A decoder-only foundation model for time-series forecasting

Anonymous authors
Paper under double-blind review

Abstract

Motivated by recent advances in large language models for Natural Language Processing (NLP), we design a time-series foundation model for forecasting whose out-of-the-box zero-shot performance on a variety of public datasets comes close to the accuracy of state-of-the-art supervised forecasting models for each individual dataset. Our model is based on pretraining a patched-decoder style attention model on a large time-series corpus, and can work well across different forecasting history lengths, prediction lengths and temporal granularities.

1 Introduction

Time-series data is ubiquitous in various domains such as retail, finance, manufacturing, healthcare and natural sciences. In many of these domains, one of the most important use-cases of time-series data is forecasting. Time series forecasting is critical to several scientific and industrial applications, like retail supply chain optimization, energy and traffic prediction, and weather forecasting. In recent times, Deep learning models (Salinas et al., 2020; Borovykh et al., 2017) have emerged as a popular approach for forecasting rich, multivariate, time series data, often outperforming classical statistical approaches such as ARIMA or GARCH (Box & Jenkins, 1968). In several forecasting competitions such as the M5 competition (Makridakis et al., 2022) and IARAI Traffic4cast contest (Kopp et al., 2021), almost all the winning solutions are based on deep neural networks.

At the same time, we are witnessing a rapid progress in the Natural Language Processing (NLP) domain on large foundation models for downstream NLP tasks. Large language models (LLMs) are growing in popularity because they can be used to generate text, translate languages, write different kinds of creative content, and answer your questions in an informative way (Radford et al., 2019). They are trained on massive amounts of data, which allows them to learn the patterns of human language. This makes them very powerful tools that can be used for a variety of downstream tasks, often in a zero-shot learning mode.

This motivates the question: "Can large pretrained models trained on massive amounts of time-series data learn temporal patterns that can be useful for time-series forecasting on previously unseen datasets?" In particular, can we design a time series foundation model that obtains good zero-shot out-of-the-box forecasting performance on previously-unseen datasets? Such a time series foundation model, if possible, would bring significant benefits for downstream forecasting users in terms of significantly reduced training data and compute requirements. It is not immediately obvious that such a foundation model for time series forecasting is possible. Unlike in NLP, there is no well defined vocabulary or grammar for time-series. Additionally, the model would need to support forecasting with varying history lengths (context) , prediction lengths (horizon) and time granularities. Furthermore, unlike the huge volume of public text data for pretraining language models, vast amounts of time series data is not readily available. In spite of these issues, we provide evidence to answer the above question in the affirmative.

In particular, we design a single foundation model for time series forecasting that, when applied to a variety of previously-unseen forecasting datasets with different temporal granularities, obtains close to state-of-the-art zero-shot accuracy (compared to the best supervised models trained individually for these datasets). Our model can work well across different forecasting history lengths, prediction lengths and time granularities at inference time. The key elements of our foundation model are twofold: 1) a time series corpus built mostly

Author names are listed in alphabetical order.

using Google Trends[1], that meets the volume and diversity of data needed for training our foundation model, and 2) a patched-decoder style attention architecture that can be efficiently pre-trained on this time series corpus. Compared to the latest large language models, our time series foundation model is much smaller in both parameter size (225M parameters) and pretraining data size (1B timepoints); yet we show that even at such scales it is possible to build a practical foundation model for forecasting whose zero-shot performance comes close to the accuracy of fully-supervised approaches on a diverse set of time series data.

2 Related Work

In the last decade, deep learning models (Salinas et al., 2020; Borovykh et al., 2017) have emerged as powerful contenders in forecasting time-series in the presence of large training datasets and have been shown to outperform traditional statistical methods such as ARIMA, Exponential smoothing (McKenzie, 1984). Forecasting models can be categorized broadly into: (i) Local univariate models that include traditional methods like ARIMA, exponential smoothing (McKenzie, 1984) and non-autoregressive models like Prophet (Taylor & Letham, 2018). These models are trained individually for each time-series in a dataset in order to predict the corresponding time-series's future. (ii) Global univariate models like DeepAR (Salinas et al., 2020), Temporal Convolutions (Borovykh et al., 2017), N-BEATS (Oreshkin et al., 2019) and long-term forecasting models such as (Nie et al., 2022; Das et al., 2023) that are trained globally on many time-series but during inference they predict the future of a time-series as a function of its own past and other related covariates. (iii) Global multivariate models that take in the past of all time-series in the dataset to predict the future of all the time-series. Such models include the classical VAR model (Zivot & Wang, 2006) as well as deep learning models like (Sen et al., 2019; Zhou et al., 2022; 2021) to name a few.

All the works cited above have primarily been applied in the supervised setting with the notable exception of PatchTST (Nie et al., 2022) and N-BEATS (Oreshkin et al., 2019). PatchTST has a section on dataset to dataset transfer learning in the semi-supervised setting. The patching in our decoder-only model is inspired by (Nie et al., 2022). (Oreshkin et al., 2021) also show that the N-BEATS architecture lends itself to transfer learn between various source-target dataset pairs. However, none of these works aim to train a single foundation model that can work on a plethora of datasets. For a more in-depth discussion about transfer learning in time-series we refer the reader to the survey in (Ma et al., 2023).

Zhou et al. (2023) show how to use the the GPT-2 backbone (Radford et al., 2019) for various tasks including time-series forecasting. (Chang et al., 2023) is a follow up works along the same lines. Both the works have a section on zero-shot forecasting on a target dataset after having trained on a source dataset. For instance Table-18 (Zhou et al., 2023) shows M4 to M3 transfer. The rest of the two papers are mostly focused on fine-tuning and to the best of our knowledge they do not train a single foundation model that shows out of the box zero-shot performance on a variety of datasets. To the best of our knowledge, the very recent work in TimeGPT-1 (Garza & Mergenthaler-Canseco, 2023) is the only known parallel work on foundation model for time-series. However the model is currently not public access, and several model details and the benchmark dataset have not been revealed.

3 Problem Definition

The task at hand is to build a general purpose zero-shot forecaster that takes in the past C time-points of a time-series as context and predicts the future H time-points. Let the context be denoted by $\mathbf{y}_{1:L} := \{y_1, \cdots, y_L\}$ where we follow a numpy like notation for indices. Similarly the actual values in the horizon is denoted by $\mathbf{y}_{L+1:L+H}$. Note that since we are building a one-fits-all model we cannot have dataset specific dynamic or static covariates during training time. However, the datetime column is ubiquitous in all time-series data, so we can optionally have date derived features like day or the week, month of the year etc processed into a vector at each time-point t, denoted by $\mathbf{x}_t \in \mathbb{R}^r$. See Appendix A.1 for details. Such features could be available for forecasting in both the context and horizon, represented as $\mathbf{x}_{1:L+H}$. The task is then to learn a capable foundation model that can map any time-series context to horizon, given by

$$f : (\mathbf{y}_{1:L}, \boldsymbol{x}_{1:L+H}) \longrightarrow \hat{\mathbf{y}}_{L+1:L+H}. \tag{1}$$

[1]https://trends.google.com

The accuracy of the prediction will be measured by a metric that quantifies their closeness to the actual values. For instance, if the metric is Mean Squared Error (MSE), then the goodness of fit is measured by,

$$\text{MSE}(\mathbf{y}_{L+1:L+H}, \hat{\mathbf{y}}_{L+1:L+H}) = \frac{1}{H}\|\mathbf{y}_{L+1:L+H} - \hat{\mathbf{y}}_{L+1:L+H}\|_2^2. \tag{2}$$

4 Model Architecture

A foundation model for time-series forecasting should be able to adapt to variable context and horizon lengths, while having enough capacity to encode all patterns from a large pretraining datasets. Transformers have been shown to be able to adapt to different context lengths in NLP (Radford et al., 2019). Inspired by the success of patch based modeling in the recent long horizon forecasting work (Nie et al., 2022) we also chose to breakdown the time-series into patches during training. However, there are several key differences in our foundation model architecture, the primary one being that our model is trained in decoder-only mode (Liu et al., 2018). We will now describe the key parts of our architecture and training methodology illustrated in Figure 1.

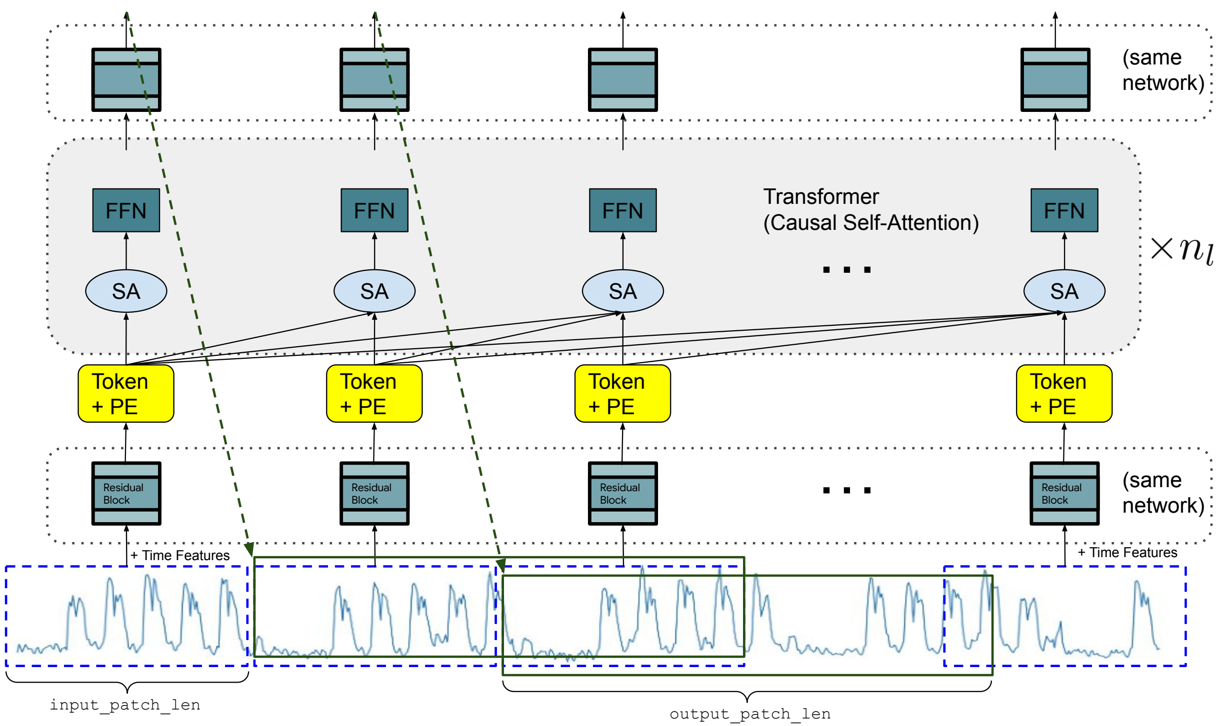

Figure 1: We provide an illustration of our model architecture during training where we show a input time-series of a certain length that can be broken down into input patches. Each patch along with (optional) time-features is processed into a vector by a residual block (as defined in the model definition) to the model dimension of the transformer layers. The vector is then added to positional encodings and fed into n_l stacked transformer layers. SA refers to self-attention (note that we use multi-head causal attention) and FFN is the fully connected layer in the transformer. The output tokens are then mapped through a residual block to an output of size `output_patch_len` which is the forecast for the time-period following the last input patch seen by the model so far.

Input Layers. The job of the input layers is to preprocess the time-series into input tokens to the transformer layers. We first break the input into contiguous non-overlapping patches. Then each patch (along with optional date derived features for that patch) is processed by a Residual Block into a vector of size `model_dim`. The Residual Block is essentially a Multi-layer Perceptron (MLP) block with one hidden layer with a skip connection as defined in (Das et al., 2023).

In other words, the inputs $\mathbf{y}_{1:L}, \boldsymbol{x}_{1:L}$ are broken down into patches of size `input_patch_len` (p). The j-th patch can be denoted as $\tilde{\mathbf{y}}_j = \mathbf{y}_{p(j-1)+1:pj}$ and $\tilde{\mathbf{x}}_j = \mathbf{x}_{p(j-1)+1:pj}$. Then the j-th input token to the subsequent transformer layers can be denoted as,

$$\mathbf{t}_j = \texttt{InputResidualBlock}(\tilde{\mathbf{y}}_j, \tilde{\mathbf{x}}_j) + \texttt{PE}_j \tag{3}$$

where $\texttt{PE}_j$ denotes the j-th positional encoding as defined in the original transformer paper (Vaswani et al., 2017). There will be $N = \lfloor L/p \rfloor$ such input tokens.

Stacked Transformer. The bulk of the parameters in our model are in n_l transformer layers stacked on top of each other. Each of these layers have the standard multi-head self-attention (MHA) followed by a feed-forward network (FFN). The main hyperparameters are `model_dim` which is equal to the dimension of the input tokens $\mathbf{t}_j$'s and number of heads (`num_heads`). We set the hidden size of the FFN's to be equal to `model_dim` as well. We use causal attention that is each output token can only attend to input tokens that come before it in the sequence (including the corresponding input token). This can be described by the equation

$$\mathbf{o}_j = \texttt{StackedTransformer}(\mathbf{t}_1, \cdots, \mathbf{t}_j), \quad \forall j \in [N]. \tag{4}$$

Output Layers. The remaining task is to map the output tokens into predictions. We train in decoder only mode i.e each output token should be able to be predictive of the part of the time-series that follows the last input patch corresponding to it. This is common for popular large language models like (Radford et al., 2019). However, one key difference in our time-series foundation model is that input patch length need not be equal to output patch length i.e we should be able to predict a larger chunk of the time-series based on the encoded information from the input patches seen so far. Let the output patch length be `output_patch_len` (h). We use another Residual Block to map the output tokens to the predictions. This can be described as,

$$\hat{\mathbf{y}}_{pj+1:pj+h} = \texttt{OutputResidualBlock}(\mathbf{o}_j). \tag{5}$$

Thus we encode all the data in $\mathbf{y}_{1:pj}$ into $\mathbf{o}_j$ and use that to predict the subsequent h time-points $\mathbf{y}_{pj+1:pj+h}$. This is done for all patches in one training mini-batch.

Loss Function. In this work, we focus on point forecasting. Therefore we can use a point forecasting loss during training like MSE as defined in Equation (2). The loss that is minimized during training can be expressed as,

$$\texttt{TrainLoss} = \frac{1}{N} \sum_{j=1}^{N} \text{MSE}(\hat{\mathbf{y}}_{pj+1:pj+h}, \mathbf{y}_{pj+1:pj+h}). \tag{6}$$

Note that if one is interested in probabilistic forecasting, then it is easy to have multiple output heads for each output patch, each head minimizing a separate quantile loss as in (Wen et al., 2017). Another approach can be to output the logits of a probability distribution family and minimize the maximum likelihood loss for probabilistic forecasting (Awasthi et al., 2021; Salinas et al., 2020).

Inference. The trained network can be used to produce forecasts for *any* horizon using auto-regressive decoding similar to large language models. Given an input $\mathbf{y}_{1:L}$ (assume L is a multiple of p for simplicity) it can first predict $\hat{\mathbf{y}}_{L+1:L+h}$. Then, we can use the concatenated vector $\tilde{\mathbf{y}}_{1:L+h} = [\mathbf{y}_{1:L}; \hat{\mathbf{y}}_{L+1:L+h}]$ as an input to the network to generate the next output patch prediction $\hat{\mathbf{y}}_{L+h+1:L+2h}$ and so on.

We name our model **Pre**trained **Dec**oder for **T**ime-series (PreDcT).

Dataset	# Time-Series	Time Length	Granularity	Context	Horizon
ETTm1	7	69680	15 Min.	512	96
ETTm2	7	69680	15 Min.	512	96
ETTh1	7	17420	1 Hour	512	96
ETTh2	7	17420	1 Hour	512	96
Wiki	115084	803	1 Day	256	56
ILI	7	966	1 Week	96	24
TourismL	555	228	1 Month	64	12

Table 1: Statistics and task definitions of the target datasets.

5 Empirical Results

We evaluate our model in zero-shot settings on well known public datasets against state-of-the-art supervised forecasting baselines. We show that a *single* pretrained model can come close or surpass the performance of baselines on the benchmarks even when the baselines are specially trained or tuned for each specific task. Subsequently, we perform ablation studies that justify different choices made in our foundation model architecture.

5.1 Zero-shot Evaluation

We are interested evaluating the performance of a single foundation model pretrained on a large time-series dataset on different target datasets never seen before by the model or, in other words, *zero-shot evaluation.* To test the generalization performance of our baselines, we choose target datasets covering a variety of data domains, scales and time granularities. We will first discuss the pretraining and target datasets, along with the baselines we use. Then we demonstrate the (Z)ero-(S)hot performance of our model PreDcT(ZS) compared against state-of-the-art models trained directly on the target datasets.

Pretraining Data. We would like our pretraining corpus to include large volumes of temporal data representing a variety of domains, trend patterns and time granularities that ideally capture the forecasting use-cases which we are interested in serving by the deployed model. It is challenging to find a large time-series dataset that meets the volume and diversity of data needed for training our foundation model. In this paper, we find that Google Trends[2] can provide a time series corpus that is ideally suited for pre-training our foundation model. Google Trends captures search interest over time for millions of queries. We choose around 22k head queries based on their search interest over 15 years from 2007 to 2022. We use the search interest over time for these queries in hourly, daily, weekly and monthly granularities to form our dataset. The date ranges are Jan. 2018 to Dec 2019 for hourly and Jan. 2007 to Dec. 2021 for the other granularities. Along with the trends data, we also add time series from several other publicly available datasets to our pretraining corpus. We add in all the granularities of the M4 dataset (Makridakis et al., 2022) and the hourly (and 15 minute) Electricity and hourly Traffic datasets (see (Zhou et al., 2021)). M4 has a good mix of granularities with around 100k time-series in total. Traffic and Electricity are large long-term forecasting datasets with > 800 and > 300 time-series each having tens of thousands of time-points. In addition, we add all the 15 min granularity traffic time series from (Wang et al., 2023).

We train on a mixture distribution over these datasets that aims to give sufficient weightage to all granularities. We train with a maximum context length of 512 whenever the length of the time-series allows that. For weekly granularity we do not have sufficiently long time-series therefore a max. context length of 256 is used. For the same reason, max. context length of 64 is used while training on monthly and higher granularity data.

Target Datasets. To benchmark our model's performance, we choose commonly used forecasting datasets of varying sizes that cover various domains, granularities, context lengths and horizon lengths, to test the generalization power of our foundation model against other baselines. The details are summarized in Table 1.

[2] `https://trends.google.com`

(Sub)Hourly. For 15 min. granularity we use the ETTm1, ETTm2 datasets and test all models on the task of predicting a horizon of 96 time-points after seeing a context of size 512. For hourly granularity, we choose the ETTh1, ETTh2 datasets and test all models on the same task as the 15 min. datasets. The datasets and the context, horizon pair have been widely used in long-term forecasting benchmarks (Zhou et al., 2021; Nie et al., 2022). Note that we used the more challenging original, unscaled versions of these datasets in order to test our model's zero-shot performance on time-series of different scales.

Daily. For the daily granularity, we use the Wikipedia web-traffic dataset from the corresponding Kaggle competition [3]. It has 115k time-series with over two years of data if we exclude the time-series with missing values. The dataset contains web-traffic to Wikipedia articles and the task is to predict the web-traffic (in log scale) on future dates. We choose a context length of 256 to predict a horizon of 56 days (8 weeks). This dataset has been used in prior multivariate forecasting papers such as like (Sen et al., 2019).

Weekly. We use the ILI dataset[4] that collects the number of patients and influenza-like illness ratio in a weekly frequency. We use a context length of 96 to predict 24 weeks into the future. This is one of the configurations used in long-term forecasting papers like (Zhou et al., 2021).

Monthly. We choose TourismL (Tourism Large) (Wickramasuriya et al., 2019) as one of the target datasets. It contains monthly tourist visit data in Australia that has been grouped into various regions. It consists of 555 time-series with very different scales. We choose the task of predicting a 12 month horizon given a context length of 64.

All the target datasets are divided into `train:validation:test` splits (periods) chronologically with the proportions being 7:1:2. We evaluate the models based on metrics resulting from rolling windows in the `test` period. Specifically, PreDcT(ZS) solely predicts in the `test` period as it has already been pretrained. The supervised learning models (per dataset) are trained on the `train` part with the hyper-parameters being tuned using the `validation` split. Then they predict in the `test` period for a head-to-head comparison with the zero-shot PreDcT(ZS).

Baselines. We compare our model against three recently published state-of-the-art supervised forecasting models PatchTST (Nie et al., 2022), TiDE (Das et al., 2023) and FEDFormer (Zhou et al., 2022) as well as the popular N-BEATS (Oreshkin et al., 2019) and DeepAR models (Salinas et al., 2020). Note that these models have already been shown (Zhou et al., 2022; 2021) to be superior to common statistical forecasting methods such as Prophet (Taylor & Letham, 2018) and ARIMA; hence we do not include them in our baselines. We train the above supervised models on the train split of each target dataset and measure their performance on the corresponding test split. For these supervised models we report the best metrics among models trained with and without date-derived features. See Appendix A.2 for the hyper-parameters used for each dataset. We compare our zero-shot metrics from PreDcT(ZS) to these state-of-the-art supervised metrics, which we denote by PatchTST(S), TiDE(S), N-BEATS(S), FEDFormer(S) and DeepAR(S) in our results below.

In PreDcT(ZS), we set `input_patch_len=32`, `output_patch_len=128`. We train a model with about 225M parameters that uses 16 head multi-head attention in each transformer layer.

Results. In Table 2 we present the main results on all our target datasets. We report normalized metrics NRMSE and WAPE, that are proportional to MSE and MAE, and are defined (for each time series) as

$$\mathrm{NRMSE}(\mathbf{y}_{L+1:L+H}, \hat{\mathbf{y}}_{L+1:L+H}) = \frac{\sqrt{\frac{1}{H}\|\mathbf{y}_{L+1:L+H} - \hat{\mathbf{y}}_{L+1:L+H}\|_2^2}}{\frac{1}{H}\|\mathbf{y}_{L+1:L+H}\|_1},$$

$$\mathrm{WAPE}(\mathbf{y}_{L+1:L+H}, \hat{\mathbf{y}}_{L+1:L+H}) = \frac{\frac{1}{H}\|\mathbf{y}_{L+1:L+H} - \hat{\mathbf{y}}_{L+1:L+H}\|_1}{\frac{1}{H}\|\mathbf{y}_{L+1:L+H}\|_1}.$$

The metrics in the table are across all time-series in the test period of the target datasets. The metrics are calculated over all rolling window (context, horizon) pairs that can be extracted from the test period. Note

[3] https://www.kaggle.com/code/muonneutrino/wikipedia-traffic-data-exploration

[4] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html

Model	ETTh1		ETTh2		ETTm1		ETTm2		Wiki		ILI		TourismL	
	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE
PatchTST(S)	**0.656**	0.380	0.245	0.161	**0.571**	**0.307**	0.180	0.114	0.115	0.081	0.414	0.132	0.590	0.203
TiDE(S)	0.663	**0.374**	0.245	0.161	0.588	0.320	0.186	0.120	**0.103**	0.070	0.455	**0.131**	0.574	0.194
N-BEATS(S)	0.687	0.421	**0.235**	0.152	0.608	0.320	**0.179**	**0.111**	**0.103**	**0.069**	**0.410**	0.137	0.666	0.213
FEDFormer(S)	0.675	0.411	0.294	0.205	0.635	0.381	0.240	0.152	-	-	0.441	0.164	1.080	0.266
DeepAR(S)	0.851	0.468	0.331	0.228	0.892	0.453	0.283	0.177	-	-	0.844	0.534	0.859	0.264
PreDcT(ZS)	0.672	0.378	0.238	**0.151**	0.591	0.310	0.199	0.123	0.107	0.070	0.477	0.152	**0.514**	**0.192**

Table 2: We present NRMSE and WAPE metrics for (S)upervised models and our (Z)ero-(S)hot model. The supervised models are trained, tuned and evaluated on the specific target datasets. PreDcT(ZS) metrics are reported on zero-shot performance i.e the model has never seen the target dataset prior to inference. The best number in each column is colored blue, and the second-best number is colored green. The worst performing model metrics per column is colored red and the second-worst is colored orange. It can be seen that the PreDcT(ZS) metrics are uniformly good across all datasets and was among the best or second-best performing model for 8 out of the 14 columns. We report standard errors for the supervised metrics in Table 7 in the appendix.

that FEDformer and DeepAR training was not able to scale to the Wiki dataset (the largest of our target datasets) and hence their metrics for Wiki in the table are left blank.

For each column in the table (corresponding to a metric computed for a dataset), we color-code the best performance by blue, the second-best performance by green, the worst performances by red and the second-worst performance by orange. We observe that PreDcT(ZS) obtained uniformly good results (in the ballpark of the best supervised models) for a majority of the target datasets. In particular, PreDcT(ZS) obtained the best performance for the TourismL dataset, and close to the best performing models for Wiki, ETTh1 and ETTh2. It was among the best or second-best performing model for 8 out of the 14 columns in the table, and never among the worst two performing models in any column except one (where it was second-worst for NRMSE on ILI). This is particularly remarkable since we use a single, pretrained model evaluated in zero-shot manner on the target datasets, and are comparing here against state-of-the-art supervised baselines trained separately on each of the datasets.

5.2 Ablation

Next, we perform several ablation studies that inform the design decisions we made for our foundation model architecture.

Different architectures on same pretraining data. (Nie et al., 2022) have shown that PatchTST can be used to learn semi-supervised representations of time-series. Similarly (Oreshkin et al., 2021) have shown that the N-BEATS architecture can be used for transfer learning in time-series. Therefore, we also consider pre-training a foundation model based on the PatchTST and N-BEATS architecture using the same pretraining dataset, and evaluate them in a zero-shot manner on the target datasets, similar to PreDcT(ZS). These baselines will be denoted by PatchTST(ZS) and N-BEATS(ZS). Note that N-BEATS(ZS) was restricted to training and inference with a fixed context length on account of being a MLP model.

The results are shown in Table 3. It can be seen that PreDcT(ZS) performs better than or similar to PatchTST(ZS) on ETTh1, ETTh2, ETTm2, Wiki and TourismL, but we are dramatically better than PatchTST(ZS) on ETTm2 and ILI. Note that because of encoder-decoder only training PatchTST can only adapt to context lengths that are used for pretraining which are 512, 256 and 64 as mentioned in the *Pretraining Data* section above. This is evident by its bad performance on the ILI dataset which has a context length of 96. This can be further seen in the study in Table 4, discussed subsequently. N-BEATS(ZS) performs slightly better than us on ETTm2, slightly worse on ETTm1, and similar on ETTh1 and ETTh2. But it cannot adapt to varying context lengths, so it could not generalize to the other datasets for Wiki, ILI and TourismL.

Adapting to different context lengths. A good foundation model should be able to adapt to a variety of different context lengths. This is possible in our model because of decoder-only training - the output token of every patch extracts features from all the patches that come before it, and is trained to predict

Model	ETTh1		ETTh2		ETTm1		ETTm2		Wiki		ILI		TourismL	
	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE
PreDcT(ZS)	0.672	0.378	**0.238**	**0.151**	**0.591**	**0.310**	0.199	0.123	**0.107**	**0.070**	**0.477**	**0.152**	0.514	0.192
PatchTST(ZS)	**0.670**	**0.374**	0.239	**0.151**	0.740	0.370	0.205	0.127	0.109	0.073	0.618	0.199	**0.505**	**0.186**
N-BEATS(ZS)	**0.670**	0.381	0.239	0.153	0.617	0.323	**0.189**	**0.119**	-	-	-	-	-	-

Table 3: We present metrics for the three different zero-shot model architectures. It can be see that PreDcT(ZS) does uniformly well across all datasets. PatchTST(ZS) does not do well on ILI on account of not being able to generalize to context 96 because of encoder-decoder mode of training. N-BEATS numbers could not be obtained on non-ETT datasets because it has a fixed context length due to its MLP architecture. The best number in each column is made **bold**.

the next output patch. In Table 4 we show the performance (in terms of NRMSE) of PreDcT(ZS) with different context lengths on the same task as before of predicting 96 time-points. We also juxtapose our performance with PatchTST(ZS) that is trained in encoder-decoder fashion. It can be seen that our model has good performance throughout which becomes progressively better with more context. On the other hand the performance of PatchTST is only good for context length 512 because it has not been optimized for other context lengths because of encoder-decoder model training. Note that because of overlapping stride the original PatchTST model does not lend itself easily to decoder-only training.

Model	ETTh1				ETTh2				ETTm1				ETTm2			
	96	192	384	512	96	192	384	512	96	192	384	512	96	192	384	512
PreDcT(ZS)	**0.779**	**0.715**	**0.692**	0.672	**0.253**	**0.250**	**0.239**	**0.238**	**0.663**	**0.631**	**0.607**	**0.591**	**0.204**	**0.202**	**0.201**	**0.199**
PatchTST(ZS)	1.429	1.125	0.995	**0.670**	0.333	0.279	0.245	0.239	1.168	1.327	1.106	0.740	0.320	0.261	0.220	0.205

Table 4: NRMSE numbers are presented for the pretrained models when the context length is varied at inference time. The prediction horizon is held fixed at 96. It can be seen that PreDcT(ZS) can adapt to different context lengths at inference time.

Input patch length. The size of `input_patch_len` represents an important trade-off. We have typically seen that increasing its value from 8 to 64 increases performance but having too high a `input_patch_len` is impractical because the model cannot be easily applied to context lengths that are less than `input_patch_len`, at inference time. In many monthly and higher granularity tasks, it is common to have small context lengths. In Table 5 we show the NRMSE of another PreDcT(ZS) model with `input_patch_len=8` on ETT datasets, which is clearly worse than our original model that uses `input_patch_len=32`.

`input_patch_len`	ETTh1	ETTh2	ETTm1	ETTm2
32	**0.672**	**0.238**	**0.591**	**0.199**
8	0.680	0.263	0.706	0.209

Table 5: Ablation with respect to input patch length. NRMSE numbers are reported.

Autoregressive decoding. In recent long-term forecasting works (Zeng et al., 2023; Nie et al., 2022; Das et al., 2023) it has been observed that directly predicting the entire forecasting horizon in one shot from a decoder can yield better results than auto-regressive decoding on long horizon benchmarks. For a foundation model the horizon length of the task is not known before inference time, therefore one-shot decoding might not be possible for very long horizons. However, by keeping the `output_patch_len` longer than `input_patch_len` one can ensure fewer autoregressive steps. This was one of the key decisions in the design of PreDcT, that is quite different from LLMs. In order to showcase this we choose the task of predicting 512 time-steps into the future for the ETT datasets. In Table 6, we present results from a model with `output_patch_len=32` vs our original model that uses `output_patch_len=128`. The former has to perform 16 autoregressive steps while the latter has to do only 4. It can be clearly seen that having a larger `output_patch_len` helps in this case.

output_patch_len	ETTh1	ETTh2	ETTm1	ETTm2
128	**0.724**	**0.286**	**0.711**	**0.256**
32	0.746	0.292	0.737	0.259

Table 6: Ablation with respect to output patch length for the task of predicting 512 steps into the future. NRMSE numbers are reported.

6 Discussion and Future Work

We train a decoder-only foundation model for time-series forecasting using a large pretraining corpus of about 1B time-points, the majority of it being search interest time-series derived from Google trends. We show that even a relatively small 225M parameter pretrained model that uses our PreDcT architecture displays impressive zero-shot performance on a variety of public benchmarks from different domains and granularities. The PreDcT(ZS) model can rival the performance of recent state-of-the-art supervised baselines that have been specifically trained on the target datasets. This is remarkable since the PreDcT(ZS) model has not seen the target datasets before inference.

In future work, it would be interesting to push the boundaries of scale in both the pretraining data as well as the number of parameters in the model. It would be a insightful to perform a scaling study in the lines of (Hoffmann et al., 2022) for time-series foundation model. An empirical analysis of probabilistic zero-shot forecasting from an appropriately trained foundation model is also an interesting direction for future work.

References

Pranjal Awasthi, Abhimanyu Das, Rajat Sen, and Ananda Theertha Suresh. On the benefits of maximum likelihood estimation for regression and forecasting. *arXiv preprint arXiv:2106.10370*, 2021.

Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

George EP Box and Gwilym M Jenkins. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 17(2):91–109, 1968.

Ching Chang, Wen-Chih Peng, and Tien-Fu Chen. Llm4ts: Two-stage fine-tuning for time-series forecasting with pre-trained llms. *arXiv preprint arXiv:2308.08469*, 2023.

Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan K Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with TiDE: Time-series dense encoder. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=pCbC3aQB5W`.

Azul Garza and Max Mergenthaler-Canseco. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Michael Kopp, David Kreil, Moritz Neun, David Jonietz, Henry Martin, Pedro Herruzo, Aleksandra Gruca, Ali Soleymani, Fanyou Wu, Yang Liu, Jingwei Xu, Jianjin Zhang, Jay Santokhi, Alabi Bojesomo, Hasan Al Marzouqi, Panos Liatsis, Pak Hay Kwok, Qi Qi, and Sepp Hochreiter. Traffic4cast at neurips 2020 - yet more on the unreasonable effectiveness of gridded geo-spatial processes. In Hugo Jair Escalante and Katja Hofmann (eds.), *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pp. 325–343. PMLR, 06–12 Dec 2021. URL `https://proceedings.mlr.press/v133/kopp21a.html`.

Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Qianli Ma, Zhen Liu, Zhenjing Zheng, Ziyang Huang, Siying Zhu, Zhongzhong Yu, and James T Kwok. A survey on time-series pre-trained models. *arXiv preprint arXiv:2305.10716*, 2023.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022.

ED McKenzie. General exponential smoothing and the equivalent arma process. *Journal of Forecasting*, 3 (3):333–344, 1984.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *International conference on learning representations*, 2022.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2019.

Boris N Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. Meta-learning framework with applications to zero-shot time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9242–9250, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *Advances in neural information processing systems*, 32, 2019.

Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jingyuan Wang, Jiawei Jiang, Wenjun Jiang, Chengkai Han, and Wayne Xin Zhao. Towards efficient and comprehensive urban spatial-temporal prediction: A unified library and performance benchmark. *arXiv preprint arXiv:2304.14343*, 2023.

Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Shanika L Wickramasuriya, George Athanasopoulos, and Rob J Hyndman. Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. *Journal of the American Statistical Association*, 114(526):804–819, 2019.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *Proceedings of the AAAI conference on artificial intelligence*, 2023.

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. In *Proceedings of the AAAI conference on artificial intelligence*, 2021.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pp. 27268–27286. PMLR, 2022.

Tian Zhou, Peisong Niu, Xue Wang, Liang Sun, and Rong Jin. One fits all: Power general time series analysis by pretrained lm. *arXiv preprint arXiv:2302.11939*, 2023.

Eric Zivot and Jiahui Wang. Vector autoregressive models for multivariate time series. *Modeling financial time series with S-PLUS®*, pp. 385–429, 2006.

A Appendix

A.1 Date Derived Features

In our study we focus on 5 date derived features: (1) month of the year, (2) day of the week, (3) hour of the day, (4) minute of the hour and (5) second of the minute. For any time point t in the context and the horizon we have $\mathbf{x}_t \in \mathbb{R}^5$. In addition for each date derived feature:

1. We mask it by -1 if its granularity is irrelevant to the time series $\mathbf{y}$. For example, the daily Wikipedia makes no use of hour of the day, minute of the hour, and second of the minute.
2. We normalize its values to $[-0.5, 0.5]$. For example, the raw minute of the hour value v is transformed to $v/60 - 0.5$.

A.2 More Details on Models

N-BEATS. All N-BEATS(S) and N-BEATS(ZS) models are trained following the N-BEATS-G hyper-parameters used in Table-18 from the original paper (Oreshkin et al., 2019), except that we use width = 1024 and stacks = 24 to match the model sizes to about 200M. All supervised N-BEATS(S) use the dataset specific input output lengths, while N-BEATS(ZS) is trained with input length = 512 and output length = 96 to predict on all ETT datasets.

PatchTST. For PatchTST(S) models in all ETT datasets and ILI we use the hyper-parameters from the original paper (Nie et al., 2022). For Wiki and TourismL, we use 16 attention heads, 10 layers, a patch-size of 64 and a stride of 32. The model dimension used is 1024. We tune the learning rate per dataset.

For PatchTST(ZS) we use we use 16 attention heads, 20 layers, a patch-size of 32 and a stride of 16. We use the same model dimension as PreDcT(ZS).

TiDE. We used the hyper-parameters from the original paper (Das et al., 2023), but tuned the model dimensions (from 256 to 1024), number of layers (from 1 to 4) and learning date per dataset.

FEDFormer. For all datasets we use the same hyper-parameters of the original paper (Zhou et al., 2022), except model dimensions. We tuned the model dimension per dataset from 64 to 1024.

DeepAR. For all datasets we use the same hyper-parameters of the original paper (Zhou et al., 2022), except model dimensions, number of layers (from 1-4) and learning rate per dataset. We tuned the model dimension per dataset from 64 to 1024.

PreDcT. We use use 16 attention heads, 20 layers, a input patch length of 32 and output patch length of 128. The model dimension is set to 1280.

A.3 Additional Empirical Results

Model	ETTh1		ETTh2		ETTm1		ETTm2	
	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE
PatchTST(S)	0.656 ± 0.0004	0.380 ± 0.0002	0.245 ± 0.0004	0.161 ± 0.0004	0.571 ± 0.003	0.307 ± 0.001	0.180 ± 0.0008	0.114 ± 0.0007
TiDE(S)	0.663 ± 0.0001	0.374 ± 0.0001	0.245 ± 0.002	0.161 ± 0.001	0.588 ± 0.0005	0.320 ± 0.0005	0.186 ± 0.0005	0.120 ± 0.0001
N-BEATS(S)	0.687 ± 0.002	0.421 ± 0.002	0.235 ± 0.0002	0.152 ± 0.0001	0.608 ± 0.007	0.320 ± 0.004	0.179 ± 0.001	0.111 ± 0.0003
FEDFormer(S)	0.675 ± 0.003	0.411 ± 0.004	0.294 ± 0.010	0.205 ± 0.01	0.635 ± 0.007	0.381 ± 0.005	0.240 ± 0.006	0.152 ± 0.004
DeepAR(S)	0.851 ± 0.005	0.468 ± 0.003	0.331 ± 0.007	0.228 ± 0.003	0.892 ± 0.015	0.453 ± 0.007	0.283 ± 0.005	0.177 ± 0.004
Model	Wiki		ILI		TourismL		-	
	NRMSE	WAPE	NRMSE	WAPE	NRMSE	WAPE	-	-
PatchTST(S)	0.115 ± 0.0002	0.081 ± 0.0002	0.414 ± 0.005	0.132 ± 0.003	0.595 ± 0.012	0.204 ± 0.003	-	-
TiDE(S)	0.103 ± 0.0004	0.070 ± 0.0006	0.455 ± 0.004	0.131 ± 0.005	0.574 ± 0.001	0.194 ± 0.002	-	-
N-BEATS(S)	0.103 ± 0.0002	0.069 ± 0.0001	0.410 ± 0.003	0.137 ± 0.002	0.666 ± 0.002	0.213 ± 0.0003	-	-
FEDFormer(S)	-	-	0.441 ± 0.0007	0.164 ± 0.0003	1.080 ± 0.002	0.266 ± 0.0007	-	-
DeepAR(S)	-	-	0.844 ± 0.007	0.534 ± 0.006	0.859 ± 0.008	0.264 ± 0.001	-	-

Table 7: NRMSE and WAPE confidence intervals (mean ± 1 standard error) of the supervised baselines.