Comparative Analysis of Deep Learning Models for Long-term Time Series Prediction in Marine Turbine Variables

Tolap Saturam

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

01/09/2023

Abstract

This dissertation investigates long-term time series prediction for turbine variables, including velocity, torque, and thrust. The robust long-term predictions offer benefits for turbine adjustments and better engine management. The distinctive traits of Turbine Marine datasets have been examined. Through a comparison of diverse methods from various deep learning model families based on mean squared error, this study evaluates their effectiveness regarding prediction accuracy and model complexity. The standout model, FDNet, shows remarkable predictive capabilities, while the more lightweight model, DLinear, also performs well. This analysis guides the researchers and practitioners in selecting models to suit their work, not only in turbine experiments but also in other similar time series prediction problems. The code is available at https://github.com/Nackalalalong/turbine2023.

Education Use Consent

Acknowledgements

I extend my sincere thanks to Dr. Gerardo Aragon Camarasa for his guidance and support in shaping this research. I am also grateful to my friends and family whose encouragement and understanding have been invaluable throughout this journey.

Contents

1 Introduction **5**

1.1 Overview & Problem Statement 5

1.2 Objectives 5

1.3 Report Structure 5

2 Background & Literature Survey **6**

2.1 Marine Turbine Data 6

2.2 Related Work 7

2.3 Models Used 7

2.3.1 NLinear 7

2.3.2 DLinear 8

2.3.3 GCFormer 8

2.3.4 TiDE 9

2.3.5 FDNet 9

3 Design & Implementation **11**

3.1 Dataset Analysis 11

3.2 Implementation Details 11

3.2.1 Repository Overview 11

3.2.2 Hyperparameter Tuning 13

3.2.3 Training Models 14

3.2.4 Evaluating Models 15

4 Experiments **16**

4.1 Experiment Settings 16

4.2 Result 16

4.3 More Analyses On Each H and T 17

4.4 RMSE of each variable 17

4.5 Model Size Comparison 17

4.6 How Far Can We Predict Efficiently? 18

5 Conclusion 19

5.1 Overview 19

5.2 Future Work 19

5.3 Lessons Learned 19

A Datasets Analysis 21

A.1 2D dataset 21

A.2 3D dataset 22

B Hyperparameter Tuning Result 24

Bibliography 26

Chapter 1: Introduction

1.1 Overview & Problem Statement

The reliance on fossil fuels for energy generation has reached a concerning level [5]. Moreover, the consumption of fossil fuels has negative effects on the environment, manifesting in both direct and indirect ways [4]. To counter these environmental challenges, researchers are actively seeking cleaner and more effective energy generation alternatives. Among these alternatives, marine turbines hold promise for harnessing electricity water stream flow. The variables that govern marine turbine performance—velocity, torque, and thrust—present intricate challenges in optimization. The ability to forecast the future values of variables far enough in advance is extremely useful, as it would allow us ample time to adjust the turbine settings to the optimal position. This, in turn, would result in the best turbine performance. To navigate these challenges, various models have been employed to enhance marine turbine performance. However, selecting the best model requires a comprehensive understanding of the pros and cons associated with each, necessitating a systematic exploration of experiments. In this context, the current study aims to contribute to the field by conducting a rigorous evaluation of different models' effectiveness in predicting marine turbine variables, thereby facilitating informed decision-making for optimal energy generation.

1.2 Objectives

This project focuses on forecasting marine turbine parameters, including velocity, thrust, and torque. This employs linear models and a range of different model approaches. Three datasets are used, each containing three parameters for prediction. The main objective is to examine the marine turbine datasets to uncover significant characteristics that can aid in model development. The subsequent goal is to assess a variety of predictive models from different families to determine their effectiveness.

1.3 Report Structure

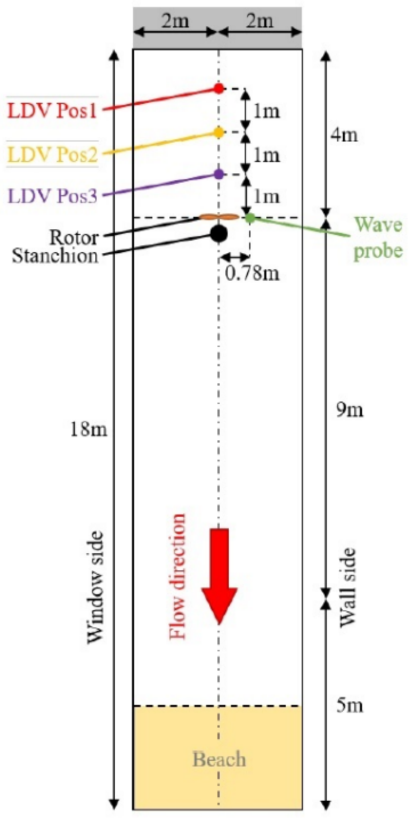
The structure of the report is outlined as follows: Chapter 1 provides an Introduction and objectives. The background and literature survey are presented in Chapter 2. Chapter 3, Design and Implementation, encompasses data analysis and the specifics of the models employed in this dissertation. Chapter 4 delves into experiments, covering the experiment settings, results, and insightful analysis. The concluding chapter, Chapter 5, summarizes the findings and draws conclusions.

Chapter 2: Background & Literature Survey

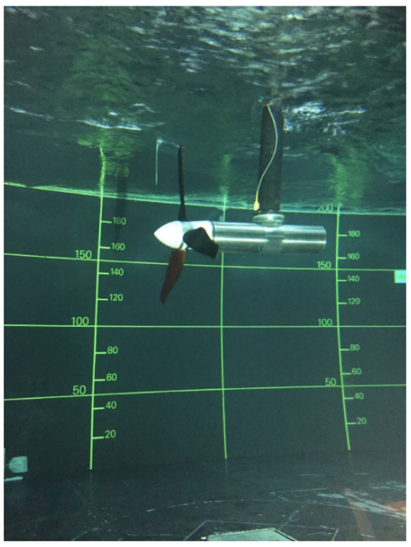
2.1 Marine Turbine Data

The dataset employed for this project was obtained through collaboration with the Energy Systems Research Unit at the University of Strathclyde, facilitated by Dr. Gerardo Aragon Camarasa, a Lecturer in Autonomous and Socially Intelligent Robotics within the Computing Science department at the University of Glasgow. This dataset encompasses measurements related to marine turbines, specifically thrust, torque, and flow velocity. These measurements were conducted in a controlled testing environment, as illustrated in figure 2.1(b) and adhered to the International System of Units (SI).

The dataset includes velocity measurements taken at three specific points upstream of the turbine, indicated as 1D, 2D, and 3D. These designations represent distances of 1 meter, 2 meters, and 3 meters, respectively, relative to the turbine's diameter as depicted in figure 2.1(a). Hence, there are 3 datasets that will be used in the experiments, including 1D, 2D, and 3D datasets. The numbers in the dataset names indicate the distances upstream of the turbines.



(a) Sketch of the top view of the setup from [11]



(b) The real turbine in a tank used in the data collection processes from [11]

Figure 2.1: The data collection setup for turbine variables

2.2 Related Work

The field of long-term multivariate time series forecasting has become a complex challenge in machine learning. The usual way to predict time series, like ARIMA [2], falls short when dealing with real-world data complexities like changes over time, unusual data points, noise, and many factors at play. The development of the backpropagation technique [14], a fundamental method for training artificial neural networks, ushered in the age of deep learning with more intricate models.

The rise of the Attention mechanism and Transformer [16] has made Transformer-based models a key player in computer vision and other fields. An early Transformer-based method for time series prediction [9] outperformed the previous best method, ARGONet [10], which combines statistical and spatio-temporal methods. Despite their impressive results in Long-Term Time Series Forecasting (LTSF), transformers face challenges such as taking sheer amount of time and memory. Informer [20], using a self-attention technique, outperforms others and speeds up LTSF predictions. FEDformer [21], a transformer-based model, outpaces earlier methods by combining the Transformer structure with other techniques. Recent research [18] shows that a simple linear model, NLinear, beats all transformer-based models by a large margin. Notably, GCFormer [19], a novel transformer-based approach, outperforms other models including FEDformer.

Furthermore, TiDE [3], a Multi-layer Perceptron (MLP) based encoder-decoder model, outperforms earlier methods on popular LTSF benchmarks with faster speeds compared to transformer-based models. Later, FDNet [15], a simpler model based on basic auto-regression principles, outperforms other models on practical benchmarks. This model is faster for predictions without using attention mechanisms like transformers.

2.3 Models Used

In this dissertation, there are five different models used in the experiments including NLinear [18], DLinear [18], GCFormer [19], TiDE [3], and FDNet [15]. These models were selected for different reasons. The two simple and powerful linear models, NLinear and DLinear, are used as strong baselines. GCFormer is the current transformer-based state-of-the-art model, so it might be able to outperform other models due to the effectiveness of the transformer. TiDE outperforms the transformer-based model but still runs much faster, so this model is interesting. The last one, FDNet, is the current state-of-the-art model without long-term dependencies in the data.

2.3.1 NLinear

This model is quite simple, relying solely on a fully connected layer, as illustrated in figure 2.2. However, to enhance the performance of the pure linear mode in addressing distribution shifts, the input sequence will be subtracted by the last value of the sequence and then passed through the model. The subtracted component will be added to the output before making predictions.

There are 2 possible configurations of NLinear. In the first configuration, the model consists of multiple fully connected layers, one for each input feature individually. We refer to this as NLinear-i, where ”i” indicates individual. In the second configuration, the model employs only one fully connected layer, shared among all input features. When input is fed to the model, the second and third dimensions are swapped and then processed through the model. The second and third dimensions of the output are swapped back. We denote this as NLinear-ni, with ”ni” indicating not individual.

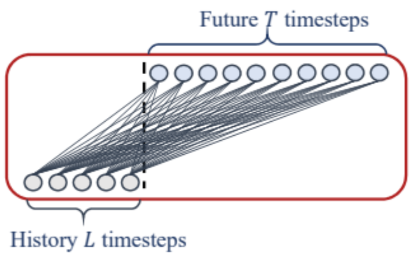


Figure 2.2: Illustration of the basic linear model from [18]

2.3.2 DLinear

This model is also rooted in the linear model but incorporates slightly more intricate elements. It incorporates a series decomposition technique utilized in FEDformer [21] in combination with linear layers. The input undergoes decomposition into a trend component through a moving average kernel and a seasonal component. These components are then passed through the linear layers. Subsequently, the values of the two outputs are combined to derive the final prediction. Much like NLinear, there are two potential configurations: DLinear-i and DLinear-ni.

2.3.3 GCFormer

As evident from figure 2.3, the model's entry comprises two branches: a global branch and a local branch. The global branch incorporates convolutional layers with a kernel size equivalent to the input size, allowing it to capture long-term dependencies in the data while maintaining sublinear parameter growth [19]. Conversely, the local branch captures short-term dependencies in the data, utilizing models such as PatchTST [12], AutoTransformer [17], DLinear [18], TCN [1], or any other model capable of capturing short-term dependencies. Subsequently, the outputs of each branch progress through the remaining layers of the model, comprising linear layers and attention mechanisms. Ultimately, the two values from the two branches are combined, passed through a reverse norm unit, and contribute to the final prediction.

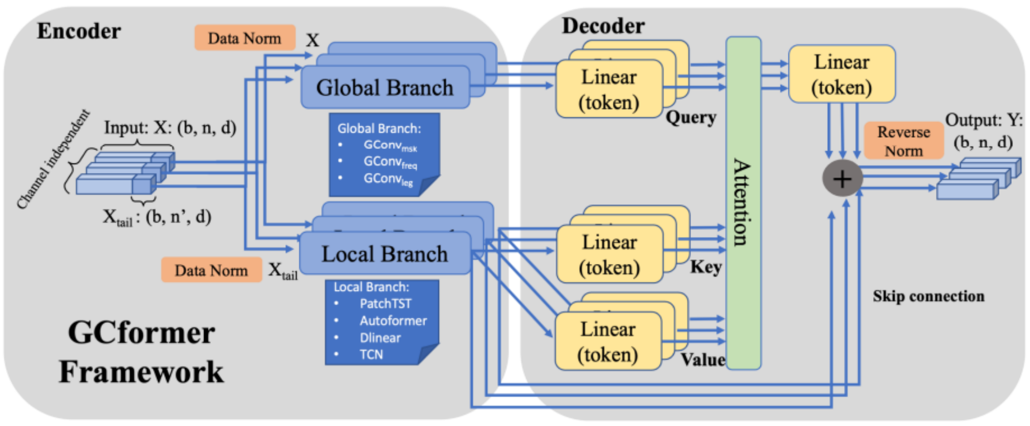


Figure 2.3: GCFormer overall framework from [19]

2.3.4 TiDE

This model is a Multi-layer Perceptron-based encoder-decoder architecture. According to figure 2.4, the model exclusively consists of linear layers. The major components of the model encompass dense encoders and decoders, forming a stack of residual blocks. These blocks incorporate dense layers, dropout, and layer normalization, along with a skip connection connecting the input and output of the dropout layer. The model's input is a concatenation of three components: the main lookback window, attributes, and covariates. The main lookback window signifies the sliding window of relevant variables, while attributes offer unchanging data information—such as retail product features. Covariates are additional features that evolve over time, with computable future values. For instance, the day of the week or week of the year for a data point, which are calculable based on the data point's timestamp. Prior to the final prediction, a residual unit and a temporal decoder are introduced.

In this dissertation, covariates are not included. The TiDE model encompasses two configurations. The first configuration, TiDE-wo-a, does not utilize attributes or covariates. The second configuration, TiDE-w-a, excludes covariates but incorporates an attribute. The attribute values are designated as 1 for 1D, 2 for 2D, and 3 for 3D datasets.

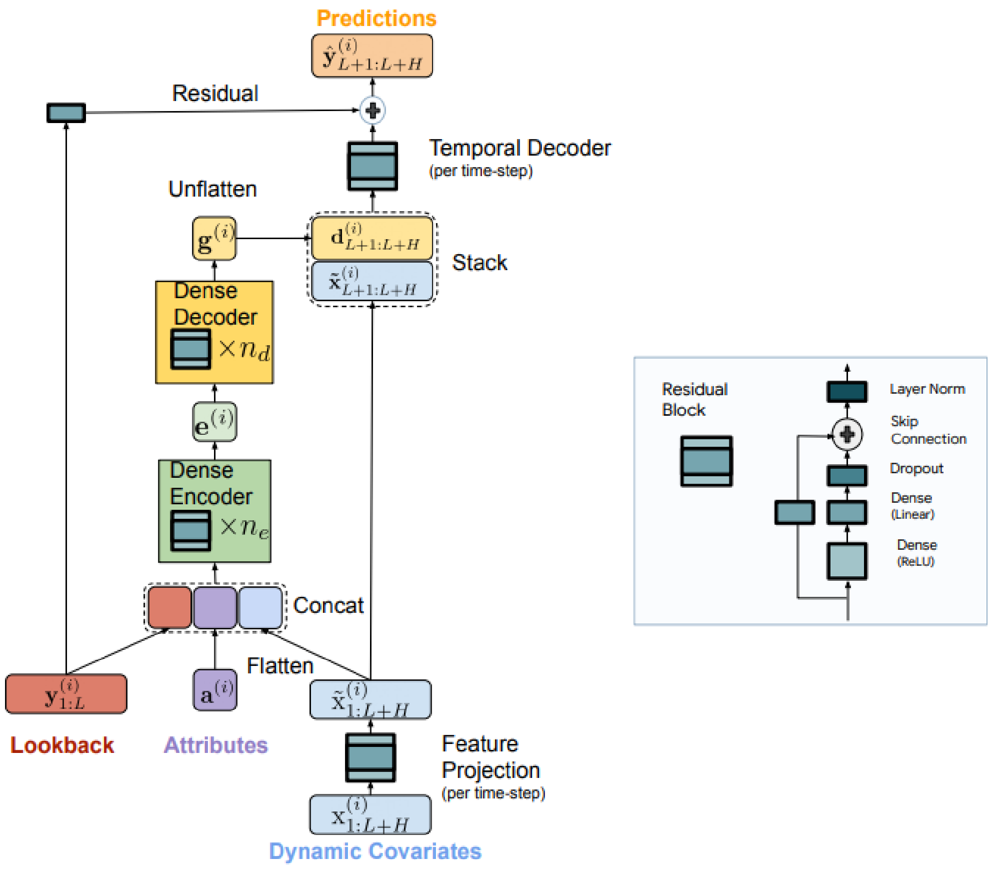


Figure 2.4: Overview of TiDE architecture from [3]

2.3.5 FDNet

This model does not focus on capturing long-term dependencies within the input. Instead, its emphasis lies on recent dependencies. It has been demonstrated that components responsible for extracting long-term dependencies are not crucial, and in fact, the model performs better without them [15]. The input sequence is divided into multiple sub-sequences, each with a progressively decreasing length. The final sub-sequence is the shortest yet contains more feature extraction layers, as depicted in figure 2.5. Each subsequence is processed by the model, and their outputs remain mutually independent. The primary layers of the model are comprised of convolutional layers, as evident in figure 2.6.

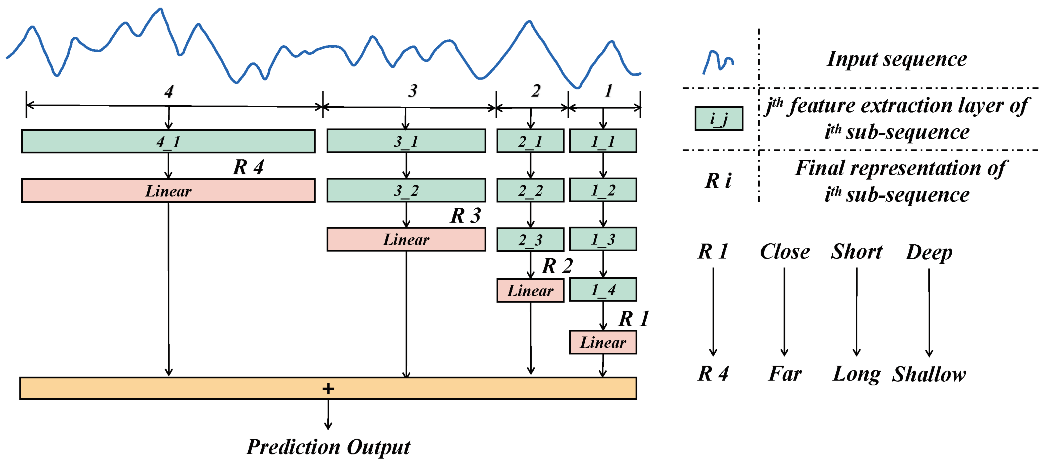


Figure 2.5: Focal input decomposition from [15]

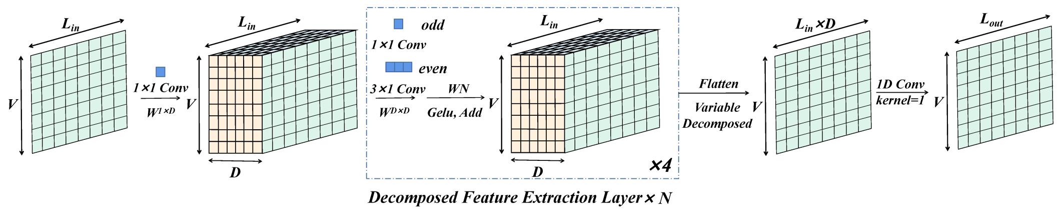


Figure 2.6: Overview architecture of FDNet from [15]

Chapter 3: Design & Implementation

3.1 Dataset Analysis

Prior to delving into the model experiments, comprehending the dataset's characteristics holds significant importance. The attributes it possesses, including noise, patterns, outliers, adherence to normal distribution, and inter-variable correlations, can be advantageous for model selection, preprocessing design, and even algorithmic enhancements within the models.

We start by noticing that the three variables—thrust, torque, and flow velocity—follow repeated patterns and sometimes have different levels of random changes. Figure 3.1 shows these repeating patterns over time. These patterns in the data suggest that there are inherent behaviors in how marine turbines work, while the random changes indicate outside factors that affect these behaviors. This interaction between patterns and randomness highlights the dataset's complexity, inviting us to explore more about where these variations come from.

Our study also reveals that these variables have a common distribution. This is shown in figure 3.2, which gives a visual representation of how often different values occur for each variable. This common distribution is important because it allows us to use certain statistical methods. By sticking to this pattern, we can confidently analyze the data and create reliable models.

Additionally, we examine how these variables relate to each other. Interestingly, flow velocity has weak connections to both thrust and torque. This suggests that changes in flow speed are not strongly connected to changes in thrust and torque. On the other hand, thrust and torque have a strong connection. This is visible in figure 3.3, where a clear straight line shows their close relationship. This strong connection is represented by a number called the correlation coefficient, which is 0.97. This means that when thrust changes, torque tends to change in a similar way, and vice versa.

It is important to note that 2D and 3D datasets are similar to 1D, but we will not delve into them here to save space. The details of 2D and 3D datasets are shown in A.

These discoveries have significant implications for understanding how marine turbines work. The mix of repeating patterns and random changes suggests that many outside influences affect turbine performance. The common distribution pattern lets us use established statistical methods. The observed connections between variables show different degrees of how they relate. These insights set the stage for more advanced analyses in later sections, where we will dig even deeper into the dataset. This will help us gain a better grasp of marine turbine dynamics and renewable energy systems.

3.2 Implementation Details

3.2.1 Repository Overview

The code has been published on GitHub at `https://github.com/Nackalalalong/turbine2023`. The experiments were conducted using Python 3.10.11. There are multiple folders and files, including:

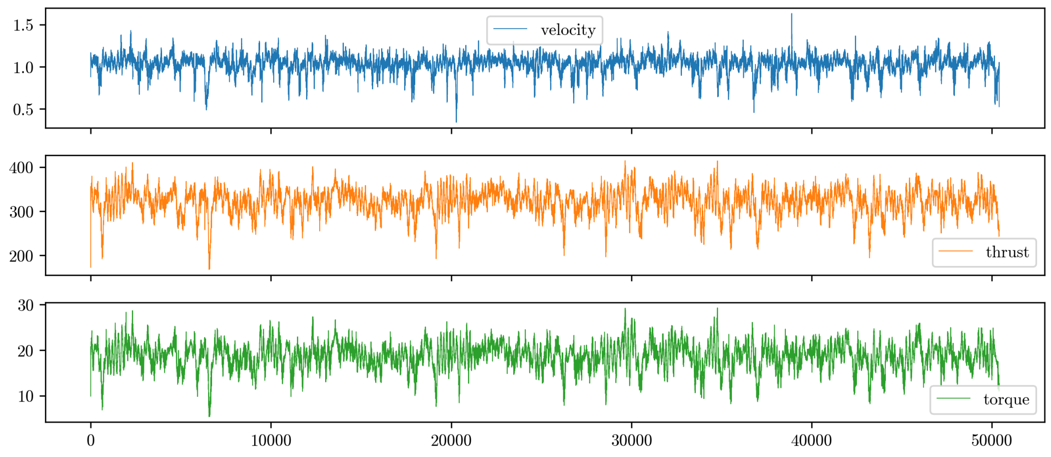


Figure 3.1: The unprocessed values of the variables within the 1D dataset across the temporal domain

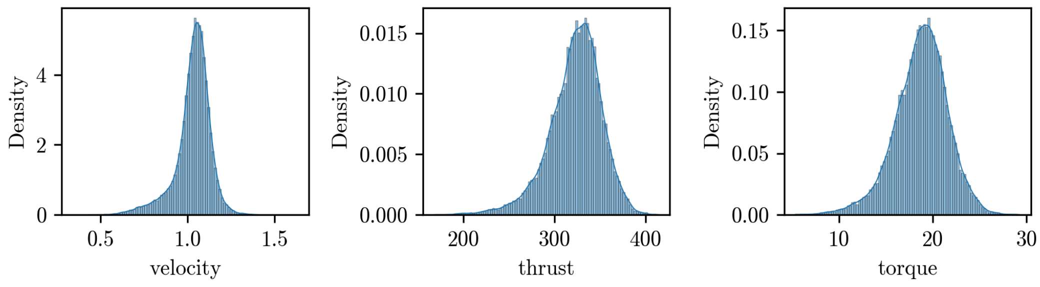


Figure 3.2: The statistical distribution of the variables within the 1D dataset, encompassing velocity, thrust, and torque.

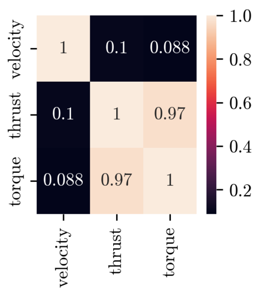


Figure 3.3: The correlation of variables in 1D dataset

- `data`: This folder contains raw data files used in this dissertation in CSV format, including `Data1D.txt`, `Data2D.txt`, and `Data3D.txt`.
- `layers`: This folder contains code for various complex layers, distinct from the layers provided by deep learning frameworks for the models.
- `models`: The implementation of the models is located in this folder.
- `utils`: This folder contains utility function code for the training and evaluation pipeline.

The files include:

- `explore.ipynb`: This Jupyter notebook contains code related to data analysis in section 3.1.
- `data.py`: This code loads data into PyTorch's DataLoader for use in training and evaluating models.
- `constants.py`: This Python file contains constant variables, such as H and T, as well as the results of hyperparameter tuning. Additionally, it includes the `Config` class, which consists of variables used to define a model's architecture and behavior.
- `tune.py`: This Python file is used to run hyperparameter tuning.
- `train.py`: This Python file contains code for training models.
- `analyse_exp.ipynb`: This Jupyter notebook contains code for plotting multiple bar charts to compare the performance of the models discussed in section 4.3.
- `eval.py`: This Python file contains code for plotting the Mean Squared Error (MSE) vs. timestep in T graph, as used in section 4.6.
- `requirements.txt`: This text file lists the required Python libraries used in the experiments.
- `.style.yapf`: This is the code that defines the YAPF format style.
- `exp_error.txt`: When there are errors while training models in parallel, the error details will be written to this file.

This project utilized `Typer`, a Python library designed for parsing command-line arguments. It also generates informative, auto-generated command-line messages, such as help messages. The help message can be displayed by running `python <python_filename> --help`.

3.2.2 Hyperparameter Tuning

This dissertation used `ray`, a Python tuning library. In the `tune.py` file, there is a `main` function, which serves as the entry point for hyperparameter tuning. This function accepts four arguments, including:

- `model`: The model to tune hyperparameters. There are eight possible values, including `nlinear-i`, `nlinear-ni`, `dlinear-i`, `dlinear-ni`, `tide-w-a`, `wide-wo-a`, `gcformer`, and `fdnet`.

- `epochs`: The number of epochs for tuning a model.
- `num_samples`: The number of hyperparameter combinations to try.
- `data`: The name of the dataset. There are four possible values, including `1d`, `2d`, `3d` for a specific dataset, and `all` for all datasets.

Normally, `ray` displays the tuning results and details in the terminal. To save these results for later use, you can use a pipe command to redirect the output from the terminal to a file. For example, simply run:

```
python tune.py nlinear-i > tune_result.txt
```

3.2.3 Training Models

In the `train.py` file, the `main` function is the entry point for training models. There are thirteen arguments in total for this function, including:

- `model`: The model name, which is the same as described in the hyperparameter tuning section (section 3.2.2).
- `data`: The dataset name. The possible values are the same as described in the hyperparameter tuning section (section 3.2.2).
- `seq_len`: The history length H.
- `pred_len`: The prediction length T.
- `long_run`: If this is True, all combinations of H and T experiments will be conducted. The default is False.
- `epochs`: The number of epochs for each experiment.
- `parallel`: If this is True, the experiments will run in parallel.
- `max_workers`: The maximum number of processes that can run simultaneously if `parallel` is True. The default is 4.
- `skip_done`: If this is True, experiments that have finished before will be skipped.
- `tensorboard_save_dir`: The folder for Tensorboard logging.
- `eval_after_train`: If this is True, it will evaluate the model with the test dataset after finishing training for each experiment. The evaluated result will be saved in nested subfolders of the `tensorboard_save_dir` folder. This is useful for plotting a graph for performance comparison later.
- `log_grad`: If this is True, the gradients of backpropagation will be logged in Tensorboard logging, but the experiment will take much longer time.
- `load_data_to_cuda`: If this is True, the entire dataset in the DataLoader will be loaded onto the GPU. Typically, data resides in the CPU memory and is transferred to the GPU when it passes through the model.

For example, to run TiDE-wo-a experiments with all datasets and all combinations of H and T in parallel with a maximum of three processes running simultaneously for 20 epochs, and log the experiments to the folder `exp`, skip the finished experiments, and evaluate with the test dataset after finishing training in each experiment, simply run:

```
python train.py --model tide-wo-a --data all --epochs 20
--tensorboard-save-dir exp --long-run --skip-done
--eval-after-train --parallel --max-workers 3
```

It is important to note that `Typer` converts underscores in an argument name to hyphens.

3.2.4 Evaluating Models

To plot the Mean Squared Error (MSE) for each H and T comparison graph (see figure 4.1), simply run the `analyse_exp.ipynb` Jupyter notebook.

Another file for visualizing the model evaluation is `eval.py`. This Python file contains multiple functions for different purposes. The function `reconstruct` is used for plotting the reconstructed values versus the actual values. However, it does not provide much informative insight for this dissertation. Another function is the `analyse_how_far` function, which is used to plot the average MSE for each timestep from 1 to 720 (see figure 4.2). This function accepts three arguments, including:

- `tensorboard_save_dir`: The Tensorboard log folder from the experiments after running `train.py`.
- `level`: This defines how we average the MSE values. There are three possible options, including `T`, `data`, and `overall`. If `level` is `T`, the MSE values for all H values will be averaged. If `level` is `data`, all MSE values for all combinations of H and T will be averaged. If `level` is `overall`, all MSE values for all combinations of H, T, and datasets will be averaged.
- `log_x`: If this is True, the x-axis will be set to a logarithmic scale.

For example, to plot figure 4.2, run the command `python eval.py analyse-how-far --level overall --log-x`

Chapter 4: Experiments

4.1 Experiment Settings

Preprocessing. In order to prepare the input matrices, all datasets will be transformed into sliding windows with history length of H and prediction length of T while $H \in \{24, 48, 72, 96, 120, 144, 168, 192, 336, 504, 672, 720\}$ and $T \in \{96, 192, 336, 720\}$. The values of H and T are taken from [18] to maintain consistent experimental settings. Some models that necessitate more additional features than the three primary features exhibit slight differences in the dimensions of their matrices.

Evaluation Metric. Mean Squared Error (MSE) is used to compare performance due to the continuity of the target values. Furthermore, MSE exhibits sensitivity to outliers. Consequently, when a model's MSE is low, this signifies a noteworthy reduction in deviations between the actual and predicted values.

Compared Methods. We include eight methods consisting of five models. DLinear, TiDE and NLinear have two settings. The methods are NLinear-i (NLinear with individual linear layer for each input feature), Nlinear-ni (NLinear with only one linear layer for all input features), DLinear-i (DLinear with individual linear layer for each input feature), DLinear-ni (DLinear with only one linear layer for all input features), TiDE-wo-a (TiDE without attribute), TiDE-w-a (TiDE with attribute), GCFormer and FDNet. NLinear-ni, the simplest model, is used as a baseline. For each compared method, hyperparameter tuning is performed using H of 168 and T of 336. The results of the hyperparameter tuning are shown in B.

4.2 Result

Dataset	T	NLinear-i	NLinear-ni	DLinear-i	DLinear-ni	TiDE-wo-a	TiDE-w-a	GCFormer	FDNet
1D	96	0.700	0.752	**0.587**	0.610	0.630	0.631	0.639	0.608
1D	192	0.971	0.996	0.752	0.746	0.762	0.761	0.847	**0.743**
1D	336	1.203	1.234	0.850	**0.822**	0.837	0.835	1.060	0.827
1D	720	1.303	1.319	0.930	**0.882**	0.896	0.892	1.188	0.891
2D	96	0.761	0.815	**0.656**	0.685	0.697	0.703	0.710	0.666
2D	192	1.130	1.155	0.899	0.884	0.898	0.900	0.999	**0.878**
2D	336	1.514	1.520	1.062	**1.026**	1.040	1.037	1.298	1.028
2D	720	1.834	1.864	1.172	**1.136**	1.151	1.148	1.637	1.141
3D	96	0.799	0.853	0.697	0.718	0.731	0.740	0.758	**0.695**
3D	192	1.094	1.119	0.907	0.885	0.896	0.905	0.991	**0.857**
3D	336	1.281	1.297	1.008	0.979	0.989	1.001	1.144	**0.941**
3D	720	1.332	1.336	1.057	1.050	1.065	1.073	1.239	**1.004**

Table 4.1: Average MSE of different dataset and T. The best results are highlighted in bold.

NLinear-i	NLinear-ni	DLinear-i	DLinear-ni	TiDE-wo-a	TiDE-w-a	GCFormer	FDNet
1.160	1.188	0.881	0.869	0.883	0.886	1.042	**0.857**

Table 4.2: Overall MSE of each model. The best result is highlighted in bold.

According to table 4.1, DLinear-i exhibits the lowest MSE in 1D dataset with T of 96 and 2D dataset with T of 96, while DLinear-ni attains the lowest MSE in 1D dataset with T of 336, 720, and 2D dataset with T of 336, 720. For the remaining experiments, FDNet demonstrates the lowest MSE. Additionally, this model boasts an overall MSE of 0.857, as evident in table 4.2. It is evident that the standout performer is FDNet, closely followed by DLinear. Among the transformer-based models, GCFormer, being the sole representative, achieves an overall MSE of 1.042, even lower than DLinear. Lastly, TiDE-w-a marginally underperforms compared to TiDE-wo-a.

4.3 More Analyses On Each *H* and *T*

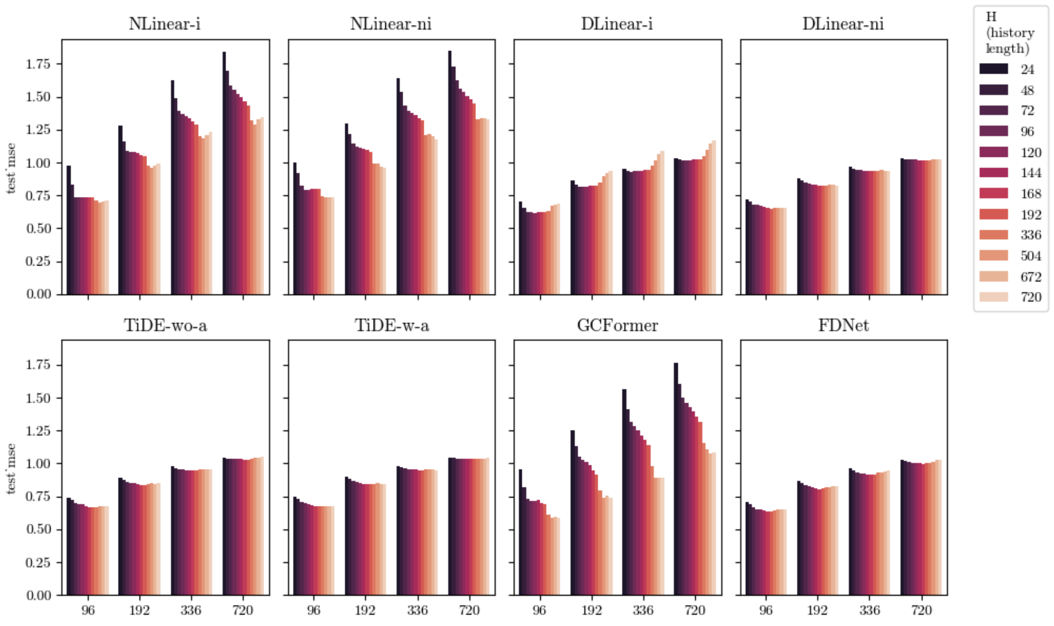


Figure 4.1: MSE of each *H* and *T*. The Y axes are shared among subplots

As depicted in figure 4.1, NLinear-i, NLinear-ni, and GCformer exhibit similar trends. The MSE at the lowest *H* value is relatively high and decreases significantly as *H* increases. However, for NLinear-i, a slight increase is noticeable as *H* reaches its maximum. On the other hand, DLinear-i, DLinear-ni, TiDE-wo-a, TiDE-w-a, and FDNet share certain characteristics in their trends. The MSE at the lowest *H* value is slightly elevated and decreases gradually as *H* increases. A minor increase is observed when *H* reaches its maximum. Notably, DLinear-i stands out with a distinct pattern. Noticeable spikes occur when *H* is near its maximum. Surprisingly, at the highest *T* value, the MSE even surpasses values at lower *H* values.

4.4 RMSE of each variable

To compare the errors of each variable in various datasets, the root mean squared error (RMSE) is favored due to its non-standardized assessment. As the turbine variables possess different units, the RMSE effectively indicates the effectiveness of a model in predicting each variable.

If we look at RMSE the of each variable in the datasets reported in table 4.3, It is interesting to see that FDNet has only one lowest RMSE value of 1.116 for velocity of 3D dataset. GCFormer has 3 lowest values of RMSE while DLinear-ni has 4 lowest values of RMSE.

4.5 Model Size Comparison

As shown in table 4.4, NLinear-i possesses the lowest number of parameters at 7.2k. In contrast, the transformer-based model GCformer has 2.3m parameters. Notably, the exceptional

Dataset	Variable	NLinear-i	NLinear-ni	DLinear-i	DLinear-ni	TiDE-wo-a	TiDE-w-a	GCFormer	FDNet
1D	velocity	1.021	1.071	0.985	**0.963**	0.977	0.970	1.026	0.968
1D	thrust	1.054	1.044	1.013	**0.920**	0.934	0.921	0.981	0.938
1D	torque	1.048	1.046	1.028	**0.926**	0.941	0.930	0.970	0.941
2D	velocity	1.234	1.261	1.097	**1.053**	1.068	1.061	1.124	1.067
2D	thrust	1.306	1.307	1.175	1.072	1.085	1.078	**1.069**	1.090
2D	torque	1.293	1.287	1.163	**1.058**	1.072	1.066	1.070	1.074
3D	velocity	1.223	1.198	1.191	1.147	1.160	1.162	1.173	**1.116**
3D	thrust	1.034	1.001	0.976	0.936	0.943	0.945	**0.913**	0.919
3D	torque	1.036	1.004	0.962	0.937	0.946	0.949	**0.919**	0.920

Table 4.3: RMSE of each dataset and variable. The best results are highlighted in bold. The value is obtained by averaging all combinations of H and T.

NLinear-i	NLinear-ni	DLinear-i	DLinear-ni	TiDE-wo-a	TiDE-w-a	GCFormer	FDNet
7.2k	2.4k	14.4k	4.8k	140k	542k	2.3m	224k

Table 4.4: The number of parameters of each model

model FDNet comprises 224k parameters, making it 31 times larger than NLinear-i and 10 times smaller than GCFormer.

4.6 How Far Can We Predict Efficiently?

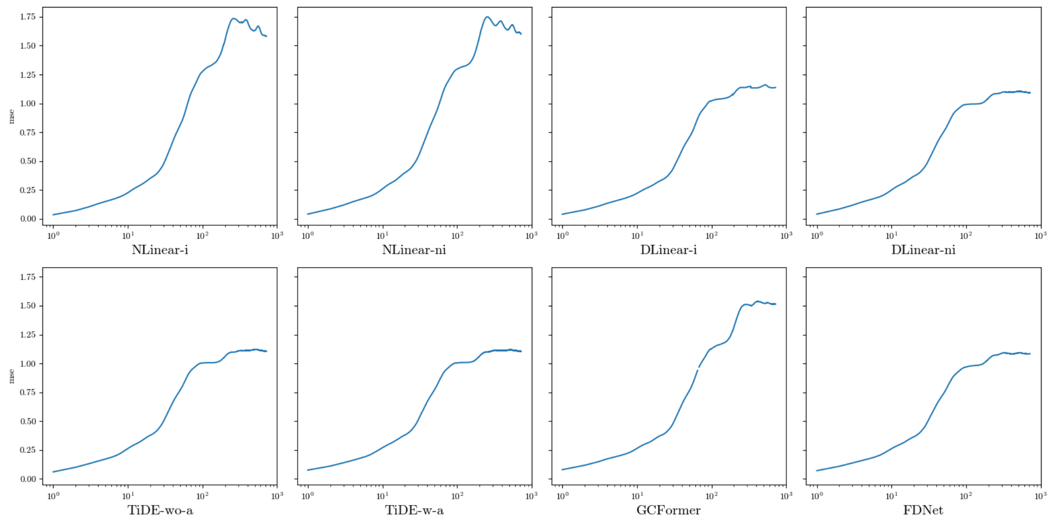


Figure 4.2: The Mean Squared Error (MSE) for each time step t, where t ranges from 1 to 720, is plotted on the x-axis. The plot utilizes a logarithmic scale to aid in pinpointing the elbow point. With a sequence of prediction matrices Y' characterized by dimensions (N, T, c) — where N represents the count of input matrices and c signifies the number of predictive variables — the data point's value at an x-axis value of t corresponds to the average of $Y'[:, t, :]$, following NumPy's indexing notation. The y-axis is consistent across the plots.

It is intriguing to observe the extent to which we can predict while maintaining satisfactory accuracy before the MSE experiences a rapid increase, rendering it highly unreliable for further extended predictions. Figure 4.2 indicates that the elbow point for all models is generally consistent, occurring approximately 20 timesteps into the future before the MSE begins to escalate. Furthermore, it is evident that the MSEs of NLinear-i, NLinear-ni, and GCFormer experience a sharp increase as we forecast further ahead.

Chapter 5: Conclusion

5.1 Overview

The objectives outlined in this dissertation have been successfully accomplished. Our analysis reveals that FDNet consistently achieves the lowest Mean Squared Error (MSE) across various forecasting scenarios, demonstrating its effectiveness. The behavior of MSE with changing prediction horizons (H) highlights distinct trends in different models: NLinear, and GCFormer exhibit significant MSE reduction as H increases, while DLinear, TiDE-wo-a, TiDE-w-a, and FDNet start with lower MSE but show more gradual MSE decline. Model complexity varies, with NLinear-i being the simplest, GCFormer the most complex, and FD-Net achieving a balanced parameter count. Additionally, there is an observed ”elbow point” around 20 timesteps into the future where MSE starts rising, and distant predictions lead to sharp MSE increases for NLinear and GCFormer. The characteristics of three marine turbine datasets have also been analysed, revealing the periodic patterns, correlation between thrust and torque, and normal distribution of all variables.

5.2 Future Work

Future research endeavors could explore models from different families that have not yet been benchmarked with the turbine datasets. Examples of such families include diffusion-based models [8], state space models[13], and graph neural network models [7][6]. It is plausible that these models may surpass the predictive performance of FDNet, while potentially exhibiting varying levels of complexity. Furthermore, considering the resemblance among the three datasets, intelligent preprocessing techniques could be employed to amalgamate multiple datasets, potentially enhancing prediction efficacy. For instance, combining 3D and 2D datasets to predict 2D dataset variables might be viable, given that the fluid stream traverses a distance of 3 meters before reaching 2 meters.

5.3 Lessons Learned

Throughout my research journey, I gained valuable insights into the field of time series prediction by exploring different machine learning models. I conducted various experiments using a range of model types, from traditional multi-layer perceptrons to advanced transformer-based architectures. This thorough exploration helped me understand the strengths and weaknesses of each model type and gave me a holistic view of their capabilities.

As I delved into the structures of these models, I uncovered the intricate layers that make them up, including both simple and highly complex designs. A significant takeaway from this exploration was that higher complexity does not always lead to better performance. This highlighted the importance of selecting a model based on the specific characteristics of the dataset, considering that the nature of the data significantly influences the optimal level of model complexity.

The experiments also emphasized the significance of a diverse perspective when evaluating model performance. Beyond traditional metrics like Mean Squared Error (MSE), I employed a comprehensive approach, considering factors such as individual variable RMSE, overall MSE, model size, and complexity. This comprehensive evaluation allowed for a more thorough assessment of model effectiveness. Interestingly, I discovered that the common assumption regarding the need to capture long-term dependencies for long-term time series prediction is not universally valid. This finding challenged established beliefs and prompted me to explore new possibilities.

Equally important were the practical skills I developed. This project enhanced my proficiency in deep learning frameworks like PyTorch and Lightning. Additionally, my ability to use LaTeX for documentation improved, enhancing my communication of technical details. I also refined my task management skills, learning how to prioritize tasks and identify those that would yield the most significant results. This process emphasized the importance of task optimization by aligning efforts with research goals, ultimately minimizing redundancy and maximizing efficiency.

Appendix A: Datasets Analysis

A.1 2D dataset

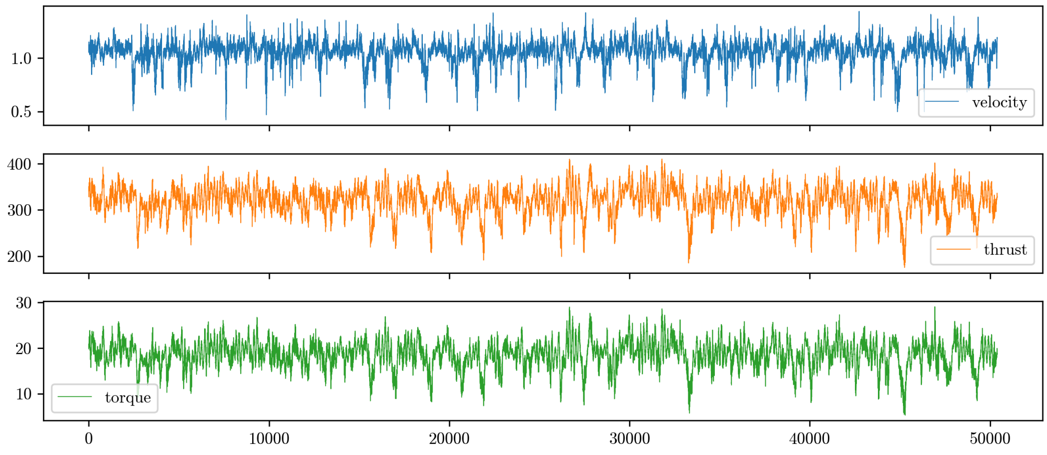


Figure A.1: The unprocessed values of the variables within the 2D dataset across the temporal domain

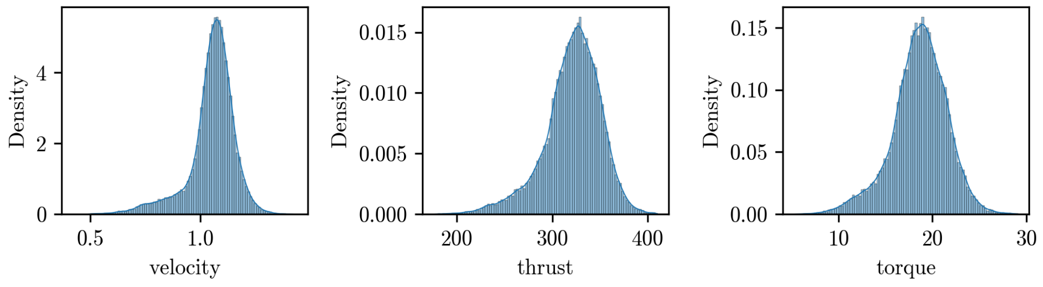


Figure A.2: The statistical distribution of the variables within the 2D dataset, encompassing velocity, thrust, and torque.

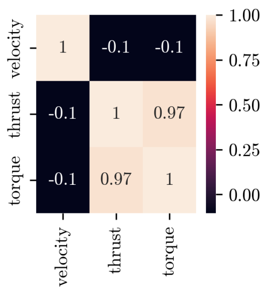


Figure A.3: The correlation of variables in 2D dataset

A.2 3D dataset

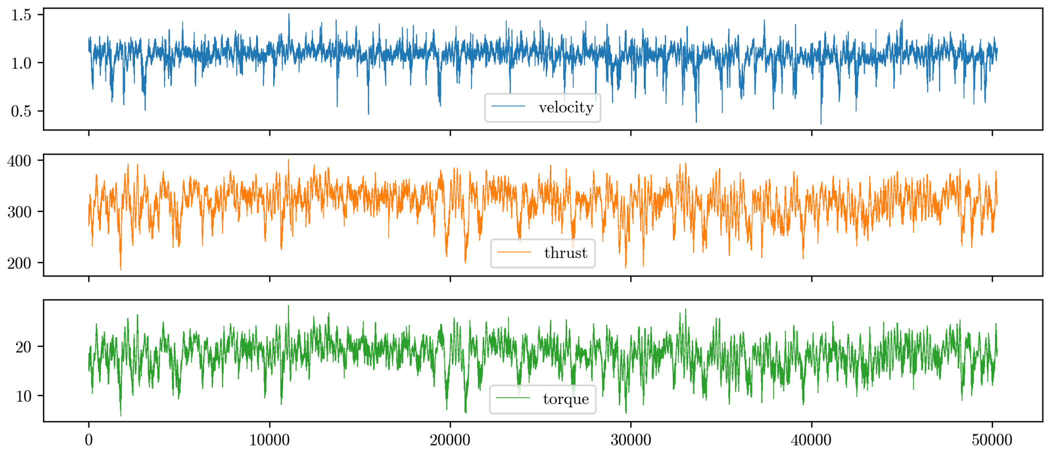


Figure A.4: The unprocessed values of the variables within the 3D dataset across the temporal domain

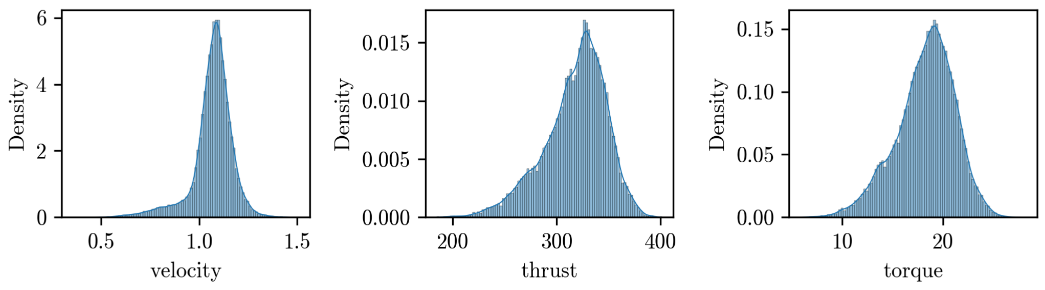


Figure A.5: The statistical distribution of the variables within the 3D dataset, encompassing velocity, thrust, and torque.

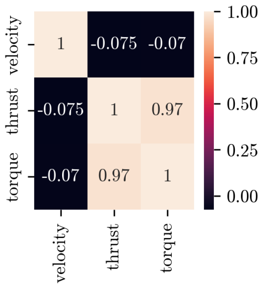


Figure A.6: The correlation of variables in 3D dataset

Appendix B: Hyperparameter Tuning Result

name	value
learning rate	0.000467785952240407
batch size	64

Table B.1: Hyperparameter tuning result of NLinear-i

name	value
learning rate	0.00010606156361434318
batch size	16

Table B.2: Hyperparameter tuning result of NLinear-ni

name	value
learning rate	0.00036642856489512316
batch size	32

Table B.3: Hyperparameter tuning result of DLinear-i

name	value
learning rate	0.00012822786597668052
batch size	64

Table B.4: Hyperparameter tuning result of DLinear-ni

name	value
learning rate	0.00036490350684871407
batch size	128
hidden_dim	64
encoder_layer_num	3
decoder_layer_num	1
temporal_decoder_hidden	32
decoder_output_dim	8
dropout_rate	0.2

Table B.5: Hyperparameter tuning result of TiDE-wo-a

name	value
learning rate	0.00021678258939580435
batch size	128
hidden_dim	128
encoder_layer_num	2
decoder_layer_num	2
temporal_decoder_hidden	64
decoder_output_dim	16
dropout_rate	0.1

Table B.6: Hyperparameter tuning result of TiDE-w-a

name	value
learning rate	0.0004530686954847177
batch size	64
n_heads	32
d_model	128
d_ff	256
patch_len	8
stride	32
dropout	0.14746611098341372
fc_dropout	0.4557664851038477
global_bias	0.4691404907245757
local_bias	0.3823220793212333
h_channel	64

Table B.7: Hyperparameter tuning result of GCFormer

name	value
learning rate	1.7917200056006403e-05
batch size	32
seq_kernel	3
attn_nums	3
d_model	32
pyramid	3
dropout	0.10660256242257105
ICOM	False

Table B.8: Hyperparameter tuning result of FDNet

Bibliography

[1] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018.

[2] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970.

[3] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with tide: Time-series dense encoder, 2023.

[4] EESI. Fact sheet: Climate, environmental, and health impacts of fossil fuels, 2021. Accessed: June 20, 2023.

[5] IEA. The global energy crisis pushed fossil fuel consumption subsidies to an all-time high in 2022, 2022. Accessed: June 18, 2023.

[6] Guangyin Jin, Yuxuan Liang, Yuchen Fang, Jincai Huang, Junbo Zhang, and Yu Zheng. Spatio-temporal graph neural networks for predictive learning in urban computing: A survey, 2023.

[7] Ming Jin, Huan Yee Koh, Qingsong Wen, Daniele Zambon, Cesare Alippi, Geoffrey I. Webb, Irwin King, and Shirui Pan. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection, 2023.

[8] Lequan Lin, Zhengkun Li, Ruikun Li, Xuliang Li, and Junbin Gao. Diffusion models for time series applications: A survey, 2023.

[9] Shunyu Liu and Lijun Wang. Time series forecasting with transformer. *arXiv preprint arXiv:2001.08317*, 2020.

[10] Fred S. Lu, Mehdi W. Hattab, and Christofer Clemente. Improved state-level influenza nowcasting in the united states leveraging internet-based data and network approaches. *Nature Communications*, 10:147, 2019.

[11] R. Martinez, M. Allmark, S. Ordonez-Sanchez, C. Lloyd, T. O'Doherty, G. Germain, B. Gaurier, and C. Johnstone. Rotor load prediction using downstream flow measurements. Unpublished Preprint, August 6, 2020.

[12] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023.

[13] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[15] Li Shen, Yuning Wei, Yangzhu Wang, and Huaxin Qiu. Fdnet: Focal decomposed network for efficient, robust and practical time series forecasting, 2023.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 30–38, 2017.

[17] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, 2022.

[18] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting?, 2022.

[19] YanJun Zhao, Ziqing Ma, Tian Zhou, Liang Sun, Mengni Ye, and Yi Qian. Gcformer: An efficient framework for accurate and scalable long-term multivariate time series forecasting, 2023.

[20] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *CoRR*, abs/2012.07436, 2020.

[21] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. *CoRR*, abs/2201.12740, 2022.