Introduction

Following on from previous work I have benchmarked the models listed in table 1 using a methodology that is consistent to the original benchmarks allowing for direct comparison of the results. The models were selected based: on how recently they have been introduced, their use as benchmarks in recent research, their adoption into popular time series libraries such as GluonTS, Darts and Nixtla's NeuralForecast and the diversity of the model architecture.

N-BEATS	Global univariate	MLP	https://arxiv.org/pdf/1905.10437
N-HITS	Global univariate	MLP	https://arxiv.org/pdf/2201.12886
TiDE	Global univariate	MLP	https://arxiv.org/pdf/2304.08424
NLinear	Multivariate	Linear	https://arxiv.org/pdf/2205.13504
DLinear	Multivariate	Linear	https://arxiv.org/pdf/2205.13504
Autoformer	Multivariate	Transformer	https://arxiv.org/pdf/2106.13008
PatchTST	Multivariate	Transformer	https://arxiv.org/pdf/2211.14730
TimesFM	Foundational	Transformer	https://arxiv.org/pdf/2310.10688v2

Table 1. The models benchmarked in this study

The objectives were to evaluate recent models which are considered to have state of the art performance in order to:

1. Compare the performance to older neural network architectures and classical statistical modelling approaches.

2. Develop a better understanding of how more recent models perform on a more diverse range of time series than is typically measured in recent research which tends to focus more on long horizon multivariate scenarios.

Experiment Setup

I have aimed to faithfully reproduce the experimental setup of previous benchmark experiments whilst not having to rely on the GluonTS library allowing these benchmarks to be fairly compared to previous work. Note that TimesFM, being a foundational model, was evaluated on zero short performance and therefore no model training was required and consequently the experimental setup described in this section does not apply.

The benchmarking has been completed using my NNTS github repository, which I have developed for performing experiments with time series models. The models benchmarked were created in this repo with PyTorch using, where available the, the source code of the official implementation of the models. This setup allows us to more direct control over the hyperparameters, the training regime and the data sampling strategy which means that for example we can sample data using a DeepAR/ GluonTs type approach and an Informer/Multivariate approach in the one code base.

With the exception of Autoformer and PatchTST all models were trained 5 times with different random seed values and the mean of the metrics for each trained model are reported. In total I have trained and logged the results of more than 3000 models.

Where possible I have retained the same configuration and hyperparameters as the previously benchmarked models:

The following values are the same as the previously benchmarked models:

1. Forecast horizon
2. Context length (Lookback window)
3. Global model data sampling strategy.
4. Train / Test data splitting with no validation set and models are selected based on the best training loss.
5. Batch size
6. Learning rate scheduler ReduceLROnPlateau with matching patience
7. Adam optimiser
8. Performance error metrics (ie mean and median MASE, sMAPE, msMAPE, RMSE, MAE)

Changes to the Experimental Setup

As I have not used GluonTS there are some changes to the setup which I describe as follows:

Loss Function

The models benchmarked previously used differing loss functions, in part this was necessary as models such as DeepAR and WaveNet produce distributional outputs and consequently use NLL as loss function, however N-BEATS which is a point forecast model would have used a loss function based on MASE and sMAPE. As all of the models I have benchmarked produce point forecasts I believe that using a common loss function for all the models allows for the fairest comparison to be made. The loss functions defined in the literature for each model vary. The multivariate models and TiDE use MSE; N-HITS uses MAE. For this study all models were benchmarked using MAE.

Context length

GluonTS's implementation of DeepAR and Transformer include a lag sequence feature which, act in addition to the context window of historical values, and specify a set of indices of historical values to include as additional features. Consequently the receptive field is extended beyond the defined context window, theoretically giving these models an advantage as they have access to additional information.

To allow for a fair comparison these benchmarks have been performed using 3 sets of context lengths the details of which are described in the appendix. Using different context lengths affects both performance and the computational demands on training since the size of the network is in part a function of the context length. Unless otherwise stated the reported results will be from models trained with context lengths matching the previous work.

Data Sampling

The existing benchmarks made on global univariate models (eg DeepAR, Wavenet and N-BEATS) were completed using GluonTS which uses a data sampling strategy described by Salinas et al., 2020.. Samples are windows of historic observations and are drawn from the entire time-series excluding the last forecast horizon which is reserved for backtesting. More uniquely it uses a weighted sampling scheme to draw windows of observations from a time-series in the dataset such that the observations are equally likely to be drawn from a time-series regardless of the length of the time-series. I have implemented a similar system to sample data for the global univariate models, the key difference between the two systems is that my sampling strategy will select a random window sample from a time-series, whereas the GluonTS scheme will stochastically sample from a given time-series where the expected number of samples drawn is 1. Initial experiments with DeepAR indicate that both approaches are equivalent in performance.

For multivariate models the data sampling scheme is necessarily different due to the input which requires each sample to consist of a sequence of historical observations for each time-series in the dataset. Consequently the input of a single example takes the form of L x C dimension matrix where L is the Context Length and C is the number of time-series in the dataset, commonly referred to in the literature as the number of channels. The official implementation of the multivariate models benchmarked all use the same code for sampling and therefore share a common sampling scheme. I have implemented the same scheme but using logic that integrates better with the nnts codebase and have verified that samples are identical between the two mechanisms for a given random seed.

Data Processing

To be consistent with previous benchmarks the raw data values are used as inputs into each model. This is unlike the experimental setup used by the authors of the papers where global normalisation is applied to scale the input data into the model to produce their published results.

I do not add any additional time-dependent or time-independent (static) features. This differs from GluonTS which by default will add some temporal features (eg day of the month) and a static feature for the unique identifier for each time-series to certain models such as DeepAR and Wavenet. As discussed, GluonTS will also add lag features (not to be confused with the sample window of historical values set by the context length) to DeepAR and the transformer models.

This decision has implications for models such as TiDE and Autoformer that can readily accept covariates, it is my belief that providing the same data to all models firstly provides the fairest way to make a comparison and secondly serves as a useful baseline for any future research that may wish to evaluate the efficacy of incorporating such features.

Hyperparameters

Table 2 details the common set of hyperparameters that were used in the training of all the models benchmarked.

Epochs	100
Batch Size	32
Loss Function	MAE
Optimiser	Adam
Weight Decay	0.
Scheduler	Reduce LR on Plateau
Scheduler Patience	10
Early Stopper Patience	30
Batches Per Epoch	50

Table 2: Common hyperparameters for all models

All configuration and settings have been logged to WandB.

Models

In total 8 model architectures (3 global univariate, 4 multivariate and 1 foundational model) were benchmarked. Univariate models generate forecasts specific to one time-series and global means that a single model is trained on a dataset of multiple time-series and can therefore forecast any time series from that dataset.

Conversely, multivariate models are trained to forecast a common forecast horizon for all time-series in a dataset (ie the output of such a model is a H x C matrix where C is the number of time-series in the dataset (channels) and H is the forecast horizon. The implications of this are that firstly, these models require a different data sampling strategy from Univariate models as discussed previously and secondly, Multivariate models are only suitable for use with multivariate datasets which are datasets containing multiple time-series where all the time-series share a common time-frame. Datasets containing time-series of varying lengths are not suitable because shorter time-series will have missing values which would need to be handled by padding or truncating the dataset to a common time-frame. For these benchmarks I have only benchmarked multivariate models on the subset of datasets that are multivariate.

Global Univariate Models

Unless otherwise stated all models are trained with a learning rate of 1e-3 and with no dropout.

N-HITS

N-HITS is a fully connected model architecture which is a development of the N-BEATS model proposed by authors from Nixtla. The model was implemented based on the Nixtla's NeuralForecast code. Table 3 lists the model specific hyperparameters used.

n_blocks	[1, 1, 1]
mlp_units	[[512, 512], [512, 512]]
n_pool_kernel_size	[1, 1, 1]
n_freq_downsample	[1, 1, 1]

Table 3: N-HITS model specific hyperparameters

TiDE

TiDE is an MLP model with an architecture that utilises residual blocks and skip connections. The architecture is designed to incorporate time-dependent and time-independent covariates, which we do not use for these benchmarks in order to keep the input into each model consistent. Future work to explore the effectiveness of the model to utilise covariates could use these results as a baseline. Table 4 lists the model specific hyperparameters used.

hidden_size	256
num_encoder_layers	2
num_decoder_layers	2
decoder_output_dim	8
temporal_decoder_dim	128
output_dim	1
dropout	0.3

Table 4: TiDE model specific hyperparameters

N-BEATS

The original benchmarks include the results of the N-BEATS model trained on GluonTS. For this benchmark I have benchmarked N-BEATS using a model developed based on the source code from the author's original implementation. The GluonTS N-BEATS is trained using [MASE, sMAPE, MAPE] as the loss, whereas this benchmark trained the models using MAE. Table 5 lists the model specific hyperparameters used.

theta_size	32
num_stacks	30
num_layers	2
layer_size	512

Table 5: N-BEATS model specific hyperparameters

Multivariate Models

The LTSF-Linear and PatchTST are multivariate models but can be configured to run in what's referred to as a channel independent mode which is used to produce the univariate results detailed in the literature of these models. I originally assumed that it would be possible to use the channel independent mode of the multivariate models to run experiments on datasets that are not multivariate. The problem is that the architecture still expects a single training example to contain an input window from each series in the dataset, meaning that it can not accommodate datasets with multiple series whose date time ranges are not aligned. To get around this limitation all the multivariate papers conduct their univariate testing by simply using a single series (the "OT" feature) from the ETT dataset and report their results accordingly. None of the experiments detailed in the literature or the associated code appear to have performed any experiments using channel indepence on the full dataset using multiple time series.

Scaling becomes a problem when using channel independence for datasets containing a lot of series as the size of the model is, in part, a function of the number of series. DLinear and NLinear have a particular issue as there are no shared parameters between the series meaning that a dataset like Kaggle Web Traffic have ~23M parameters divided into 145,063 channels which results in 145,063 individual weight matrices resulting in an extremely computationally expensive computation. Consequently it was not possible to benchmark all the multivariate models when using channel independence because of resource constraints.

Autoformer + RevIn

Autoformer is a transformer based model architecture which followed various time-series transformer architectures such as Informer. The model hyperparameters were chosen based on the authors selected parameters of the ETTh script in the official source code report. The initial testing with the model has extremely poor performance which I suspected was likely to be caused by no scaling of the input. We are not performing global normalisation of the input as was the case with the experiments detailed in the paper. Consequently, I took the decision to implement the same Instance Normalisation as PatchTST which improved the performance and these are the figures reported in my results. Table 6 lists the model specific hyperparameters used.

d_model	512
n_heads	8
e_layers	2
d_layers	1
d_ff	2048
factor	1
Moving_avg (kernel size)	25
Embedding Type	Token Embedding + Position Embedding
Activation function	gelu

Table 6: Autoformer model specific hyperparameters

LTSF

LTSF is a family of models and I have benchmarked the two most commonly referenced models: DLinear and NLinear. They are simple 1-layer linear models developed as baselines for multivariate models and as such should be treated as multivariate models. Like PatchTST these models have a "channel independent" configuration which isolates the model parameters to a dedicated channel (time-series), however being 1-layer models there are no shared parameters at all, effectively meaning that each channel is a "local" model.

In addition to the multivariate configuration I thought it would be interesting to configure these models as Univariate so they could be trained against all the datasets. To achieve this the models were set to have one channel and then trained using our univariate data sampling strategy. Consequently the weights of the network are shared by all the series in the training set. The authors did not propose such a configuration and so I do not believe that this is something that they have considered.

The authors have an official implementation of the paper developed in pytorch, the source code of which was used as the basis for my implementation.

DLinear

DLinear uses decomposition to split the input sequence into trend and seasonal components using a moving average that is somewhat reminiscent of classical time-series decomposition. There is just one hyperparameter, kernel size, which defines the size of the moving average window used when decomposing the input into seasonal and trend signals. In the literature this is set to 25 which I have retained for the benchmarks. I would question whether this is optimal and would think that using the seasonality value would be more suitable, but I leave this for a future study.

Moving avg (kernel size)	25

Table 7: DLinear model specific hyperparameters

NLinear

NLinear is similar to DLinear, but instead of decomposing the input sequence a simple local scaling function is applied on the temporal dimension.

PatchTST

PatchTST is a transformer based model incorporating a segment based patching system which reshapes the temporal input sequence to increase the receptive field and an Instance Normalisation transformation known as Revin. There are some things worth noting concerning the source code provided by the authors. Firstly, they use One-Cycle scheduling in the training regime to dynamically vary the learning rate. This is in contrast to LTSF-Linear and Autoformer models which use a Stepped learning rate scheduler. To be consistent with the other experiments we have used a reduced learning rate on plateau schedulers.

The model hyperparameters were chosen based on the authors selected parameters of the univariate ETTh script.

The computational demands to train a PatchTST with channel independence are significantly greater than the non-transformer based model. For example, traffic hourly took 19 hours to train and the model has more than 60M parameters.

Foundational Models

TimesFM

TimesFM is a foundational model with a transformer based architecture incorporating patching of the input with an autoregressive output like many LLM's. TimesFM has been pretrained on publicly available time series datasets including, amongst others, the M4, Electricity and Traffic datasets all of which are used in these benchmarks. This should be taken into account when evaluating the results as data leakage on at least some of the datasets in the benchmarks would have occurred during the training of the model.

The benchmarking was performed using zero shot (i.e. no additional fine tuning) with the timesfm-1.0-200m version using the TimesFM python package which is described in the official github repo.

Most of the hyperparameters for the model are fixed and are described in Table 8.

Input patch length	32
Output patch length	128
Num layers	20
Model dims	1280
backend	cpu

Table 8 hyperparameters for TimesFM

In addition a context length needs to be specified for the TimesFM model. This is related to but not to be confused with the context length described in this document. For clarity I will refer to this as the "input context length" and it is a parameter that is a parameter required by the model that must be a multiple of the input patch length and be longer than the "context length". The manuscript recommends setting this to be as small as possible in order to maximise performance and therefore I used a value that was the smallest multiple of the input patch length that was larger than the context length.

Results

Model Performance

Comparing performance of the models in this study

A summary of the MASE performance for the models is detailed in table B of the Appendix. N-HITS and N-BEATS (which has been previously benchmarked) perform best on 27 of the 41 datasets whilst TiDE has the lowest MASE on 1 dataset. This may be an indication that TiDE requires additional covariates to perform optimally.

TimesFM is somewhat difficult to evaluate as the precise details of the training data used is not publicly available. It is however stated that all M4, Traffic Hourly and Electricity Hourly dataset were used. For this study we only consider the 18 datasets reported in the TimesFM own benchmarking. Interestingly TimesFM performs best on 6 of these datasets.

Of the 15 multivariate datasets, one (Kaggle Web Traffic) was not possible to benchmark on any multivariate model because the resource demands were too great and an additional 3 (Solar 10 mins, Traffic Hourly and Temp Rain) were benchmarked with a subset of the multivariate models and the univariate models. A multivariate model performed best in 5 datasets (2 PatchTST, 2 DLinear and 1 NLinear). Autoformer was the only model not to perform best on any dataset.

Comparing performance to previously benchmarked models

The models in this study improved upon previously benchmarked MASE in 8 datasets with N-HITS scoring best in 4. Table 9 details the improvements to the existing benchmarked results.

Dataset	**Previous Best MASE (model)**	**New best MASE (model)**
Covid	5.326 (ETS)	5.176 (NLinear multivariate)
Dominick	0.531 (Wavenet & Transformer)	0.507 (N-HITS)
Kaggle Weekly	0.622 (TBATS)	0.582 (N-HITS)
M4 Daily	1.141 (FFNN)	1.125 (N-BEATS)
Pedestrians	0.247 (Wavenet)	0.241 (N-HITS)
Traffic Weekly	1.084 (Prophet)	1.040 (TimesFM)

US Births	1.453 (TBATS)	1.438 (N-BEATS)
Weather	0.631 (DeepAR)	0.621 (N-HITS & TimesFM)

Table 9 Comparison of best MASE error for previously benchmarked results with the results from this study in datasets where there has been an overall improvement.

All configuration and settings have been logged to WandB.
Results are also summarised in Google Sheets.

Comparison of LTSF configurations (independent, multivariate, global and local)

Whilst testing the independent mode of the LTSF-Linear models it became clear that dedicating channels to serve specific time-series (ie no sharing of parameters between series in the dataset) effectively resulted in a collection of local models stored in a single artefact. I thought that in this case the training regime employed could be detrimental to performance, as data from all time-series is being sampled, and then the loss measured and optimisation occurs for data from all time series. To understand if this was the case I ran a series of experiments which trained individual models for each time series in a dataset in order to compare the results. Since the metrics average the loss per time series in the benchmarks it is possible to make a direct comparison.

The DLinear MASE of these experiments are shown in Table 9 alongside the other configurations: Independent, Multivariate, and Global. The best error for each data set is shown in bold and the second best in underlined. Unsurprisingly the Local and Independent results do appear to be closely related, however there also seems to be an association between the Multivariate and the Global results. For example the two best performing configurations for Car parts are Multivariate and Global both with a MASE of ~0.75 and Local and Independent have similar errors of ~1.3 and ~1.4 respectively. NLinear has the same pattern of behaviour. Clearly the characteristics of certain datasets tend to benefit from sharing information between time-series. These results seem to suggest some evidence that for these Linear models information can be shared using univariate or multivariate methods to a similar effect. Given the limitations of multivariate models it would be interesting to see if this observation holds for other multivariate models.

model	**Local**	**Independent**	**Multivariate**	**Global**
Car parts	1.271	1.403	<u>0.752</u>	**0.747**
Covid deaths	9.354	9.200	**5.601**	<u>5.974</u>
Electricity hourly	<u>1.881</u>	1.883	**1.880**	2.012
Electricity weekly	1.049	1.096	**0.780**	<u>0.827</u>
Fred md	**0.512**	<u>0.548</u>	0.627	0.735
hospital	0.909	0.931	**0.800**	<u>0.808</u>

Nn5 daily	**0.957**	0.960	0.961	0.958
Nn5 weekly	1.060	1.126	0.879	**0.862**
rideshare	4.603	4.580	4.314	**4.114**
Solar 10 minutes	1.788	1.793	**1.611**	1.620
Solar weekly	1.116	**1.102**	1.157	1.152
Traffic hourly	0.965	0.968	0.923	**0.918**
Traffic weekly	1.454	1.487	**1.096**	1.130

Table 10 DLinear MASE of Local, Global Univariate, Channel Independent Multivariate and Multivariate configurations. Bold denotes lowest loss. Underlined is the second lowest loss.

Comparison of context length

It would seem reasonable to assume that as the context length increases there should be a corresponding improvement in performance, however our results suggest that this is not always the case. Table 11 details the MASE for the DLinear, N-HITS and NLinear models on a selection of datasets with varying context lengths. Interestingly in a number of datasets: Car Parts, NN5 Daily, Rideshare, Solar Weekly, Traffic Hourly the MASE is actually larger with the longest context lengths. Additionally the models do not always respond to the change in context length in the same way, for example, DLinear and NLinear on the Electricity Hourly dataset see a reduction in error as the context length increases and conversely N-HITS MASE increases with a longer context length. The reasons behind this I will leave for a future investigation, but one possible explanation in some instances could be that the increase in context length reduces the number of training samples available. In the extreme case where the context length and the forecast horizon are as long as the time-series there would be only one training example available per time-series in the dataset. In these scenarios the models are likely to be overfitting on the training set.

dataset	**context length**	**DLinear independent**	**N-HITS**	**NLinear independent**
Car parts	15	1.404	**0.748**	1.397
	24	1.500	0.750	1.572
Covid deaths	9	9.200	7.362	6.287
	38	11.897	7.648	7.135
	60	14.060	**6.260**	7.927
Electricity hourly	30	1.883	2.073	1.891
	210	1.599	2.304	1.647
	336	1.606	2.627	**1.594**
Electricity weekly	16	1.446	1.510	1.412
	65	1.101	**0.986**	0.998
Fred md	15	0.547	0.660	0.499

	24	0.551	0.698	**0.493**
hospital	15	0.929	0.802	0.856
	24	1.045	**0.797**	0.898
Nn5 daily	9	0.960	0.908	0.957
	70	0.897	**0.833**	0.892
	112	1.035	0.869	1.036
Nn5 weekly	16	1.043	0.869	1.015
	65	1.120	**0.864**	1.074
rideshare	210	4.579	**4.041**	4.486
	336	4.645	4.065	4.573
Saugeen river flow	9	1.602	1.668	1.443
	38	1.776	1.647	1.446
	60	1.646	1.633	**1.442**
Solar 10 minutes	50	1.793	2.406	1.781
	2016	1.280	1.198	1.179
Solar weekly	6	1.102	1.384	1.312
	10	**0.970**	1.082	1.327
	65	1.260	2.478	2.626
sunspot	9	0.118	0.166	0.083
	38	0.080	0.048	0.067
	60	0.062	**0.036**	0.059
Traffic hourly	30	0.968	**0.850**	0.926
	210	1.102	0.998	1.084
	336	1.138	1.069	1.133
Traffic weekly	16	1.209	1.144	1.182
	65	1.487	**1.083**	1.355
us_births	9	2.166	1.510	2.145
	38	1.549	1.565	1.569
	60	1.578	**1.509**	1.569

Table 11 MASE for the DLinear, N-HITS and NLinear models on a selection of datasets with varying context lengths

Execution times

All model training was conducted using an Apple M3 Pro CPU processor with 18Gb Memory.

Model	Training Time (s)
Autoformer	50,544.87
DLinear independent	3,538.82
DLinear multivariate	746.87
N-BEATS	7,245.35
N-HITS	1,129.08
NLinear independent	2,336.44
NLinear multivariate	295.09
PatchTST independent	5,200.37
PatchTST multivariate	4,841.04
TiDE	1,265.03

Table 12 - Total mean time to train 15 datasets common to all models.

Appendix

Context Length Modes

The benchmarks have been conducted using 3 schemes for context lengths. The method of calculating the context length for each mode is as follows:

0: The context lengths used in the original benchmarks.
1: A heuristic where the context length = 2 * forecast horizon
2: A heuristic where the context length = 1.25 * min(forecast horizon, seasonality)

Table A details the specific context length and forecast horizon for each mode. The benchmark results are from context mode 0 (i.e. the same context lengths as in previous benchmarking work).

dataset	context length 0	context_length 1	context length 2	forecast horizon
australian_electricity_demand	420	672	420	336

bitcoin	9	60	38	30
car_parts	15	24	15	12
cif_2016	15	24	15	12
covid_deaths	9	60	38	30
dominick	10	16	65	8
electricity_hourly	30	336	210	168
electricity_weekly	65	16	65	8
fred_md	15	24	15	12
hospital	15	24	15	12
kaggle_web_traffic	10	16	65	8
kdd_cup	210	336	210	168
m1_monthly	15	36	23	18
m1_quarterly	5	16	10	8
m1_yearly	2	12	8	6
m3_monthly	15	36	23	18
m3_quarterly	5	16	10	8
m3_yearly	2	12	8	6
m4_daily	9	28	18	14
m4_hourly	210	96	60	48
m4_monthly	15	36	23	18
m4_quarterly	5	16	10	8
m4_weekly	65	26	65	13
m4_yearly	2	12	8	6
nn5_daily	9	112	70	56
nn5_weekly	65	16	65	8
pedestrian_counts	210	48	30	24
rideshare	210	336	210	168
saugeen_river_flow	9	60	38	30
solar_10_minutes	50	2016	1260	1008
solar_weekly	6	10	65	5
sunspot	9	60	38	30
temperature_rain	9	60	38	30
tourism_monthly	15	48	30	24

tourism_quarterly	5	16	10	8
tourism_yearly	2	8	5	4
traffic_hourly	30	336	210	168
traffic_weekly	65	16	65	8
us_births	9	60	38	30
vehicle_trips	9	60	38	30
weather	9	60	38	30

Table A: Dataset context lengths for each mode with the forecast horizon.

Mean MASE results

Dataset	Multivariate	Autoformer	DLinear independent	DLinear multivariate	N-BEATS global	N-HITS global	NLinear independent	NLinear multivariate	PatchTST independent	PatchTST multivariate	TiDE global	TimesFM(ZS) foundational	TimesFM(ZS) Paper's Benchmark	TimesFM(ZS) Comments
Aus. Elecdemand					**1.094**	1.143					1.217	2.109	Y	
Bitcoin					6.408	6.066						**5.366**	Y	
Carparts	Y	1.247	1.403	0.752	0.753	0.748	1.398	1.045	1.142	1.075	**0.747**	0.852		
CIF 2016					2.261	1.622					2.352	**1.261**	Y	
COVID	Y	7.221	9.200	5.601	7.911	7.362	6.287	**5.176**	8.918	8.111	9.069	8.794	Y	
Dominick					0.511	**0.507**					0.550	0.528		
Electricity Hourly	Y	2.400	1.883	**1.880**	2.005	2.073	1.891	1.882	2.131	2.138	2.127	2.265		
Electricity Weekly	Y	0.929	1.096	**0.780**	0.884	0.986	1.001	0.792	0.996	0.846	0.811	0.924		
FRED-MD	Y	0.621	0.548	0.627	0.666	0.660	0.499	0.843	**0.470**	0.573	0.901	0.635	Y	
Hospital	Y	0.930	0.931	0.800	0.786	0.802	0.857	0.803	0.855	0.809	0.850	**0.771**	Y	
Kaggle Weekly	Y				0.613	**0.582**					0.720	0.602		
KDD					1.252	**1.194**					1.304	1.361		
M1 Monthly					**1.253**	1.257					1.501	1.223		
M1 Quarterly					2.543	**2.369**					2.777	1.882		
M1 Yearly					5.525	**4.840**					6.258	5.257		
M3 Monthly					0.916	**0.906**					1.066	0.983		
M3 Other														
M3 Quarterly					**1.134**	1.136					1.522	1.290		
M3 Yearly					3.024	**2.824**					3.272	3.428		
M4 Daily					**1.125**	1.142					1.324	1.328		

M4 Hourly					**2.756**	3.781					3.081	0.812		
M4 Monthly					**0.970**	0.987					1.122	1.063		
M4 Quarterly					**1.209**	1.226					1.469	1.370		
M4 Weekly					0.484	**0.473**					0.560	0.463		
					3.430	**3.344**					3.698			
NN5 Daily	Y	1.087	0.960	0.961	**0.901**	0.908	0.957	0.957	1.156	1.017	0.962	1.483	Y	
NN5 Weekly	Y	1.032	1.126	0.879	**0.824**	0.864	1.098	0.847	1.011	0.961	0.848	0.857	Y	
Pedestrians					0.245	**0.241**					0.245	0.257	Y	
Rideshare	Y	7.680	4.580	4.314	4.108	4.041	4.485	4.418	**3.617**	3.753	4.164	2.985		
Saugeen		1.546	1.602	1.602	1.587	1.668	**1.443**	**1.443**	1.492	1.492	1.536	2.225	Y	
Solar 10 Mins	Y		1.793	1.611	2.379	2.406	1.781	1.771	1.439	**1.437**	1.473	1.456		
Solar Weekly	Y	1.767	1.102	1.157	1.306	1.384	1.312	1.248			1.099	**0.970**	Y	
Sunspot		0.133	0.118	0.118	0.192	0.166	**0.083**	**0.083**	0.119	0.119	0.174	0.272		
Temp. Rain	Y			1.093	**0.791**	0.795		1.725		0.861	1.201	1.050		
Tourism Monthly					**1.465**	1.476					1.705	2.273	Y	
Tourism Quarterly					1.570	**1.535**					2.844	2.193	Y	
Tourism Yearly					109.908	**3.195**					5.358	3.259	Y	
Traffic Hourly	Y		0.963	0.923	0.873	**0.850**	0.922	0.918	0.964	0.896	0.924	0.799	Y*	Data Leakage
Traffic Weekly	Y	1.476	1.487	1.096	1.117	1.083	1.357	1.103	1.266	1.168	1.124	**1.040**	Y	
US Births		1.688	2.166	2.166	**1.438**	1.510	2.145	2.145	2.607	2.607	2.723	4.475	Y	
Vehicle Trips					1.747	**1.728**					2.015	2.164		
Weather					0.625	**0.621**					0.648	0.621	Y	

Table B. MASE results for each model and dataset benchmarked in this study. Best results are shown in **bold.** Datasets not in TimesFM's own benchmarks are not considered as best results due to the likelihood that these were part of the training data.