A decoder-only foundation model for time-series forecasting

Abhimanyu Das [1] **Weihao Kong** [1] **Rajat Sen** [1] **Yichen Zhou** [1]

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 decoder style attention model with input patching, using a large time-series corpus comprising both real-world and synthetic datasets. Experiments on a diverse set of previously unseen forecasting datasets suggests that the model can yield accurate zero-shot forecasts across different domains, forecasting horizons 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; Oreshkin et al., 2019) 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) deep network based solutions performed very well.

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 ? Such a pretrained time-series foundation model, if possible, would bring significant benefits for downstream forecasting users in terms of no additional training burden and significantly reduced 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, such a 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 *TimesFM*, a single foundation model for time-series forecasting that, when applied to a variety of previously-unseen forecasting datasets across different domains, 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 large-scale time-series corpus built using both real-world (mostly time-series data from web search queries[1] and Wikipedia page visits[2]) and synthetic data, which meets the volume and diversity of data needed for training our foundation model, and 2) a decoder style attention architecture with input patching, that can be efficiently pre-trained on this time-series corpus.

[1]Google Research. Correspondence to: Rajat Sen <senrajat@google.com>, Yichen Zhou <yichenzhou@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

Author names are listed in alphabetical order.

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

[2]https://wikimedia.org/api/rest_v1/

Compared to the latest large language models, our time-series foundation model is much smaller in both parameter size (200M parameters) and pretraining data size (O(100B) timepoints); yet we show that even at such scales, it is possible to pretrain 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. Our work also suggests that unlike recent work (Gruver et al., 2023) that recommends Large Language Models such as GPT-3 and LLama-2 as out-of-the-box zero-shot forecasters, foundation models trained from scratch exclusively on time-series data can obtain much better zero-shot performance at a tiny fraction of its costs.

2. Related Work

In the last decade, deep learning models (Salinas et al., 2020; Oreshkin et al., 2019) 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 and 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; Chen et al., 2023) 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. 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 an in-depth discussion about transfer learning in time-series we refer the reader to the survey in (Ma et al., 2023).

There has been some very recent work on re-using or fine-tuning large language models for time-series forecasting. In particular, Gruver et al. (2023) benchmarks pretrained LLMs like GPT-3 and LLaMA-2 for zero-shot forecasting performance. As we show later, our model obtains much superior zero-shot performance at a tiny fraction of these model sizes. Zhou et al. (2023) and Chang et al. (2023) show how to fine-tune a GPT-2 (Radford et al., 2019) backbone model for time-series forecasting tasks. With the exception of a transfer-learning study (forecasting on a target dataset after having trained on a source dataset), these papers mostly focus on fine-tuning a pretrained model on target datasets, and not on pretraining a single foundation model with good 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 other parallel work on a zero-shot foundation model for time-series forecasting. However the model is 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 are denoted by $\mathbf{y}_{L+1:L+H}$. Note that since we are building a single pre-trained model, we cannot have dataset specific dynamic or static covariates during training time. The task is then to learn a foundation model that can map any time-series context to horizon,

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

The accuracy of the prediction can be measured by a metric that quantifies their closeness to the actual values, for instance, Mean Absolute Error (MAE) defined in Equation 6.

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). However, there are several time-series specific design choices. The main guiding principles for our architecture are the following:

Patching. Inspired by the success of patch based modeling in the recent long horizon forecasting work (Nie et al., 2022) we also choose to break down the time-series into patches during training. A patch of a time-series is a natural analogue for a token in language models and patching has been shown to improve performance. Moreover this improves inference speed as the number of tokens being

fed into the transformer is reduced by a factor of the patch length. On the other hand, increasing the patch length all the way to the context length moves us away from decoder-only training and the efficiencies that come with it. We delve into this further in Section 6.2.

Decoder-only model. A key difference between our architecture and PatchTST (Nie et al., 2022) is that our model is trained in decoder-only mode (Liu et al., 2018). In other words, given a sequence of input patches, the model is optimized to predict the next patch as a function of all past patches. Similar to LLMs this can be done in parallel over the entire context window, and automatically enables the model to predict the future after having seen varying number of input patches.

Longer output patches. In LLMs the output is always generated in an auto-regressive fashion one token at a time. However, in long-horizon forecasting it has been observed that directly predicting the full horizon yields better accuracy than multi-step auto-regressive decoding (Zeng et al., 2023). But this is not possible when the horizon length is not known apriori, as in the case of zero-shot forecasting which is our primary goal.

We propose a middle ground by allowing our output patches for prediction to be longer than the input patches. As an example, suppose the input patch length is 32 and output patch length is 128. During training, the model is simultaneously trained to use the first 32 time-points to forecast the next 128 time-steps, the first 64 time-points to forecast time-steps 65 to 192, the first 96 time-points to forecast time-steps 97 to 224 and so on. During inference, suppose the model is given a new time-series of length 256 and tasked with forecasting the next 256 time-steps into the future. The model will first generate the future predictions for time-steps 257 to 384, then condition on the initial 256 length input plus the generated output to generate time-steps 385 to 512. On the other hand, if in a model the output patch length was fixed to the input patch length of 32, then for the same task we would have to go through 8 auto-regressive generation steps instead of just the 2 above. However, there is a trade-off. If the output patch length is too long, then it is difficult to handle time-series whose lengths are less than the output patch length for instance monthly, yearly time-series in our pretraining data.

Patch Masking. If we use patches naively, the model might only learn to predict well for context lengths that are multiples of the input patch length. Therefore we make a careful use of masking during training. Parts of patches as well as entire patches from the beginning of the context window can be masked in a data batch. We employ a specific random masking strategy (described later) during training that helps the model see all possible context lengths starting from 1 to a maximum context length.

Now that we have mentioned the guiding principles, we next formally describe each component of our model architecture (illustrated in Figure 1), which we name as TimesFM (**T**ime-**s**eries **F**oundation **M**odel).

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 is processed by a Residual Block into a vector of size `model_dim`. Along with the input, we also supply a binary padding mask $\mathbf{m}_{1:L}$ where 1 denotes that the corresponding input in $\mathbf{y}_{1:L}$ should be ignored and vice-versa. The Residual Block is essentially a Multi-layer Perceptron (MLP) block with one hidden layer with a skip connection, similar to that defined in Das et al. (2023).

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

$$\mathbf{t}_j = \texttt{InputResidualBlock}(\tilde{\mathbf{y}}_j \odot (1 - \tilde{\mathbf{m}}_j)) + \texttt{PE}_j \tag{2}$$

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 `num_layers` (n_l) transformer layers stacked on top of each other. Each of these layers have the standard multi-head self-attention (SA) 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 FFNs 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

$$\mathbf{o}_j = \texttt{StackedTransformer}((\mathbf{t}_1, \dot{m}_1), \cdots, (\mathbf{t}_j, \dot{m}_j)), \tag{3}$$

for all $j \in [N]$. $\dot{m}_j$ is the masking indicator for the j-th token defined as $\min\{\mathbf{m}_{p(j-1)+1:pj}\}$ i.e if a patch has any non-masked time-point the corresponding token marked as not being masked. All patches that are masked out completely are not attended to by the causal self attention.

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

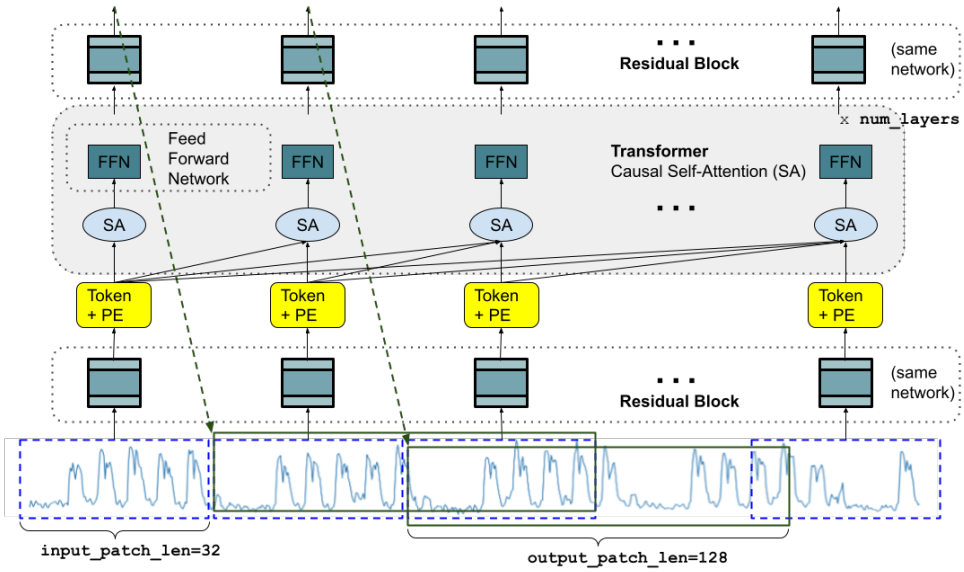


Figure 1. We provide an illustration of the TimesFM model architecture during training, where we show a input time-series of a specific length that can be broken down into input patches. Each patch along is processed into a vector by a residual block (as defined in the model definition) to the model 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 window following the last input patch seen by the model so far.

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,

$$\hat{\mathbf{y}}_{pj+1:pj+h} = \texttt{OutputResidualBlock}(\mathbf{o}_j). \quad (4)$$

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 Mean Squared Error (MSE). The loss that is minimized during training can be expressed as,

$$\texttt{TrainLoss} = \frac{1}{N}\sum_{j=1}^{N} \text{MSE}(\hat{\mathbf{y}}_{pj+1:pj+h}, \mathbf{y}_{pj+1:pj+h}). \quad (5)$$

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).

Training. We train the model with standard mini-batch gradient descent in decoder-only fashion, that goes through all windows for a time-series and across time-series. The only non-standard part is the way we sample the mask during training. For each time-series in the batch, we sample a random number r between 0 and $p-1$. Then we set the $\mathbf{m}_{1:r} = 1$ and the rest as zero i.e we mask out a fraction of the first input patch. However, this is sufficient to cover all input context lengths from 1 to the maximum training context length. We explain this using an example below:

Suppose the maximum context length is 512 and $p = 32$. Then if $r = 4$, the output prediction after seeing the first patch (from $\mathbf{o}_1$) is optimized to predict after seeing $28 = 32-4$ time-points, the output of the next patch (from $\mathbf{o}_2$) is optimized to predict after seeing $28 + 32$ time-points, and so on. When this argument is repeated for all such r's, the model has seen all possible context lengths till 512.

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 pre-

dict $\hat{\mathbf{y}}_{L+1:L+h}$. 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. If L is not a multiple of p, we simply append zeros to make it a multiple of p and mark the corresponding entries in the mask as 1.

5. Pretraining Details

We would like our pretraining corpus to include large volumes of temporal data representing a variety of domains, trend and seasonality 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. We address this problem by sourcing the bulk of data used to train our models from three major sources: Google trends, Wiki Pageview statistics and synthetic time-series. In summary the main data sources are:

Google Trends. Google Trends [3] 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. Beyond these head queries the time-series become more than 50% sparse. We download 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. The trends datasets amounts to roughly 0.5B time-points.

Wiki Pageviews. Wiki Pageviews [4] captures the hourly views of all Wikimedia pages. We download all pageview data from Jan. 2012 to Nov. 2023, clean and aggregate the views by page into hourly, daily, weekly and monthly granularities, and filter out pageview time-series with excessive zeros. The final corpus contains roughly 300B time-points.

Synthetic Data. Another major component of our pretraining data is of synthetic origin. We create generators for ARMA (McKenzie, 1984) processes, seasonal patterns (mixture of sines and cosines of different frequencies), trends (linear, exponential with a few change-points) and step functions. A synthetic time-series can be an additive combination of one or more of these processes. We create 3M synthetic time-series each of length 2048 time-points. More details about our synthetic data generation are presented in Appendix A.7.

Other real-world data sources. Along with the wiki and trends data, we also add time-series from several other publicly available datasets to our pretraining corpus. We add all the granularities of the M4 dataset (Makridakis et al., 2022), the hourly and 15 minute Electricity and the hourly Traffic datasets (see Zhou et al. (2021)). We also add the 10-minute granularity Weather dataset used for evaluations in 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).

Table 1. Composition of TimesFM pretraining dataset.

Dataset	Granularity	# Time series	# Time points
Synthetic		3,000,000	6,144,000,000
Electricity	Hourly	321	8,443,584
Traffic	Hourly	862	15,122,928
Weather (Zhou et al., 2021)	10 Min	42	2,213,232
Favorita Sales	Daily	111,840	139,179,538
LibCity (Wang et al., 2023)	15 Min	6,159	34,253,622
M4 hourly	Hourly	414	353,500
M4 daily	Daily	4,227	9,964,658
M4 monthly	Monthly	48,000	10,382,411
M4 quarterly	Quarterly	24,000	2,214,108
M4 yearly	Yearly	22,739	840,644
Wiki hourly	Hourly	5,608,693	239,110,787,496
Wiki daily	Daily	68,448,204	115,143,501,240
Wiki weekly	Weekly	66,579,850	16,414,251,948
Wiki monthly	Monthly	63,151,306	3,789,760,907
Trends hourly	Hourly	22,435	393,043,680
Trends daily	Daily	22,435	122,921,365
Trends weekly	Weekly	22,435	16,585,438
Trends monthly	Monthly	22,435	3,821,760

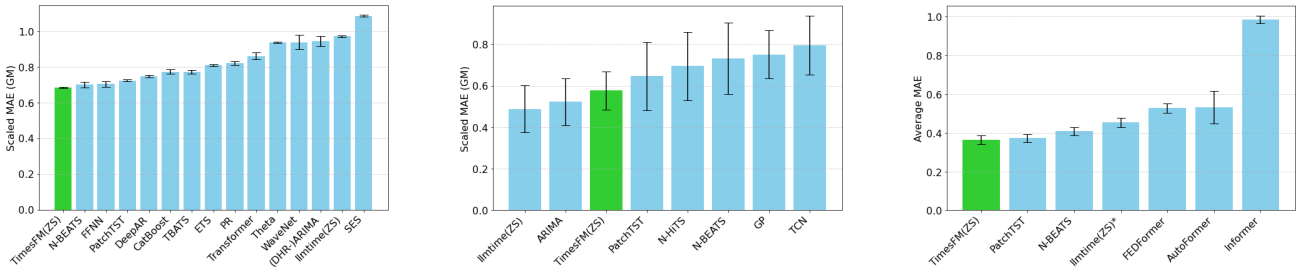
Dataset Mixing and Training. We train on a mixture distribution over these datasets that aims to give sufficient weight to all granularities and datasets. The training loader samples 80% real data and 20% synthetic, with the real data mixture providing equal weights to the groups: hourly + sub-hourly, daily, weekly, and monthly datasets. 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 maximum context length of 256 is used. For the same reason, a maximum context length of 64 is used while training on $\geq$ monthly granularity data. We also use only the standard normalization part of reversible instance normalization (Kim et al., 2021) – i.e, the context of each time-series is scaled by the context mean and standard deviation of the first input patch in the context.

6. Empirical Results

We evaluate our model in zero-shot settings on three groups of well known public datasets against the best performing baselines for each group. These datasets have been intentionally held out from our pretraining data. We show that a *single* pretrained model can come close or surpass the performance of baselines models on the benchmarks even

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

[4] https://en.wikipedia.org/wiki/Wikipedia:Pageview_statistics



(a) Monash Archive (Godahewa et al., 2021) (b) Darts (Herzen et al., 2022) (c) ETT (Horizons 96 and 192) (Zhou et al., 2021)

Figure 2. We report average performance in three groups of datasets. In all figures, the lower the metric the better and the error bars represent one standard error. Note that among the baselines only TimesFM and llmtime are zero-shot. In (a) we report results on the Monash datasets. Since the datasets have different scales, we take the Geometric Mean (GM) of the MAE's scaled by the MAE of a naive baseline. We can see that TimesFM is the top model. In (b), we report the similarly scaled MAE on the Darts benchmarks. TimesFM is within significance of the best performing methods which are ARIMA and llmtime in this case. Note that these datasets have one time-series each and therefore statistical methods are competitive with deep learning ones. Finally, in (c) we report the average MAE for 96 and 192 horizon prediction tasks on 4 ETT datasets i.e 8 tasks in total. TimesFM and PatchTST are the best performing models

when the baselines are specially trained or tuned for each specific task. Subsequently, we perform ablation studies that justify different choices made in our architecture.

6.1. Zero-shot Evaluation

To benchmark our model's performance, we choose three groups of commonly used forecasting datasets that cover various domains, sizes, granularities, and horizon lengths: Darts (Herzen et al., 2022), Monash (Godahewa et al., 2021) and Informer datasets (Zhou et al., 2021), to test the generalization power of our foundation model against other baselines.

In all cases, we report performance on the official metrics and scalings of the datasets, using either their standard test splits or common test splits in other literature. We present a summary of the results below - more details can be found in Appendix A.4. We provide the hyper-parameters and other details about our model in Appendix A.5. A version of TimesFM has been released on Hugging at `timesfm-1.0-200m`, with corresponding inference `code`.

Monash (Godahewa et al., 2021). Monash archive is a collection of 30 datasets of different training and prediction lengths that covers granularities ranging from minutes to years and domains including finance, demand forecasting, weather and traffic. The archive reports four official metrics for several statistical baselines such as Exponential Smoothing(ETS) and ARIMA, as well as supervised ML baselines like CatBoost (Prokhorenkova et al., 2018), DeepAR (Salinas et al., 2020) and WaveNet (Oord et al., 2016). Following llmtime (Gruver et al., 2023) we start from the Monash Huggingface repository [5] and filter out the datasets that contain missing values. This leaves us with 18 datasets which we specify in Appendix A.4.2.

Out of the four official metrics, following prior work (Gruver et al., 2023), we report our performance in terms of mean MAE (see Appendix A.1). As the datasets have massively different scales, for each dataset we normalize the metric by the metric achieved by a naive baseline that just constantly predicts the last value in the context for each time-series. Then the scaled MAE's are averaged across all datasets. The scaled aggregation was also used in Gruver et al. (2023). In Figure 2a, we use the Geometric Mean (GM) for averaging since it is more robust for normalized metrics (Fleming & Wallace, 1986). We also report the Arithmetic Mean based aggregated metrics in Figure 7 in the appendix.

The mean scaled MAE across all datasets is plotted in Figure 2a along with standard error bars. We compare the performance of TimesFM with the baseline models implemented in Monash, and the zero-shot llmtime (Gruver et al., 2023) model that uses GPT-3 (Radford et al., 2019) with a specific prompting technique. Note that the zero-shot models are marked as (**Z**ero-**S**hot). TimesFM is the top model even though we never trained on these datasets. It is slightly better but within significance of N-BEATS but outperforms deep supervised models like DeepAR (Salinas et al., 2020), and improves on llmtime's performance by more than 25%.

Darts (Herzen et al., 2022). This is a collection of 8 univariate datasets which include interesting seasonalities and additive+multiplicative trends. We report performance of several baselines implemented in the Darts package like

[5] `https://huggingface.co/datasets/monash_tsf`

TCN (Lea et al., 2016), N-HiTS (Challu et al., 2023) and N-BEATS (Oreshkin et al., 2019). All these baselines are supervised. As before, we also report zero-shot forecasting results from llmtime (Gruver et al., 2023) using GPT-3 (Radford et al., 2019). Other supervised baselines in (Gruver et al., 2023) like SM-GP (Wilson & Adams, 2013) and ARIMA (McKenzie, 1984) are also added.

We report the official metric for this dataset group that is MAE for each individual dataset in Appendix A.4. In Figure 2b, we present the average scaled MAE across all 8 datasets, as we did for the Monash datasets. TimesFM is within statistical significance of the best models that is llmtime and seasonal ARIMA in this case. Note that since there are only 8 individual time-series in this dataset group, the standard errors are not sharp and therefore does not provide a clear ordering among the models. Also, note that for ARIMA, the seasonality needs to be encoded correctly in the parameters for the best results, which needed manual tuning. Further, since these datasets are used in numerous time series blog posts for illustrative purposes, data contamination for llmtime cannot be ruled out.

Informer (Zhou et al., 2021). The Informer datasets have been widely used for benchmarking various supervised long-horizon forecasting methods. A few of these datasets are used in pretraining, so we focus on the other datasets in this collection (ETTm1, ETTm2. ETTh1 and ETTh2) related to electricity transformer temperatures over a two year period in 1 hour and 15 minutes granularities. Note that the long horizon baselines usually report rolling validation results on the test set which would amount to millions of tokens for evaluating llmtime (Gruver et al., 2023) and would be too expensive. Therefore, following llmtime, we compare all methods on the last test window. Also, it is reasonable to directly average the MAE for these datasets since the results are reported on standard normalized dataset (using the statistics of the training portion).

We consider the task of predicting horizon length 96 and 192, given a context length of 512 for all methods. The MAE averaged over all 8 tasks (4 datasets with two horizons each) is presented in Figure 2b. TimesFM performs the best and the supervised PatchTST (Nie et al., 2022) baseline (which is a state-of-the-art long horizon deep forecasting method) is within significance of it. The other long horizon methods are quite a bit worse even though they have been trained these datasets. llmtime is better than FEDFormer but worse than PatchTST with statistical significance.

We present visual examples of our forecasts along with baselines in Appendix A.8.

6.2. Ablation

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

Scaling. Performance curves with respect to number of parameters in a model have been a keenly studied area in the context of LLMs. Kaplan et al. (2020) established a power law like relationship between the number of parameters in a language model and its downstream performance i.e the more the number of paramaters the better the performance. However, Hoffmann et al. (2022) established a more nuanced scaling law that lays down methods to train compute optimal models based on the number of tokens available in a training dataset.

We perform a preliminary scaling study where we train three TimesFM models of sizes 17M, 70M and 200M parameters, using the same pre-training dataset till 1.5M iterations with a global batch-size of 4096. Then we collect checkpoints that represent varying number of FLOPS (Floating Point OPerationS) across the different model runs. Then we plot the performance on Scaled MAE (GM) on Monash as a function of FLOPS, in Figure 3. This is now a standard way to perform scaling studies in LLMs (see recent work like Gu & Dao (2023)). It can be clearly seen that the errors decrease monotonically with the number of FLOPS (in log scale). All experiments were performed on a TPUv5e[6] setup with 16 tensor-cores. For the 200M model it takes 2 days to complete 1.5M iterations on our setup.

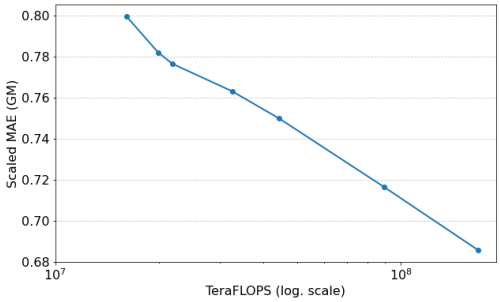


Figure 3. Scaled MAE (GM) on Monash datasets as a function of FLOPS across three model sizes 17M, 70M and 200M. The first 4 points are from 17M and 70M checkpoints while the last 3 are from 200M.

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, there-

[6] https://cloud.google.com/tpu/docs/v5e-training

fore one-shot decoding might not be possible for very long horizons. However, as mentioned earlier, 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 TimesFM, 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 on the original rolling validation task of the ETT test sets (Zhou et al., 2021). In Figure 4, we present results from models with output_patch_len varying from 8 to 128. We see a monotonic decrease in average MAE with output_patch_len.

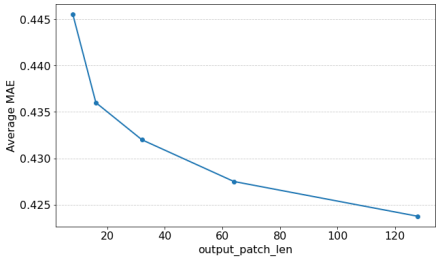


Figure 4. Ablation with respect to output patch length for the task of predicting 512 steps into the future on ETT datasets on the original test set in Zhou et al. (2021). We report the average across all 4 ETT datasets.

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 32 increases performance but having too high a `input_patch_len` is impractical since that makes the model shift from decoder only training more towards encoder-decoder style training. Note that in the "Training" paragraph of Section 4, we describe the mask sampling strategy to support any context length. If in the extreme case p is set the maximum context length we have to individually sample all possible context windows from 1 to maximum context length, which would be required for encoder-decoder style of training.

In Figure 5, we show the mean scaled MAE (GM) TimesFM(ZS) - 70M model on Monash with `input_patch_len` varying from 8 to 128. Note that both models have been trained to about 1.5M steps even though the p=8 model is three times slower to train. We can see that $p = 16, 32$ marks the best performance, with the error increasing towards either end. Note that $p = 32$ model is almost twice as fast to train compared to $p = 16$ and thus constitutes a prudent choice.

Dataset Ablation. Next we showcase the need for synthetic data. Intuitively, the majority of our real datasets have commonly found granularities like hourly, daily etc which have specific periodic patterns like 24 time-point period for hourly data. This can make the model not generalize well to underrepresented frequencies. We train a

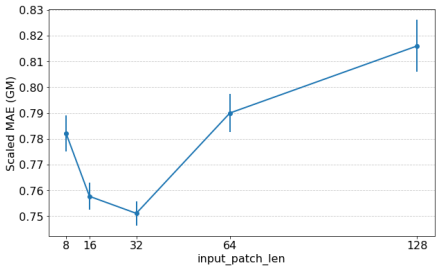


Figure 5. Scaled MAE (GM) for our 70M models on Monash datasets for different input patch lengths. We also plot error bars denoting one standard error.

200M model with no synthetic data added in the mix and showcase the performance on Monash and ETT datasets in Figure 6. It can be seen that there is a performance drop on Monash because many of the datasets in Monash have under-represented granularities like quarterly, yearly or 10 minutes etc. Perhaps even more compelling is the comparison on ETT datasets. We can see that there is almost no difference between the two models on the hourly ETTh datasets which has a well represented granularity. However, for the 15min ETTm datasets the model with synthetic data performs quite a bit better.

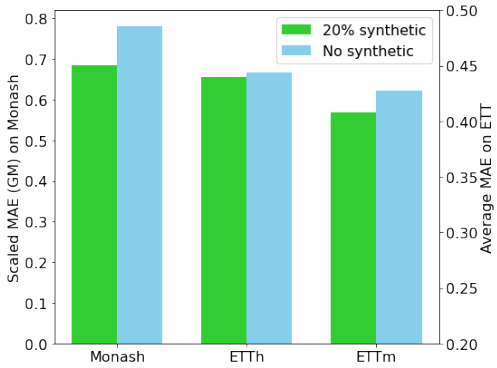


Figure 6. Average scaled MAE for Monash on the left and average MAE on ETT datasets on the right. We compare the performance of 200M model with and without the synthetic data.

We provide a finetuning study in the same setting as Zhou et al. (2023) in Appendix A.2, where our model performs better than all baselines on all the reported datasets. This shows the utility of our model on downstream tasks.

7. Conclusion

In this paper, we presented TimesFM, a practical foundation model for forecasting whose zero-shot performance comes close to the accuracy of fully-supervised forecasting models on a diverse set of time-series data. This model is pretrained on real-world and synthetic datasets comprising

O(100B) timepoints. The limitations and future directions can be categorized under the following,

Prompt Tuning. In LLMs it is well known that prompt tuning techniques like chain-of-thought (Wei et al., 2022) can drastically improve performance in cases where the model is inaccurate with simple prompts. Such techniques are less clear for time-series foundation model. We can tune simple hyper-parameters like context length as the moment. However, with probabilistic forecasting we might be able to output different statistics as well as come up with techniques that align more with user's expectations while not decreasing likelihood.

Probabilistic Forecasting. It should be straightforward to train with probabilistic loss functions in our framework as detailed in the "Loss Function" part of Section 4. However, being one of the first works of building a single foundation model for forecasting, this was not our main focus and is left to future explorations. Note that as mentioned before we plan to release our model weights and after that such loss functions (Salinas et al., 2020; Awasthi et al., 2021) can be added during finetuning.

Covariate handling. Currently the model is not pretrained with covariates as one of the key challenges is finding large volumes of pretrained data with meaningful covariates (apart from date features). We also need methods to have a joint universal representation of covariates. Currently there are two simple techniques we can think of for handling covariates (i) In a zero-shot setting at inference time we can predict in-context and linearly regress the residual on covaraites. Then our model + the residual model can be used for forecasting in the horizon. (ii) during finetuning it is straightforward to handle covariates by adding them as inputs to the input and output residual blocks. Categorical variables can be added as embeddings.

More finetuning studies. We perform a fintuning study in Appendix A.2 following a prior work. However, a more in depth study that involves finetuning in the presence of covariates would be beneficial. This being one of the first works of building a single foundation model for forecasting, this was not our main focus and is left to future explorations. Ideas in recent work such as (Chen et al., 2023) could be useful in this regard.

Other architectures. Given the cost of training foundation models we did not perform much hyper-parameter tuning in our pretraining, while following some well established best practices for training transformers. In a similar vein, it would also be interesting to try out alternatives like the exciting directions of all MLP structures like Chen et al. (2023) or efficient linear state space models like Mamba (Gu & Dao, 2023) (and references there in).

Interpretability. Deep foundation models trained on a huge corpuses of data could be inherently less interpretable compared to statistical methods like ARIMA, ETS (Box & Jenkins, 1968). In this regard methods like LOCO, SHAP (see (Verdinelli & Wasserman, 2023) and references there in) could be used to some extent to attribute feature importances to different lags in the context supplied to the model. However, this does not solve the problem to a full extent and one of the best things to do would be to open source a version of the model with a proper model card. (Mitchell et al., 2019).

Impact Statement

This paper shows that it is possible to train a single pretrained model that has phenomenal zero-shot performance on a variety of forecasting tasks, thus opening up exciting possibilities for downstream applications. Therefore it is crucial to discuss ethical and societal considerations of using such a model and how some of the related concerns can be mitigated.

Data Privacy. Note that most of our data sources are publicly available and are aggregated i.e no individual user activity constitutes a time-point. Further the Google Trends data is differentially private.

Bias. Biases can creep into a foundation model through a variety of sources especially through data. The model might perpetuate these biases in forecasts, leading to unfair outcomes. Biased forecasts can have real-world consequences. For example, a biased forecast of crime rates in a neighborhood could lead to increased police presence, disproportionately impacting certain communities. Note that our model is not trained with any covariates so some of these sensitivities are reduced but cannot be ruled out.

In this regard, we believe that it is best to release the exact details of the datasets used for training and therefore we have summarized our data sources in Table 1. Moreover we plan to do a open weights release of our model so that the community can analyze the model for downstream tasks. We will ensure that we release a good model card (Mitchell et al., 2019). This will also aid in finetuning the model with more diverse data-sources.

Training cost. Our largest model has 200M parameters which is much smaller compared to even smaller scale LLMs. Our data volume is quite large but still smaller compared to SOTA LLMs. We have revealed the exact computation requirements for the models i.e 16 core TPUv5e for 2 days. Note that experimentation and trial runs of course cost more than the final model run. Therefore in the interest of equitability we would like to release the weights of our model in a responsible manner.

Lastly we would like to note that similar to LLMs there

could be inputs on which the model does not perform well or hallucinates. Therefore in many critical use cases it might be recommended to use the model in a human-in-the-loop fashion or alternatively do a wide range of testing or finetuning.

References

Awasthi, P., Das, A., Sen, R., and Suresh, A. T. On the benefits of maximum likelihood estimation for regression and forecasting. *arXiv preprint arXiv:2106.10370*, 2021.

Borovykh, A., Bohte, S., and Oosterlee, C. W. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

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

Challu, C., Olivares, K. G., Oreshkin, B. N., Garza, F., Mergenthaler, M., and Dubrawski, A. NHITS: Neural Hierarchical Interpolation for Time Series forecasting. In *The Association for the Advancement of Artificial Intelligence Conference 2023 (AAAI 2023)*, 2023. URL https://arxiv.org/abs/2201.12886.

Chang, C., Peng, W.-C., and Chen, T.-F. Llm4ts: Two-stage fine-tuning for time-series forecasting with pre-trained llms. *arXiv preprint arXiv:2308.08469*, 2023.

Chen, S.-A., Li, C.-L., Yoder, N., Arik, S. O., and Pfister, T. Tsmixer: An all-mlp architecture for time series forecasting. *arXiv preprint arXiv:2303.06053*, 2023.

Das, A., Kong, W., Leach, A., Mathur, S. K., Sen, R., and Yu, R. 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.

Fleming, P. J. and Wallace, J. J. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986.

Garza, A. and Mergenthaler-Canseco, M. Timegpt-1. *arXiv preprint arXiv:2310.03589*, 2023.

Godahewa, R., Bergmeir, C., Webb, G. I., Hyndman, R. J., and Montero-Manso, P. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

Gruver, N., Finzi, M., Qiu, S., and Wilson, A. G. Large language models are zero-shot time series forecasters. *arXiv preprint arXiv:2310.07820*, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Herzen, J., Lässig, F., Piazzetta, S. G., Neuer, T., Tafti, L., Raille, G., Van Pottelbergh, T., Pasieka, M., Skrodzki, A., Huguenin, N., et al. Darts: User-friendly modern machine learning for time series. *The Journal of Machine Learning Research*, 23(1):5442–5447, 2022.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kim, T., Kim, J., Tae, Y., Park, C., Choi, J.-H., and Choo, J. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

Kopp, M., Kreil, D., Neun, M., Jonietz, D., Martin, H., Herruzo, P., Gruca, A., Soleymani, A., Wu, F., Liu, Y., Xu, J., Zhang, J., Santokhi, J., Bojesomo, A., Marzouqi, H. A., Liatsis, P., Kwok, P. H., Qi, Q., and Hochreiter, S. Traffic4cast at neurips 2020 - yet more on the unreasonable effectiveness of gridded geospatial processes. In Escalante, H. J. and Hofmann, K. (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.

Lea, C., Vidal, R., Reiter, A., and Hager, G. D. Temporal convolutional networks: A unified approach to action segmentation. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pp. 47–54. Springer, 2016.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Ma, Q., Liu, Z., Zheng, Z., Huang, Z., Zhu, S., Yu, Z., and Kwok, J. T. A survey on time-series pre-trained models. *arXiv preprint arXiv:2305.10716*, 2023.

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

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

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., and Gebru, T. Model cards for model reporting. In *Proceedings of the conference on fairness, accountability, and transparency*, pp. 220–229, 2019.

Nie, Y., Nguyen, N. H., Sinthong, P., and Kalagnanam, J. A time series is worth 64 words: Long-term forecasting with transformers. *International conference on learning representations*, 2022.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

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

Oreshkin, B. N., Carpov, D., Chapados, N., and Bengio, Y. 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.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

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

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Verdinelli, I. and Wasserman, L. Feature importance: A closer look at shapley values and loco. *arXiv preprint arXiv:2303.05981*, 2023.

Wang, J., Jiang, J., Jiang, W., Han, C., and Zhao, W. X. Towards efficient and comprehensive urban spatial-temporal prediction: A unified library and performance benchmark. *arXiv preprint arXiv:2304.14343*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837, 2022.

Wen, R., Torkkola, K., Narayanaswamy, B., and Madeka, D. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Wilson, A. and Adams, R. Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pp. 1067–1075. PMLR, 2013.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

Zeng, A., Chen, M., Zhang, L., and Xu, Q. Are transformers effective for time series forecasting? *Proceedings of the AAAI conference on artificial intelligence*, 2023.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, 2021.

Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., and Jin, R. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In *International Conference on Machine Learning*, pp. 27268–27286. PMLR, 2022.

Zhou, T., Niu, P., Wang, X., Sun, L., and Jin, R. One fits all: Power general time series analysis by pretrained lm. *arXiv preprint arXiv:2302.11939*, 2023.

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

A. Appendix

A.1. Metrics

The metrics that are used for reporting results in this paper are:

- MAE (Godahewa et al., 2021)

$$\text{MAE}(\mathbf{y}_{L+1:L+H}, \hat{\mathbf{y}}_{L+1:L+H}) = \frac{1}{H}\|\mathbf{y}_{L+1:L+H} - \hat{\mathbf{y}}_{L+1:L+H}\|_1. \tag{6}$$

- msMAPE (Godahewa et al., 2021)

$$\text{msMAPE}(\mathbf{y}_{L+1:L+H}, \hat{\mathbf{y}}_{L+1:L+H}) = \frac{1}{H}\sum_{i=1}^{H}\frac{2|y_{L+i} - \hat{y}_{L+i}|}{\max\{|y_{L+i}| + |\hat{y}_{L+i}| + \epsilon, 0.5 + \epsilon\}}. \tag{7}$$

  In Monash benchmarks (Godahewa et al., 2021) $\epsilon = 0.1$ was used. This metric is used in order to avoid undefined values in other normalized metrics like MAPE. In multivariate datasets the metrics are calculated for each time-series and then we take the mean or the median. In this paper we only use the mean versions.

Aggregating across datasets. Since the datasets have wildly different scales averaging unnormalized metrics like MAE is not kosher. Therefore following (Gruver et al., 2023) we scale the metric of each baseline for a dataset by the same metric achieved by a naive baseline on that dataset. The naive baseline just makes the constant prediction y_L repeated across the prediction length. We did not need to do that for the Informer datasets since on these datasets metrics are usually reported on standard normalized data (Nie et al., 2022).

A.2. Finetuning study on ETT

In this section, we test whether TimesFM can be fintuned on a small fraction of a dataset to provide even better performance. We follow the same protocol as in GPT4TS (Zhou et al., 2023) (see Table 13 in their paper). (Zhou et al., 2023) finetune GPT2 input and output blocks on long-term forecasting benchmarks on 10% of the of the original datasets and compare it against models trained from scratch on the same data. Then the models are evaluated on the original test set task of (Zhou et al., 2021). We also tune the input and output residual blocks on 10% of the training set and present the results in Table 2. We can see that our model performs the best by a large margin. In ETTh1, ETTh2, ETTm1 our finetuned model is better than 18%, 3% and 12% better than GPT4TS, respectively. In fact we can see our 10% finetuned model's performances are comparable or better than that of most baselines trained on the whole training dataset as reported in Table 14 of (Zhou et al., 2023). This shows that the inductive biases encoded in our model weights by finetuning on a large time-series corpus are better for downstream forecasting task than an off the shelf language model like GPT2, even though our model is orders of magnitude smaller.

A.3. Pretraining PatchTST

Since TimesFM applies a similar patching strategy as PatchTST (Nie et al., 2022), for an ablation study we use the same data loader and pretrain a PatchTST model of 200M parameters to the same number of FLOPS as the final 200M TimesFM model. We denote it as PatchTST(ZS). The two models share the same hyperparameters of the transformer stack. For PatchTST(ZS) we use the same input patch length = 32, and a stride of length half of input patch size (i.e. stride = 16) as done in the original PatchTST paper.

We report the detailed results on Monash and ETT in Appendix A.4.2 and A.4.3. It can be seen that the results are not that good for PatchTST(ZS) on Monash. This is expected since our pretrain data loader will predominantly have context lengths of 512 instead of shorter context lengths as in Monash. Moreover the PatchTST model does fewer iterations at the same number of FLOPS. On ETT datasets, the PatchTST(ZS) model is performs similarly to TimesFM(ZS) and PatchTST. This is also expected since the context length for this study is indeed 512.

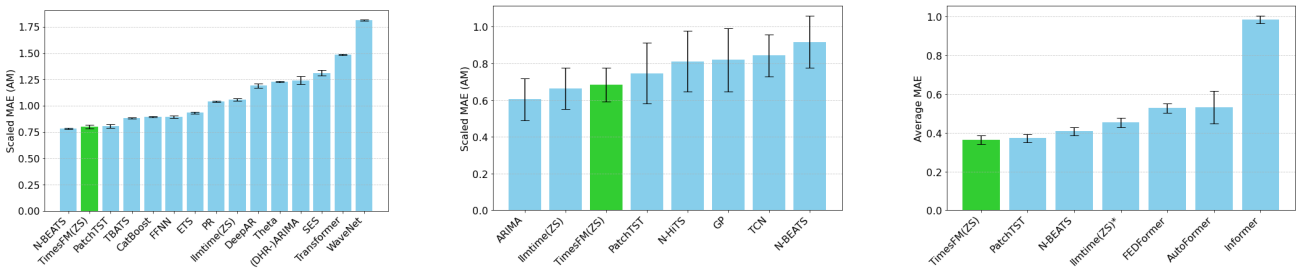
As PatchTST(ZS) is an encoder-decoder model, to pretrain it for zero shot forecasting one should theoretically prepare all possible context lengths and horizon lengths in the pretrain datasets. Pretraining it to its maximum performance requires much more compute and likely more careful tuning compared to pretraining TimesFM.

A.4. Additional Empirical Results

In this section, we provide more detailed tables for our zero-shot datasets and experiments described in Section 6.1. The AM based aggregated metrics are presented in Figure 7.

Table 2. MAE of different methods on ETT datasets. All methods use 10% of the original training set for training or finetuning. The baseline numbers are from Table 13 in (Zhou et al., 2021).

Dataset		TimesFM(FT)	GPT4TS(FT)	DLinear	PatchTST	TimeNet	FEDFormer	Autoformer
ETTh1	96	0.398	0.456	0.495	0.485	0.628	0.499	0.552
	192	0.424	0.516	0.538	0.524	0.593	0.555	0.598
	336	0.436	0.535	0.622	0.550	0.648	0.574	0.619
	720	0.445	0.591	0.743	0.610	0.641	0.614	0.616
	Avg	**0.426**	0.525	0.600	0.542	0.628	0.561	0.596
ETTh2	96	0.356	0.374	0.411	0.389	0.409	0.416	0.451
	192	0.400	0.411	0.519	0.414	0.467	0.474	0.477
	336	0.428	0.433	0.572	0.441	0.494	0.501	0.543
	720	0.457	0.464	0.648	0.480	0.491	0.509	0.523
	Avg	**0.410**	0.421	0.538	0.431	0.465	0.475	0.499
ETTm1	96	0.345	0.404	0.392	0.419	0.501	0.518	0.614
	192	0.374	0.423	0.412	0.434	0.528	0.546	0.592
	336	0.397	0.439	0.434	0.454	0.568	0.775	0.677
	720	0.436	0.498	0.477	0.556	0.549	0.579	0.630
	Avg	**0.388**	0.441	0.429	0.466	0.537	0.605	0.628
ETTm2	96	0.263	0.269	0.303	0.274	0.285	0.399	0.454
	192	0.309	0.309	0.345	0.317	0.323	0.379	0.691
	336	0.349	0.346	0.385	0.353	0.353	0.559	1.407
	720	0.415	0.417	0.440	0.427	0.449	0.614	1.166
	Avg	**0.334**	0.335	0.368	0.343	0.353	0.488	0.930



(a) Monash Archive (Godahewa et al., 2021) (b) Darts (Herzen et al., 2022) (c) ETT (Horizons 96 and 192) (Zhou et al., 2021)

Figure 7. We report average performance in three groups of datasets. In all figures, the lower the metric the better and the error bars represent one standard error. Note that among the baselines only TimesFM and llmtime are zero-shot. In (a) we report results on the Monash datasets. Since the datasets have different scales, we take the Arithmetic Mean (AM) the MAE's scaled by the MAE of a naive baseline. We can see that TimesFM is within significance of the top model N-BEATS. In (b), we report the similarly scaled MAE on the Darts benchmarks. TimesFM is within significance of the top of method which is ARIMA in this case. Note that these datasets have one time-series each and therefore statistical methods are competitive with deep learning ones. Finally, in (c) we report the average MAE for 96 and 192 horizon prediction tasks on 4 ETT datasets i.e 8 tasks in total. TimesFM and PatchTST are the best performing models in this case.

A.4.1. Darts

We present the MAE results individually from all 8 datasets in Table 3. It can be seen that TimesFM performs well for all datasets with clear seasonal patterns. On an average we are within significant level of the best model. Note that there are only 8 time-series as a whole in Darts and theerfore these evaluations have very wide confidence intervals.

In Figure 11 we present visual comparisons of our forecasts vs some of the baselines.

A.4.2. Monash

In Table 4 we present the actual MAE numbers that are behind the main Figure 2a. In Figure 12, we present some examples of our zero-shot forecasts. For most datasets, we set the context window to be the maximum length of the series in the

Table 3. MAE for Darts datasets. We also include the naive baseline that predicts the last values in the context repeatedly.

	GP	ARIMA	TCN	N-BEATS	N-HiTS	llmtime(ZS)	TimesFM(ZS)	PatchTST	NAIVE
AirPassengersDataset	34.67	24.03	54.96	97.89	59.16	34.37	62.51	44.65	81.45
AusBeerDataset	102.05	17.13	30.90	10.39	34.23	16.13	11.94	21.97	96.35
GasRateCO2Dataset	2.27	2.37	2.64	2.63	3.85	3.50	2.50	2.67	2.29
MonthlyMilkDataset	30.33	37.19	70.86	33.64	32.73	9.68	28.09	42.60	85.71
SunspotsDataset	53.74	43.56	51.82	73.15	49.93	47.34	41.40	62.33	48.24
WineDataset	4552.06	2306.70	3287.14	4562.02	3909.51	1569.32	2871.33	2498.69	4075.28
WoolyDataset	649.98	588.78	1158.79	903.01	382.09	808.73	728.92	542.28	1210.33
HeartRateDataset	5.65	5.56	5.49	6.57	6.10	6.21	5.85	6.74	5.92
Scaled MAE (Arithmetic Mean)	0.8193	0.6045	0.8427	0.9176	0.8109	0.6641	0.6829	0.7462	1.0000
Scaled MAE (Geometric Mean)	0.7509	0.5219	0.7946	0.7316	0.6936	0.4882	0.5767	0.6458	1.0000

Table 4. We present the mean MAE results for our methods along size Monash baselines. We also include the naive baseline that predicts the last values in the context repeatedly.

Dataset	llmtime(ZS)	SES	Theta	TBATS	ETS	(DHR-)ARIMA	PR	CatBoost	FFNN	DeepAR	N-BEATS	WaveNet	Transformer	PatchTST(ZS)	TimesFM(ZS)	NAIVE
australian electricity demand	459.96	659.60	665.04	370.74	1282.99	1045.92	247.18	241.77	258.76	302.41	213.83	227.50	231.45	382.23	448.81	659.60
bitcoin	1.75e18	5.33e18	5.33e18	9.90e17	1.10e18	3.62e18	6.66e17	1.93e18	1.45e18	1.95e18	1.06e18	2.46e18	2.61e18	1.11e18	1.3e18	7.77e17
pedestrian counts	70.20	170.87	170.94	222.38	216.50	635.16	44.18	43.41	46.41	44.78	66.84	46.46	47.29	51.27	40.71	170.88
weather	2.32	2.24	2.51	2.30	2.35	2.45	8.17	2.51	2.09	2.02	2.34	2.29	2.03	2.07	2.07	2.36
nn5 daily	9.39	6.63	3.80	3.70	3.72	4.41	5.47	4.22	4.06	3.94	4.92	3.97	4.16	3.77	3.54	8.26
nn5 weekly	15.91	15.66	15.30	14.98	15.70	15.38	14.94	15.29	15.02	14.69	14.19	19.34	20.34	17.00	14.67	16.71
tourism yearly	140081.78	95579.23	90653.60	94121.08	94818.89	95033.24	82682.97	79567.22	79593.22	71471.29	70951.80	69905.47	74316.52	224411.89	109977.29	99456.05
tourism quarterly	14121.09	15014.19	7656.49	9972.42	8925.52	10475.47	9092.58	10267.97	8981.04	9511.37	8640.56	9137.12	9521.67	21276.98	12102.04	15845.10
tourism monthly	4724.94	5302.10	2069.96	2940.08	2004.51	2536.77	2187.28	2537.04	2022.21	1871.69	2003.02	2095.13	2146.98	4596.21	3183.77	5636.83
cif 2016	715086.33	581875.97	714818.58	855578.40	642421.42	469059.49	563205.57	603551.30	1495923.44	3200418.00	679034.80	5998224.62	4057973.04	8374813.14	773980.44	386526.37
covid deaths	304.68	353.71	321.32	96.29	85.59	85.77	347.98	475.15	144.14	201.98	158.81	1049.48	408.66	348.60	209.80	353.71
fred md	2013.49	2798.22	3492.84	1989.97	2041.42	2957.11	8921.94	2475.68	2339.57	4264.36	2557.80	2508.40	4666.04	4965.51	947.12	2825.67
traffic hourly	0.03	0.03	0.03	0.04	0.03	0.04	0.02	0.02	0.01	0.01	0.02	0.02	0.01	0.01	0.01	0.03
traffic weekly	1.17	1.12	1.13	1.17	1.14	1.22	1.13	1.17	1.15	1.18	1.11	1.20	1.42	1.23	1.12	1.19
saugeenday	28.63	21.50	21.49	22.26	30.69	22.38	25.24	21.28	22.98	23.51	27.92	22.17	28.06	22.33	24.63	21.50
us births	459.43	1192.20	586.93	399.00	419.73	526.33	574.93	441.70	557.87	424.93	422.00	504.40	452.87	1193.28	437.27	1152.67
hospital	24.62	21.76	18.54	17.43	17.97	19.60	19.24	19.17	22.86	18.25	20.18	19.35	36.19	20.87	19.41	24.07
solar weekly	2049.09	1202.39	1210.83	908.65	1131.01	839.88	1044.98	1513.49	1050.84	721.59	1172.64	1996.89	576.35	1093.46	1258.27	1729.41
Scaled MAE (Arithmetic Mean)	1.0588	1.3115	1.2295	0.8824	0.9337	1.2427	1.0382	0.8924	0.8942	1.1925	0.7844	1.8119	1.4840	2.1419	0.8005	1.0000
Scaled MAE (Geometric Mean)	0.9715	1.0855	0.9371	0.7736	0.8104	0.9449	0.8218	0.7733	0.7044	0.7477	0.7005	0.9384	0.8619	1.0557	0.6846	1.0000

dataset capped at 512 (similar to statistcal models used in the official Monash baselines). For some datasets, we did some inference time-tuning of the context length, i.e we predict the last horizon length number of points in the training set with context lengths 32, 64 and maximum allowed and chose the best one in terms of this validation metric. This is fair as most Monash DL baselines use different context lengths for different datasets during training and our model is completely zero-shot. The max context lengths used for these datasets are (cif 2016, 32), (tourism yearly, 32), (covid deaths, 32), (bitcoin, 32), (tourism monthly, 32) and (tourism monthly, 64).

A.4.3. INFORMER

We present the MAE on the last split of the test set for all dataset, horizon pairs considered in Table 5. Owing to expensive evaluations for llmtime, the results are reported on the last test window of the original test split, as done in (Gruver et al., 2023).

Table 5. MAE for ETT datasets for prediction horizons 96 and 192. Owing to expensive evaluations for llmtime, the results are reported on the last test window of the original test split.

Dataset	llmtime(ZS)*	PatchTST	PatchTST(ZS)	FEDFormer	AutoFormer	Informer	TimesFM(ZS)
ETTh1 (horizon=96)	0.42	0.41	0.39	0.58	0.55	0.76	0.45
ETTh1 (horizon=192)	0.50	0.49	0.50	0.64	0.64	0.78	0.53
ETTh2 (horizon=96)	0.33	0.28	0.37	0.67	0.65	1.94	0.35
ETTh2 (horizon=192)	0.70	0.68	0.59	0.82	0.82	2.02	0.62
ETTm1 (horizon=96)	0.37	0.33	0.24	0.41	0.54	0.71	0.19
ETTm1 (horizon=192)	0.71	0.31	0.26	0.49	0.46	0.68	0.26
ETTm2 (horizon=96)	0.29	0.23	0.22	0.36	0.29	0.48	0.24
ETTm2 (horizon=192)	0.31	0.25	0.22	0.25	0.30	0.51	0.27
Avg	0.45	0.37	0.35	0.53	0.53	0.99	0.36

A.5. More Details on Models

We now present implementation details about TimesFM and other baselines.

TimesFM. For our main 200M model we 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. We train with layer norm and a cosine decay learning rate schedule with peak learning rate of $5e-4$. The hyper-parameters of TimesFM for various sizes are provided in Table 6. Note that the settings are for the base models and not ablation models. The hidden dims of both the residual block and the FFN in the transformer layers are set as the same as model dimensions. We keep layer norm in transformer layers but not in the residual blocks.

Table 6. Hyper-parameters for TimesFM

Size	num_layers	model_dims	output_patch_len	input_patch_len	num_heads	dropout
200M	20	1280	128	32	16	0.2
70M	10	1024	128	32	16	0.2
17M	10	512	128	32	16	0.2

Monash Baselines. The raw metrics for the Monash baselines are directly taken from Tables 9 and 11 of the supplementary material of the original paper (Godahewa et al., 2021). For llmtime, we use the precomputed outputs provided by the authors of (Gruver et al., 2023).

Darts Baselines. For all the Darts baselines we use the precomputed outputs provided by the authors of (Gruver et al., 2023). For more details please see Section C.1 in that paper.

Informer Baselines. For FEDFormer (Zhou et al., 2022), Autoformer (Wu et al., 2021), Informer (Zhou et al., 2021) and PatchTST (Nie et al., 2022) we use the original hyperparameters and implementation. The results presented in the main paper are obtained on the last test window of length horizon length as stated in the llmtime (Gruver et al., 2023) paper.

We generate the llmtime predictions using the code provided by the authors [7] but adapted to the ETT datasets. Note that as of January 2024, OpenAI has discontinued access to GPT-3, therefore we had to use the GPT-3.5-Turbo model.

A.6. Date Features

As we mentioned earlier, since we are building a single pre-trained 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 technically have date derived features like day of the week, month of the year etc processed into a vector at each time-point t, denoted by $\mathbf{x}_t \in \mathbb{R}^r$.

If so, the learning task can be rewritten as

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

There are many options to incorporate these features into the model, one being to directly concatenate them after the time-points in each patch. For this paper we decide to focus on the univariate time-series input, and will investigate this enhancement in the future.

A.7. Synthetic Data

We create the synthetic data to reflect common time-series patterns using traditional statistical models. We start with four simple times series patterns:

- Piece-wise linear trends (I), where the number of the piece-wise linear components is randomly chosen between 2 and 8.
- ARMA(p, q) (II), where $1 \leq p, q \leq 8$ and the corresponding coefficients are generated from either a multivariate Gaussian or a uniform, then normalized.

[7] https://github.com/ngruver/llmtime/blob/main/experiments/run_monash.py

- Seasonal patterns. In particular we create the sine (III) and the cosine (IV) waves of different random periods between 4 and max context length / 2 time-points and time delays.

We then randomly enable / disable these four components (I) - (IV), generate their time-series of length 2048 respectively, and sum them up using uniformly sampled random weights to create each times series in the synthetic datasets. We also choose to apply the trend multiplicatively 50% of the times the trend component is chosen.

A.8. Illustrative Examples

We conduct a visual inspection of the forecasts generated by TimesFM, first on some synthetic examples and then on the benchmark datasets.

In Figure 8 we show 4 different synthetic curves: (1) sum of 5 sine curves of different periods, (2) a sine curve linearly scaled, (3) a sine curve with a linear trend, and (4) minimum of two sine curves with a linear trend. Our results suggests that TimesFM picks up the trend and seasonal components readily interpretable by humans, while ARIMA and (to a lesser extent) llmtime fail in some of the instances.

As illustrated in Figure 9, TimesFM also effectively captures these subtle characteristics within both the trend and seasonal patterns of the depicted real world time-series. For instance, in the Air Passenger dataset, TimesFM correctly captures the amplitude increase with trend –this is also reflected by the fact that it attains the best MAE on this dataset (see Table 3). In the traffic hourly example on the left, it can be seen that TimesFM can correctly identify the seasonal peaks even in the presence of outliers in the context, while llmtime is thrown off.

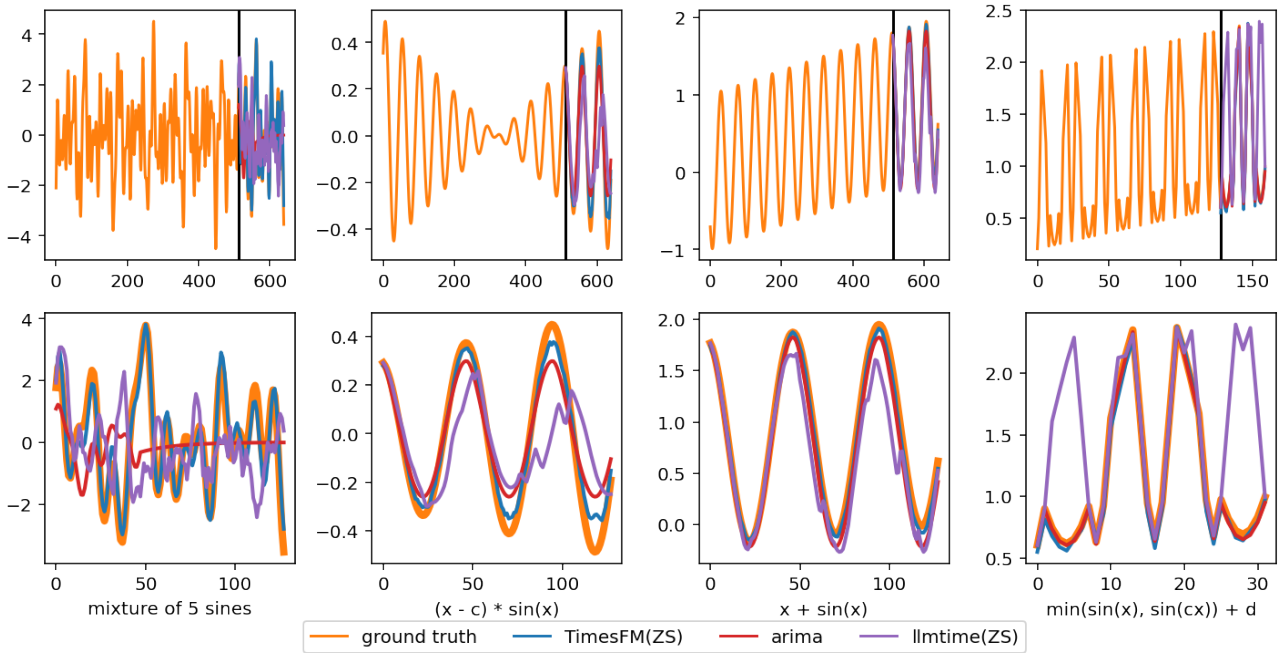


Figure 8. Forecasts visualized on synthetic curves. The bottom row plots zoom in on the prediction horizon for the sake of clarity.

We provide more visualization in Figure 10, Figure 11 and Figure 12.

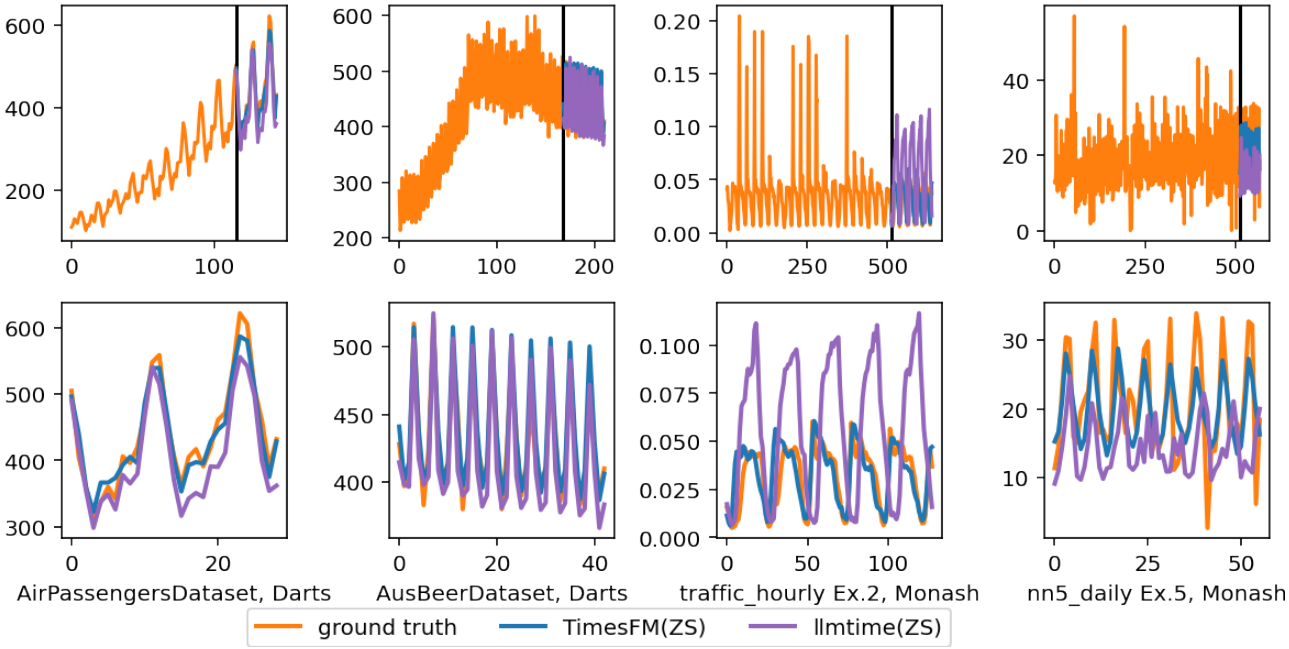


Figure 9. Forecasts visualized on Darts and Monash. The bottom row plots zoom in on the prediction horizon for the sake of clarity.

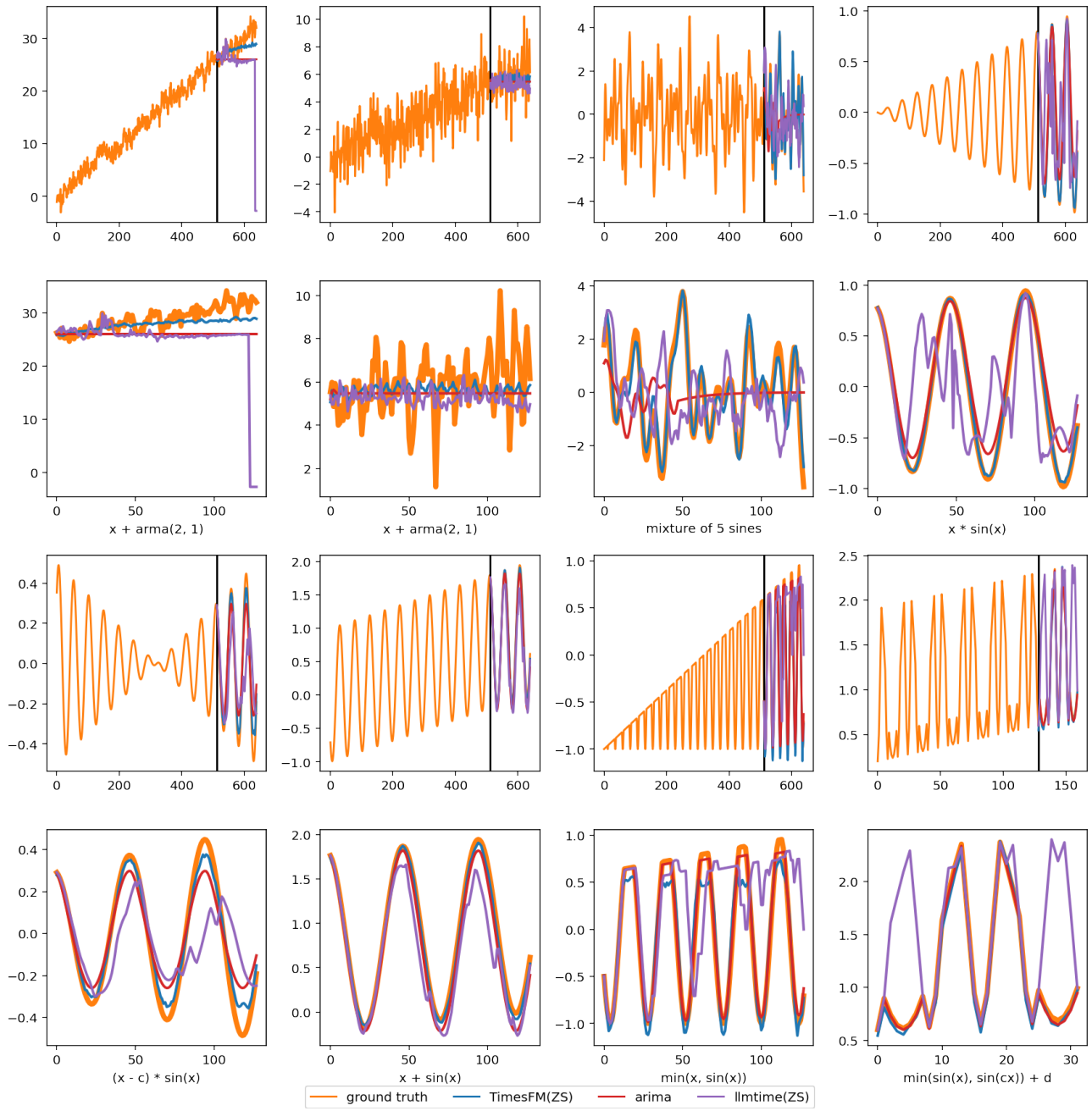


Figure 10. Forecasts visualized on synthetic curves. The second row plots zoom in on the prediction horizon for the sake of clarity.

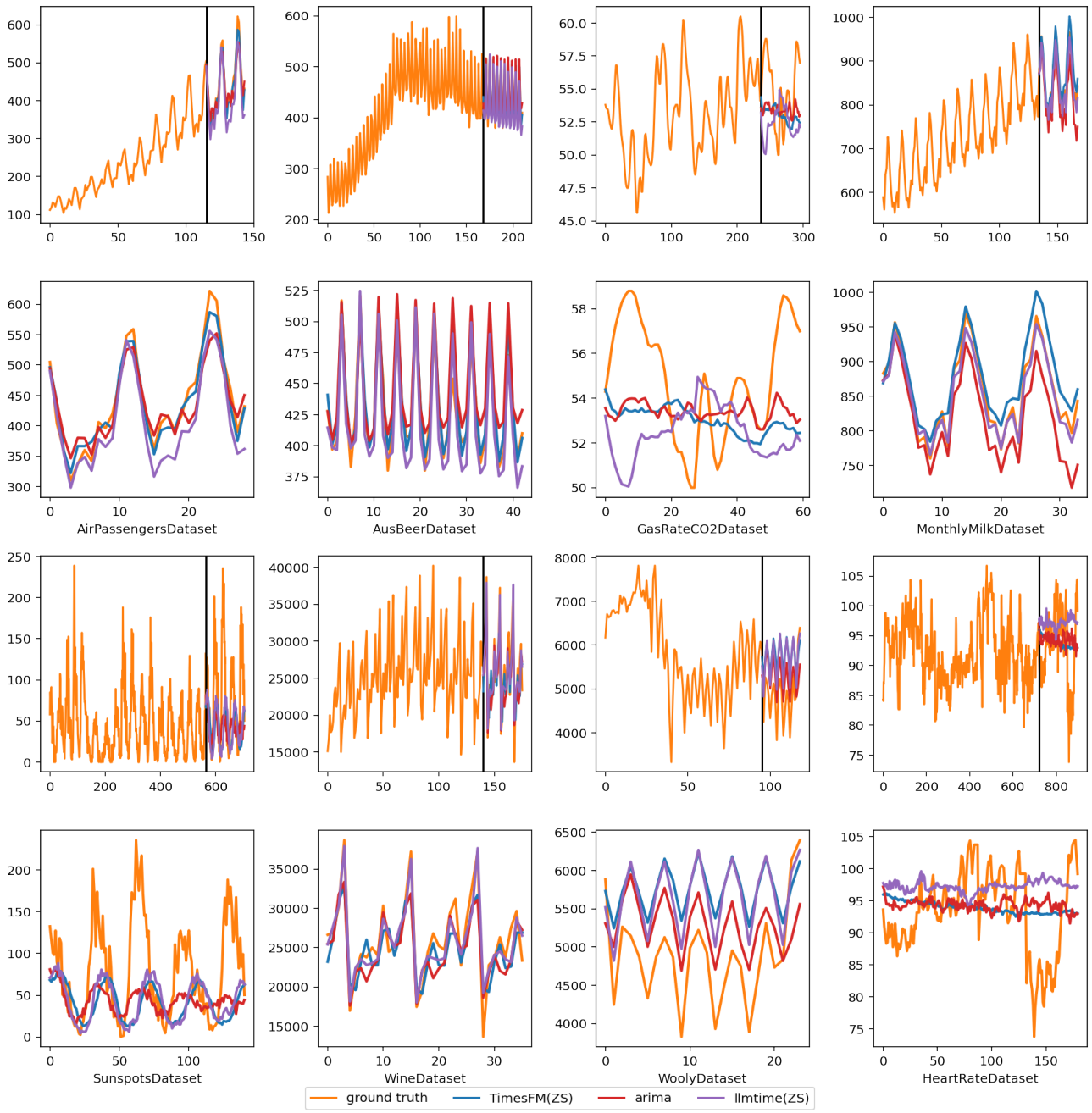


Figure 11. Forecasts visualized on all Darts datasets. The second row plots zoom in on the prediction horizon for the sake of clarity.

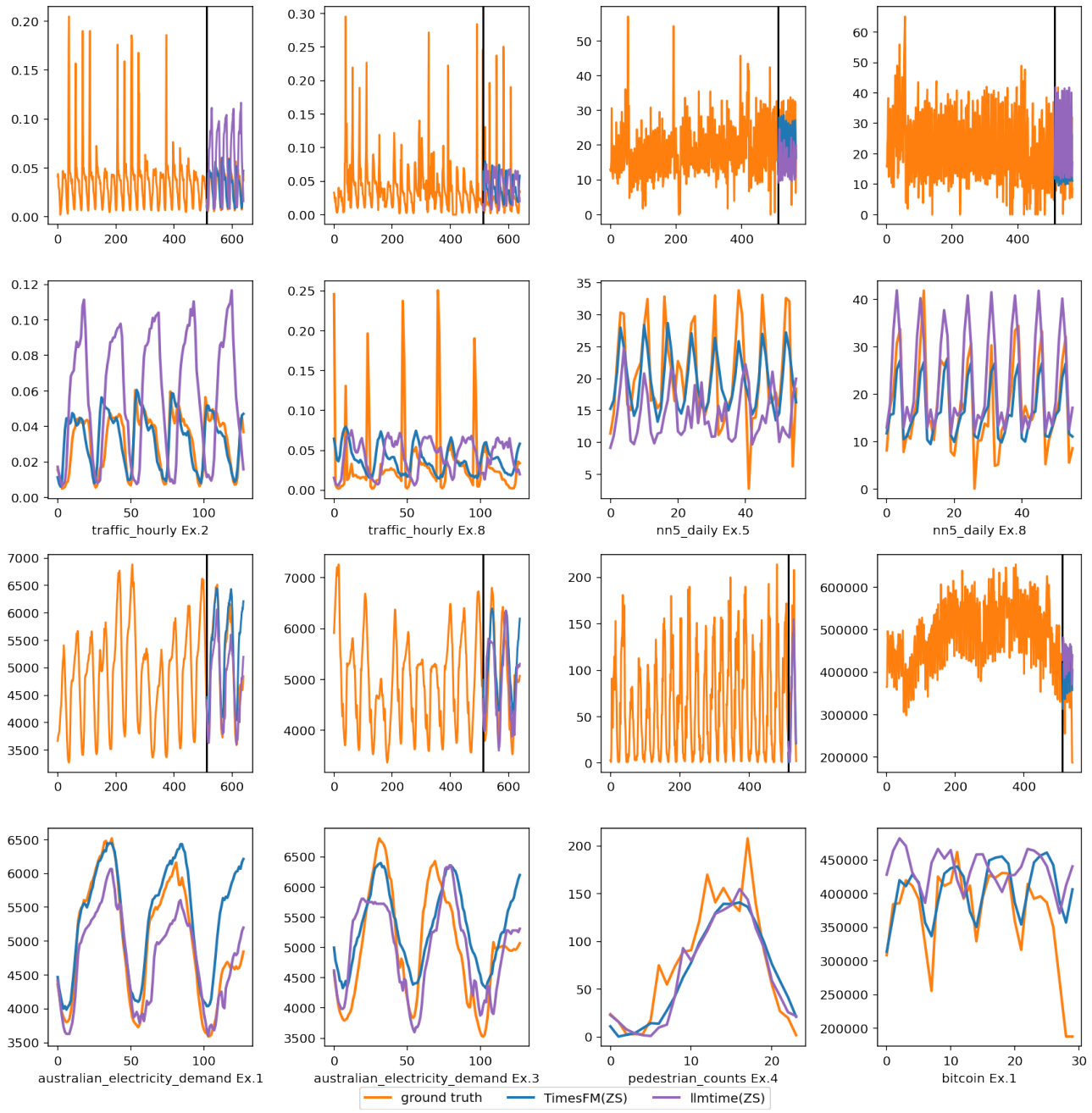


Figure 12. Forecasts visualized on a few Monash datasets. The second row plots zoom in on the prediction horizon for the sake of clarity.