with LLM models. The input length (lookback window) is fixed to 512 and prediction length is in {96, 192, 336, 720}. Significant values are in bold.

As shown in Table 2, MultiPatchFormer consistently achieves low error rates for datasets with strong seasonality and highly correlated datasets with many variates (Electricity, Traffic, Weather). This can be attributed to the multi-scale embedding and temporal Transformer blocks, which effectively captures temporal correlations at various granularities and channel-wise attention, which represents the complex dependencies between time series channels through learning of the attention scores between channel pairs, extracting meaningful features. However, our model exhibits relatively lower performance on datasets with dominating short-term dependencies and limited training data, such as the ETTh2 dataset. This could be associated with the multi-scale embedding and multi-head attention, which, while being effective for capturing long-range patterns, may not show significant effectiveness for data with simpler temporal patterns and limited training samples.

In time series forecasting, size of the input series determines the historical information that a model receives to forecast. Different input lengths are configured to verify effectiveness of MultiPatchFormer to deal with various input lengths. As the input length increases, the error measures of MultiPatchFormer continue to decrease, demonstrating its robustness in handling long and noisy sequences evidenced by Table 5, which indicates an MSE reduction of more than 15%, 11% and 10% achieved on the ETTm1, Weather and Electricity datasets when prolonging the input length from 96 to 720. In this experiment, the best performing predictive models are selected and the results for input length of 48, 192 and 336 are visualized in Fig. 4. According to this figure, MultiPatchFormer consistently outperforms other baselines in most cases by using longer input lengths. This serves as evidence of efficiency of MultiPatchFormer in utilizing information of long input sequences thanks to multi-scale analysis and capturing cross-variable dependencies. MultiPatchFormer is also compared with other best performing baselines using a shorter input length (e.g. 48) on Electricity dataset. As Fig. 4 illustrates, our model forecasts different prediction lengths by using a short input window (H = 48) with the lowest error, indicating its capability in capturing temporal correlations from shorter input sequences.

Ablation studies

To verify the effectiveness of the components in our design of MultiPatchFormer, we conduct ablation studies by removing the multi-scale embedding, channel-wise encoder and multi-step decoder from the main model. Time series forecasting results using our model without those components are reported in Table 6. As evidenced by the table, each of the multi-scale embedding, channel-wise encoder and multi-step decoder modules contribute to performance promotion. For example, in ETTh1 forecasting dataset, multi-scale embedding improves the MSE error rate by approximately 2% in prediction length of 720 and the channel-wise encoder promotes the prediction accuracy (MSE) by 2.5%. Our multi-step decoder, improves the prediction error in most cases, specifically when the forecast horizon is long, e.g. 720. For example, in traffic forecasting, consisting of 862

Methods	Ours		SDGL		MTGNN	
Metric	MSE	MAE	MSE	MAE	MSE	MAE
Traffic						
96	**0.438**	**0.260**	0.557	0.271	0.660	0.437
192	**0.456**	**0.268**	0.581	0.287	0.649	0.438
336	**0.475**	**0.276**	0.611	0.302	0.653	0.472
720	**0.514**	**0.295**	0.702	0.345	0.639	0.437
Avg	**0.470**	**0.275**	0.613	0.301	0.650	0.446
Electricity						
96	0.146	**0.233**	**0.145**	0.238	0.217	0.318
192	0.163	**0.247**	**0.159**	0.255	0.238	0.352
336	**0.178**	**0.263**	0.185	0.284	0.260	0.348
720	**0.213**	**0.293**	0.232	0.314	0.290	0.369
Avg	**0.175**	**0.259**	0.180	0.273	0.251	0.347
ETTh1						
96	**0.378**	**0.389**	0.446	0.428	0.515	0.517
192	**0.430**	**0.420**	0.471	0.452	0.553	0.522
336	**0.473**	**0.442**	0.506	0.475	0.612	0.577
720	**0.475**	**0.466**	0.623	0.548	0.609	0.597
Avg	**0.439**	**0.429**	0.512	0.476	0.572	0.553
Weather						
96	0.157	0.197	**0.153**	**0.194**	0.230	0.329
192	**0.207**	**0.242**	0.212	0.257	0.263	0.322
336	0.267	**0.286**	**0.263**	0.294	0.354	0.396
720	**0.345**	**0.339**	**0.352**	0.360	0.409	0.371
Avg	**0.244**	**0.266**	0.245	0.276	0.314	0.355

Table 4. Comparison result of our model with graph models. The input length (lookback window) is set to 96 and prediction length is in {96, 192, 336, 720}. Significant values are in bold.

Input length	96		192		336		512		720	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	0.396	0.389	0.365	0.379	0.352	0.372	**0.350**	**0.372**	0.353	0.376
ETTm2	0.279	0.321	0.263	0.315	0.253	**0.307**	**0.252**	0.309	0.256	0.313
Weather	0.249	0.270	0.232	0.259	0.224	0.254	**0.222**	**0.254**	0.224	0.257
ECL	0.175	0.259	0.163	0.250	0.161	0.249	0.159	0.249	**0.158**	**0.248**

Table 5. Multivariate time series forecasting results of MultiPatchFormer (ours) with different input length {96, 192, 336, 512, 720} averaged over four prediction lengths. Significant values are in bold.

variables across 720 future timestamps, the utilization of a multi-step decoder yields an MAE error reduction of 1%. We utilize a different kernel size for each dataset in the channel summarization part of the channel-wise attention in order to project the key and values, depending on the performance improvement. In some cases, e.g., Electricity dataset, the kernel size is set to 1, since it gives the best results compared to the larger kernels. But, in datasets with highly correlated channels (such as Traffic with 862 variates), large kernels (e.g., 21) yield lower error rates by reducing channel dimension in key and value of the channel-wise attention. We study the impact of varying number of scales on time series forecasting and indicate the results in Table 7. In some cases, considering only one dominant scale is enough, but in some datasets, utilizing two or more than two scales helps to boost the performance and capture cross-scale information. In addition, more experiments are conducted in order to verify the effect of hyper-parameters, including the learning rate, hidden (model) dimension and number of training epoch. The results are illustrated in Fig. 5.

Model efficiency

Our model is versatile in terms of speed and memory usage efficiency. Specifically, MultiPatchFormer is comparable with other most recent models, including PatchTST[7] and Pathformer[8] in terms of training time and memory footprint. As illustrated in Fig. 6, our model is faster than its main competitor model (Pathformer) in terms of training time, and its memory footprint is lower than most of the baselines, being close to memory usage of the linear models (DLinear). In time-series forecasting of ETTh1 dataset, with input length of 96 and prediction length of 96, our model is more than ten times faster than Pathformer and FEDformer, demonstrating

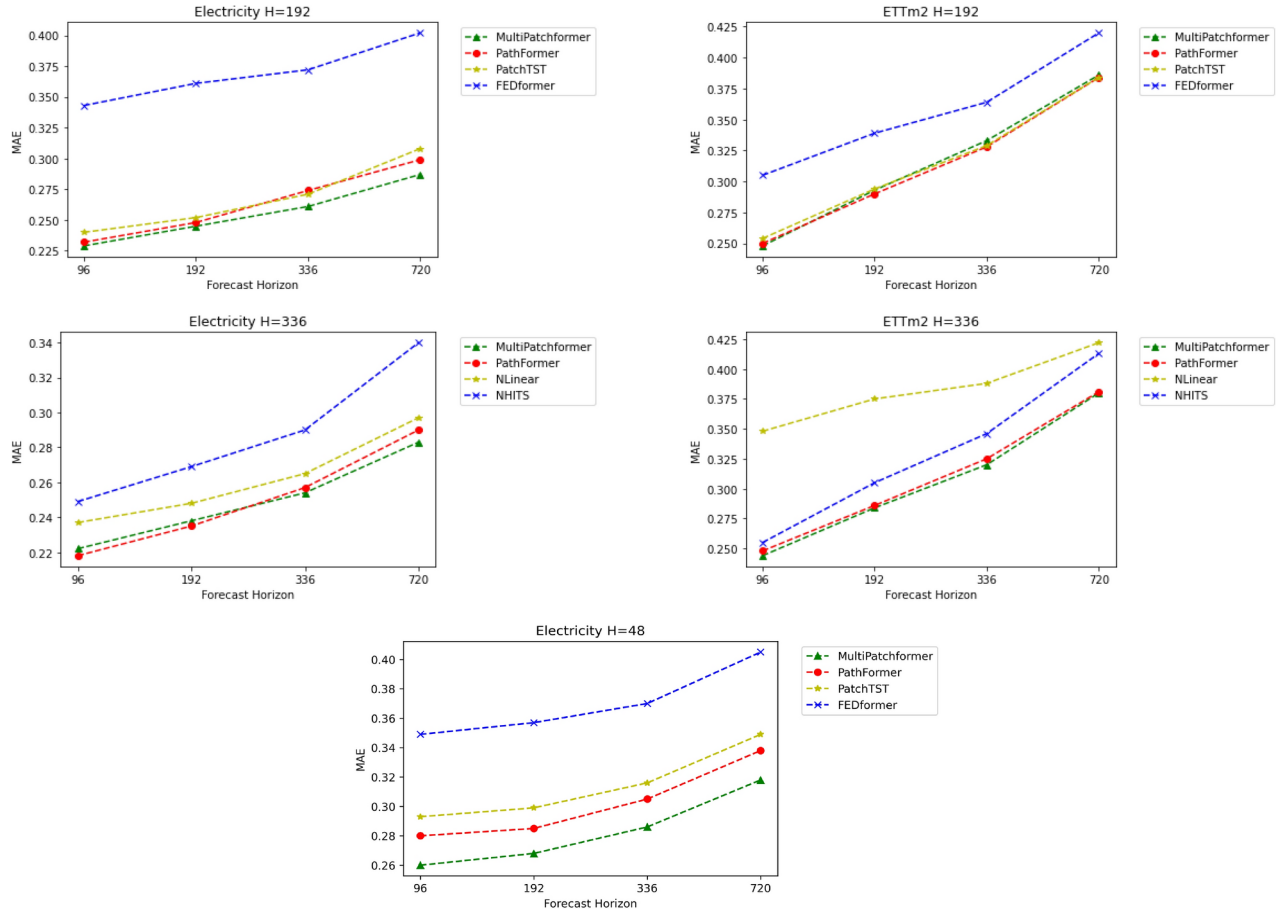

Fig. 4. Time series forecasting results of MultiPatchFormer with different input lengths (H = 48, 192 and 336) on ETTm2 and Electricity datasets.

	W/O multi-scale embedding		W/O channel-wise encoder		W/O multi-step decoder		MultiPatchFormer	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1								
96	0.383	0.394	0.387	**0.393**	0.382	0.395	**0.380**	**0.393**
192	0.435	**0.424**	0.443	0.425	0.435	0.427	**0.432**	**0.424**
336	0.475	**0.444**	0.485	0.445	0.474	0.449	**0.472**	**0.444**
720	0.478	0.470	0.481	**0.464**	0.473	0.471	**0.469**	**0.464**
Traffic								
96	**0.437**	0.264	0.490	0.311	0.439	**0.260**	0.439	**0.260**
192	0.460	0.271	0.493	0.306	**0.456**	**0.269**	0.457	**0.269**
336	0.483	0.280	0.503	0.301	**0.477**	0.278	0.478	**0.277**
720	0.516	0.300	0.537	0.319	0.516	0.299	**0.515**	**0.297**
Electricity								
96	0.147	0.235	0.167	0.248	**0.146**	0.234	**0.146**	**0.233**
192	0.164	0.248	0.181	0.259	**0.163**	**0.247**	**0.163**	**0.247**
336	0.179	0.265	0.196	0.275	**0.178**	0.264	**0.178**	**0.263**
720	0.216	0.295	0.235	0.308	0.216	0.295	**0.213**	**0.293**

Table 6. Ablation of MultiPatchFormer model without different components: multi-scale embedding, channel-wise attention and multi-step decoder. Significant values are in bold.

Number of scales	1		2		4	
Dataset	MAE	MSE	MAE	MSE	MAE	MSE
Electricity						
96	0.138	0.229	**0.128**	**0.218**	0.131	0.222
192	0.150	**0.239**	0.155	0.244	**0.149**	**0.239**
336	0.165	0.255	0.168	0.258	**0.162**	**0.253**
720	0.195	0.280	0.192	**0.276**	**0.191**	0.277
ETTm1						
96	**0.309**	**0.338**	0.310	0.339	0.313	0.343
192	**0.362**	**0.369**	0.368	0.370	0.365	**0.369**
336	0.398	**0.393**	**0.396**	**0.393**	**0.396**	**0.393**
720	0.472	0.434	**0.465**	**0.433**	0.470	0.434
ETTh1						
96	0.380	0.390	0.379	**0.389**	**0.378**	**0.389**
192	0.432	**0.420**	0.432	0.421	**0.430**	**0.420**
336	0.477	0.444	0.474	0.442	**0.473**	**0.441**
720	0.482	0.471	0.486	0.476	**0.476**	**0.468**

Table 7. Ablation study on the number of scales in multi-scale embedding. Significant values are in bold.

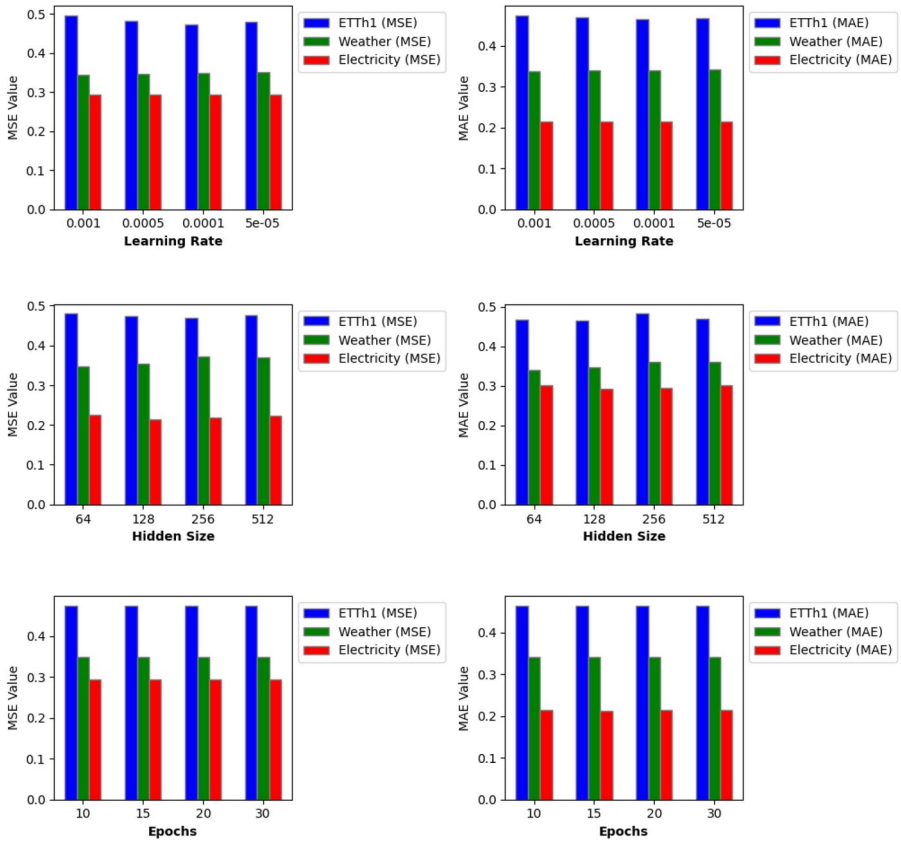

Fig. 5. Influence of hyperparameters on time series forecasting of MultiPatchFormer.

lower memory consumption, while delivering the similar or better prediction accuracy. This highlights MultiPatchFormer's efficiency while providing accurate forecasts.

Visualization

We visualize prediction results of MultiPatchFormer for Electricity and Traffic datasets. As illustrated in Figs. 7 and 8, the prediction curves align well with the ground truth ones in various cases and prediction horizons, indicating MultiPatchFormer's capability in handling complex trends and patterns.

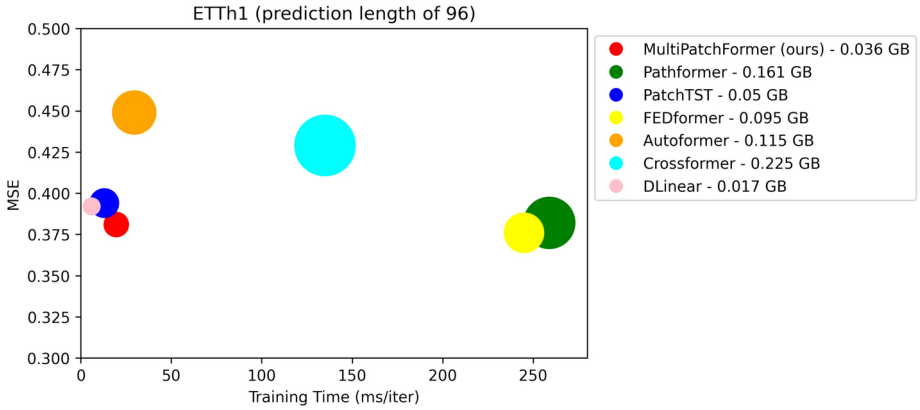

Fig. 6. Model efficiency comparison. Greater circle diameter represents larger memory footprint (in GB).

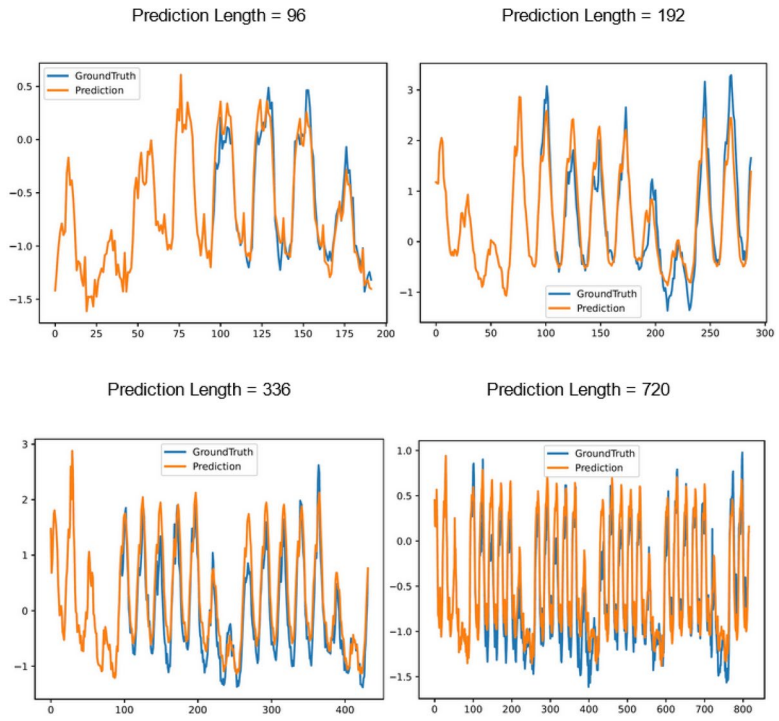

Fig. 7. Prediction results of our model applied to the Electricity dataset with different forecasting horizons.

Conclusion

We propose MultiPatchFormer, a Transformer based time series forecasting model, which integrates temporal dependencies associated with different temporal scales and captures intricate correlations among time series channels. In addition, we utilize 1-dimensional convolution to reduce channel dimension of key and value in the channel-wise multi-head attention to decrease noise effect and computational burden. A novel multi-step decoder is further devised to reduce over-fitting effect in dealing with long forecasting horizons. These innovative components collectively empower our model to produce accurate forecasts in variety of domains. Extensive experiments show the superior performance and robustness of our model in different settings and with varying input lengths.

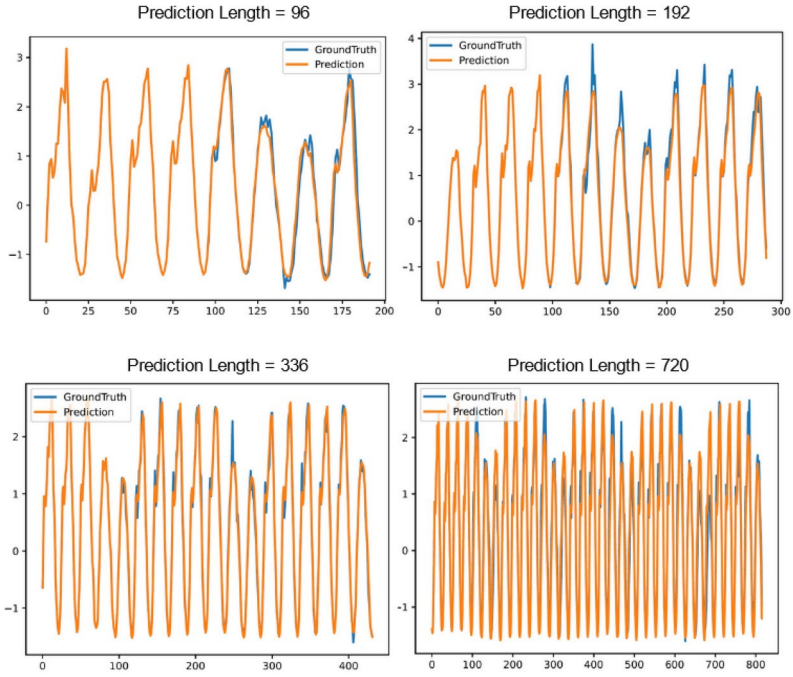

Fig. 8. Prediction results of our model applied to the Traffic dataset with different forecasting horizons.

Data availability

All the datasets used in this work are publicly available[24]. The link of the aforementioned repository to download the datasets is: https://drive.google.com/drive/folders/1ZOYpTUa82_jCcxIdTmyr0LXQfvaM9vIy.

Code availability

Accession codes The code is available at: https://anonymous.4open.science/r/MultiPatchFormer-DD43.

Received: 6 September 2024; Accepted: 5 December 2024
Published online: 10 January 2025

References

1. Brown, T. et al. Language models are few-shot learners. *Adv. Neural Inform. Process. Syst.* **33**, 1877–1901 (2020).
2. Dosovitskiy, A. An image is worth 16×16 words: Transformers for image recognition at scale. *arXiv preprint*. arXiv:2010.11929 (2020).
3. Vaswani, A. *et al.* Attention is all you need. *Adv. Neural Inform. Process. Syst*. **30** (2017).
4. Zhou, T. *et al.* Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. in *International conference on machine learning*, 27268–27286 (PMLR, 2022).
5. Zhou, H. et al. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proc. AAAI Conf. Artif. Intell.* **35**, 11106–11115 (2021).
6. Zeng, A., Chen, M., Zhang, L. & Xu, Q. Are transformers effective for time series forecasting?. *Proc. AAAI Conf. Artif. Intell.* **37**, 11121–11128 (2023).
7. Nie, Y., Nguyen, N. H., Sinthong, P. & Kalagnanam, J. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint*. arXiv:2211.14730 (2022).
8. Chen, P. *et al.* Pathformer: Multi-scale transformers with adaptive pathways for time series forecasting. *arXiv e-prints* arXiv-2402 (2024).
9. Ferreira, M. A., Higdon, D. M., Lee, H. K. & West, M. Multi-scale and hidden resolution time series models. *Bayesian Anal.*, 947–967 (2006).
10. Liu, Y. *et al.* itransformer: Inverted transformers are effective for time series forecasting. in *The Twelfth International Conference on Learning Representations* (2023).
11. Shabani, A., Abdi, A., Meng, L. & Sylvain, T. Scaleformer: Iterative multi-scale refining transformers for time series forecasting. *arXiv preprint*. arXiv:2206.04038 (2022).
12. Elsaraiti, M., Ali, G., Musbah, H., Merabet, A. & Little, T. Time series analysis of electricity consumption forecasting using arima model. in *2021 IEEE Green technologies conference (GreenTech)*, 259–262 (IEEE, 2021).
13. Hyndman, R. J. & Khandakar, Y. Automatic time series forecasting: The forecast package for r. *J. Stat. Softw.* **27**, 1–22 (2008).
14. Salinas, D., Flunkert, V., Gasthaus, J. & Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *Int. J. Forecasting* **36**, 1181–1191 (2020).
15. Wu, H. *et al.* Timesnet: Temporal 2d-variation modeling for general time series analysis. in *The Eleventh International Conference on Learning Representations*. (2022).
16. Liu, M. et al. Scinet: Time series modeling and forecasting with sample convolution and interaction. *Adv. Neural Inform. Process. Syst.* **35**, 5816–5828 (2022).
17. Das, A. *et al.* Long-term forecasting with tide: Time-series dense encoder. *arXiv preprint*. arXiv:2304.08424 (2023).
18. Wu, Z. *et al.* Connecting the dots: Multivariate time series forecasting with graph neural networks. in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 753–763 (2020).

19. Guo, S., Lin, Y., Wan, H., Li, X. & Cong, G. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. *IEEE Trans. Knowl. Data Eng.* **34**, 5415–5428 (2021).
20. Wang, S., Zhang, M., Miao, H., Peng, Z. & Yu, P. S. Multivariate correlation-aware spatio-temporal graph convolutional networks for multi-scale traffic prediction. *ACM Trans. Intell. Syst. Technol. (TIST)* **13**, 1–22 (2022).
21. Li, Z. L., Zhang, G. W., Yu, J. & Xu, L. Y. Dynamic graph structure learning for multivariate time series forecasting. *Pattern Recognit.* **138**, 109423 (2023).
22. Zhao, K. et al. Multiple time series forecasting with dynamic graph modeling. *Proc. VLDB Endowment* **17**, 753–765 (2023).
23. Miao, H. *et al.* Less is more: Efficient time series dataset condensation via two-fold modal matching–extended version. *arXiv preprint*. arXiv:2410.20905 (2024).
24. Wu, H., Xu, J., Wang, J. & Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Adv. Neural Inform. Process. Syst.* **34**, 22419–22430 (2021).
25. Cirstea, R.-G. *et al.* Triformer: Triangular, variable-specific attentions for long sequence multivariate time series forecasting–full version. *arXiv preprint*. arXiv:2204.13767 (2022).
26. Jin, M. *et al.* Time-llm: Time series forecasting by reprogramming large language models. *arXiv preprint*. arXiv:2310.01728 (2023).
27. Zhou, T. et al. One fits all: Power general time series analysis by pretrained lm. *Adv. Neural Inform. Process. Syst.* **36**, 43322–43355 (2023).
28. Radford, A. et al. Language models are unsupervised multitask learners. *OpenAI Blog* **1**, 9 (2019).
29. Liu, C. *et al.* Spatial-temporal large language model for traffic prediction. *arXiv preprint*. arXiv:2401.10134 (2024).
30. Wang, S., Zhang, M., Miao, H. & Yu, P. S. Mt-stnets: Multi-task spatial-temporal networks for multi-scale traffic prediction. in *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, 504–512 (SIAM, 2021).
31. Kim, T. *et al.* Reversible instance normalization for accurate time-series forecasting against distribution shift. in *International Conference on Learning Representations* (2021).
32. Zhang, Y. & Yan, J. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting. in *The Eleventh International Conference on Learning Representations* (2022).
33. Liu, S. *et al.* Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. in *International Conference on Learning Representations* (2021).
34. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inform. Process. Syst.* **32** (2019).

Author contributions

V.N. conceived the experiment(s), V.N. and A.D. conducted the experiment(s), M.B. and A.D. analysed the results. All authors reviewed the manuscript.

Declaration

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to A.B.D.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

© The Author(s) 2024