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
iTransformer	Multivariate	Transformer	https://arxiv.org/pdf/2310.06625
PatchTSMixer	Multivariate	MLP	https://arxiv.org/pdf/2306.09364
TSMixer	Multivariate	MLP	https://arxiv.org/pdf/2303.06053

TimeMixer	Multivariate	MLP	https://openreview.net/pdf?id=7oLshfEIC2
TimesFM	Foundational	Transformer	https://arxiv.org/pdf/2310.10688v2
TinyTimeMixers	Foundational	MLP	https://arxiv.org/pdf/2401.03955

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.

In order to run multivariate models on all the dataset we set the number of channels to be 1 which then allows the model to be treated as a univariate model.

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.

iTransformer

iTransformer is a transformer based model which applies the attention and feed forward network on the inverted dimensions. The source code used is the official implementation.

d_model	512
embed	timeF
use_norm	True
class_strategy	projection

factor	1
n_heads	8
output_attention	True
d_ff	512
Activation function	gelu
e_layers	2
enc_in	1
dec_in	1

Table 7: iTransformer model specific hyperparameters

PatchTSMixer

PatchTSMixer was originally named TSMixer and was proposed by researchers from IBM. It is an MLP based model. Source code was adapted from the official Hugging Face implementation

d_model	48
num_layers	3
expansion_factor	3
dropout	0.5
head_dropout	0.7
mode	common_channel
scaling	std

Table 8: patchTSMixer model specific hyperparameters

TSMixer

TSMixer was a model architecture proposed by researchers from Google. It is an MLP based architecture inspired by MLP-Mixer from the Computer Vision field.

activation_fn	relu
blocks	4

num_blocks	4
ff_dim	256
output_channels	None
norm_type	batch
normalize_before	True
revin	True
affine	False
subtract_last	False

Table 9: TSMixer model specific hyperparameters

TimeMixer

Is an all MLP model architecture

revin	True
affine	True
subtract_last	False
task_name	short_term_forecast
label_len	0
down_sampling_window	2
down_sampling_layers	1
d_model	32
d_ff	32
e_layers	4
factor	3
enc_in	1
dec_in	1
c_out	1
down_sampling_method	avg
channel_independence	1

moving_avg	25
embed	timeF
freq	None
use_future_temporal_feature	0
decomp_method	moving_avg
use_norm	1
num_class	None

Table 10: TimeMixer model specific hyperparameters

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 11 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.

Tiny Time Mixers

Tiny Time Mixers (TTM) is a foundational model proposed by researchers at IBM and is based on using blocks of PatchTSMixer and is therefore not Transformer based. The benchmarking was completed using the R2 version of the model from Hugging Face. The experiments were run as Zero Shot (ie we did not perform any fine tuning). A number of the Monash datasets were used in the training process and these are denoted in *italics* in the results to indicate the presence of data leakage These models predict a maximum forecast horizon of 96 timesteps and 512 timesteps as a context length. As a result some datasets were not suitable for inference. Context windows were leading zero padded to ensure the input was of the correct shape.

freq_prefix_tuning	True
prefer_l1_loss	True
prefer_longer_context	True
force_return	zeropad

Table 12 hyperparameters for Tiny Time Mixers