



Evaluating feature attribution for multivariate time series data

OA de Villiers



[orcid.org/ 0000-0002-7242-9848](https://orcid.org/0000-0002-7242-9848)

Dissertation accepted in fulfilment of the requirements for the degree
Master of Engineering in Computer and Electronic Engineering
at the North-West University

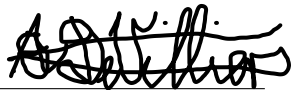
Supervisor: Prof MH Davel

Co-supervisor: Prof SI Lotz

Graduation: November 2025

Declaration

I, Ockert Almaro de Villiers, hereby declare that the dissertation entitled 'Evaluating attribution for multivariate time series' is my own original work and has not been submitted to any other university or institution for examination.



OA. de Villiers

Student number: 28449517

Signed on the 1st day of January 2024 at Potchefstroom.

Abstract

Understanding why a Deep Neural Network (DNN) produces a specific prediction can be as important as making an accurate prediction. The development of such interpretability strategies for DNNs is an active field of research. The current study aims to develop a testbed that can be used to examine the performance of deep learning interpretability techniques for multivariate time series data.

Apart from some transformer-based techniques, most existing interpretability techniques were not specifically developed for time series data, but were rather adapted from other contexts, such as computer vision. In this study, we create a synthetic testbed that can be used to evaluate existing interpretability techniques specifically for time series data. The testbed provides a controlled environment within which datasets with known and tunable characteristics can be created. The purpose of the testbed is to evaluate the performance and accuracy of interpretability techniques, to allow for a fair comparison of their strengths and weaknesses.

We approached this task by first identifying the characteristics of time series data that were important to control and then develop a system capable of generating the required data. A quantitative metric was then developed to compare existing techniques. This metric's effectiveness is demonstrated by applying it to synthetic test datasets. Each dataset represents known characteristics that can be used as ground truth during the evaluation of interpretability techniques.

The key concepts explored in this research include time series data, existing interpretability techniques, temporal convolutional networks, and time-shifted multilayer perceptrons. The research results aim to inform the development of interpretability techniques specifically designed for time series data and advance the understanding of the challenges and opportunities in this field.

Keywords: *Interpretability, deep neural networks, additive feature attribution, multivariate time series.*

Contents

List of Figures	x
List of Tables	xiv
List of Abbreviations	xv
1 Introduction	1
1.1 Background	2
1.1.1 Time series data	2
1.1.2 Deep learning	3
1.1.3 Challenges of interpretability in deep learning	4
1.1.4 Interpretability techniques	5
1.1.5 Time series interpretability	5
1.2 Problem statement	7
1.3 Research questions	7
1.4 Objectives	7
1.5 Project scope	8
1.6 Research methodology	8
1.7 Research design	9
1.8 Dissertation overview	10

2	Background	12
2.1	Overview	12
2.2	Time series data	13
2.3	Time series modelling	15
2.3.1	Recurrent Neural Networks	16
2.3.2	Time-Shifted Multilayer Perceptrons	16
2.3.3	Temporal Convolutional Networks	16
2.3.4	Transformers	17
2.4	Interpreting time series models	17
2.4.1	Motivation	19
2.4.2	Feature attribution of DNNs	19
2.4.3	Specific feature attribution techniques	21
2.4.4	Evaluating feature attributions	23
2.4.5	Shapley values	24
2.5	Synthetic time series interpretability test benches	25
2.6	Conclusion	34
3	Methods and techniques	36
3.1	Overview	36
3.2	DNNs for time series analysis	37
3.2.1	DNN training	37
3.2.2	Time-Shifted Multilayer Perceptron (TS-MLP)	38
3.2.3	Temporal Convolutional Network (TCN)	40
3.3	Shapley values	44
3.3.1	Analytic Shapley values	45
3.3.2	Exact Shapley values	47
3.3.3	DeepLIFT	48

3.3.4	DeepSHAP	50
3.3.5	Reference values	52
3.4	KnowIt	53
3.4.1	Usage	53
3.4.2	Functionality	54
3.5	Conclusion	55
4	Test bench design	56
4.1	Overview	56
4.2	Main components and interactions	57
4.3	Data	58
4.3.1	Data Configuration	58
4.3.2	Model Configuration	59
4.3.3	Training Configuration	59
4.3.4	Train Data	59
4.3.5	Test Data	60
4.3.6	Attribution Data	60
4.3.7	Ground Truth Data	60
4.4	Processes	61
4.4.1	Data Generator	61
4.4.2	Model Trainer	61
4.4.3	Model Evaluator	62
4.4.4	Attribution Generator	62
4.4.5	Ground Truth Generator	63
4.4.6	Attribution Evaluator	63
4.5	Conclusion	63

5	Synthetic time series data	65
5.1	Overview	65
5.2	Data design	65
5.2.1	Goals	66
5.2.2	Defining synthetic time series features	66
5.2.3	Specifications to consider	68
5.2.4	Data description	69
5.2.5	Limitations	75
5.3	Examples of generated data	78
5.3.1	Linear trend with noise and lag	78
5.3.2	Seasonal (cyclical) data with lead	79
5.3.3	Combining trends and cyclical components	82
5.3.4	Nonlinear combinations of trends and cyclical features	84
5.3.5	Complex nonlinear input and output relationships	87
5.4	Conclusion	90
6	Evaluating interpretations	91
6.1	Overview	91
6.2	Demonstration: TS-MLP on simple polynomial	92
6.2.1	Goal	92
6.2.2	Polynomial description	92
6.2.3	Expected Shapley values	92
6.2.4	Data generation	94
6.2.5	Trained TS-MLPs	94
6.2.6	Interpretations	96
6.3	Important considerations	99
6.3.1	Scaling	99

6.3.2	Reference values	107
6.3.3	Local and global attribution values	108
6.3.4	Sample size	110
6.4	Evaluation metric	112
6.4.1	Methodology	112
6.4.2	Results	114
6.4.3	Discussion	115
6.5	Conclusion	116
7	Test bench demonstrations	118
7.1	Overview	118
7.2	TS-MLP on time-dependent polynomial	118
7.2.1	Goal	119
7.2.2	Methodology	119
7.2.3	Results	121
7.3	TCN on time-dependent polynomial	125
7.3.1	Goal	125
7.3.2	Methodology	125
7.3.3	Results	125
7.3.4	Conclusion	126
7.4	Solar wind – magnetopause standoff distance analysis	128
7.4.1	Background	128
7.4.2	Magnetopause standoff distance	133
7.4.3	Informed magnetopause standoff distance system	135
7.4.4	Ill-informed magnetopause standoff distance system	141
7.4.5	Evaluation metrics	145
7.5	Conclusion	148

8	Conclusion	150
8.1	Overview	150
8.2	Summary	150
8.3	Key findings	152
8.4	Evaluation against research goals	153
8.5	Future Work	155
8.6	Concluding remarks	156
A	Supplemental content	158
A.1	Shapley value calculation	158
A.1.1	Case study: Shapley value applied to two variables	159
A.1.2	Case study: Shapley value with three features	161
A.2	Model configuration	163
A.2.1	Identification parameters	164
A.2.2	Data parameters	164
A.2.3	Architecture parameters	164
A.3	Training configuration	165
A.3.1	Optimisation parameters	165
A.3.2	Batch and data split parameters	165
A.4	Synthetic time series data generation configuration	166
	References	171

List of Figures

2.1	Time series decomposed into its trend, seasonal, cyclical, and random components.	14
2.2	Categorisation of various interpretability techniques used for time series models. The green elements highlight the core focus areas of this study, while the arrows indicate the relationships and derivations between different categories of techniques.	18
2.3	Example from the synthetic dataset of Turbé et al. [34]. Each channel is generated from sinusoidal components with time-varying amplitude, frequency, and phase, often confined to short windows and perturbed by noise. A dataset item is thus a multivariate time series constructed from these sinusoidal components. The classification task is to predict whether the sum of the frequencies of two designated sinusoidal components exceeds a given threshold.	27
2.4	Comparison between the ground truth interaction matrix (left) and average attention weights (β) learned by Seq2Graph (right) on Dataset 4 of Baric et al. [15]. The x-axis shows the influencing time series, and the y-axis shows the target time series. Strong self-dependence (high values along the diagonal) is expected under the ground-truth generative model. In contrast, Seq2Graph's β values are biased toward time series index 0 rather than showing uniform self-dependence, indicating deviation from the ground truth in its attention assignments.	30
2.5	Examples of simulated time series data generated with GRATIS at different intervals (annual, quarterly, monthly, and weekly). For each simulated series, the most similar real-world series from the M4 competition is shown in the same colour. This illustrates that the synthetic data produced by GRATIS can realistically reproduce the patterns, seasonality, and variability found in real-world time series, making it a useful framework for benchmarking and experimentation [16].	32
2.6	The middlebox dataset is generated using various time series processes. The first row illustrates how each feature evolves in different samples, while the bottom row corresponds to the heatmap of each sample, where red indicates informative features [35].	33

3.1	Simplified structure of TS-MLP structure. In this figure, blue blocks denote lagged input features from previous time steps, while the red block indicates the model's forecasted value at the time step.	39
3.2	The type of workflow and the structure of simplified TCN. The input time series data is represented by the blue blocks at the different time steps, and the red blocks represent the convolutional layers that capture temporal features. The last red block represents what the model will predict of the signal on the next time step using temporal dependencies learned by the convolutional filters. . . .	42
4.1	Functional design of synthetic testbed. The purple-highlighted boxes show newly developed systems, and the light blue boxes show pre-existing systems.	57
5.1	Plot of the dynamically shifted feature x_1 using the expression $\frac{(2 \cdot \sin(0.0015 \cdot t))}{(3 \cdot \cos(0.001 \cdot t))}$. The top plot shows the feature effect, and at the bottom, the time shift values at a certain time are illustrated, while some regions appear dark blue due to intervals of very high-frequency oscillations.	73
5.2	Generated input feature data and the corresponding transfer function for a linear trend with noise and a lag.	80
5.3	Input feature generated (blue) and the correlating transfer function output for seasonal data (red).	81
5.4	Input feature generated (blue and green) and the correlating transfer function output (red) for combined trend and seasonal data.	83
5.5	The output of the transfer function combines trend and cyclical components in a nonlinear way using the example values in Table 5.1.	86
5.6	The output of the transfer function combines transformed input features to create complex synthetic time series patterns using the example values in Table 5.1. . .	89
6.1	Output feature of the polynomial from the smallest to largest value of y	94
6.2	Interpretation heatmap of a random representative single sample, showing us how these values might look as a matrix, important to understand for the sections to follow.	97
6.3	A comparison of the G-DeepSHAP Values (DSVs)('DeepSHAP') and Analytic Shapley Values (ASVs) ('Exact Shapley') for x_1 . The 'Ground Truth' corresponds to the polynomial component $f(x_1) = x_1$	98
6.4	A comparison of the DSV('DeepSHAP') and ASVs ('Exact Shapley') for x_2 . The 'Ground Truth' corresponds to the polynomial component $f(x_2) = x_2^2$	99
6.5	A comparison of the DSV('DeepSHAP') and ASVs ('Exact Shapley') for x_3 . The 'Ground Truth' corresponds to the polynomial component $f(x_3) = x_3^3$	100

6.6	Comparison of the TS-MLP interpretations when using scaled (left column) versus unscaled data (right column). The blue points depict the Generalised DeepSHAP (G-DeepSHAP) approximations, the red points represent the Exact Shapley estimations, and the green line indicates the ground truth equation (covered by the red dots). The first row shows the attribution values for x_1 , the second row for x_2 , and the third for x_3 features. Note the green line is hidden behind the red line in the upper right figures	106
6.7	Comparative scatter plots of model interpretations across baseline types. The first column represents SHAP values calculated with the mean baseline, the second column with a zero baseline, and the third column with the G-DeepSHAP-like baseline. Blue, red, and green dots denote the SHAP values for each baseline condition.	109
6.8	A 4x3 grid of images displaying model interpretations. The first row represents results with two baseline samples, the second with 10 baseline samples, the third with 100 baseline samples, and the fourth with 1,000 baseline samples. The first column corresponds to the x_1 , the second column to x_2 , and the third column to the x_3 feature. The side legend indicates colour codes: blue for G-DeepSHAP, green for ground truth, and red for Exact SHAP.	111
6.9	The plot shows the relationship between the mean squared error (MSE) and the number of baseline samples used in calculating G-DeepSHAP. The mean MSE is plotted on the y-axis, with the number of baseline samples represented on a logarithmic x-axis (from 2 up to 1,000). The baseline sample size trend line shows that interpretability performance becomes better as the baseline sample size increases. Note further iterations have shown this plot line to not be as smooth, however follow the same trend	115
7.1	Mean absolute attribution heatmap for time-dependent feature contributions of the TS-MLP.	122
7.2	Average absolute effect of each feature over time.	123
7.3	A comparison of feature contributions for x_1 , x_2 , and x_3 using ASV and DSV from the TS-MLP model.	124
7.4	A comparison of x_1 , x_2 , and x_3 feature inputs with their ASVs (from the polynomial equation), DSVs (DeepSHAP, from the TCN model), and true equation values. Feature contributions are extracted from the model's interpretation matrix and sorted by input magnitude.	127
7.5	OMNI values for $\mathbf{v}_{sw}$ and $\mathbf{n}_p$, with $\mathbf{r}_{mp}$ calculated from the (dashed) original values using Equation 7.3. The shifted $\mathbf{v}_{sw}$ and $\mathbf{n}_p$ features are time-lagged using the simplified time shift into past time steps.	132
7.6	Mean absolute heatmap of interpretations, showing the significance of each feature over time (x-axis starting from one second ago on the right-most tick).	137

7.7	Mean absolute interpretation heatmap of the DSV from the ground truth, evaluated over 1,000 samples of the validation set. The left pane shows feature importance over time, the middle pane highlights feature contributions to specific prediction points, and the right pane captures time step importance to the final prediction.	138
7.8	Evaluation of storm data using G-DeepSHAP: (a) Gaussian smoothing time series absolute feature time points of true ASVs, (b) Model predictions G-DeepSHAP time series absolute points for time points. Both figures use 1,000 sample points for reference values as outlined in Section 6.3.4.	139
7.9	Mean absolute interpretation heatmap of the ASVs from the ground truth, evaluated over 1,000 sample points.	141
7.10	Evaluation of storm data: true and predicted values, alongside the sum of DSVs, using 1,000 sample points as discussed in Section 6.3.4. Only evaluation set points between sample indices 40,344 and 41,344 are plotted here.	143
7.11	Mean absolute heatmap of feature contributions for the model without time shift information.	144
7.12	Time step distribution for the most relevant features across samples, showing the model's response to the lack of explicit time shift information.	144
A.1	Six ways of forming a three-player coalition	163

List of Tables

2.1	Summary of synthetic time series interpretability test benches	35
5.1	Summary of configurable parameters for synthetic time series generation.	72
6.1	Hyperparameter values selected through manual tuning for optimal model performance of a TS-MLP for no temporal dependencies	95
6.2	Model training configurations and corresponding MSE values for a Multilayer Perceptron (MLP) with unscaled data	105
6.3	$\mathcal{L}_{MSE}^{\text{mean}}$ for different sample sizes of a MLP model trained on simple polynomial	114
7.1	TS-MLP model training hyperparameters for polynomial with temporal effects	120
7.2	TCN model training hyperparameters for polynomial with temporal effects	126
7.3	TCN model training hyperparameters for storm data with temporal effects	136
7.4	Per-sample evaluation metric	146

List of Abbreviations

AED Average Exact Difference

AI Artificial Intelligence

ASV Analytic Shapley Value

AUCSE Area Under the Curve of the Score Area

CNN Convolutional Neural Network

DeepLIFT Deep Learning Important FeaTures

DL Deep Learning

DNN Deep Neural Network

DSV G-DeepSHAP Value

GRU Gated Recurrent Unit

G-DeepSHAP Generalised DeepSHAP

LIME Local Interpretable Model-Agnostic Explanations

LSTM Long Short-Term Memory network

LRP Layer-wise Relevance Propagation

ML Machine Learning

MLP Multilayer Perceptron

MSE Mean Square Error

NITheCS National Institute for Theoretical and Computational Sciences

QII Quantitative Input Influence

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SHAP SHapley Additive exPlanations

TCN Temporal Convolutional Network

TS-MLP Time-Shifted Multilayer Perceptron

VAR Vector Autoregressive

Chapter 1

Introduction

‘The formulation of the problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skill.’ — Albert Einstein

As Machine Learning (ML) is increasingly applied across various domains, offering users trustworthy explanations of model behaviour becomes essential. Deep Neural Networks (DNNs), specifically, are often employed in high-risk domains, such as autonomous driving and the health-care sector, which makes it crucial for humans to comprehend and be able to explain how the algorithms function [1, pp. 17, 38].

Many techniques have been devised to interpret the predictions of trained DNNs, particularly by identifying the critical feature combinations that inform their decision-making processes [1, p. 236]. The literature does not consistently differentiate between ‘explainability’ and ‘interpretability’. The two terms are, therefore, considered almost interchangeable. Following Molnar [1, p. 15], we use the terms interpretability and explainability interchangeably to refer to the extent to which a model’s decisions can be understood by humans. Additionally, we will be using the term ‘explanation’ to describe reasoning about individual predictions.

Comparing different interpretability strategies is generally complicated since there is usually no gold standard to measure against, especially for multivariate time series data, which consist of different variables that evolve with time. These variables are not only dependent on their past

results but also depend on other variables [2]. Furthermore, the visualisation of interpretations in multivariate time series regression tasks is not as intuitive as laying saliency maps on images or highlighting certain related words in the text [3].

Establishing true attribution values is a significant hurdle when assessing feature attribution techniques on real-world data. To address this, we develop a system that provides a synthetically controlled test environment for evaluating and comparing feature attribution techniques. Although the system is capable of assessing the efficacy of these techniques under various different controlled conditions, we focus on demonstrating its potential using simplified use cases.

1.1 Background

Time series data contains various observations that connect past values to future predictions through time-stamped measurements. A broad range of industries today examine patterns through time series data to make predictive insights across various fields, from finance to weather forecasting and healthcare. Time series data analysis faces challenges such as capturing trends, seasonality, and evolving patterns over time. Special techniques exist for the appropriate analysis and interpretation of this type of data.

1.1.1 Time series data

Time series data can be described as a set of observations made over time, usually at equally spaced intervals [4, p. 4]. Such data is used in various fields, such as finance and economics, meteorology, medicine, and others [5, p. 170]. Because data observations are ordered temporally, a time series exhibits unique characteristics, making the time variable a key factor and equally important in the analysis.

Time series data often exhibits characteristics of temporal dependence, seasonality, trend, cross-correlation, and stationarity. These are defined as follows:

- **Temporal dependence:** This means that the current data point is affected by the previous data points [4]. For example, one might find that today’s temperature is similar to that of the previous day since the temperature increases or decreases gradually.

-
- **Seasonality:** Seasonality can be defined as periodic changes that are cyclic; that is, they occur at certain intervals [4, p. 51]. Shop sales, for instance, grow during the holidays.
 - **Trend:** The general direction or pattern of data over time is referred to as a trend [4, p. 50]. For example, the global temperature has risen slowly, historically, but is predicted to increase.
 - **Cross-correlation:** Analysing multiple time series data simultaneously may show that one series influences another. For instance, the rise of oil may lead to an increase in the price of goods [4, p. 59].
 - **Stationarity:** A stationary time series is one whose statistical properties are constant with time. For data that are non-stationary, the statistical properties change; they also need further operations to make them stationary for use in modelling [4], [6].

1.1.2 Deep learning

Artificial Intelligence (AI) is the common name for a broad set of methods aiming to create machines with certain faculties that humans commonly recognise as ‘intelligence’. ML, a subset of AI, represents techniques that learn from data in order to perform better over time. The ML technique we focus on is Deep Learning (DL), which is a subset of ML that uses a neural network with multiple layers to learn complex patterns in larger datasets.

A Multilayer Perceptron (MLP) is one of the simplest kinds of DNNs, which has an input layer, one (or more) hidden layers, and an output layer. Every layer is made up of neurons, in which every neuron transforms the input it receives and performs a mathematical operation on that input (a weighted sum that is passed through a non-linear activation function). Then, the output is transmitted to the following layer. The whole network is able to learn and make predictions from data [5, p. 35].

Various DNN architectures are adapted to time series modelling. These include Recurrent Neural Networks (RNNs) that were developed to model sequences using hidden states. Important variants are Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs) [7]. Currently, popular techniques include Temporal Convolutional Networks (TCNs) and transformers. In addition, a simple MLP can be adjusted to model sequences by providing it with both

past and current inputs. These architectures are described in more detail in Section 2.3.

1.1.3 Challenges of interpretability in deep learning

Even though these models are successful, DL models are often referred to as ‘black boxes’ in that they can make accurate predictions, but it is not clear how they make their decisions. This transparency is lacking in challenging applications such as healthcare, finance, and autonomous driving, where they do not just need to produce correct predictions but also need to provide the rationale behind a decision.

As mentioned earlier, ‘interpretability’ and ‘explainability’ are used interchangeably to describe a model’s capacity to provide insights into its behaviour and decisions [1, pp. 15–21]. Several approaches to building DL models that can be explained assume users should be able to understand what models concluded and, more specifically, how specific input attributes contributed to these conclusions [1, pp. 15–21].

When ML models are used in high-risk applications, such as healthcare or autonomous systems like self-driving cars, it is crucial to understand and explain their behaviour to ensure trust and accountability. Model exposure serves both purposes of achieving user trust as well as enabling developers to debug models while identifying bias, in addition to fulfilling regulatory transparency needs. In particular, these techniques are useful for complex models, including DNNs for which the internal mechanisms are often obscure to human understanding. This plays a role in establishing characteristics such as [8]:

- **Accuracy:** Interpretation methods need to generate explanations that are usable for new data collections while also delivering accurate representations of model decisional reasoning.
- **Fidelity:** Model explanations need to show authentic model behaviour instead of using simplification or approximations.
- **Consistency:** All models used for a single task need to generate comparable explanations to maintain consistency between training instances.
- **Stability:** When inputs change slightly, the explanations provided for similar forecasting

predictions should remain consistent.

- **Novelty:** Technical interpretation systems should reveal distinctive model behaviour features, especially during new model versus existing model comparisons.

1.1.4 Interpretability techniques

Interpretability techniques are not replacements for ML models: they assist in explaining these models. One may think of interpretability techniques as models by themselves used to explain ML models.

Instance-specific insights into the individual features that contribute to a particular model’s prediction of a specific instance are given by feature attribution methods [1]. This is typically expressed as the change in the model’s output compared to a reference output when a feature takes on a specific value; thus, the effect can be positive or negative. In additive feature attribution methods, the feature attributions sum to the predicted value plus the specific reference used. (This implies that – for techniques where the reference value is 0 – the feature attributions automatically sum to the output value.)

On the other hand, feature importance is the relative contribution of each input variable to the output, regardless of whether the influence is positive or negative. Feature importance is often expressed as a percentage and reports a global perspective by aggregating contributions over a set of predictions to reason about how each feature helps the model perform [9].

Together, these methods provide complementary views: feature importance typically provides a global perspective (for the model itself), while instance-specific feature attributions typically provide a local perspective. In Section 2.4, we provide an expanded elaboration of interpretability techniques organised according to their taxonomy [1].

1.1.5 Time series interpretability

Interpretability is, to some extent, well explored in other fields of ML, such as vision and language processing [10]. The main focus of this dissertation concentrates on time series data because they create specific interpretability challenges. The way time series differ from other

data types requires an ability to detect temporal patterns where present data affects current observations. Such a temporal structure typically makes conventional interpretability methods ineffective. These challenges have been studied in the case of time series data in several prior works. For example, the studies by Ismail et al. [3] and Barredo et al. [11] show that original static interpretability approaches such as SHapley Additive exPlanations (SHAP) and Local Interpretable Model-Agnostic Explanations (LIME) are not easy to extend to consider temporal dimensions. The analysis of feature importance through these models lacks embedded temporal features, because of which their analyses cannot detect essential time-dependent dependencies. Lundberg and Lee, in their SHAP-based research, also note that simply integrating SHAP with dynamic data types such as time series means making modifications that are currently still in development [9].

Recently, transformer models have offered an improved way of working with sequential data (such as time series). Transformers use self-attention to learn the weight of each time step. As a result, transformers can learn these time steps and capture long-range dependencies and complex interactions that traditional time series models (such as RNNs or LSTM networks) handle poorly. However, transformers are not easy to interpret over time series of data. These advanced models also require more specialised interpretability techniques [12], [13].

Although progress has been made, there is still a considerable gap in the development of interpretability techniques, specifically for time series. Many existing methods designed to explain static data (see Che et al. [14]) fail to capture dynamics and, therefore, produce incomplete or inaccurate explanations in most cases. Such a limitation puts time-sensitive applications at risk because an incorrect interpretation can have serious consequences. A gap also exists in prior work concerning measuring feature importance on temporal datasets, with recent work by Ismail et al. [3], Baric et al. [15], and Kang et al. [16] seeking to fill this gap using state-of-the-art interpretability tools.

This work was conducted as part of the National Institute for Theoretical and Computational Sciences (NITheCS) project ‘Knowledge Discovery in Time Series Data’. The aim here is to design and assess methods for extracting new information about intricate tasks from DNN-based systems. The ‘KnowIt’ toolkit¹ was consequently designed to train models, acquire interpretations, and visualise the outcomes.

¹<https://github.com/MUST-Deep-Learning/KnowIt>

1.2 Problem statement

Most interpretability techniques were not developed in the context of time series data but were instead applied to time series data after being developed in other contexts, such as computer vision. It is unclear how well these techniques perform when explicitly applied to time series data, to correctly provide reasons for predictions of deep learning models.

1.3 Research questions

The above problem statement raises the following main research questions:

1. What aspects (such as temporal dependence, seasonality, or stationarity) should be considered when examining interactions in time series?
2. What criteria would a test bench need to adhere to to be valid for testing interpretability techniques?
3. How can a platform be designed to test the accuracy of time series interpretability techniques?
4. What practical considerations influence the accuracy of the evaluation process?
5. How is a time series test bench itself evaluated?

1.4 Objectives

The primary objective of this study was to construct a synthetic test bench to assess the performance of feature attribution techniques in interpreting DL models applied to multivariate time series data.

The specific objectives of this research were to:

- Define characteristics of the synthetic data: while keeping key properties in perspective, also addressing aspects such as noise levels, interaction time delays, and types of noise.

-
- Design and generate synthetic data with specific characteristics.
 - Develop quantitative metrics for evaluating the effectiveness of feature attribution techniques.
 - Design an end-to-end process for evaluating the accuracy of feature attributions and implement it as a test bench.
 - Demonstrate the use of the test bench for evaluating feature attribution techniques.

1.5 Project scope

To ensure the study remains focused, we limited the scope as follows:

- Interpretability techniques: Interpretability techniques are limited to feature attribution.
- Dataset: The study evaluates feature attribution techniques using synthetic and limited real-world datasets, such as those used to predict solar storms.
- Toolkit: As far as possible, we used well-established packages and libraries. Frameworks such as the KnowIt toolkit were used to train DL models. These primary frameworks also integrate and adapt other relevant toolkits.
- Architectures: The primary emphasis is on classification, with architectures primarily being TCNs and Time-Shifted Multilayer Perceptrons (TS-MLPs), a type of MLPs adapted for time series.

1.6 Research methodology

In this work, we pursue an experimental approach to quantify the usefulness of modern feature attribution techniques in the context of DL-based multivariate time series analysis. The focus is on understanding how the characteristics of time series data influence the performance of attribution methods, motivating the need for tailored strategies.

The methodology consists of the following key components:

-
- **Literature review:** We start by reviewing key topics, including time series data and interpretability methods, and TCN-based deep learning models. This then gives the necessary theoretical foundation.
 - **Iterative experimental development:** The research follows an iterative approach to hypothesis testing. With each stage, the experiments are progressively refined, adding increasing complexity. To do this, certain hypotheses are specified, controlled datasets are designed, model performance is evaluated, and the experimental setup is refined according to results.
 - **Quantitative analysis:** The effectiveness of attribution techniques is assessed using quantitative metrics such as correlation coefficients, mean absolute error (MAE), and root mean squared error (RMSE). In addition, expert feedback is incorporated to assess interpretability in relation to known relationships in the synthetic datasets.
 - **Real-world application:** Beyond synthetic data experiments, the system is demonstrated on a real-world problem. Specifically, we collaborate with NITheCS on a case study involving the prediction of solar storm events, using experimental findings to improve interpretability in practical applications.

This methodology is structured to systematically explore the strengths and weaknesses of feature attribution techniques for time series data, bridging the gap between theoretical understanding and real-world applicability.

1.7 Research design

The following steps define the main experimental process:

- **Establishing ground truth attributions:** We design synthetic datasets with well-defined feature-to-outcome relationships. This permits the evaluation of attribution methods against a known reference for precise results.
- **Synthetic data generation:** Time series datasets are generated using NumPy and pandas libraries with controlled interaction levels, noise levels, and time lags. The datasets that we use here are meant to be realistic, but with some key variables controlled.

-
- **Model training:** DL models, specifically TCNs and MLPs, are trained on the synthetic datasets to establish baseline performance. These models serve as reference points for evaluating attribution techniques.
 - **Feature attribution methods:** Using post-hoc attribution techniques like SHAP and DeepSHAP, we explain the predictions of trained models to understand their interpretability.
 - **Performance evaluation:** Attribution methods are evaluated using multiple metrics, developed and refined as part of this study.
 - **Iterative refinement of experiments:** The experiments are iteratively modified to include additional complexities. Refining dataset characteristics, such as noise levels or time dependencies, tuning attribution method parameters (e.g., number of samples for SHAP), and reevaluating evaluation metrics are part of this.
 - **Real-world application execution:** Finally, we apply the developed methodology to a real-world case study in collaboration with NITheCS, using experimental insights to improve interpretability in the context of solar storm prediction.

This provides a structured workflow for probing the performance of feature attribution techniques on time series data.

1.8 Dissertation overview

The remainder of this dissertation is structured as follows:

- **Chapter 2: Background**

This chapter offers background on time series data by defining it, explaining why it is complicated, and describing where it can be used. It also covers various time series models and provides an overview of attribution techniques, especially Shapley values. It concludes with a brief literature review on synthetic data set generators and existing synthetic time series test benches.

- **Chapter 3: Methods and techniques**

Here, the methods and techniques used in the study are introduced in more detail. TS-MLPs and TCNs are described and discussed. Detail is also provided on the following interpretation techniques: Analytic Shapley Values, Exact Shapley Values, DeepLIFT, DeepSHAP, and reference values.

- **Chapter 4: Test bench design**

This chapter provides an overview of the test bench’s requirements, structure, design, and implementation. It includes descriptions of each component’s functionality and configuration, and the test bench’s inputs and outputs.

- **Chapter 5: Synthetic time series data**

This chapter describes the process of designing the synthetic dataset generator. It provides examples of generated data and discusses options to configure these datasets.

- **Chapter 6: Evaluating interpretations**

In this chapter, a metric to evaluate interpretations is developed using a simplified setting: an MLP on simple polynomial datasets. This extends to the analysis of a TCN trained on data generated using a time-dependent polynomial.

- **Chapter 7: Test bench demonstrations**

The intended use of the proposed test bench is demonstrated through the evaluation of MLP and TCN models with polynomial and other nonlinear time series data.

- **Chapter 8: Conclusion**

Finally, the study is summarised, and its conclusions are presented regarding overall relevance and purpose, suggestions for further research, and practical implications.

In conclusion, the test bench is a platform that can be used to evaluate (and potentially improve) the accuracy and effectiveness of interpretability techniques for time series data processing.

Chapter 2

Background

‘Understanding is the first step to acceptance, and only with acceptance can there be recovery.’
— J.K. Rowling

2.1 Overview

This chapter provides background to the study of this dissertation, but technical details will be explained in Chapter 3. It defines time series data and describes approaches to modelling it using DL. Concepts regarding the interpretability of time series models are reviewed before discussing closely related work, such as existing synthetic time series test benches.

Section 2.2 defines what time series data are, and Section 2.3 discusses DL models. Section 2.4 discusses techniques that can be used to interpret these DL models, and Section 2.5 discusses some important studies.

2.2 Time series data

As introduced in Section 1.1.1, time series data consist of a set of items indexed in time order [4, p. 4]. For instance, a time series enables one to understand what causes variation in certain variables from one period to another. As an introductory example, we illustrate a time series equation with input features that are dependent on each other and on time. We show the Vector Autoregressive (VAR) model equation representation [17, p. 13]:

$$z(t) = c + A_1 z(t-1) + A_2 z(t-2) + A_3 z(t-3) + \dots + A_p z(t-p) + \varepsilon(t)$$

where $z(t)$ is a vector of time series values, c is a fixed number, $A_1, A_2, A_3, \dots, A_p$ are matrices of VAR coefficients, p is the order of the VAR model, and the residual errors are represented by $\varepsilon(t)$.

In this example, one can observe the features of the VAR model that account for temporal relations that exist in the time series through previous steps that impact the current values. There are two types of time series data: stationary and non-stationary. Identifying these characteristics is vital in selecting the appropriate analytical methods [4, p. 14]. Non-stationary time series data display changing statistical characteristics over time, such as trends or seasonality, leading to instability in mean and variance [6, pp. 221–234]. Here, more than just random error influences the data's variability. In contrast, data that is defined as stationary has a constant mean and variance, as well as autocorrelation characteristics that are not a function of time. This is easily dealt with analytically. In stationary time series data, random errors are the only source of variation. One such method of working with stationary time series is classical decomposition, which decomposes a time series over a given period into four basic components [4, pp. 50–51]:

- **Trend:** This is a permanent shift in the level of a time series. It refers to a progression that is positive and firm of a time series that has a definite inclination.
- **Seasonality:** From the perspective of cyclicity, seasonality means activities or phenomena that are performed or happen at a fixed time interval, say monthly or quarterly. This component basically captures fluctuations that repeat themselves over a period of a week, probably due to seasonal changes.
- **Cyclical:** The cyclical component resembles the seasonal one but differs in that it does not repeat at fixed intervals. Instead, it represents long-term fluctuations around the

trend line, often linked to phenomena such as business cycles. For example, an economic recession may take 8–10 years to run its course, whereas seasonality recurs in a predictable calendar pattern.

For example, Figure 2.1 illustrates a yearly employment data time series decomposed into its trend, seasonal, cyclical, and irregular components. Note that the classical approach, depicted in Figure 2.1, assumes stationary time series data.

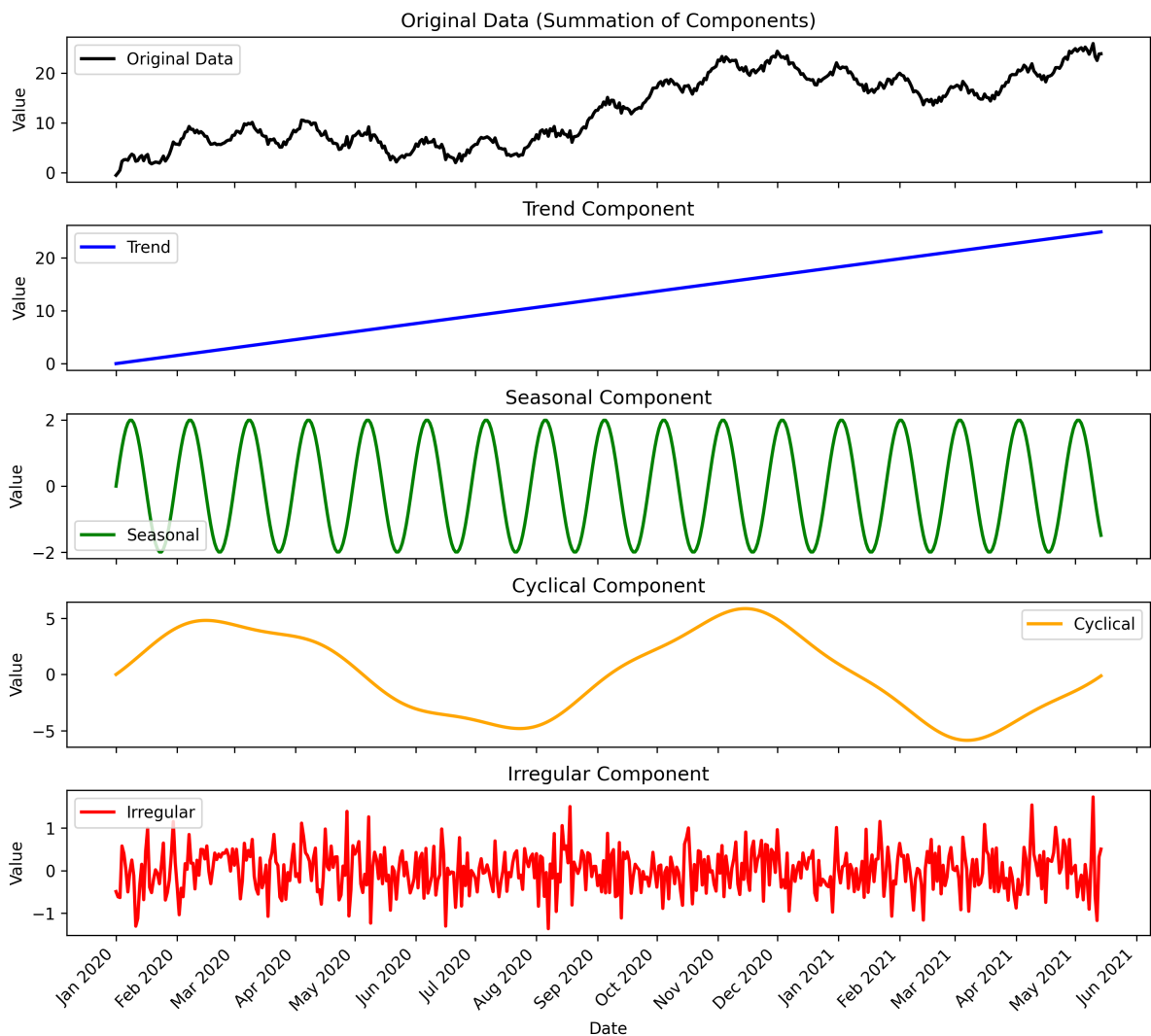


Figure 2.1: Time series decomposed into its trend, seasonal, cyclical, and random components.

Time series data can also be grouped by the number of time-dependent variables:

-
- **Univariate time series** involves a single time-dependent variable.
 - **Bivariate time series** contains two time-dependent variables.
 - **Multivariate time series** contains multiple time-dependent variables where each variable must depend on its own past to incorporate the data of other variables at the same time.

This categorisation is very helpful in selecting the right models and the right method of decomposition, considering the complexity and structure of time series data. Some of the multivariate time series data include climatic data, economic data, and data monitoring a patient’s health [4], [18], [19].

2.3 Time series modelling

Time series models are continuously refined to improve the accuracy of future predictions [2]. In general, time series analysis refers to the process of evaluating time-dependent data to extract useful statistics and data features. There are three main tasks commonly performed on time series data:

- **Time series classification** assigns a label to an entire sequence, based on patterns in the input time series [2], [20].
- **Time series regression** predicts a continuous value from one or more input variables, using relationships learned from past data.
- **Time series forecasting** is a specific type of regression where the goal is to predict future values of the same variables, using their previously observed values [6, pp. 103–156].

In forecasting, we use past observations to forecast future values for the target variable, and this may or may not include historical values of the target variable and possibly some more related variables. Specifically, we apply a regression model upon this relationship to forecast future time steps given new input.

Over the last few years, several DL models have been proposed to model time series data. Also note that these architectures are developed to cope with the inconvenience of long-term

dependencies, dealing with sequences with diverse length scales, and boosting the efficiency of computation. In this section, we provide a brief overview of some of the most important models related to the current research and connected to the methods applied in this work. In Chapter 3, we discuss the models used in this dissertation in more depth.

2.3.1 Recurrent Neural Networks

Among the DL architectures designed for sequential data, RNNs are noteworthy. They can process inputs in a recurrent fashion, making them suitable for capturing temporal relations, which are essential in time series analysis. However, RNNs have some drawbacks, such as vanishing gradients, which make learning long sequences difficult. This has led to the improvement of other models such as LSTMs and GRUs, which introduced ways of keeping and controlling information over longer sequences. Although these models are widely used in the field, they are not the main focus of this study [7], [21].

2.3.2 Time-Shifted Multilayer Perceptrons

One of the most common neural network types is the MLP, and although it is more commonly used for static data, it can be applied in time series forecasting. By lagging the input features, TS-MLPs can incorporate the temporal dynamics into the model without complicated connections such as recurrent or convolutional connections. Even though TCNs and RNNs are more elaborate, TS-MLPs are useful for specific time series problems, especially those where the focus is on short- to medium-term relationships [22], [23].

In this work, TS-MLPs are used alongside TCNs, which gives another perspective to the time series forecasting problem. Because they are easy to implement and fast to compute, they are useful when there is not a great deal of historical information in the data.

2.3.3 Temporal Convolutional Networks

TCNs have convolutional structures, which makes it possible to work with temporal data. While TCNs applies convolutions to the data they take as input, they can work with long sequences of

input data without the recurrent connection. This can lead to exploding gradients, which occur when the gradients in a neural network become too large during training. These architectures are particularly beneficial for tasks where past events have long-term effects and are employed in several time series forecasting domains [24]. TCNs have been employed in this dissertation because they enable the handling of long-term dependencies while simultaneously minimizing the computational complexity that is involved [24], [25].

2.3.4 Transformers

Transformers, first introduced in the context of natural language processing, have recently been applied to many other sequence-based problems because of their capacity to capture long-range correlations via a component called self-attention [13]. RNNs and TCNs work sequentially while transformers work on the entire input at once, consequently making them suitable for large data sets. Owing to their adaptability and high-end results on various tasks, they are used extensively in more complicated time series tasks. Although transformers are the state-of-the-art solution, they are not the main concern of this dissertation.

2.4 Interpreting time series models

Figure 2.2 provides a high-level representation of various techniques for interpreting time series models, particularly those involving DNNs. It categorises the interpretation approaches into several areas, which are explained in the subsequent sections. We focus on the green elements illustrated in Figure 2.2.

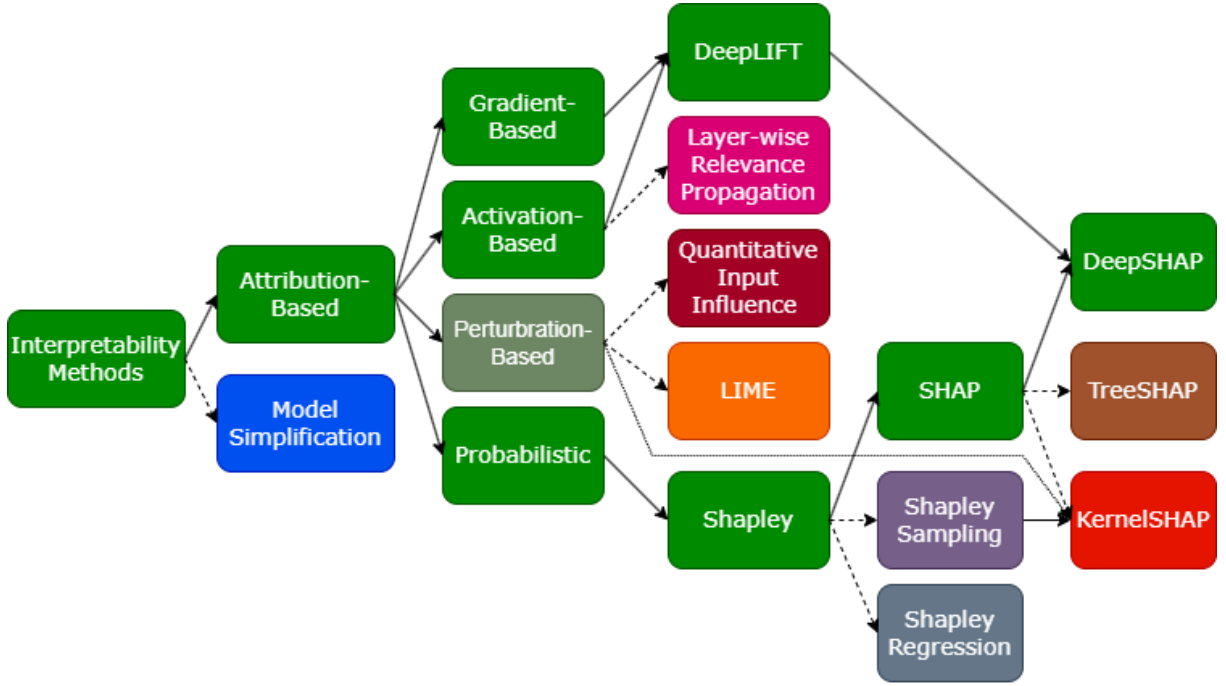


Figure 2.2: Categorisation of various interpretability techniques used for time series models. The green elements highlight the core focus areas of this study, while the arrows indicate the relationships and derivations between different categories of techniques.

Post-hoc interpretability is concerned with applying methods of model explanation that would become effective only after the model has already been built. We focus on two categories: attribution-based strategies and model simplification strategies [1], [9, p. 21], although many more exist.

Attribution-based methods examine how much input each feature has had on the output of the model and how much data has affected the result. For example, methods for estimating feature attributions estimate how much each feature contributed to predictions for certain tasks of a model. These methods are helpful in tracking how models could come to such a conclusion, as they show the process with which certain conclusions are made step by step [1, p. 15].

Then there are model simplification techniques that attempt to reverse engineer a simpler, more easily understandable model of the feature relevance while still preserving accuracy. These methods are further classified into two types: They are: model-agnostic, which can be used for any model, regardless of its type, and model-specific, which is developed to be used for some types of models [1, p. 1].

In this section, we introduce feature attribution techniques in the context of time series and regression models. Section 2.4.1 outlines the motivation for interpretability in these settings. Section 2.4.3 provides an overview of major attribution categories, including gradient-based, perturbation-based, activation-based, and probabilistic methods. Section 2.4.2 organises these methods into a structured taxonomy. Section 2.4.5 explains key techniques such as SHAP, DeepSHAP, and LIME, with a focus on their application to time series. Finally, Section 2.4.4 discusses how attribution methods are evaluated for reliability and relevance, and addresses common challenges and solutions.

2.4.1 Motivation

In time series and regression models, feature selection is important to interpret the output of the model. The issue is that in several datasets, the features that may prove useful at one period of time will either not prove useful at another, or they may become useful. This leads to static feature selection methods that are unaware of the changing data pattern, losing potential model performance.

For this reason, dynamic feature relevance assessment is required for maintaining the model performance, hence the use of feature attribution techniques. In other words, they allow one to show which features are important factors, and with those important factors, to help change the models when the importance of features changes. It makes sense of the results, catching model bias and going for on-the-fly reoperations on the fairness front.

2.4.2 Feature attribution of DNNs

Feature attribution in DL is a way to quantify how much a given feature contributes to the model’s predictions. The techniques produce metrics that define the impact each feature has on the predictive model. Shapley values distribute contribution scores among features based on their marginal contributions to the prediction, following fairness principles defined in cooperative game theory. This results in local interpretability as it shows how features contribute to particular forecasts. Conversely, collating these interpretations across multiple predictions provides global interpretability, revealing broader feature importance trends [26].

In the field of neural networks, this approach measures the significance of every input factor in the production of the model result [26]. Thus, it helps users understand the network decision-making process, allowing them to identify which features influenced the prediction.

The feature attribution methods inspect prediction model results through feature input alterations while using baseline reference values for comparison. These methods enhance model interpretability, especially in complex DL models where internal processes are not easily understood [27]. Feature attribution techniques quantify the feature’s contribution to a decision made by a model. In this way, these methods show which input features or the regions of the input are more important to the model for generating particular predictions [27].

Feature attribution techniques can be broadly classified into several categories [28]:

1. **Gradient-based:** These estimate the gradient of the prediction of the model with respect to features to see which features are important. To do this, they perform reverse mode differentiation and propagate gradients back through the network to study how the output reacts to variations in certain features. This approach can be used to clarify which time series features or which regression inputs were the most important for the model output [1], [9].
2. **Perturbation-based:** These techniques estimate the effect a specific feature produces on the model’s output prediction. They change one thing at a time and see how the output will change if that one thing changes vs. everything else. In particular, this is useful for time series or regression models, where both the passage of time and changes in features affect the prediction [1].
3. **Activation-based:** These techniques interpret neural network activations on the intermediate layer level for feature contribution analysis. On the other hand, they explain how the features are important by visualising the activation level of neurons at different layers of the network for different inputs of the features.
4. **Probabilistic:** These methods use Bayesian or Monte Carlo approaches to estimate feature contributions. They are particularly useful when time series data involve variability or uncertainty, since feature relevance is expressed in probabilistic terms.

Gradient-based and perturbation-based are the two most common feature attribution methods in

practice. Gradient-based methods use a gradient to work out how much the features input to the model need to change in order to improve the model outputs. When dealing with complicated systems such as neural networks, this approach is extremely helpful in discovering the most important features. On the contrary, the perturbation-based method flips the roles, modifying the input features and observing the model output thereby. For example, in time series data, we can investigate the effect of this individual time step, or feature, on the model output by manipulating it.

Feature relevance methods also include additive attribution methods. Here, the methods identify contributions to a model result from specific features such that the sum of all feature scores equals the model result [1, 146–162]. Additive attribution may be useful for understanding how much of each particular feature or time point contributes to the output, which can be very valuable in time series contexts.

2.4.3 Specific feature attribution techniques

An interesting approach with regard to attribution methods is that of Shapley values. Shapley values ensure that each input feature is fairly credited for its role in the model’s predictions and provide a reliable way to measure feature importance [1]. Attribution methods aim to show how a model makes its predictions by giving each input feature a share of the credit. Some methods do this by changing feature values, others by looking at gradients or tracing values back through the network, but the overall goal is always to make complex models easier to understand. Shapley value is used to measure the contributions of input features to the output of an ML model at the instance level. The concept is discussed later in Section 2.4.5 but is also further detailed below.

In the context of time series problems, various attribution methods and their variants provide unique ways of interpreting model predictions. Below, each technique is described in terms of its underlying approach and how it can be applied to understand the contributions of input features:

- SHAP is derived from the use of Shapley values in estimating the proportional contribution of each feature to the model result. SHAP is distinguished from other attribution methods

because it satisfies key properties such as local accuracy, consistency, and fairness, which make the interpretation of individual predictions reliable [1], [9]. SHAP can explain, in a case study of time series forecasting and regression, how each feature affects the prediction at each time step or data point. This proves useful in understanding what features (temporal or otherwise) affect model outputs.

- Shapley regression values are designed for regression models, in particular, adapting the general framework to the properties of regression tasks. Their benefit lies in their refined interpretation of how different features may contribute to the logged values, which is good for accurately explaining the regression model behaviour [9].
- Shapley sampling values: These are an alternative to calculating actual Shapley values for time series data sets, which would be computationally expensive. Evaluation of the output of the model in a random subset of input features is performed by selecting the input features in a subset at random. Each feature’s average contribution across all subsets is an approximate Shapley value [9].
- Layer-wise Relevance Propagation (LRP) is a way to trace back the predicted value of a given network by propagating the relevance of the output neuron through the layers and assigning a relevance score to each neuron according to its contribution to the output [29]. On its own, LRP is useful for deep learning models, including applications in time series analysis.
- Quantitative Input Influence (QII) assesses how sensitive a model is to its input features and declares which are most important in determining the output of the time series model. Unlike previous methods of general feature attribution, QII takes into account variability and distributional aspects of input data to assign a probabilistic measure of feature importance. QII can be quantified by techniques such as sensitivity analysis, given changes to the output as a result of perturbations to inputs, and Shapley values that fairly distribute the variance of the output across features [1], [9]. Finding this can be useful to understand what’s happening deep inside the DL model and find where that model might be improved.
- LIME, explains individual predictions by building a simple, interpretable model around the instance we want to understand. It works by slightly changing the input data and observing how the predictions change, then fitting a simple model such as linear regression on this perturbed data. In this way, LIME shows which features are most influential for

that particular prediction, which is especially useful when working with complex black-box models [1, 168–171].

- **Deep Learning Important Features (DeepLIFT)** is a useful tool for interpreting a complex model that is found in a neural network used to perform time series forecasting, and is a comparison of the output of a neural network with a reference input and attributes differences to the input features. We use this technique to explain the predictions without depending on the details of the internal working of the network [27].

These Shapley value variants and attribution variants are important tools for making ML interpretable and explainable. In time series, they offer a deeper insight into feature contribution, as well as the impact on model prediction, which helps to improve the understanding of decision-making processes.

2.4.4 Evaluating feature attributions

Quantifying interpretability in DL is difficult, and work on the evaluation of interpretability of DL have been done [1, p. 25]. Key criteria for an ideal trustworthy interpretability technique should include the following as mentioned in Section 1.1.3 [8]: Accuracy, fidelity, consistency, stability and novelty.

Two approaches are commonly used to evaluate interpretability methods effectively: measuring against ground truth and perturbation-based evaluation (also known as masking).

- **Measuring against ground truth:** This approach involves comparing the results of the interpretability technique against the actual explanations, normally garnered from domain specialists’ expert opinions or by human intuition. This assesses to what level the method incorporates the technique with human conforming cognitive processes and knowledge [30].
- **Perturbation-based evaluation (masking):** We take this approach by removing or ‘masking’ a portion of the input or model parameters and observing how the model’s predictions and explanations change. This method evaluates the sensitivity of the interpretability technique to variations of the input parameters or the trained model and robustness to a small perturbation. The Parameter Randomization test, part of the class

of perturbation-based evaluations, is shown to be effective for some datasets and models but not for others, suggesting such tests should be used on a case-by-case basis [31].

Additional evaluation variations are ablation studies, which systematically remove or mask parts of the input to see their effect on model performance, along with metric-based evaluations and user studies [3], [32]. Gevaert et al. [30] make a case for choosing the right metrics and methods given specific use cases and datasets through their general benchmarking approach.

Criticism

Evaluation of interpretability techniques often relies on approaches like ground truth comparison or perturbation-based evaluation. It is difficult to achieve a ground truth explanation in practice, and ground truth-based measurements are criticised for being too resource-expensive and time-consuming, as well as limited to a certain number of scenarios or use cases [9]. Perturbation-based evaluation is criticised as being difficult to interpret, sensitive to the particular perturbation used, not informative if the model was not trained on similar examples to the perturbed example, not informative in terms of human understanding, hard to decide what perturbation level to use, and – in general – sensitive to the particular perturbation used [9]. The explanations of a model are best interpreted if evaluated with a number of different techniques. The selection of evaluation techniques depends on the interpretability techniques employed and the problem domain of interest [28].

2.4.5 Shapley values

Shapley values, as introduced in Section 2.4.3, can be assigned to Lloyd Shapley. Shapley developed this concept in 1953 when dealing with cooperative game theory to help explain how features or players of a game contribute to the final result [33]. In ML, they help interpret the predictive model by providing an average contribution of each feature across. This ensures a balanced view of feature importance and supports model explainability.

Computing Shapley values can be demanding because the number of possible feature subsets grows rapidly with high-dimensional data. In response, several practical implementations have been proposed that enable near-optimal solutions while preserving the method’s interpretability

advantages. Here, we focus on the conceptual foundations of Shapley value computations and their role in interpretability, while Chapter 3 provides the detailed mathematical formulations.

Shapley values variants

To overcome computational challenges in large models, key development approaches are:

- **SHAP (Shapley Additive exPlanations)**: Provided is an approximation-based method for Shapley value computations with some acceleration algorithms specific to the models.
 - **KernelSHAP**: Approximation of Shapley values by a model-agnostic approach based on weighted linear regression.
 - **TreeSHAP**: The specializations were designed for tree-based models like random forest, and gradient boosting machines, and exact Shapley output could be computed in polynomial time.
- **Shapley sampling**: Randomly samples feature subsets and exposes an approximation of Shapley values with less computing time.
- **Shapley regression**: Reduces the non-possibility of using Shapley values in the regression models to tell people about individual predictors' contribution to the model output.
- **DeepSHAP**: DeepLIFT and Shapley values are integrated in order to explain DL models. DeepSHAP optimises the Shapley value computations using backpropagation and emphasises the significance of inputs as a means to disembodiment neural networks.

These implementations thus enable the use of Shapley values for complex models and large data sets, preserving the practicality of the approach to understanding feature contributions in high-dimensional situations.

2.5 Synthetic time series interpretability test benches

Evaluation and validation of interpretability methods of time series data can utilise synthetic time series test benches. These frameworks produce synthetic datasets and evaluation scenarios.

This review describes how synthetic test benches can be used to tackle interpretability, examining relevant studies.

‘Evaluation of Post-Hoc Interpretability Methods in Time-Series Classification’

In a recent study by Turbé et al. [34], post-hoc interpretability techniques were evaluated through synthetic datasets for classification problems. Each feature in the dataset is a sine wave, which simplifies the analysis of the temporal relationships of multivariate time series features. In order to increase the complexity of the classification task, sine waves with various frequencies and amplitudes were summed, and noise was added to create the dataset. This design would enable control and tuning of dataset complexity so that interpretability methods could be evaluated in the presence of varying numbers of features.

The performance of interpretability methods is evaluated via metrics like the Area Under the Curve of the Score Area (AUCSE), which quantifies error in identifying regions of input data that make a difference to model predictions. This is a ‘score error’, the difference between what the ground truth or benchmark thought a region contributed and what the interpretability method thought that region contributed. This error score is calculated by plotting cumulative importance scores for regions given in input compared against their true contribution and measuring the area under this curve. The lower the AUCSE, the closer the identified regions are to their true impact on predictions and, therefore, the better the interpretability method based on this performance measure.

In addition to AUCSE, the reliability of interpretability techniques is ascertained using precision and recall metrics. These are metrics about how accurately the techniques can correctly identify and separate important input regions in data sets across a range of levels of complexity. Recall measures whether you have identified all the significant regions, and precision measures whether the regions you have identified are all significant.

Turbé et al. [34] evaluated multiple interpretability methods, including Shapley Value Sampling, Integrated Gradients, and LIME [34]. In particular, they find that Shapley Value Sampling and Integrated Gradients perform well at accurately realizing the features of the model used for predictions, even under high data complexity. Nevertheless, all tested methods failed to deal with temporal dependencies as the data became noisier or multivariate. This dataset is tunable

in complexity and multivariate, and it is highly suited for controlled experiments.

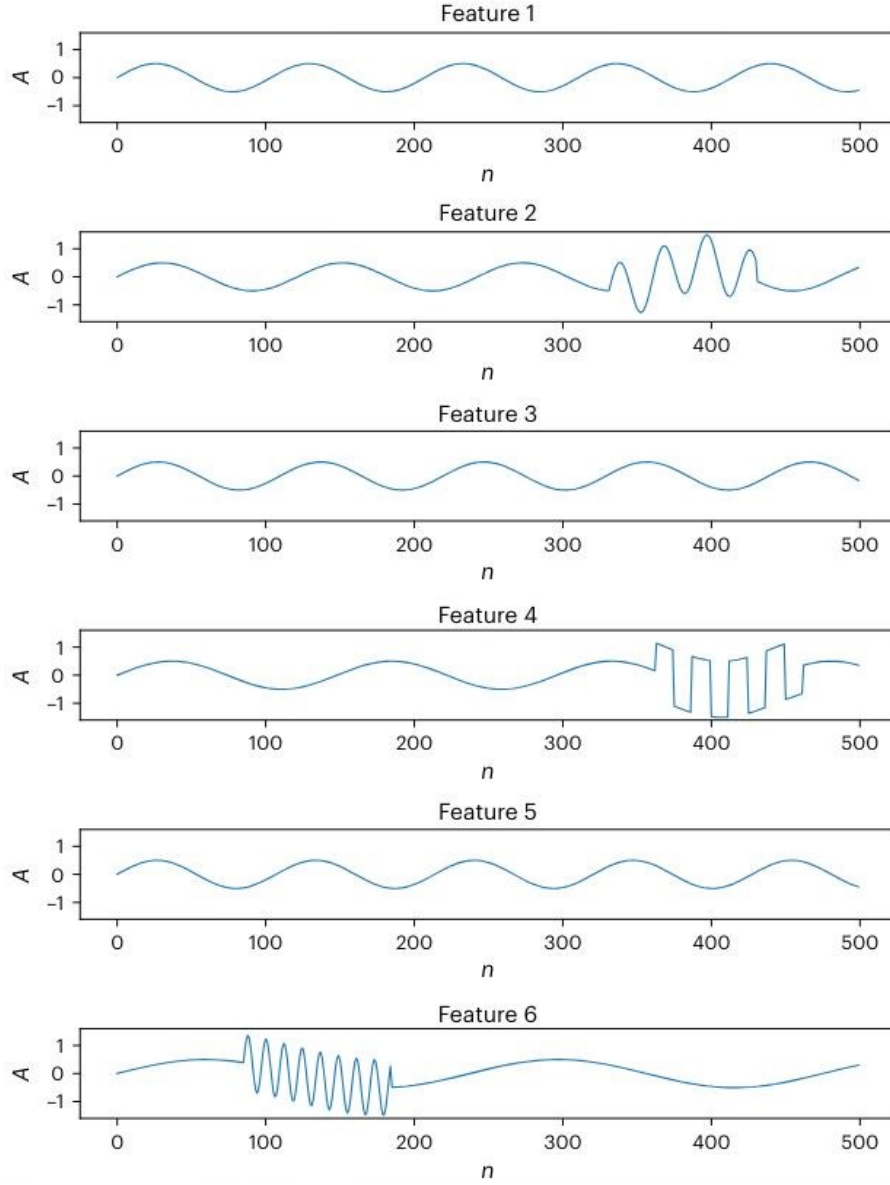


Figure 2.3: Example from the synthetic dataset of Turbé et al. [34]. Each channel is generated from sinusoidal components with time-varying amplitude, frequency, and phase, often confined to short windows and perturbed by noise. A dataset item is thus a multivariate time series constructed from these sinusoidal components. The classification task is to predict whether the sum of the frequencies of two designated sinusoidal components exceeds a given threshold.

An example synthetic dataset used by Turbé et al. [34] (Figure 2.3) is shown. The sine waves' components are shown centered in the figure, with their frequencies representing six features.

A single data item is a multivariate time series with six feature channels and 500 time steps. Two of the features contain short sine waves with a compact support of 100 time steps placed at random in time. The classification problem consists of determining if the sum of the frequencies of two sine waves is greater than a given threshold. This dataset has a clear temporal structure and multiple features, making it well-suited for testing how interpretability methods deal with temporal dependence.

This study establishes an important foundation from which to assess the interpretation of temporal relationships and also highlights the importance of datasets with tunable complexity. In Chapter 5, we describe how the desired input characteristics informed the dataset generation design and how we expanded Turbé et al.’s [34] work to include architectures and tasks more aligned with our research goals.

Turbé et al. [34] report that Shapley sampling consistently achieves the best performance across synthetic, ECG, and FordA datasets, as well as across Bi-LSTM, CNN, and Transformer architectures, using their two proposed metrics for relevance identification ($\text{AUC } \tilde{S}_{\text{top}}$ and $\text{F1 } \tilde{S}$). The only noteworthy exception is that DeepLiftShap performs better than Shapley on the ECG dataset using a CNN. The results show that Integrated Gradients is the second-best method for Bi-LSTM models, DeepLiftShap is the second-best method for CNNs, and GradShap and Integrated Gradients perform very similarly for Transformer models. Beyond these rankings, their analysis of relevance attribution reveals that none of the methods are entirely calibrated: the relevance values are a trustworthy ranking of which steps are more or less important, rather than directly mapping to the quantitative contribution of time steps. Last but not least, the evaluation prevents distribution shift between training and corrupted samples because the models were trained with random perturbations. This guarantees that the observed performance degradation actually represents information removal rather than artifacts of changes in data distribution.

‘Benchmarking Attention-Based Interpretability of Deep Learning in Multivariate Time Series Predictions’

To address shortcomings in existing benchmarks for attention-based interpretability techniques, such as being unable to handle multivariate data, a lack of ground truth against which to validate

a technique, and poor evaluation of temporal dependencies, Baric et al. [15] present a synthetic time series test bed. A test bed that they proposed is a controlled environment to systematically evaluate attention mechanisms in time series models, with tunable complexity and well-defined ground truth to validate interpretability methods. Through an examination of how attention mechanisms regulate the processing of different time steps and features, this test bed examines that issue within the framework of how attention operates generally.

In this study, the dataset used was synthetically generated by having multiple time series interact with no autoregression. For a clear ground truth of expected influences, a predefined relationship was used to make time series depend on one another. Further increasing the complexity of the dataset was done by adding noise and controlling the number of interactions in the series. This design served as a benchmark for multivariate, tunable, and interpretable evaluation of interpretability in attention-based models.

Attention, weight distribution accuracy, and sensitivity analysis were used to see how well attention mechanisms captured relationships in the data. In the study, one of the key evaluation techniques used was the seq2graphs model, a neural model that represents sequential data as graphs, which was tested for its ability to recover the ground truth relationship between the time series. It was found that seq2graphs was robust under simpler conditions but became biased when noise and complexity increased, as seen in Figure 2.4.

In Dataset 4 (Figure 2.4), the ground truth values (left side) and the average β were obtained from the seq2graphs model (right side). In each of these cases, β corresponds to the attention weights the seq2graphs model gives to attention to another time series describing the contribution of one time series to another. In particular, the β values report the magnitudes of the relationships between the promoting time series (x-axis) and the target time series (y-axis). Under the dataset’s design, strong self-dependence is expected, which means high β values along the main diagonal of the matrix (where each time series should primarily depend on itself). Instead, the observed bias toward time series index 0 indicates that this expected ground truth is not captured in the attention weights, throwing doubt on the model’s interpretability.

This study highlights that attention mechanisms can indeed extract relevant features from time series data, but it also shows that their reliability drops as noise and complexity increase. In this way, the proposed synthetic framework serves as a means of analyzing how well attention-based

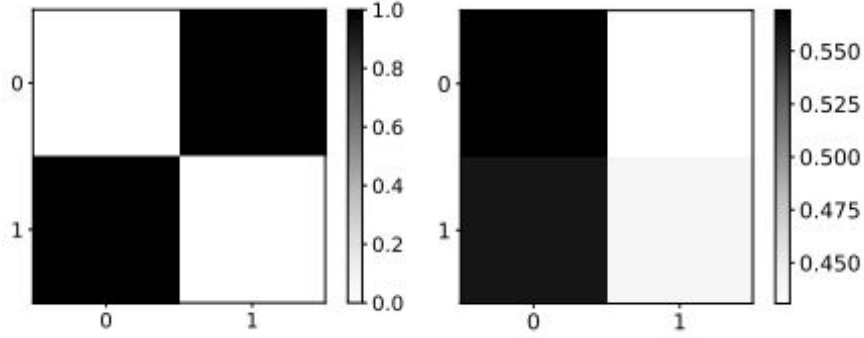


Figure 2.4: Comparison between the ground truth interaction matrix (left) and average attention weights (β) learned by Seq2Graph (right) on Dataset 4 of Baric et al. [15]. The x-axis shows the influencing time series, and the y-axis shows the target time series. Strong self-dependence (high values along the diagonal) is expected under the ground-truth generative model. In contrast, Seq2Graph’s β values are biased toward time series index 0 rather than showing uniform self-dependence, indicating deviation from the ground truth in its attention assignments.

models perform, and of pointing out both their strengths and their weaknesses. While this framework is designed for attention-based architectures, there would be massive changes needed to extend it to other approaches to interpretability, such as DeepSHAP.

In contrast to this, our work involves interpretability techniques that do not require attention mechanisms and, therefore, require a different ground truth and evaluation framework. Despite the value of Baric et al.’s [15] datasets for attention-based interpretability, they do not apply to more general models.

‘GRATIS: GeneRAtIng TIme Series’

The GRATIS structure, introduced by Kang et al. [16], is aimed at generating synthetic time-series data with measurable time heterogeneity and variability. GRATIS then synthesises time series with various behaviours: Gaussian noise, mixture autoregressive processes, and many other behaviours using mixture autoregressive (MAR) models. It turns out that by adjusting the parameters, the given function can be tuned to produce a time series with any specified trends, seasonality, and levels of noise, which makes it a powerful tool for simulating time series data displaying specific properties.

To evaluate the results produced by GRATIS, the datasets that are produced have been compared to benchmark time series data from the M4 dataset, which is used in most of the forecasting competitions. The M4 dataset also has time series data from different disciplines, such as finance, economics, and demographics. In order to compare the patterns of the simulated and real data, the authors employ a number of visual and statistical techniques to such an extent that the synthetic data hardly differs from the real-world data.

As illustrated in Figure 2.5, GRATIS generates time series data on a range of intervals, namely annual, quarterly, monthly, and weekly. For each simulated time series, the most similar real-world series from the M4 dataset is also shown in the same colour. The plots omit the y-axis to standardise some of the variability across the plots, and the x-axis is time in years. The comparison of these plots to real-world data demonstrates that the patterns found in real datasets are realistically replicated by the framework for producing synthetic data.

GRATIS finds its application in time series research by offering a flexible framework to generate datasets that can be used to evaluate the performance of forecasting and classification models. However, the framework does not directly evaluate interpretability methods; instead, it provides a testbed for different methods on a diverse set of time series with tunable complexity. Though GRATIS serves as a tool for generating artificial datasets, it is not optimized as a solution to interpretability hurdles, for example, finding which features contribute to model predictions or interpreting model predictions.

‘Benchmarking Deep Learning Interpretability in Time Series Predictions’

Ismail et al. [3] investigate how discriminatory information useful for classification is distributed across time steps and feature dimensions in multivariate time series data. Their work provides a framework for evaluating interpretability methods based on how well they reveal what the model has learned. They demonstrate that both temporal and feature-level relevance can be distinguished. Importantly, they emphasise the need for methodologies that avoid conflating these dimensions, as overlapping temporal and feature information can obscure interpretability. Their findings support our focus on evaluating model contributions along both the temporal and feature dimensions. Building on their approach, this work is based on systematically studying these dimensions with the help of several synthetic time series datasets. An important aspect

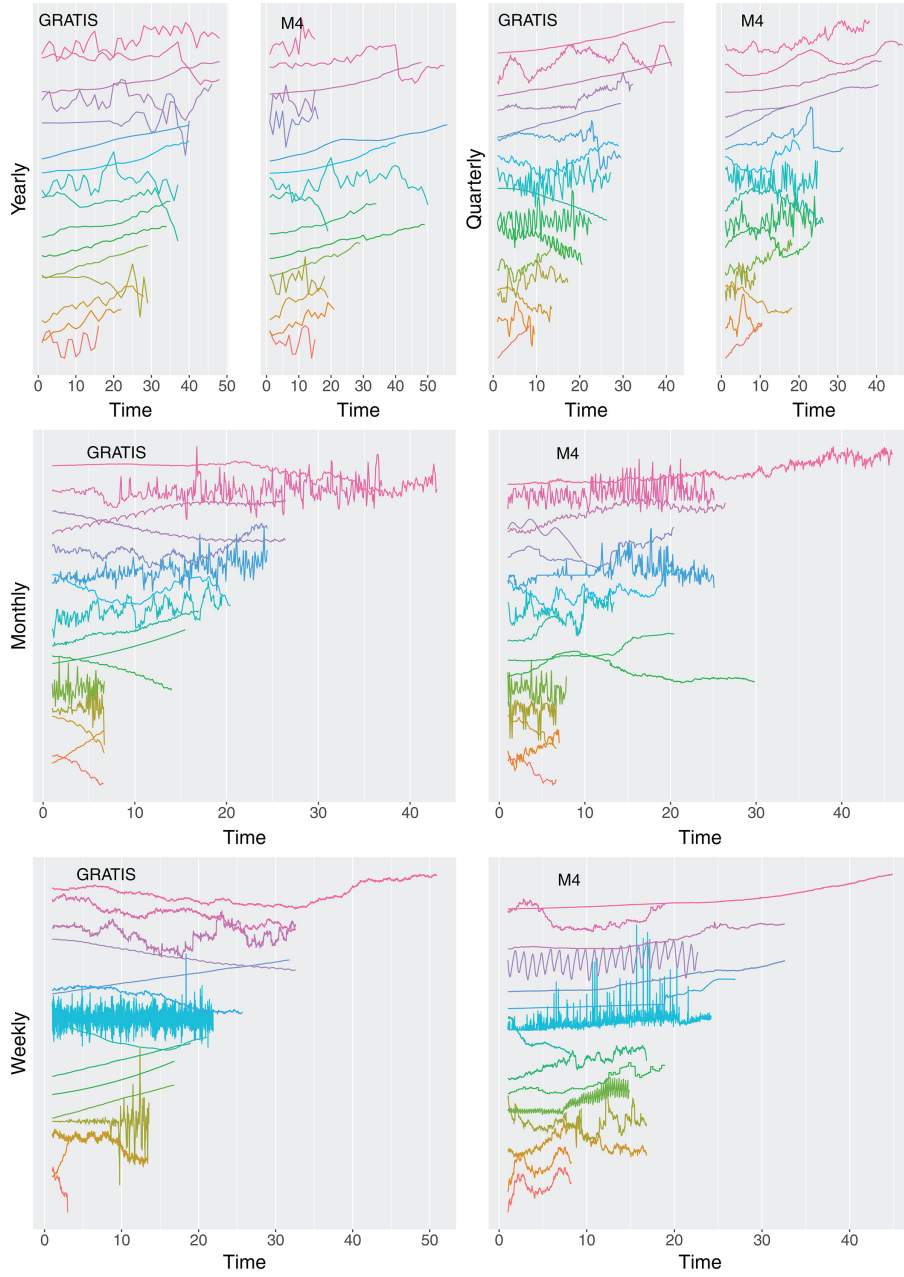


Figure 2.5: Examples of simulated time series data generated with GRATIS at different intervals (annual, quarterly, monthly, and weekly). For each simulated series, the most similar real-world series from the M4 competition is shown in the same colour. This illustrates that the synthetic data produced by GRATIS can realistically reproduce the patterns, seasonality, and variability found in real-world time series, making it a useful framework for benchmarking and experimentation [16].

of effective interpretability is the coordination of time and feature contributions that must be clearly separated.

Ismail et al. [3] introduce the so-called middlebox dataset, a synthetic benchmark generated from several time series processes, including autoregressive, sinusoidal, and random noise components. To evaluate interpretability methods, they propose precision and recall metrics that measure how well the most important features and temporal dynamics are captured. In particular, precision is the proportion of correctly highlighted important features, while recall is the proportion of true important features that are accurately identified by the interpretability method. They also evaluate temporal accuracy, assessing how well interpretability methods reflect changes in feature importance over time.

In order to understand how different model types affect the performance of saliency-based interpretability methods, their framework systematically tests these metrics (precision, recall, and temporal accuracy) on different neural network topologies, including RNNs, TCNs, and Transformers. Ismail et al. [3] show how these metrics, combined with synthetic datasets meant to supply ground truth on feature importance and temporal dynamics, apply to various architectures and under varying degrees of dataset complexity.

They demonstrate that many interpretability methods fail to reliably identify important features in cases where time and feature dimensions overlap. For that, the authors present a two-step Temporal Saliency Rescaling (TSR) approach. First, this method considers the importance of each time step, then, concerning the importance of each feature at a specific time step, disentangles temporal and feature domains to achieve better interpretability performance.

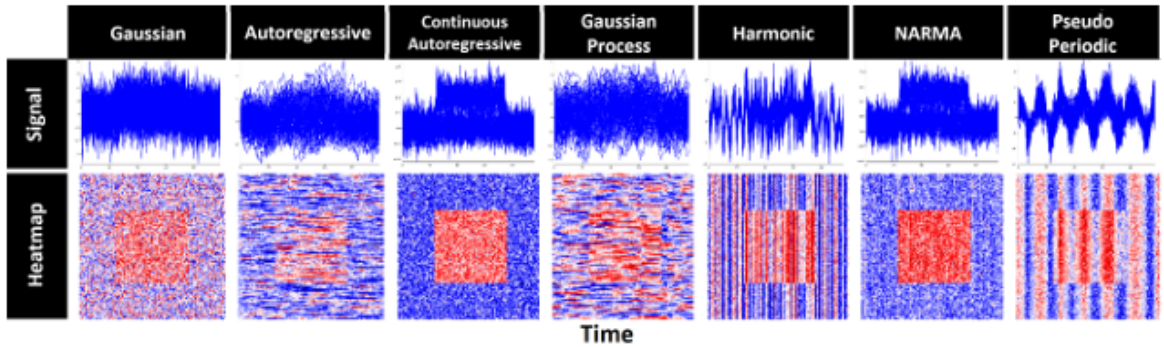


Figure 2.6: The middlebox dataset is generated using various time series processes. The first row illustrates how each feature evolves in different samples, while the bottom row corresponds to the heatmap of each sample, where red indicates informative features [35].

To generate the middlebox dataset on which this work is based, as shown in Figure 2.6, different

time series processes are used. The top row shows how each feature traverses along different samples to distinguish temporal variations. Each heatmap associated with each sample is displayed in the bottom row, where informative features are highlighted in red. The heatmaps are generated by encoding the ground truth importance of each feature–time step pair, with red highlighting the informative regions. The reason for using such a visual is that it allows us to see how models attend to features over time steps, which is important when evaluating saliency-based interpretability methods.

Ismail et al.’s [3] framework provides useful insights on how to evaluate saliency-based interpretability techniques, in particular, how to separate the evaluation of temporal and feature dimensions. In this study, we systematically evaluate these methods on synthetic datasets with known ground truth and show how saliency methods compare with one another on different types of neural network architectures, including RNNs, TCNs, and Transformers.

Their framework is strong in building a strong base for interpretability evaluation, but it mainly concerns saliency-based methods; it is unaffected by attribution-based techniques such as DeepSHAP. We extend their framework to our setting, adapting their synthetic data generation and evaluation strategies to TCNs and TS-MLPs while still working to minimize the need for external data acquisition during generalisation. We specifically test DeepSHAP attributions against defined ground truths to evaluate the interpretability of these architectures and develop extensions to their approach that satisfy our needs.

Summary

We summarise the main elements of these studies in Table 2.1.

2.6 Conclusion

This chapter provided a basic understanding of time series data, possible underlying modelling approaches, and challenges in interpreting DL models for time series analysis. It has reviewed existing synthetic test benches and studies to analyse existing interpretability techniques and, by extension, feature attribution techniques in relation to the application context.

Table 2.1: Summary of synthetic time series interpretability test benches

Study	Data Generation	Models Used	Interpretability Methods	Evaluation Metrics
Turbé et al. [34]	Combination of sine waves	Various classifiers	Shapley Value Sampling, Integrated Gradients, LIME	AUCSE, Precision, Recall
Baric et al. [15]	Synthetic time series with controlled dependencies	Attention-based models	Seq2Graphs	Attention weight distribution, Sensitivity analysis
Kang et al. [16]	Mixture autoregressive processes (MAR)	Various forecasting models	Not focused on interpretability	Similarity to real-world M4 dataset
Ismail et al. [3]	Synthetic datasets (e.g., middlebox dataset)	RNN, Transformers, TCN	Saliency maps, Temporal Saliency Rescaling (TSR)	Precision, Recall, Temporal accuracy

In order to evaluate interpretability methods, several synthetic test benches were proposed by Turbé et al. [34], Baric et al. [15], Kang et al. [16], and Ismail et al. [3] to experiment with datasets of known ground truth data. However, these frameworks are able to systematically evaluate interpretability techniques, yet most of them concentrate on saliency or attention-based approaches. This leads them often to ignore the special characteristics and difficulties of making attribution in particular styles of architecture, for instance, TCNs and TS-MLPs.

These reflect the limitations of needing more targeted synthetic datasets and evaluation approaches that explicitly acknowledge the 2 dimensions involved, the temporal and feature, , where the temporal dimension refers to when in the sequence something matters and the feature dimension refers to which variable contributes to the prediction. To make transparent and trustworthy models more appropriate for the actual interpretability requirements, filling these gaps is important.

Chapter 3

Methods and techniques

‘Give me six hours to chop down a tree, and I will spend the first four sharpening the axe.’

— *Abraham Lincoln*

3.1 Overview

In this chapter, we introduce the methods and techniques to develop a framework to evaluate the accuracy of interpretability mechanisms in time series analysis. The objective here is to figure out how well these mechanisms can explain contributions of temporal features when DNNs is used.

The chapter is organised as follows: Section 3.2 introduces TCNs and TS-MLPs, which were selected for their simplicity and suitability in demonstrating the interpretability framework for sequential data. We discuss and describe Shapley values and their approximations in Section 3.3. Finally, Section 3.4 presents the Know-it toolkit used to implement and evaluate these models and methods.

3.2 DNNs for time series analysis

Both TCN and TS-MLP methods address challenges in time series data and are a form of DNNs: TCNs leverages sequential patterns, while TS-MLPs modifies the sequence data to improve forecast accuracy.

3.2.1 DNN training

In Section 2.3, DNNs were described as layers of nodes searching for structure in the time series. Training these networks means adjusting their parameters so that the predicted values $\hat{y}(t)$ match the true outputs $y(t)$. This is done by minimising a loss function, most often the mean squared error (MSE):

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i(\theta))^2,$$

where θ represents the model's weights. Optimisation is carried out with gradient descent, where the weights are updated step by step:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_{\theta} \mathcal{L}(\theta^{(k)}),$$

with the learning rate η deciding how big each update step is. We calculate these gradients through backpropagation, which moves errors backwards through the layers.

Regularisation is needed to prevent overfitting. One common method is weight decay, which adds a penalty to the loss based on the squared size of the weights:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|_2^2,$$

where λ is the strength of the penalty. Another popular approach is dropout, where a random fraction of neurons is turned off during training, which forces the network to learn more robust patterns.

We also use normalisation layers. They rescale activations to have zero mean and unit variance, which prevents oscillations during learning and helps models generalise better.

These training principles are general to most DNNs and also apply directly to the TCNs and TS-MLP discussed in the following sections. Since the focus of this dissertation is on interpretability, the discussion here is limited to the essentials of training

3.2.2 TS-MLP

A TS-MLP is a neural network architecture for sequence prediction. Compared to the conventional MLPs, TS-MLP is used for tasks involving sequential input data where temporal relationships are to be learned. This adaptation allows MLPs to work with time series.

TS-MLP adapts traditional MLPs by adding time-shifted input features that make it easier for the model to learn time dependencies. An input layer, one or more hidden layers, and an output layer form the architecture [36]. Unlike standard MLPs, a TS-MLP enables the input layer to consider not only the current input values but also previous values of the input features, encoding temporal relations in the data.

How TS-MLP works

In a MLP, the input features are passed through a series of connected linear layers for processing and are comprised of applying a linear transformation followed by a nonlinear activation function. Mathematically, we can represent the output of the l -th layer as:

$$h^{(l)} = \sigma \left(W^{(l)} h^{(l-1)} + b^{(l)} \right) \quad (3.1)$$

where $W^{(l)}$ and $b^{(l)}$ are the weights and biases, $h^{(l-1)}$ is the input to the layer, and σ is the activation function.

We construct this architecture and use a TS-MLP that leverages lagged features (time-shifted versions of its input features) to perform temporal dynamics modelling. For example, if we have the original input x_t , we force the past inputs $x_{t-1}, x_{t-2}, \dots, x_{t-n}$ to be added. This will enable the model to have access to current as well as historical data patterns, both of which will help in learning sequential dependencies in time series.

Lagged features are one of the important building blocks of a TS-MLP. The defining feature of TS-MLP is to use lag features to capture temporal dependencies. The current input data is used to build the model, but with the last values of the input features to try to factor in historical trends and patterns. The lookback period (or number of lagged steps)—to look at how far in the past you want to use for your prediction—is a hyperparameter that matters a lot for tasks where the future is related to the past.

The general architecture of a TS-MLP is depicted in Figure 3.2. Focus is on the prior time steps, denoted by blue blocks; output is the next time step the model learns, denoted by the red block. The dense connections give temporal dependencies and non-linear activations, making it ideal for use in time series forecasting and sequence modelling.

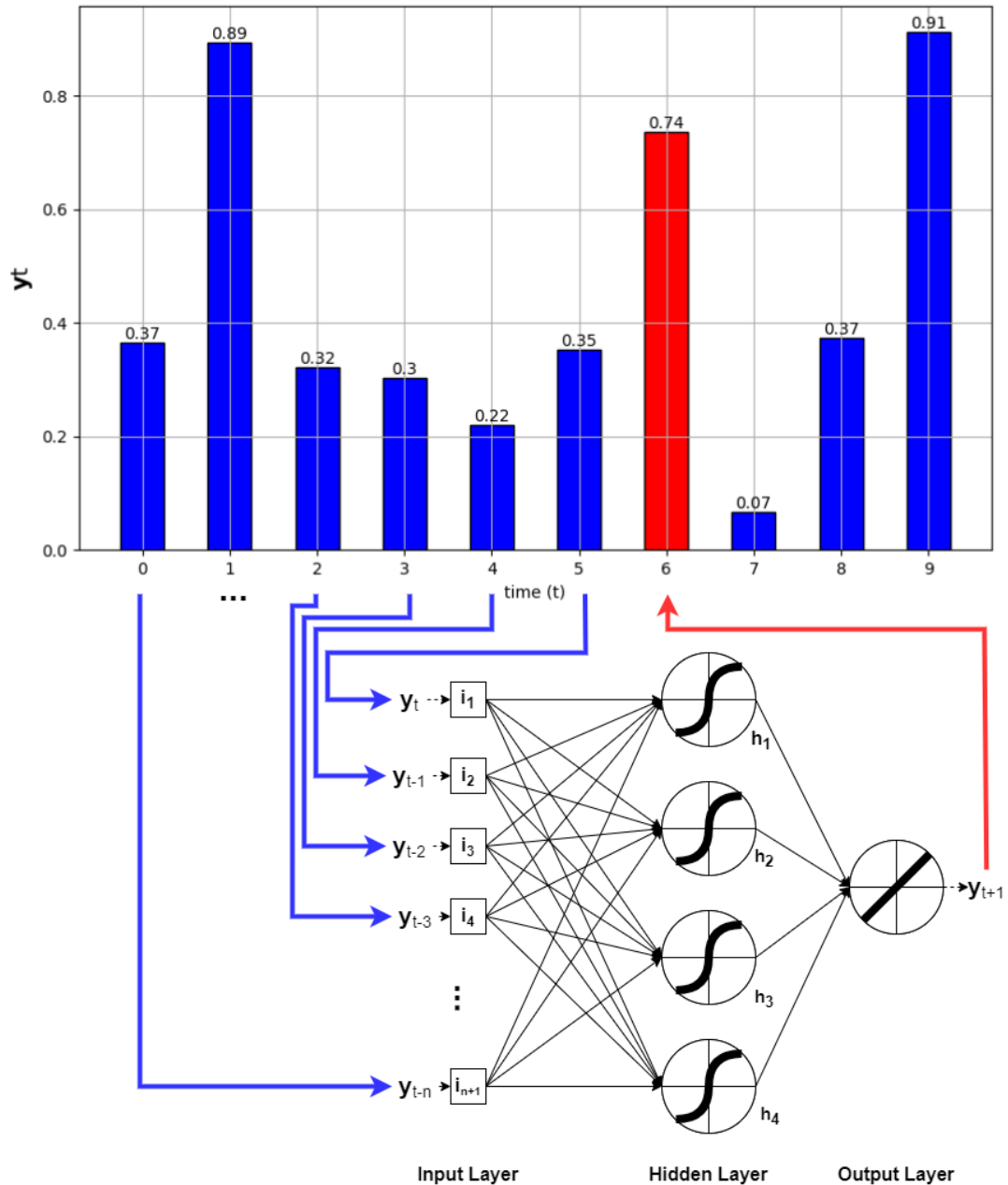


Figure 3.1: Simplified structure of TS-MLP structure. In this figure, blue blocks denote lagged input features from previous time steps, while the red block indicates the model's forecasted value at the time step.

Hyperparameters of TS-MLPs

Other than general hyperparameters of training that steer gradient descent (i.e., learning rate, batch size), TS-MLP also requires architectural choices specific to time series. These include the number of layers, the number of units per layer, the activation function, and the number of back steps (or time lags). The back steps determine how much historical data is used as input, and thus form an important hyperparameter: one can capture temporal structure by selecting a suitable lag size with little added complexity [37]. Along with model depth and layer size, these hyperparameters must be carefully set to allow the model to uncover nonlinear relationships in the time series.

3.2.3 TCN

A TCN is a DL model made to work with sequence model techniques. It will be useful to know what Convolutional Neural Networks (CNNs) are before we proceed to elaborate on TCNs. A CNN is a type of DNN and is mostly used in image recognition and processing. The main characteristics of CNNs include the learning of spatial hierarchies of features by applying convolution layers for assessment [7, p. 234]. Each layer applies a number of small filters that slide across the input data, combining their results to extract important local features. TCNs use the CNN’s structure for sequential data. Therefore, unlike other CNN, which are applied to 2D data such as images, TCNs are designed to process sequences by convolutional layers that work along the time channel.

The notion of TCNs emerged from the requirement of extending the usage of CNNs for sequential data. A CNN can learn spatial features in images through convolutional layers. By applying the same concept to the temporal domain, we can overcome some of the limitations of typical sequence models like RNNs. The use of TCNs has improved the ability to capture long-range dependencies and has enabled better parallelization of computations, thereby reducing training time [38].

How TCNs work

Because CNNs can be flexible architectures and adapted to many circumstances, the principal advantage of TCNs is that they are tailored to consume sequential data, which maintains their temporal aspects so that they can learn both the short and long-term dependencies within time series tasks [39], [40].

The temporal convolution takes a sequence and applies one-dimensional convolution to it along the temporal direction. Patterns that are specific to the temporal structure of a sequence are found by filters in these layers that are slid along the temporal axis. TCNs achieves this by applying a causal convolution, meaning that the predictions from a given time step rely only on the current or previous inputs, which is useful when the data is streaming, like forecasting tasks where you cannot access future data [40].

Likewise, a TCN's dilated convolutions allow its receptive field to grow without blowing up its parameters too much. Earlier, Lea et al. [25] and Shrikumar et al. [40] introduced dilated convolutions with gaps between filter weights so that at each layer, the network can capture the dependencies for larger swathes of time while only handling short and long-range dependencies.

Overcoming the vanishing gradient problem in deep networks with residual connections is essential to TCNs. These connections lift the constraint of local gradients and the ability to learn long-distance dependencies due to the ability to backpropagate through these connections and through these layers to learn dependencies that span long distances [41].

Figure 3.2 shows a simplified TCN architecture, showing how convolutional layers learn temporal dependencies, followed by the last layer predicting the next time step based on these dependencies.

While TCNs bring normalization methods such as Layer Normalization into training to stabilize training and improve model performance. The nonlinearity required to model complex temporal relationships in the data is due to introducing nonlinear activation functions like Rectified Linear Unit (ReLU) [40].

By design, pooling layers are optional in TCNs. A pooling layer works by combining the outputs of nearby neurons into a single value, for example, by taking their maximum (max-pooling) or

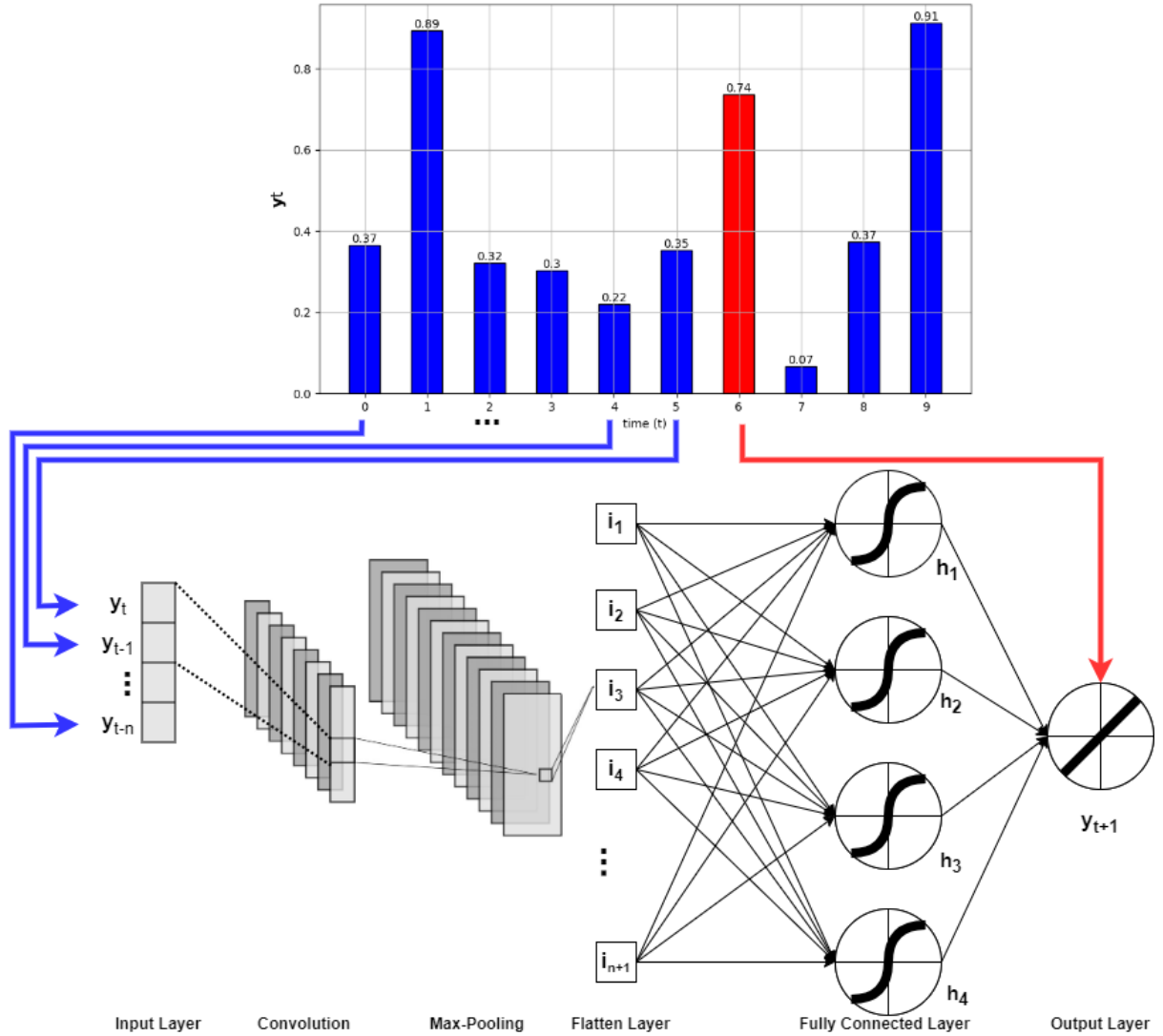


Figure 3.2: The type of workflow and the structure of simplified TCN. The input time series data is represented by the blue blocks at the different time steps, and the red blocks represent the convolutional layers that capture temporal features. The last red block represents what the model will predict of the signal on the next time step using temporal dependencies learned by the convolutional filters.

average (average pooling). This reduces the dimensionality of the data and highlights the most relevant activations. In some cases, such layers can be present in TCNs to decrease the number of features unrelated to the temporal representation. They can also improve generalisation when they help summarise information over finite time windows.

When combined with causal and dilated convolutions, TCNs computes temporal convolutions

for all timesteps at once (rather than step-by-step) and backpropagates through shorter residual paths rather than lengthy recurrent chains, which frequently results in higher accuracy and quicker, more stable training on lengthy sequences [41].

Hyperparameters of TCN

How this network performs in modelling temporal data depends a lot on the hyperparameters of TCNs. There is a set of important hyperparameters: the number of temporal convolutional layers that equals the depth of the TCN. More complex and deeper networks can identify more complex temporal dependencies, but usually overfit the data [42], so the balance is critical. Just as important as filter size and stride (the step size by which the filter moves across the sequence) in the convolutional layers, they determine the receptive field and the level of temporal information the model can grasp. The small filters are, in fact, particularly nice for finding fine-grain features, while larger filters are perfect for finding patterns. Also important to the hyperparameter is the dilation rate, which controls the spacing between filter elements in a convolution. By skipping input positions at fixed intervals, dilated convolutions expand the receptive field without increasing filter size, allowing the model to capture long-term dependencies efficiently.

Normalization and activation function are among the main parts of network optimization. Non-linearity is brought into the basic linear model that the layers offer to the neural network through activation functions like ReLU in order to enable the network to replicate intricate relations within the data. That is where normalization techniques come into play; they guarantee getting a similar result, and Layer Normalization makes the training phase stable and converging. If pooling is incorporated in the TCN, then we have to define hyperparameters such as the type of pooling (max pooling, average pooling) and the size of pooling to maintain critical information while reducing the dimensionality. With TCNs, typical regularization techniques such as dropout or weight decay apply. For example, dropout is a technique where, during training, a specific proportion of neurons is randomly set to zero to prevent overfitting, and weight decay is when we penalize large weights so that we generalize better. Section 3.2.1 details these techniques in detail.

3.3 Shapley values

At the heart of our investigations for interpretability techniques, the Shapley values constitute a mathematically grounded way to fairly apportion a model's prediction to each feature, as discussed in Chapter 2.

Shapley values explain a machine learning model by viewing input features as players in a game whose contributions are quantified by how much additional value the feature adds to different feature combinations. The Shapley value ϕ_i of a feature i is calculated as according to (3.2):

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} [v(S \cup \{i\}) - v(S)] \quad (3.2)$$

where N is the set of all features, and $v(S)$ is the model's prediction using only the features in subset S . The formula ensures that every feature is, in fact, getting a score with regard to the average incremental effect of the added feature to the model output for any potential subset of features.

The Shapley value calculation follows these key steps:

1. **Permutations:** Evaluate different coalition scenarios by considering all possible orderings of features. In the case of time series, these features can include both variables and their lagged values, so the permutations reflect different possible orders in which past observations are assumed to contribute to the prediction.
2. **Marginal contributions:** It calculates the marginal contribution of a feature by how much it adds to the linear combination of a coalition.
3. **Average contribution:** We average over all permutations of the marginal contributions of each feature.
4. **Shapley value:** The average marginal contribution for each feature is the Shapley value, equal to the sum over all permutations.

These values satisfy four fundamental properties:

- **Efficiency:** These are the total sum of all feature contributions, the difference between the model's prediction and a baseline value.

-
- **Symmetry:** There are two features whose contribution from all coalitions is equal, thus giving them the same Shapley value.
 - **Dummy:** In fact, features with zero marginal contribution (i.e., Shapley value=0) are features with no marginal contribution.
 - **Additivity:** The value of the total model can also be computed, and so can the Shapley values of the individual components combined with the model's total value.

The concept of Shapley values is so instrumental to this study that we have provided an in-depth explanation and example for Shapley values in Appendix A.1.

3.3.1 Analytic Shapley values

Analytic Shapley Values (ASVs) are an analytical process of determining the contribution of certain features towards the prediction of a sample. As it has been seen from the linear model, it is relatively simple to ascertain the influence of the predictors. Here is an example of a linear model prediction for a single data point [1]:

$$\hat{f}(x) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \quad (3.3)$$

where x represents the instance for which the contributions are to be computed. Every x_j is a feature value where $j = 1, \dots, p$. The β_j is the weight linked to feature j . Here X_j represents the baseline value for that feature. The contribution ϕ_j of the j -th feature on the prediction $\hat{f}(x)$ is then:

$$\phi_j(\hat{f}(x)) = \beta_j x_j - E(\beta_j X_j) \quad (3.4)$$

where $E(\beta_j X_j)$ is an approximation of the mean effect for feature j . The contribution is the difference between the feature effect and the average effect. The following is obtained when we total together all of the feature contributions for one instance:

$$\sum_{j=1}^p \phi_j(\hat{f}(x)) = \sum_{j=1}^p (\beta_j x_j - E(\beta_j X_j)) = \hat{f}(x) - E(\hat{f}(x)) \quad (3.5)$$

In this case, the value for this is the projected value minus the average expected value for the data point x . We now expand on some other cases, other than linearity cases, albeit ones that involve multiple steps.

This solution is referred to as the ASV. In the remainder of this section, we use the term ASV to distinguish these exact computations from approximate methods.

Shapley values in detail

In Section 2.4.5, we discussed the Shapley value assigning a value function, val , to players in the set S . It averages the feature value across all possible value combinations of the feature, hence computing the value added by a feature value. A feature value is the numerical or categorical value assigned to a feature and a specific instance of the feature.

$$\phi_j(val) = \frac{1}{p!} \sum_{S \subseteq \{1, \dots, p\} \setminus \{j\}} |S|!(p - |S| - 1)!(val(S \cup j) - val(S)) \quad (3.6)$$

with p numbers of features in the instance to be explained, x is the term for the feature vector, and S represents a subset of features used to compute their contribution to the model's prediction. $val_x(S)$ is a forecast for feature values in set S that are marginalized over feature values in set S that are not marginalized:

$$val_x(S) = \int_{x_{min} \notin S}^{x_{max} \notin S} (\hat{f}(x_1, \dots, x_p)) d\mathbb{P}_{x \notin S} - E(\hat{f}(x)) \quad (3.7)$$

Several integrations are done for each missing feature represented in the subset S , which captures the contributions of those features to the model's output. The value function, therefore, takes the role of the payoff function for the players' coalitions, with the players being feature values. The Shapley value may then be converted into a SHAP-like format, and the following efficiency theorem is obtained:

$$\hat{f}(x) = g(x') = \phi_0 + \sum_{j=1}^p \phi_j x'_j \quad (3.8)$$

In this case, ϕ_0 can be regarded as a constant; for convenience, one may consider it the average of the model outcome, but other criteria can also be used. The discrepancy between the base value (in this study, mean) and the model output is quantified by the sum of the Shapley values of the significance of features.

Computation challenges of Shapley values

The computation of the Shapley value is a time-consuming process. Indeed, in most of these problems, obtaining the true Shapley value solution is infeasible. The main drawback of the

accurate computation of Shapley values lies in the fact that, as the number of feature value interactions grows, the number of possible coalitions could be enormous. Further, to delete a feature, it is required to generate some random samples, such that the variation in the Shapley value is an approximation of and not the direct ASV. Present approaches to address the issues of computational complexity are the use of coalition sampling and the ability to set a certain limit on how many iterations will take place. Therefore, the decrease in iterations improves the computation time while, at the same time, it enlarges the error of Shapley values. There are no guidelines on how to properly select the optimal number of iterations; this depends on the desirability of attaining highly accurate Shapley value results as much as is feasible.

Interpretation challenges of Shapley values

Shapley’s values are often misunderstood. It is important to understand that the Shapley value of a feature is not the variation between the predicted value and the one computed based on the feature not participating in the model training [1]. The estimated Shapley value represents the value addition of a *particular feature value* in the prediction difference between the *actual* and the *average prediction* based on the current feature values. This difference in interpretation is subtle.

3.3.2 Exact Shapley values

The Exact explainer, available as part of the SHAP package¹, is a post-hoc interpretability technique that can theoretically compute ASV for any model of interest. This implementation relies on Shapley values to provide mathematically grounded feature attributions. However, the disadvantage of this is that it will come up with all the possible masking patterns, and this takes exponential time in computing Shapley values. The exact method described here has a time complexity of $O(2^p)$, assuming p as the number of input features. Therefore, the Exact explainer can handle only models that have no more than 15 input features, as stated in its official description. The analysis of the Exact explainer can be described by the same formula as the Shapley value presentation in Section 3.3.1.

Unfortunately, it is worth noting that the Exact explainer in the SHAP library is relatively slow

¹<https://github.com/slundberg/shap>

and highly computational in its usage. So, it could only be employed in instances where a high level of accuracy is demanded, irrespective of the computational leverage. This is different from other techniques, which bring a level of approximation into a procedure to facilitate computation, particularly in cases where the data set is large or has several parameters. As a result, its application is restricted to models that have a finite number of features or to which it is convenient to use the exact Shapley values for modelling explanation.

Consequently, the Exact explainer turns out to be a strategy for computing Shapley values with the greatest degree of accuracy. Nevertheless, the strategies for conflicting constraints preserve the sense of balance between realism and the need for accurate values. In real life, one would like to look for a less precise technique unless the particular solution requires the use of the Exact explainer. We will always employ the Exact explainer to compute attribution values using ASVs for analytical transfer functions.

3.3.3 DeepLIFT

The ideas presented here are based on the work done by Shrikumar et al. [27]. DeepLIFT and, hence, DeepSHAP define how input features deviating from their base values cause the model output to change relative to the base. The reference (baseline) values for input features are different based on the level of control in calculating feature attributions. The reference can be generic or provided by subject matter experts. In most cases, when DeepSHAP is applied to a validation set or test set (which is the case here), the reference values come from the training data, and the feature attributions are computed from the validation or test data.

DeepLIFT provides an explanation to every change from the reference by assigning each difference from the reference a contribution value $C_{\Delta x_i \Delta t}$ for feature x_i and Δx_i concerning the output t and Δt . These contributions adhere to the property of ‘summation-to-delta’ (Equation 3.9) holds that the sum of feature contributions is equal to the difference in the model’s output from its baseline output [9]. Δt represents the time shift.

$$\sum_i C_{\Delta x_i \Delta t} = \Delta t \quad (3.9)$$

For some contribution $C_{\Delta x \Delta t}$, there is a multiplier that satisfies the property of summation-to-delta:

$$m_{\Delta x \Delta t} = \frac{C_{\Delta x \Delta t}}{\Delta x} \quad (3.10)$$

This *multiplier* denotes the influence of Δx on Δt concerning Δx . Interpreting the multiplier $m_{\Delta x \Delta t}$ as the finite differences' partial derivative $\frac{\partial t}{\partial x}$ also allows for applying the *chain rule for multipliers*.

The *chain rule for multipliers* states that in a neural network with inputs $x_1, \dots, x_n$, a hidden layer having nodes $y_1, \dots, y_m$, and a goal output node t , with given the values for $m_{\Delta x_i \Delta y_j}$ and $m_{\Delta y_j \Delta t}$, the following applies:

$$m_{\Delta x_i \Delta t} = \sum_j m_{\Delta x_i \Delta y_j} m_{\Delta y_j \Delta t} \quad (3.11)$$

and this is consistent with the property of summation-to-delta. With the help of the contributions, multipliers, and summation to the delta principle, we can find how much a given set of input features $C_{\Delta y \Delta t}$ changes the output of any neural network.

We cannot say that every single contribution will be positive: there will be times when it will not be. Determines the areas of positive and negative contribution of the contribution of some *difference-from-reference* Δy to a Δt , Δy is differentiated into its positive and negative components [43]:

$$\Delta y = \Delta y^+ + \Delta y^- \quad (3.12)$$

The contribution $C_{\Delta y \Delta t}$ has two components:

$$C_{\Delta y \Delta t} = C_{\Delta y^+ \Delta t} + C_{\Delta y^- \Delta t} \quad (3.13)$$

The principle of feature contributions that DeepSHAP follows is based on the reveal-cancel rule, and it mainly performs positive and negative management using the reveal-cancel rule. We will not discuss the application of the other guidelines, like using the linear and rescale rules.

The reveal-cancel technique divides the positive and negative effects of $C_{\Delta x \Delta y}$ separately. It first considers the effect of the positive part of Δx separately and then considers the negative part of Δx separately. This method guarantees the accurate estimation of the positive and negative impacts of $C_{\Delta x \Delta y}$.

DeepLIFT defines the reveal-cancel rule with the following [43]: If we have a nonlinear neuron $y = f(x)$, with input value x , reference input value x^0 , and the components of either positive or negative then difference-from-reference Δx , (Δx^+ and Δx^-), then the positive and negative

components of $C_{\Delta x \Delta y}$ (Δy^+ and Δy^-) is to be computed as:

$$\begin{aligned}\Delta y^+ &= \frac{1}{2}(f(x^0 + \Delta x^+) - f(x^0)) \\ &\quad + \frac{1}{2}(f(x^0 + \Delta x^- + \Delta x^+) - f(x^0 + \Delta x^-)) \\ \Delta y^- &= \frac{1}{2}(f(x^0 + \Delta x^-) - f(x^0)) \\ &\quad + \frac{1}{2}(f(x^0 + \Delta x^+ + \Delta x^-) - f(x^0 + \Delta x^+))\end{aligned}\tag{3.14}$$

The multipliers for the components (positive or negative) of the contribution $C_{\Delta x \Delta y}$ are then defined as:

$$\begin{aligned}m_{\Delta x^+ y^+} &= \frac{C_{\Delta x^+ \Delta y^+}}{\Delta x^+} = \frac{\Delta y^+}{\Delta x^+} \\ m_{\Delta x^- y^-} &= \frac{C_{\Delta x^- \Delta y^-}}{\Delta x^-} = \frac{\Delta y^-}{\Delta x^-}\end{aligned}\tag{3.15}$$

3.3.4 DeepSHAP

The explanation of DeepSHAP provided in this dissertation is based on Shrikumar et al.'s [27] research on DeepLIFT and Lundberg et al.'s [9] work on DeepSHAP, as well as that of Molnar [1].

SHAP is a derivation of DeepLIFT based on DNN models, where feature attributions are determined using the backpropagation process [14], [27], [44]. DeepSHAP is a simplification of the traditionally computationally intensive Shapley value estimation process by integrating DeepLIFT's computation principles. Unlike the Shapley value, it interweaves DeepLIFT's calculation rules within the process of estimation of the Shapley value.

DeepSHAP replaces DeepLIFT's reveal-cancel rule with a variant that estimates Shapley values from the top layer and backpropagates to do the same in the lower layers [14]. Examining Equation 3.14, we can see that it looks very much like the Shapley values presented in Equation 3.8 in Section 3.3.1. This comparison essentially calculates the incremental contributions of Δx^+ and Δx^- to Δy^+ and Δy^- , and consequently to Δy . Consequently, DeepLIFT's reveal-cancel rule offers a computationally efficient method for Shapley value estimation within the framework of a DNN model. Equation 3.14 shows that $C_{\Delta x^+ \Delta y} = \Delta y^+$ and $C_{\Delta x^- \Delta y} = \Delta y^-$ and therefore Δy^+ and Δy^- defined in Equation 3.14 are the contributions of an input x to the output y in the positive and negative respectively.

For DeepSHAP, the above rule is applied in two steps to determine the contribution of an input value x_i to the output value y of a node that has undergone an activation function f . First,

it assesses the consequence of a linear function $h = \sum x_i w_i$ acting as a single input to the activation function f . This input h is then split into its positive and negative constituents h_+ and h_- based on whether or not $w_i \Delta x_i > g$ for each x_i , where w_i represents the weight assigned to that input value, and g typically stands at 0 [43]:

$$h = \sum w_i x_i = h_+ + h_- \quad (3.16)$$

$$\begin{aligned} h_+ &= \sum_i w_i x_i [w_i \Delta x_i > g] \\ h_- &= \sum_i w_i x_i [w_i \Delta x_i \leq g] \end{aligned} \quad (3.17)$$

The baseline value for h , denoted as h^0 , is computed as follows:

$$h^0 = \sum w_i x_i^0 = h_+^0 + h_-^0 \quad (3.18)$$

h_+^0 and h_-^0 are determined through the following computations:

$$\begin{aligned} h_+^0 &= \sum_i w_i x_i^0 w_i \quad \text{if } [\Delta x_i > g] \\ h_-^0 &= \sum_i w_i x_i^0 w_i \quad \text{if } [\Delta x_i \leq g] \end{aligned} \quad (3.19)$$

with the difference-from-reference for h contributing to Δy being:

$$\begin{aligned} \Delta h &= h - h^0 \\ \Delta h_+ &= h_+ - h_+^0 \\ \Delta h_- &= h_- - h_-^0 \end{aligned} \quad (3.20)$$

Therefore, regarding h as the fresh input to the activation function f , we determine the Shapley values of h_+ and h_- 's impacts on Δy by applying an adjusted version of DeepLIFT's reveal-cancel rule, outlined as follows:

$$\begin{aligned} \phi_{h_+}(\Delta y) &= \frac{1}{2}[(f(h_+ + h_-) - f(h_+^0 + h_-)) + \\ &\quad (f(h_+ + h_-^0) - f(h_-^0 + h_+))] \\ \phi_{h_-}(\Delta y) &= \frac{1}{2}[(f(h_+ + h_-) - f(h_+ + h_-^0)) + \\ &\quad (f(h_+^0 + h_-) - f(h_+^0 + h_-^0))] \end{aligned} \quad (3.21)$$

To determine the exact contribution of its to Δy , the Shapley values obtained on each of the non-negative or negative quantities stated in Equation 3.21 are then propagated linearly back

to the input value x_i . This propagation direction is determined on whether $w_i\Delta x_i > g$: This propagation direction is determined based on whether $w_i\Delta x_i > g$:

$$\phi_{x_i}(\Delta y) = \begin{cases} \frac{\phi_{h_+}}{\Delta h_+} w_i \Delta x_i & \text{if } w_i \Delta x_i > g \\ \frac{\phi_{h_-}}{\Delta h_-} w_i \Delta x_i & \text{if } w_i \Delta x_i < g \end{cases} \quad (3.22)$$

Then, the same is done for each node in a layer and all layers up to the next layer. For further contributions, these outputs are fed back to the neural network's input features. Since DeepLift adheres to the chain rule in Equation 3.11, the product of each allows us to define a multichain-rule multiplier that acts upon the neurons. Thus, writing the DeepSHAP definition in a more general form as inputs to output, we have:

$$\phi_i(y, x_i) = m_{x_i y}(x_i - E(x_i)) \quad (3.23)$$

where $y = f(x)$ is the model or equation output, x_i is the model input feature, the reference value $E(x_i)$ is the mean of the feature effect and $m_{\Delta x_i \Delta h} = \frac{\Delta y}{\Delta x}$. Effectively, a linear approximation of the Shapley values with characteristics of both Equation 3.8 and Equation 3.9

3.3.5 Reference values

The reference values used in feature attribution are very important, for it is against this point that each feature's contribution is determined. The baseline works by defining what is missing or neutral and how we compute feature contributions. Common options for baselines include:

- **Domain knowledge baseline:** Based on an expert's domain knowledge, features are set to a value.
- **Zero baseline:** Absence of a feature is simulated by setting its value to zero.
- **Mean baseline:** The mean of the training data is used as a neutral reference.
- **Dataset samples:** To capture feature interactions better, we choose representative samples from the dataset.

Since we are primarily working with DeepSHAP, we will use the dataset samples as a baseline. Since the validity and credibility of these feature attributions are dependent on their baseline values, choosing the right baseline values is indispensable for DeepSHAP and Generalised

DeepSHAP (G-DeepSHAP) (explained in later sections). Shapley values are different depending on baseline choices.

3.4 KnowIt

As introduced in Section 1.1, KnowIt (**Knowledge discovery In Time series data**)² is a Python toolkit built specifically for training DNNs and analyzing their results for time series data. It leverages two key libraries: For model development, there is PyTorch Lightning³, and for model interpretation, there is Captum⁴, which incorporates a vast and well-structured functional module set for temporal data analysis.

On top of PyTorch, of which it is a fork, PyTorch Lightning unburdens DL research by handling much of the repetitive code with the help of a higher-level API, which enables researchers to spend more time on models and experiments, not code and infrastructure [45]. Captum, derived from the Latin word for comprehension, provides several interpretability methods, DeepLift and DeepSHAP, that help explain what the PyTorch model conveys. It nicely combines with domain-specific libraries, including torch-vision and torch-text, to support model explanations across diverse domains [46]. Both tools add value to the creation and analysis of time series models and thus form part of the KnowIt platform.

3.4.1 Usage

KnowIt supports a variety of tasks on equidistantly spaced time series:

- **Time series regression:** Uni- and multivariate regression problem, one-step or multi-step predictions, auto-regressive model handling.
- **Time series classification:** It can perform multivariate classification tasks, particularly single-step commands to build prediction models, and can deal with both binary and other classification classes.

²<https://github.com/MUST-Deep-Learning/KnowIt>

³<https://lightning.ai/docs/pytorch/stable/>

⁴<https://captum.ai/>

-
- **Time series forecasting:** Based on past data, mathematical models such as autoregressive time series regression with delay are used to predict future values.

3.4.2 Functionality

A key functionality of KnowIt includes:

- **Importing datasets:** The data import arguments can also be provided by custom data structures to allow users to import their custom datasets as well.
- **Training models:** The main focus involves training DL models using custom-defined architectures, such as TCN or multilayer perceptrons, to perform forecasting and classification. Features such as hyperparameter tuning, early stopping, and real-time updates of training curves are supported. KnowIt includes these capabilities as well.
- **Generating predictions:** Given models, the users predict unknown values given new features and one-step or multi-step ahead values with KnowIt. Through learned temporal patterns, these predictions are made in areas like anomaly detection, demand forecasting, or risk assessment in an efficient manner.
- **Interpreting model predictions:** However, these are featured through the use of the Captum library, which now will interpret the model's decisions philosophically with regard to feature importance, transitioning from one state to another, and the logical way the model has chosen to make its decision. This is a very substantial functionality for users to explain model operations, justify predictions, and control model operations in the field of finance and health processes.
- **Evaluating model performance:** In addition to training models and interpreting results, KnowIt delivers a suite of evaluation criteria for time series data, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), etc. As a result, users are able to compare the actual and estimated data and change it accordingly if it does not fit.

On top of that, know-it also has convenient ways to connect user-defined PyTorch architectures and Pandas-based datasets. The toolkit is flexible in its form, which allows its tuning in terms of matching research and also areas of specialization like finance, health care, industrial analysis,

etc. Artificial intelligence solutions in the time-critical applications for interpreting complex models is another area that know-it focuses on.

3.5 Conclusion

In this chapter, we described the DL techniques that are central to this study. We reviewed two key architectures, TS-MLP and TCNs, which will be revisited in Chapters 6 and 7. We also described the theoretical foundation of Shapley values and their applications, as well as the know-it toolkit, which will be used in Chapters 6 and 7.

Chapter 4

Test bench design

‘Plans are nothing; planning is everything.’ - Dwight D. Eisenhower

4.1 Overview

The test bench is developed to compare and validate model interpretability techniques through the creation of synthetic data, training of the ML models, and extraction of the feature attributions for the purpose of evaluating the quality of the feature attributions. The implemented workflow is sequential, data is configured as input, and evaluations are yielded as output, yet it is sufficiently versatile to be suitable for various applications.

This chapter presents the details of the functional design of the test bench, with a particular focus on controlling data flow and model configuration, in addition to the shared processes that integrate the components.

4.2 Main components and interactions

The test bench is organised into three main tasks: data generation, model training, and evaluation of attributions. The system using this format is also flexible, in that all of its subsystems have defined inputs and outputs, which may be used in different ways.

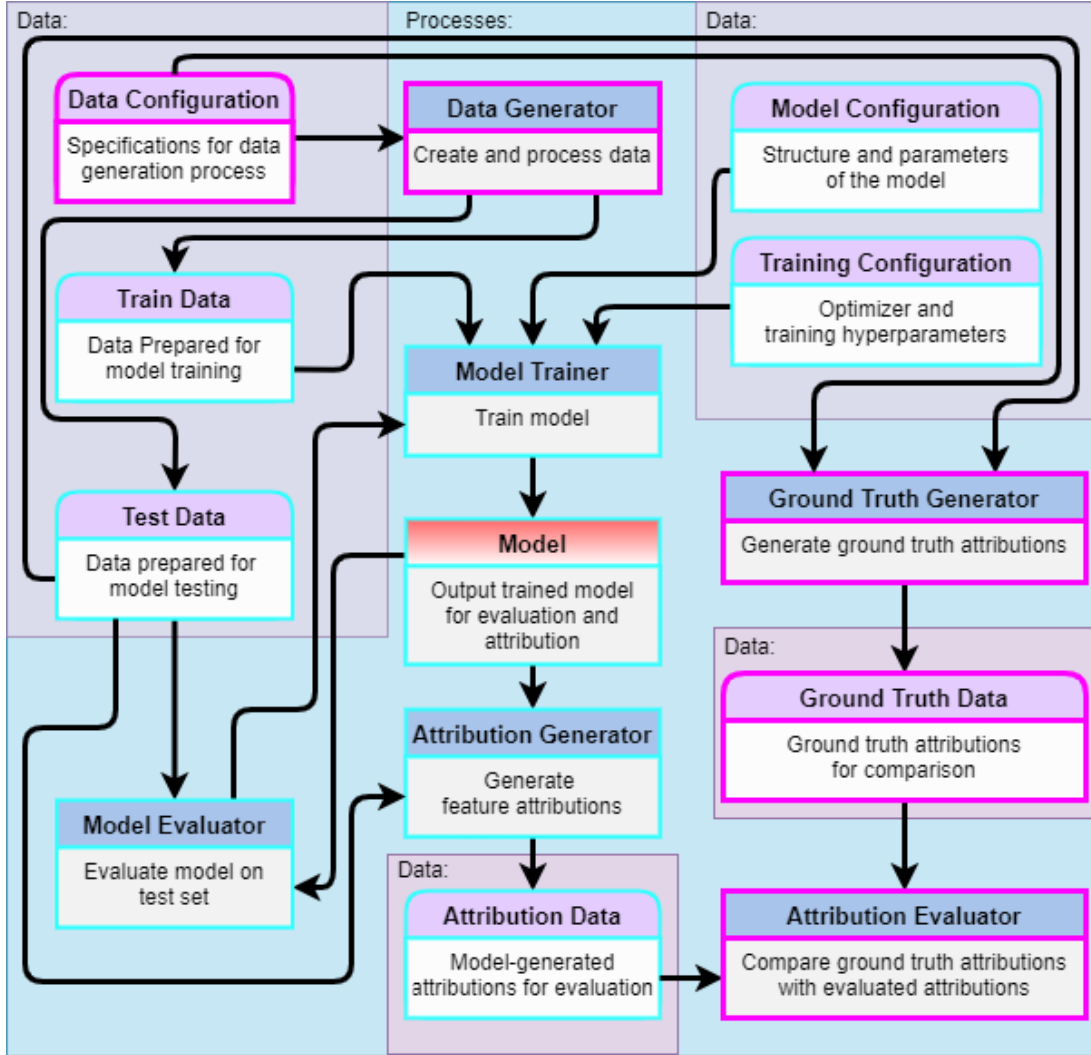


Figure 4.1: Functional design of synthetic testbed. The purple-highlighted boxes show newly developed systems, and the light blue boxes show pre-existing systems.

A high-level view of functional flow in the testbed design is provided in Figure 4.1, depicting how each of the modular components works together to accomplish the evaluation process. The *Data Configuration* is used to create the datasets for training and testing. The *Data Generator* uses this configuration to create the *Train Data* and *Test Data* sets. Thereafter, the *Model Trainer* trains the model using *Train Data*, with the provided *Model Configuration* and *Training*

Configuration, resulting in a trained model that can be further evaluated.

Then, the *Model Evaluator* assesses the accuracy of the model, using *Test Data* to generate performance metrics. The *Attribution Generator* calculates feature attributions for this trained model. The *Ground Truth Generator* uses analytical methods to produce theoretical feature attributions (*Ground Truth Data*) to serve as a baseline for evaluating the model-generated attributions. Finally, the *Attribution Evaluator* takes *Model Attribution Data* and assesses the data quantitatively against *Ground Truth Data* to determine whether a model’s interpretation was correct.

The generated data, in addition to the components developed specifically as part of this study, are highlighted with a purple border in Figure 4.1, while standard components, which are based on existing software packages, are shown with a turquoise border. Because we carefully control these components, the test bench guarantees flexibility and consistency in generating datasets for different use cases.

4.3 Data

Data is central to the evaluation of model interpretability. The evaluation includes both parameter configuration (what data will be generated) and generated datasets. The test bench carefully controls these components to create datasets that are flexible and consistent for different use cases. Data-related components consist of configurations and generated data. Each component is described separately in the following sections.

4.3.1 Data Configuration

Data Configuration is the configuration of the parameters that determine how data is generated, structured, and used in model training and testing. The number of samples, the distribution over each feature (regular, uniform, or otherwise), and the total available training and testing sets. In addition, a transfer function is defined that takes the input features and creates outputs, simulating the relation to be learned.

The *Data Configuration* setup is designed in Chapter 5.3. *Data Configuration* ensures that the

data which are produced and used will fit the user’s requirements through the user’s ability to specify data aspects. For example, to model some data for predicting economic indicators, the data set would include aspects that are close to the real economy, such as flows and seasonality. The ability to customise this enables users to recreate complex real-world phenomena, such as economic cycles, and to develop scenarios for experiments to understand model sensitivity to variations in feature distribution.

4.3.2 Model Configuration

Model Configuration specifies the model type (TCN, MLP) and model architecture, such as the number of layers, number of filters, etc, which are specific to these models.

This is the source of how the model processes and learns from data, and is a preexisting component. Model configuration is described in more detail in Appendix A.2.

4.3.3 Training Configuration

Training Configuration configures the hyperparameters to train a model. Settings describe aspects such as chosen optimiser, learning rate, batch size, number of training epochs, or choice of regulariser. These hyperparameters define the learned model to a great extent, as well as the performance of the model, because they determine how to learn and optimise. Again, this is a pre-existing configuration component. A discussion of specifications for training configuration can be found in Appendix A.3.

4.3.4 Train Data

This is the training dataset used to train the model. It is a model that will learn to associate a set of input variables with a set of output variables. This data matches the generated data structure and matches the format of the training data set. Typically, the training dataset is split into two parts. The first is a training subset, and the second is a validation subset. This ensures that we do not train the model on test data (and maximise performance on the train subsets while checking that the model performs as well as possible on the validation subsets).

The most common use of this is when performing the train/validation split early in the data preparation.

4.3.5 Test Data

With *Test Data*, the system evaluates the performance of the model. During training, the model could not see the test data, giving us an indication of the model’s ability to generalise on unseen data. The test data serves two purposes: evaluation data obtained are used to assess the model’s predictive performance, and as data for interpretability analysis to extract and compare feature attributions.

4.3.6 Attribution Data

The *Attribution Data*, referring to feature attributions generated when the trained model is applied to the test data, is obtained when the feature attributions are derived for the test data. These attributions can then tell us what each feature contributed to the model prediction and how the model could have come to its decision. From the attribution data, the users can see which features have the highest impact on the model output and can further interpret the model.

4.3.7 Ground Truth Data

Ground truth feature attributions (labelled ‘*Ground Truth Data*’ in Fig 4.1) are usually achieved using ASVs as explained in Section 2.4.5. The ground truth for our comparisons of model-generated attributions is the set of these attributions. The system can compare the ground truth data with the attribution data to evaluate how good the model’s feature attributions are and give a precise measure of how well the model’s decisions are aligned with theoretically sensible feature importance values.

4.4 Processes

The components in this section process and execute tasks. All the processes in the value chain have well-defined inputs, and each activity fulfils a particular function. It is then passed out to the next step.

4.4.1 Data Generator

The *Data Generator* is responsible for generating datasets that are used for both training and testing the model. These datasets are procedurally generated, meaning that their size and distribution are controlled by the user. (See Section 5.2.4 for the layout of the datasets.)

- Inputs: *Data Configuration* (data size and distribution, the features to select, and the transfer function to link to the output).
- Outputs: *Train Data* for model training and *Test Data* for subsequent model evaluation and attribution generation.

Data generation is intentionally designed with a great deal of versatility and can be used to create datasets similar to real-life situations.

4.4.2 Model Trainer

With the *Train Data* created earlier, the *Model Trainer* trains a DNN. This work focuses explicitly on TCN and TS-MLP. However, the system is flexible, allowing users to choose different architectures.

- Inputs: *Train Data*, *Model Configuration*, and *Training Configuration*.
- Output: A trained *Model* ready for evaluation and attribution generation if it succeeds.

After training, the trained *Model* is employed in the other stages to assess its efficiency and to produce feature attributions.

4.4.3 Model Evaluator

The *Model Evaluator* uses the *Test Data* to evaluate the model's performance and its ability to generalise effectively. Performance is determined by the existing result of the prediction or outcome and the model's performance in data that has not been learned during data training through performance indicators like mean squared error.

- Inputs: *Model* and *Test Data*.
- Performance metrics include test loss and test accuracy for classification tasks, as well as test mean squared error (MSE) and test root mean squared error (RMSE) for regression tasks.

The results can be used to assess whether this model is suitable for other research or to indicate what has to be changed in the model to make it more precise.

4.4.4 Attribution Generator

The *Attribution Generator* estimates the feature attributions of a chosen model for *Test Data*. This component can support multiple attribution methods, such as DeepSHAP, Integrated Gradients, and LIME, therefore providing users with different ways to understand the decisions made by the model.

- Inputs: *Trained Model* and *Test Data*.
- Outputs: *Attribution Data* (Model-generated Attributions).

Attributions of these features exhibit how the features affect specific predictions and, accordingly, contribute to understanding model behaviour. The Attribution Generator is developed in Chapter 6, and uses the theoretical background as described in Section 3.3.4.

4.4.5 Ground Truth Generator

The *Ground Truth Generator* establishes the initial attributions to link to the Test Data. The *Ground Truth Attributions* are used as a baseline for evaluating the attributions coming out of the model without the risk of over-optimistically assessing the model’s interpretability.

- Inputs: *Test Data* and *Data Configuration*
- Outputs: *Ground Truth Data* (e.g., ASVs).

These attributions offer a theoretical guideline for spreading the importance of features relative to which model-generated attributions can be evaluated.

4.4.6 Attribution Evaluator

A quantitative assessment of the accuracy and similarity of the model-generated attributions with the *Ground Truth Data* is done by means of the *Attribution Evaluator*. It presents a means of evaluating the reliability of the attribution method by comparing the estimated feature importance to the baseline using test MSE.

- Inputs: *Attribution Data* and *Ground Truth Data*.
- Outputs: Metrics of attribution accuracy are designed to measure how closely the attributions of a model align with the ASVs. These metrics are further detailed in Section 6.4.

In this evaluation, the attribution method is assessed to determine how reliably it reflects the actual contributions of each feature in the model’s decisions. Section 6.4 describes the design of the evaluation process.

4.5 Conclusion

This chapter provided a functional design of the test bench system, including its main components, data structures, and processes. The system comprises three distinct tasks: data gener-

ation, model training, and attribution evaluation, all of which contribute towards the ultimate goal of systematically evaluating model interpretability techniques. Configurable parameters in data and model configurations are reused in the test bench, allowing the test bench to be adapted to different experimental needs.

The upcoming chapters are based on the functional design described in this chapter. The next chapter discusses the design of synthetic datasets and experiment configurations to detail how the test bench generates realistic data from which interpretability analysis can be performed. Subsequent chapters explain the evaluation metrics and attribution methods used.

These requirements form the foundation of the test bench design, which aims to provide a robust framework for evaluating and interpreting DL model behaviours through rigorous testing and analysis.

Chapter 5

Synthetic time series data

‘Data is a precious thing and will last longer than the systems themselves.’ - Tim Berners-Lee

5.1 Overview

In the real world, time series data tend to be noisy, exhibit missing values, or have insufficient data for model training [1]. We can use synthetic time series data to overcome these limitations by creating high-quality data sets through simulations with controlled characteristics [47]. A synthetic data test bench provides the ability to generate data and models in a structured manner within a controlled environment, enabling precise control over the relationship between input and output features, which is a central focus of this work. The data design process and results are described in Section 5.2 before providing examples of generated data in Section 5.3.

5.2 Data design

We start the design process by defining the goals of synthetically generating data in Section 5.2.1. Section 5.2.2 then presents the essential constituents of synthetic time series data from time effects to input features and transfer functions. We specify the data generation parameters

in Section 5.2.3, including the trends, oscillations, and phase variations. The system implementation and its functionality are described in Section 5.2.4, including details of the configurable parts of the system that allow the creation of customised datasets to suit different users' needs.

5.2.1 Goals

The data is designed for the evaluation of interpretability techniques. The design process consists of defining relationships between input and output features, specifying dependencies among different input features, and specifying the dependency of input features on their time variables. This design allows for the systematic evaluation of DL interpretability techniques through controlled experiments.

5.2.2 Defining synthetic time series features

We discuss how we generate time series data synthetically and how the interesting dynamics of the data arise from the key features of the synthetic process. In particular, we introduce elementary data building blocks of time effect, input features, and temporal features. These combine to structure and dynamic synthetic data, which we call the transfer function. We want to build these elements into time series data to build controlled datasets that will introduce dependencies and patterns that would help us see how this would affect the DL models.

Input features (X)

Each input feature consists of a vector $\mathbf{x}_i$ of time-specific values. The entire set of input values X is denoted $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ with n being the number of features. Each feature $\mathbf{x}_i$ takes one of a particular value at every time instance t and can be expressed as a function of time $f_i(t)$:

$$x_i(t) \equiv f_i(t) \tag{5.1}$$

t here we denote as discrete as we state it in either seconds or hours. From these input features, we build the time series that constitute the system dynamics we are studying.

Temporal features (Z)

The input features can affect the output with different types of time dependencies (for example, delays). We propose a general class of temporal features derived from input features by applying some fixed time shifts or transformations so that we can consider both past and future information inside the model. For each input feature x_i , a time shift function τ_j is used to generate a new temporal feature:

$$z_j(t) \equiv x_{index^j}(\tau_j) \quad (5.2)$$

$z_j(t)$ defines coordinates in the feature matrix X , in other words a feature (column in X) at a time (row in X). Note that there may be a different number of input and temporal features. Z itself is therefore a 3-dimensional tensor: each j associated with 2 values.

Because of this shifting, the model is able to use knowledge from other time points and exploit temporal patterns and dependencies.

Time effect (τ)

The first component of a time series is the *time effect*, denoted by τ , which captures temporal dependencies within time series. It stands for temporal displacement between observations. A time effect can be a leading or a lagging impact for a time variable, that is, whether it relates past observations to the current state or future observations are considered. A simple example of the time effect would be the delayed signals received when communicating via cellphone with someone far away. We mathematically define time effects as follows:

$$\tau_{\text{lag}} \equiv t - k \quad (\text{lagging effect}) \quad (5.3)$$

$$\tau_{\text{lead}} \equiv t + k \quad (\text{leading effect}) \quad (5.4)$$

$$\tau_j \equiv g_j(t) \quad (\text{custom dynamic effect}) \quad (5.5)$$

Here, k is a fixed time interval, and $g_j(t)$ represents a more complex time-dependent function.

Transfer function and target variable (Y)

The relationship between the temporal features and the target variable Y is embodied in a transfer function $\mathcal{F}()$. This is a description of how time-shifted input features are transformed

to explain the target variable:

$$\mathbf{y}(t) \equiv \mathcal{F}(Z, t) \quad (5.6)$$

where $Z = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_m\}$ and $Y = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_k\}$ for k output variables. If there is only a single output, this is simplified to $\mathbf{y}$.

This approach allows us to develop a fine-grained control over the relationship between the input and target variables while including temporal dynamics to model systems where the observations evolve. We further explain the input features, temporal features, and time effects to Equation 5.6 in the following sections.

5.2.3 Specifications to consider

Several features should be considered before generating synthetic time series data because they add complexity and data interdependency. These aspects include:

- **Polynomial components:** Introducing polynomials of long-term trends allows for their effects on the model predictions [48]. Linear, quadratic, and such trends are polynomial functions that can present an increase or decrease in data values over time and put pressure on the model to recognise those patterns.
- **Oscillations:** Cosine and sine functions can be added to the model to represent seasonal or cyclic behaviour. These oscillations bring in certain levels of lithium bump that could be part of cyclic fluctuations familiar with time series data in real-world settings and which models must consider and correct for [49].
- **Trends:** To stand for changes up or down through time, linear or nonlinear trends must be modelled. Instead, the trend components offer examinations of these aspects to determine whether a model can identify and extend these long-term trends [48], [50].
- **Phase changes:** modelling sudden changes, or regime shifts, often presents challenging conditions that strain models' adaptive and stable characteristics. The assumptions behind these breaking points or structural changes in data mimic real-world situations when the data-generating processes of systems undergo dramatic shifts. However, this means that models need to learn and quickly adapt to a new data environment [48].

-
- Length: All datasets include time series of different lengths, either short or long-term sequences. This variation is tested with different temporal spans and sample sizes to test the model performance [51].

Constructing datasets with these aspects is essential to supporting a thorough assessment of both model performance and interpretability.

5.2.4 Data description

The process for synthesising different patterns and behaviours in the data is configurable, so several patterns and behaviours can be applied to the synthetic time series data. Key parameters and configuration options include:

- Frequency: This parameter decides the frequency of the oscillation as would be p in $\sin(pt)$. A larger p is equivalent to a higher frequency as it means there are more oscillations per second, t . This is relevant to the periodicity of real-world time series with the seasonal or cyclical characteristics described in Section 5.2.3.
- Amplitude: The amplitude parameter enables one to generate a time series with different levels of volatility indicated by its height. This parameter is connected with the intensity of oscillations discussed in detail in Section 5.2.3, as it defines the range of cyclic changes in the vertical direction. Symbolically, the amplitude is represented by the letter A in $A \sin(pt)$, and A is the measure of the height of the oscillations.
- Noise level: Changing the structure of the time series, the noise level parameter introduces random noise into the generated time series to simulate imperfections found in real-world data. This is consistent with the imperfections modelled for the robustness test in Section 5.2.3. Mathematically, noise is added by sampling from a chosen distribution. For example, Gaussian noise with mean μ and variance σ^2 is denoted as $\mathcal{N}(\mu, \sigma^2)$, where the variance controls the spread of the noise around the mean.
- Trend type: This option specifies the form of the long-term behaviour of the time series, i.e., how the values evolve as time progresses. By setting the trend type, the generator can impose linear, polynomial, seasonal, or exponential patterns on the data. These trends

are consistent with the long-term changes described in Section 5.2.3, where the model is tested for its ability to capture and extrapolate trends. Mathematically, these trends can be expressed as:

- *Linear trend*: $y(t) = c + mt$, where c is the intercept and m is the slope.
 - *Polynomial trend*: $y(t) = \sum_{i=0}^n c_i t^i$, where n is the degree of the polynomial, and c_i are coefficients.
 - *Exponential trend*: $y(t) = ce^{rt}$, where c is the initial value, and r is the growth (or decay) rate.
 - *Seasonal component*: $y(t) = A \sin\left(\frac{2\pi}{P}t + \phi\right)$, where A is the amplitude, P is the period, and ϕ is the phase shift. This captures repeating seasonal or cyclical patterns in the series.
- **Phase change points**: Specific time points at which abrupt changes occur in the time series pattern, introducing discontinuities and structural breaks, are defined by the phase change points. This corresponds to the phase changes discussed in Section 5.2.3, which model sudden regime shifts and test a model’s adaptability to learn patterns. Mathematically, these can be represented as:

$$y(t) = \begin{cases} f_1(t), & \text{if } t < t_c, \\ f_2(t), & \text{if } t \geq t_c, \end{cases}$$

where t_c is the phase change, $f_1(t)$ is the behaviour of the time series before t_c , and $f_2(t)$ is the behaviour after t_c . The change at t_c introduces a break in the sense that the statistical behaviour of the series (its mean, slope, variance, or generating function) shifts abruptly, creating a regime change that the model must adapt to.

- **Samples**: The final synthetic dataset size is determined by the number of sample parameters, so a time series of various sizes can be created. This connects to the length variation described in Section 5.2.3, which includes short, medium, and long-term sequences to test model performance.

Through a customised configuration file, we generate synthetic time series data. The data configures this file to be as precise as possible characteristics of the generated data is assumed regarding the number of features, number of data samples, and the type of temporal dynamics

on each feature. Beyond simple patterns like linear trends, the configuration supports a variety of complex waveforms, including ones such as sine and cosine. It also induces random noise, asymptotic behaviour, and time shifts.

Python implements the system, using widely used libraries such as NumPy¹ and pandas² for fast data operations. The generated data is visualised using the matplotlib³ library. One of the system's main features is the ability to create feature interactions and complex time-dependent behaviours via custom mathematical expressions. Python's built-in `eval()` function is employed to evaluate these expressions, enabling us to dynamically construct mathematical models as part of the data generation process.

Table 5.1 summarises the key configurable parameters in the synthetic data generation system:

Time shift function (`my_shift`)

A user-customisable time-shifting feature, `my_shift` modifies user-specified temporal changes to features in the time series. It offers the flexibility of fixed and dynamic shifts for generating of various temporal patterns. The `my_shift` function accepts a parent so the feature name, shift type, and a custom expression to define the shift can be passed in, for instance, the expression:

```
my_shift(feature_1, 'custom', '((2 * np.sin(0.0015 * t))/(3 * np.cos(0.001 * t))))')
```

A dynamically shifted version of `x_1` is produced based on a periodic function, which permits a rich temporal structure.

The resulting pattern is visualised in Figure 5.1. Within this experiment, we show, using `dynamic_shift` as an example, the effects of the dynamic shift on `x_1`. To generate these plots, the following mathematical definitions were applied:

- **Time variable:** The time variable t was defined from 0 to 30,000 units with 10,000 equal spacing points.

```
t = np.linspace(0, 30000, 10000)
```

¹NumPy: <https://numpy.org>

²pandas: <https://pandas.pydata.org>

³Matplotlib: <https://matplotlib.org>

Table 5.1: Summary of configurable parameters for synthetic time series generation.

Parameter	Description	Example Values
num_features	The number of features to generate in the time series	Integer (e.g. 3)
num_samples	The length of sample data (time steps) that will be created	Integer (e.g. 10,000)
time_index_frequency	The frequency of the time index in a DateTime string format	String (e.g. 1S for every second)
data.type	Specifies the type of data generation per feature. Supported types include: <i>sine</i> , <i>cosine</i> , <i>line</i> , <i>random</i> , <i>polynomial</i> , <i>exponential</i> , <i>step</i> , and <i>impulse</i> . Each type uses specific parameters: - sine/cosine : amplitude, frequency, initial_value - line : slope, initial_value - polynomial : degree, initial_value - exponential : base, asymptote, initial_value - step : step_size, initial_value - impulse : impulse_position, amplitude General parameters that apply across all types include: mean , std , and offset .	String (e.g., cosine, line, or random).
mean	The mean value for generating random data	Float (e.g. 5.0)
std	The standard deviation for data generation	Float (e.g. 1.0)
frequency	Frequency parameter for wave-based features (sine/cosine)	Float (e.g. 0.0015)
coefficient	Scaling factor applied to feature values	Float (e.g. 2.0)
value	Initial value for linear and polynomial features	Float (e.g. 1.0)
asymptote	Asymptotic value approached by exponential features over time	Float (e.g. -1)
gradient	Gradient applied to features with linear growth or decline	Float (e.g. 0.0005)
equation	Custom mathematical expressions defining interactions or shifts among features, evaluated using Python's <code>eval()</code> function	<code>@my_shift(x_1, 'custom', '((2 * np.sin(0.0015 * t))/(3 * np.cos(0.001 * t)))')</code>
transfer	Transfer functions applied to modify the outputs of equations, also evaluated using <code>eval()</code>	<code>"110.2 * ((expression1**2) * (expression3))**(-1/6)"</code>

- **Original sine wave:** The modelling of the original sine wave: x_1

$$x_1 = \sin(0.001 * t)$$

- **Dynamic shift function:** The dynamic shift is given by :

$$\text{dynamic_shift} = 2 * \sin(0.0015 * t) / (3 * \cos(0.001 * t))$$

The top plot in Figure 5.1 and the comparison between the original sine wave (x_1) and the dynamically shifted sine wave and illustrates how the applied shift may change the temporal structure of the wave is shown in Figure 5.1. In the bottom plot, the dynamic shift values themselves are visualised, which map the time-dependent variation introduced by the shifting function.

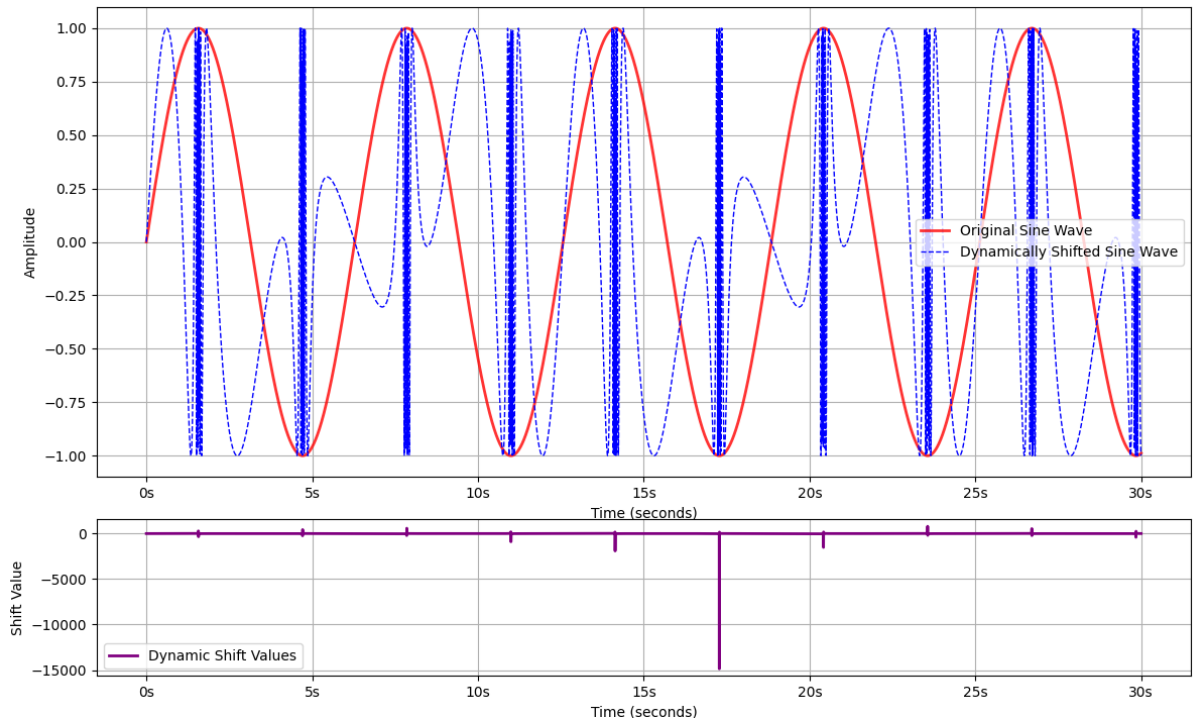


Figure 5.1: Plot of the dynamically shifted feature x_1 using the expression $\frac{(2 \cdot \sin(0.0015 \cdot t))}{(3 \cdot \cos(0.001 \cdot t))}$.

The top plot shows the feature effect, and at the bottom, the time shift values at a certain time are illustrated, while some regions appear dark blue due to intervals of very high-frequency oscillations.

Equation and transfer functions

Users can define custom mathematical expressions using the `equation` parameter, in which case it is possible to have inter-feature interactions or dependencies. The `transfer` applies transformations on the outputs of these `equations`, imparting more flexibility and complexity to the synthetic data. Python's `eval()` function both evaluates the expression based on `equation` and `transfer`.

We demonstrate the application of these parameters with a step-by-step example:

1. **Specify the number of samples and time units:** The number of samples is used for time index generation by the system, (`num_samples`) and the chosen time unit (`time_unit`). For example, we specify:

```
num_samples = 10000 and time_unit = "S"
```

results in a time index ranging from 0 to 10,000 seconds with 10,000 points.

2. **Specify the feature or features:** The `feature` parameter generates a feature by combining the user specifications that may be anything readable by Python's `eval` function. We illustrate;

```
equation = "np.sin(0.001 * t) + np.cos(0.0015 * t)"
```

This equation produces a time series feature that is cyclical in behaviour with two periodic components.

3. **Apply a transfer function:** The `transfer` generated feature is modified by a parameter using a nonlinear transformation. For instance:

```
transfer = "110.2 * (feature**2)**(-1/6)"
```

This scales the squared feature values and applies an inverse exponent. Using the feature value computed above, the transformation evaluates as:

$$\text{transformed_feature} = 110.2 \cdot \left([\sin(0.001 \cdot 1000) + \cos(0.0015 \cdot 1000)]^2 \right)^{(-1/6)}$$

4. **Introduce a custom time shift:** The `my_shift()` function can be applied to introduce dynamic time shifts. For example:

```
my_shift(feature, 'custom', '((2 * sin(0.0015 * t))/(3 * cos(0.001 * t))))')
```

The shift is computed dynamically on the basis of a periodic function. It relieves the feature so that its temporal structure is modified by shifting it forward or backward in time at each point.

5. **Add noise:** An additional applied random noise can be used to introduce the variability of the features. For example, the user specifies the standard deviation of the noise for every feature with:

```
noise = (mean=0, std=0.1)
```

The final feature becomes:

```
final_feature = feature + noise
```

6. **Generate synthetic data:** The above components are combined to generate synthetic time series data with complex interactions, nonlinear transformations, time shifts, and stochastic variability, and the system then produces that data. It supports different needs for simple patterns and complex behaviour.

This example shows the flexibility of the parameters of the `equation` and `transfer`, as well as the time-shifting and noise functionality. Full configuration details, including these parameters, are provided in Appendix A.4.

5.2.5 Limitations

The following section outlines the requirements met by this system and those that this synthetic time series data generation system did not meet.

Requirements met and unmet

The synthetic data generation system meets a number of important requirements, including:

-
- **Configurable patterns and behaviours:** Frequency, amplitude, noise levels, and trend types are customizable settings supported by the system, allowing the generation of many different synthetic data patterns. Of course, this makes mimicking common trends and cycles in time series data, such as linear, seasonal, and polynomial trends, much more accessible.
 - **Phase change points:** By incorporating phase change points, the system may introduce sudden shifts in the data about phase change points, representing structural breaks or regime shifts. While this is achieved through fixed points, dynamic detection of phase shifts based on data patterns or real-world event triggers is possible in the future.
 - **Feature interactions and temporal manipulations:** Complex interactions between features are simulated using the system, which includes lagging and leading effects, through the use of custom expressions and time-shifting functions such as `my_shift`. Although these capabilities span a broad class of time dependencies, further work may extend the class of complex causal interactions the system can handle.
 - **Dynamic phase shifts:** Static phase shifts are used via configurable points, but dynamic or adaptive phase shifts based on conditional rules (e.g., when data passes a certain threshold) were not provided. In concept, we might introduce conditional logic to the transfer functions responding to particular data patterns.

One requirement is not explicitly met:

- **Sophisticated nonlinear interactions:** While the system supports nonlinear combinations, multiplicative seasonal effects or stochastic volatility are not amongst the more advanced nonlinear dynamics supported. These could be implemented into models to provide additional realism for models that need complex feature interdependencies.

Comparison with other test benches

With respect to the scope of this system, it is helpful to also compare it to other test benches with different functionality:

-
- **Synthetic testbeds for attention-based models:** Barić et al. [15] developed a synthetic testbed that creates time-series data by simulating multiple variables with specific dependencies [15]. Their goal was to test whether attention-based deep learning models correctly focus on the most relevant time relationships. Rather than explicitly tagging key time points, their datasets are designed so that certain dependencies naturally act as important signals. While our dataset also generates structured dependencies among variables, it is designed with a more general interpretability focus rather than specifically for attention mechanisms. Unlike their work, which evaluates how models allocate attention across time steps, our dataset supports broader attribution-based methods. Additionally, our data allows for more flexible modifications of feature interactions, making it adaptable to different modelling approaches beyond attention-based learning. This difference makes our dataset complementary to theirs but applicable to a wider range of interpretability techniques.
 - **Randomised Time Series Generation (GRATIS)** [16]: The GRATIS framework generates synthetic time series based on stochastic processes. It uses a mixture of autoregressive (MAR) models to simulate data patterns with different characteristics and also contains more predefined options for data characteristics. Our system produces structured outputs by combining deterministic patterns with predefined noise levels. The system we implement can produce the same results as the GRATIS framework, but not with predefined options, but rather through equation manipulation. It is, therefore, more flexible but requires more work to construct a dataset. While our system guarantees reproducibility and the ability to evaluate explanations quantitatively, incorporating methods to better capture randomness could make it more suitable for testing models in unpredictable conditions.
 - **Post-hoc interpretability analysis tools:** The study ‘Evaluation of Post-Hoc Interpretability Methods in Time-Series Classification ’ [34] examines various post-hoc interpretability techniques for time-series models, including DeepSHAP, the method we focus on. Turbe et al. [34] also generate synthetic data in addition to real-world data. Synthetic data mainly uses combinations of sine waves with time shifts or disturbances for classification tasks. Sine waves are included in our data generation options, but we extend this with additional time-series elements. We focus on regression tasks rather than classification, specifically.

Potential extensions

This system is designed to produce basic components found in typical time series, but can be extended to include:

- **Event-based phase shifts:** The system could trigger phase changes by event-based indicators, which expands available types of regime shifts and structural breaks during simulated real-world events within the time series.
- **Advanced stochastic components:** These components strengthen the system’s unpredictable volatility processing capabilities. A GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model works as an adjustment method for temporal variability by detecting changes in volatility throughout time. Markov chains can be integrated into the system to handle state transitions, so the system becomes applicable to financial market analysis requiring unpredictable volatility assessment.

Briefly, the current system produces a synthetic time series with configurable parameters. Though capable of the task, its place within the broader research context reveals areas where it can be improved and made more consistent with advanced test benches.

5.3 Examples of generated data

In this study, the synthetic data generation system provides various features, such as generating realistic time series with trends, seasonality, noise, and time shifts. The system is flexible and generates examples of the type of data it can provide. We start by providing examples of very simple trends; when combined, these quickly result in more complex datasets.

5.3.1 Linear trend with noise and lag

The system can generate linear trends as well as add noise and then send back in time a time lag, as responses tend to be delayed. The relationship is defined as:

$$x(t) = m \cdot x(t - \text{lag}) + c + \text{noise} \quad (5.7)$$

Here, the input feature $x(t)$ is a measure related to the slope m and the intercept c ; noise is understood as a random perturbation.

Finally, in Figure 5.2, we demonstrate the input feature as a linear trend, and its corresponding transfer function shifted in time with added noise representing the system's ability to simulate lagged relationships in time series data. While this is a simple trend, combining such simple trends quickly results in complex surface behaviour.

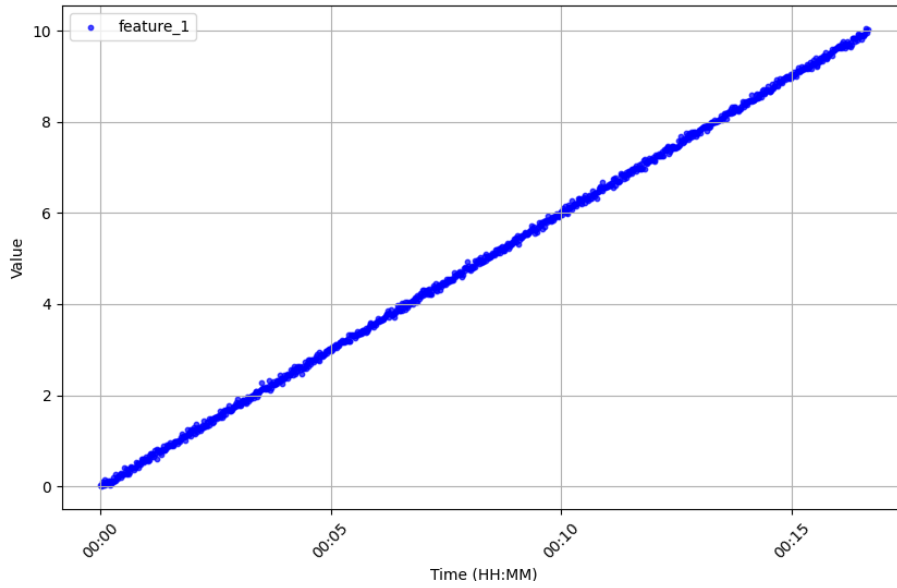
5.3.2 Seasonal (cyclical) data with lead

The synthetic data generation system enables modelling of periodic (cyclical) patterns found in real-world time series, such as seasonal sales data, temperature fluctuations, or economic cycles. The system can simulate repeating oscillations over time by producing a sine wave of a configurable frequency and amplitude.

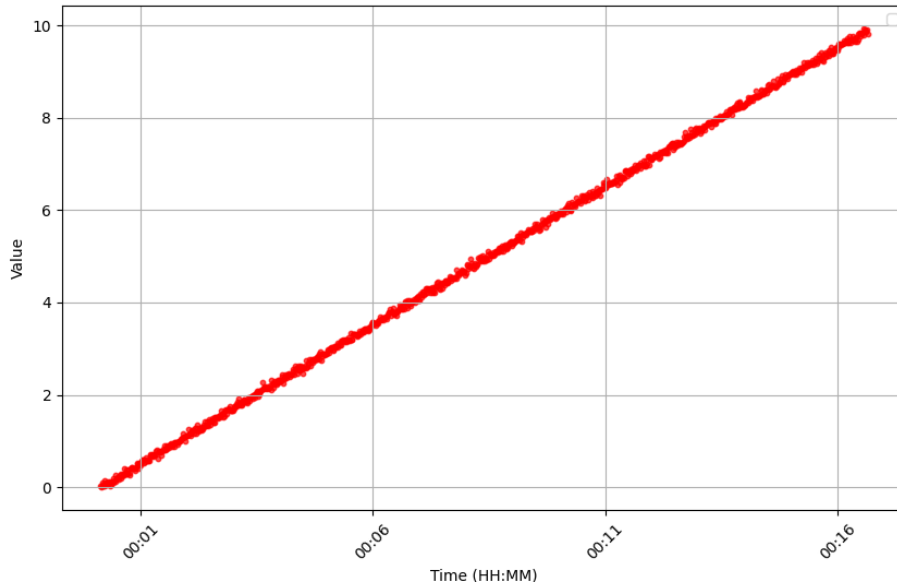
In this case, cyclical behaviour is approximated by a sine wave. Then, applying a lead effect complicates the system further in that the sine wave now behaves as if it is shifting forward in time. In this work, we achieve this shift with a custom function `@my_shift`, which shifts input values forward in time by several time steps. and, in essence, simulates the data ahead in time. This approach is helpful when the application demands using future events ('look ahead') to explain current events. This is to demonstrate the lead effect, even though the lag effect is more typical.

Key characteristics:

- Cyclical behaviour is a pattern that is generated as a sine wave that oscillates with a given frequency and amplitude.
- Using the `@my_shift` function to shift the generated data forward by four time steps, the system illustrates the lead effect. The lag effect, lead effect, or a combination of the two is also possible, as well as a dynamic time effect. The peaks of the sine wave occur before time, and, as such, the data looks more forward-looking.
- A sine wave is created with some included 'small' noise added (user-determined, with the user specifying the amplitude and the distribution of the noise).



(a) Input Feature Data



(b) Transfer Function Data

Figure 5.2: Generated input feature data and the corresponding transfer function for a linear trend with noise and a lag.

The result is then an approximation of real-world cycles, including some external disturbance of the sine wave. In Figure 5.3, we see the sine wave input and how the system can produce seasonal behaviour (cyclical if another feature is added) and a leading time shift. The frequency,

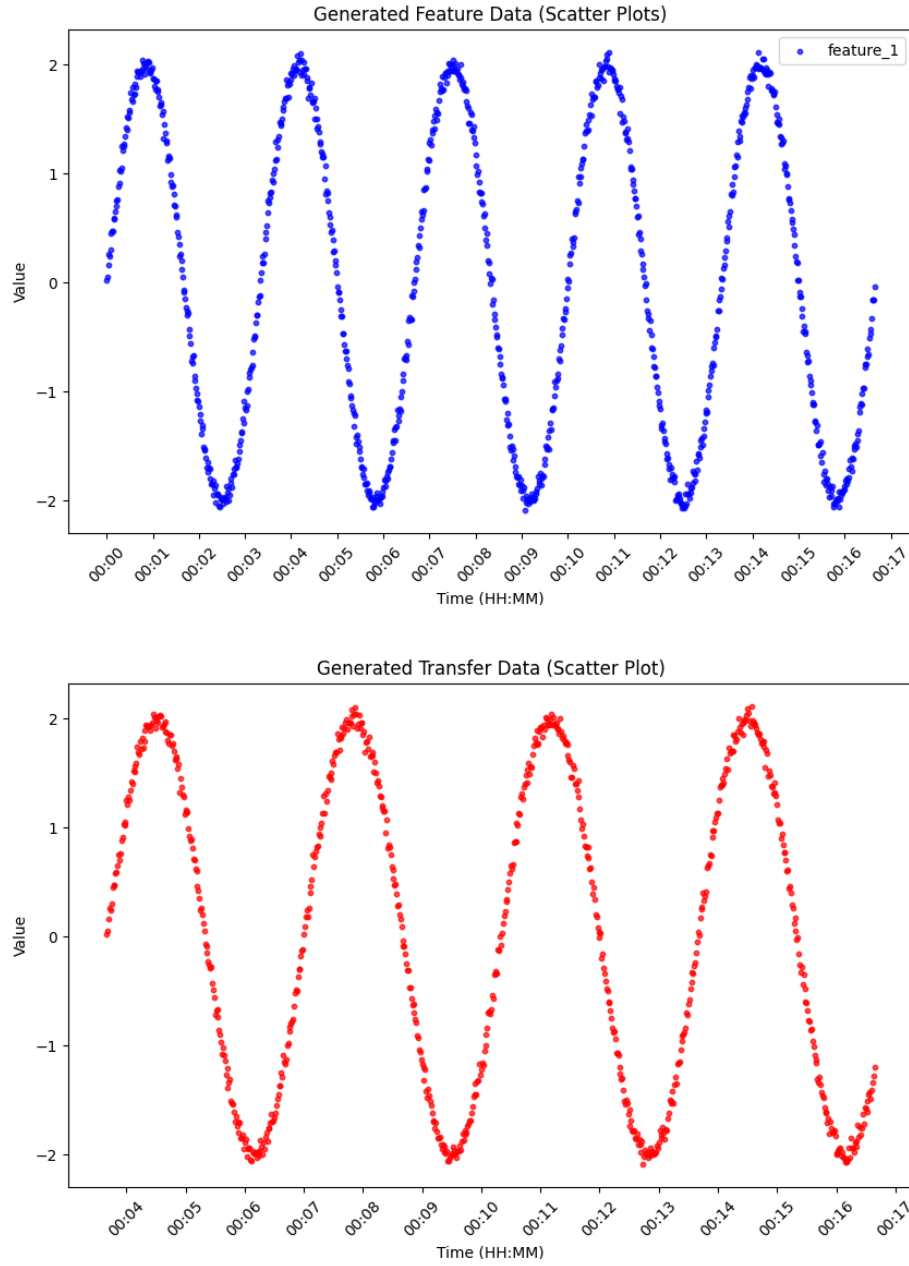


Figure 5.3: Input feature generated (blue) and the correlating transfer function output for seasonal data (red).

amplitude, and lead are changed to produce various seasonal data, such as daily cycles or longer seasonal fluctuations.

5.3.3 Combining trends and cyclical components

Long-term growth or decline is given in cyclical fluctuations in many real-world systems. The general synthetic data generation system is then enabled to generate more varied datasets from input, which combine trend and cyclical components for closer simulation of real-world phenomena such as stock market prices, climate patterns, or economic datasets.

For now, we add a linear trend with a cyclical sine wave to create growth or decline with a regular oscillation. Since this is a set of data, it is useful for testing models that want to better separate long-term trends from short-term fluctuations.

Key characteristics:

- Trend component: This is an indication of a linear trend, an increase (or decrease) over time. However, control over different change rates over time allows one to dink it around, so this is excellent for longer-term growth/decline simulation.
- Cyclical component: If we have a sine wave, then we've got a repeating oscillation. This component can be tuned to frequency and amplitude based on short-term cycle behaviour and cyclically over seasonal behaviour.
- Combination of components: By adding trend and cycle, these data include both the long-run directional signal and short-run variations, which give a more realistic description of many natural systems.

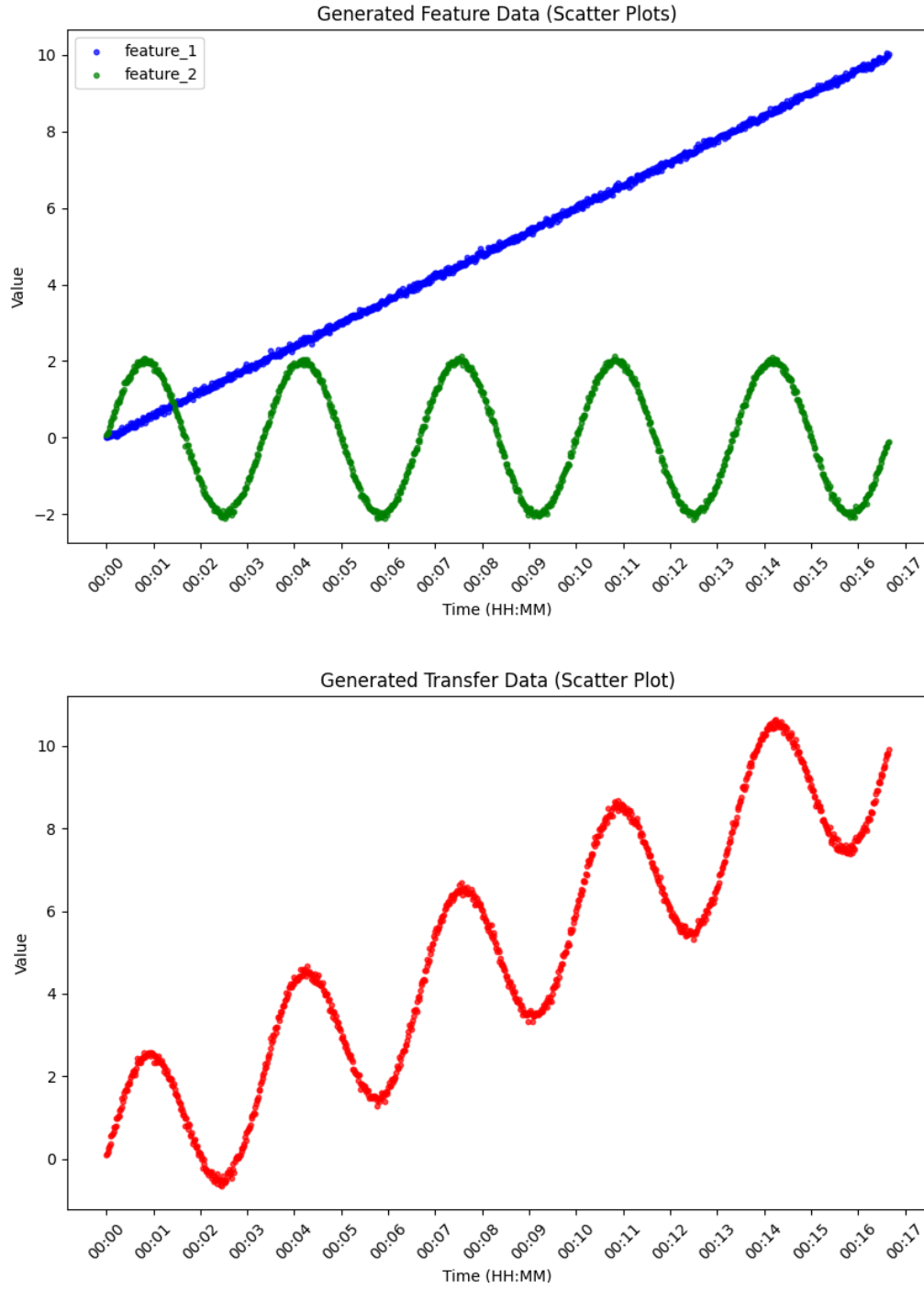


Figure 5.4: Input feature generated (blue and green) and the correlating transfer function output (red) for combined trend and seasonal data.

Figure 5.4 illustrates the system's capability of generating and combining independent features in a transfer function for output.

Data of this variety can be helpful for tasks where this type of distinction matters as much as it does for financial forecasting, where an analyst has to consider both regular market growth and short-term temporal impulses.

5.3.4 Nonlinear combinations of trends and cyclical features

Here, we demonstrate that such a system facilitates the creation of complex, nonlinear interactions, trends, and cyclical components and has the ability to support **time shifts** (lags and leads) via a custom nonlinear time-shifting function. The features themselves are a mix of **linear trends** plus **cyclical behaviours** (sometimes represented simply as sine and cosine waves) passed through **nonlinear equations** to yield complex patterns.

Non-linear transfer function

A nonlinear *transfer function* is applied to combine the shifted features:

$$\text{transfer1} = 110.2 \cdot (\text{x_1}^2 \cdot \text{x_3})^{-1/6}$$

The final output is a result of a nonlinear combination of the time-shifted cosine wave (by **x_1**) and the time-shifted linear trend (by **x_3**) in a way in which the square root of the time-shifted cosine wave is passed as input together. As shown in Figure 5.6, the system possesses the ability to combine multiple dimensions of nonlinear features for an overall nonlinear output and can produce customised time widgets, allowing for multi-dimensional random datasets that depend on features and time. In Figure 5.6, note that the system is able to process linearly merged feature columns and nonlinear transforms and create a convoluted data set with features that may not even be important to the output (**x_2**). This makes the data it generates nonlinear in the sense that the interactions between trends and seasonality look like those that occur in real-world financial markets, weather patterns, etc.

The configuration used to generate this specific example includes the parameters below:

- **Cosine wave** (x_1): Generated with a coefficient of 3 and a frequency of 0.001 to represent cyclical behaviour.
- **Sine wave** (x_2): Generated with a coefficient of 2.0 and a frequency of 0.0015.

-
- **Linear trend** (x_3): A linearly increasing feature with an initial value of 1.0, an asymptote of -1, and a gradient of 0.0005, simulating a steady long-term trend.

Custom time shifts (lead and lag) The system applies custom *time shifts* to these features using dynamic shifting formulas:

- **x_1** (cosine wave) is time-shifted using the formula:

$$\Delta t_1 = \frac{2 \sin(0.0015 \cdot t)}{3 \cos(0.001 \cdot t)}$$

This is a formula for how many shifts there are at each time of a trigonometric function and at each other time of a cosine wave. It is delayed or advanced irregularly at different times, so as it were, it changes.

- **x_2** (sine wave) undergoes a similar time shift:

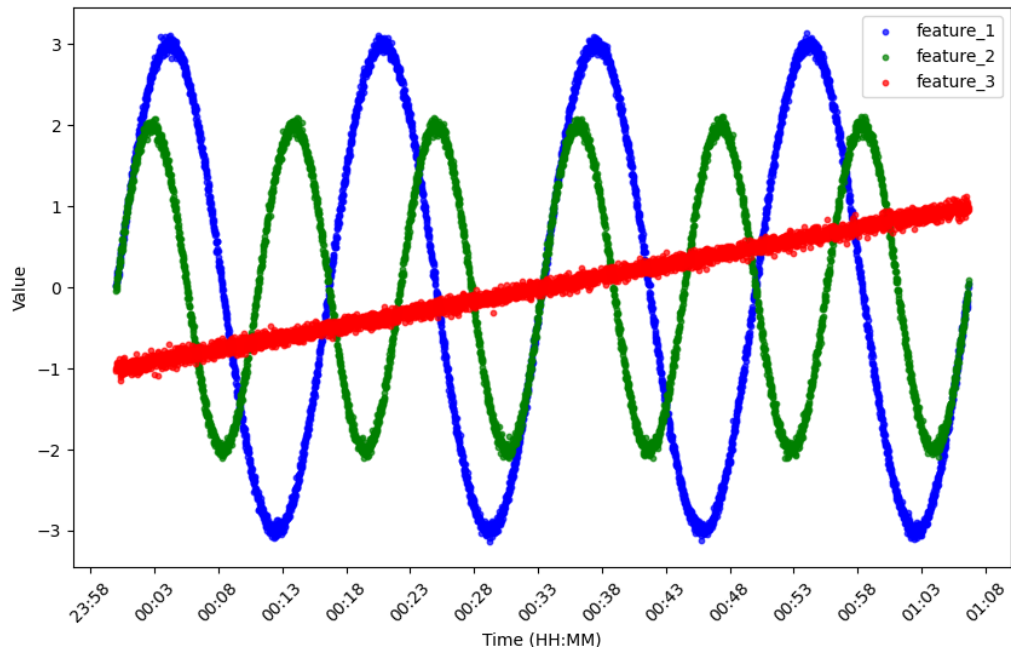
$$\Delta t_2 = \frac{2 \sin(0.0015 \cdot t)}{3 \cos(0.001 \cdot t)}$$

A nonlinear shift of the sine wave is obtained by dynamically adapting the points in time forward and backward according to the sine and sinus function oscillations.

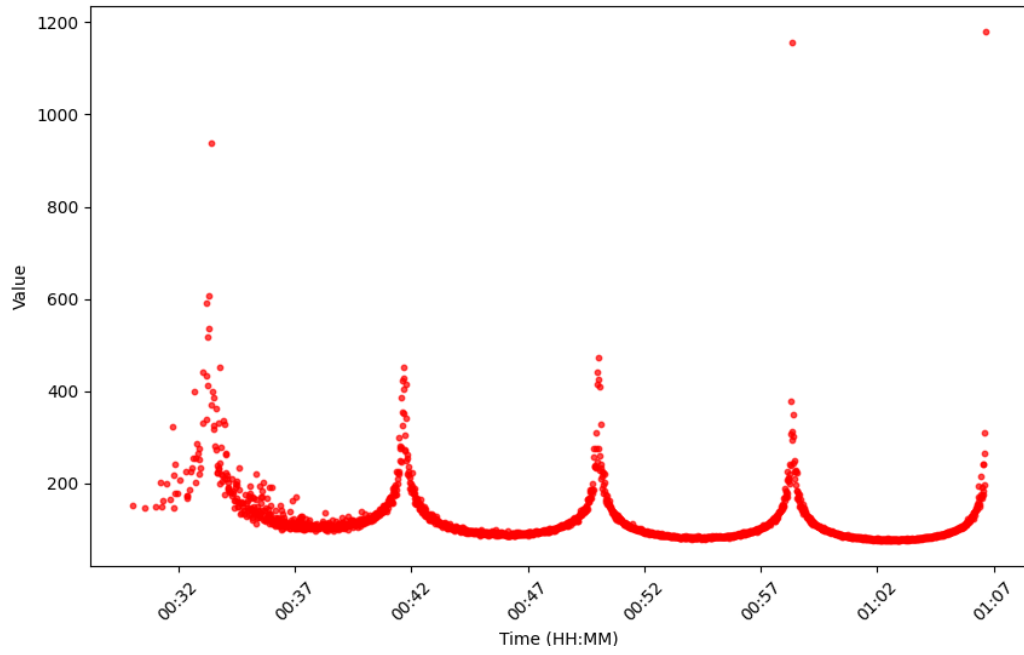
- **x_3** (linear trend) is time-shifted using a simple lead, where a one-time step shifts the data forward:

$$\Delta t_3 = 1$$

This creates a time lead for the entire series, simulating a constant future prediction behaviour.



(a) Input features.



(b) Transfer function.

Figure 5.5: The output of the transfer function combines trend and cyclical components in a nonlinear way using the example values in Table 5.1.

5.3.5 Complex nonlinear input and output relationships

In this configuration, we create complex, nonlinear synthetic time series data by applying advanced mathematical transformations to each feature of the system. Various components of nominal, logarithmic, trigonometric, seasonal, and noise are taken to induce variability, therefor representing real-world data complexity. We develop a system that generates time-shifted features via lag and lead. It combines them through various equations to derive the intricate patterns nominated for applications that require the computations on high-dimensional temporal dynamic datasets.

Input features:

- **x_1** (x_1): Defined as

$$12 + (\exp(\sin(t \cdot 0.01))^2) \cdot 0.5 \cdot \log(1 + t \cdot 0.01) \cdot 5 \cdot \cos(t \cdot 0.02)^2 + \sin(t \cdot 0.05) \cdot 4 + \mathcal{N}(0, 0.5)$$

This feature does an exponential rise to the sine of the time index and cosine wave for oscillatory behaviour, and times this by a logarithmic growth. There is, moreover, a low-frequency seasonal sine wave, as well as random noise. To capture the delayed dependencies, we lag this feature by 4 time steps.

- **x_2** (x_2): Defined as

$$50 + (\log(1 + t \cdot 0.01) \cdot 5 \cdot \cos(t \cdot 0.02)^2) \cdot \sqrt{|\tan(t \cdot 0.01)|} - (\exp(\sin(t \cdot 0.01))^2) + 7 \cdot \cos(t \cdot 0.03) \cdot 3 + \mathcal{N}(0, 0.5)$$

This feature combines logarithmic growth, trigonometric transformations, high-amplitude seasonal cosine waves, and noise. It is time-shifted by three lag time steps to simulate forecasting scenarios.

- **x_3** (x_3): Defined as

$$350 + ((\exp(\sin(t \cdot 0.01))^2)^2) + \sin(\sqrt{|\tan(t \cdot 0.01)|}) + 5 \cdot \sin(t \cdot 0.02) \cdot 5 + \mathcal{N}(0, 1.0)$$

Additional variability in this feature is provided by squared exponential transformations, trigonometric functions, and high-amplitude seasonal sine waves with added noise. It is lagged by 1 time step.

Transfer function: Nonlinear transformation of the input features yields complex output data with interactions beyond simple additive or multiplicative combinations:

$$\text{transfer} = 110.2 \cdot \text{expression1} \cdot \left(\frac{\text{expression3}}{1 + |\text{expression2}|} \right)^{-\frac{1}{6}}$$

This transformation function scales **expression1** by a factor of 110.2, multiplying it with the output of **expression3** divided by the sum of 1 and the absolute value of **expression2**. The expression projects ranges of the features nonlinearly onto one another through the resulting nonlinear interaction, creating a balance between the variability and stability of nonlinear synthetic data.

Figure 5.6 illustrates the system's ability to synthesise highly complex input and output features. This has been demonstrated to offer utility in pattern making, whereby real-world patterns, such as changes to the market, seasonal effects, and/or cycles, can be simulated.

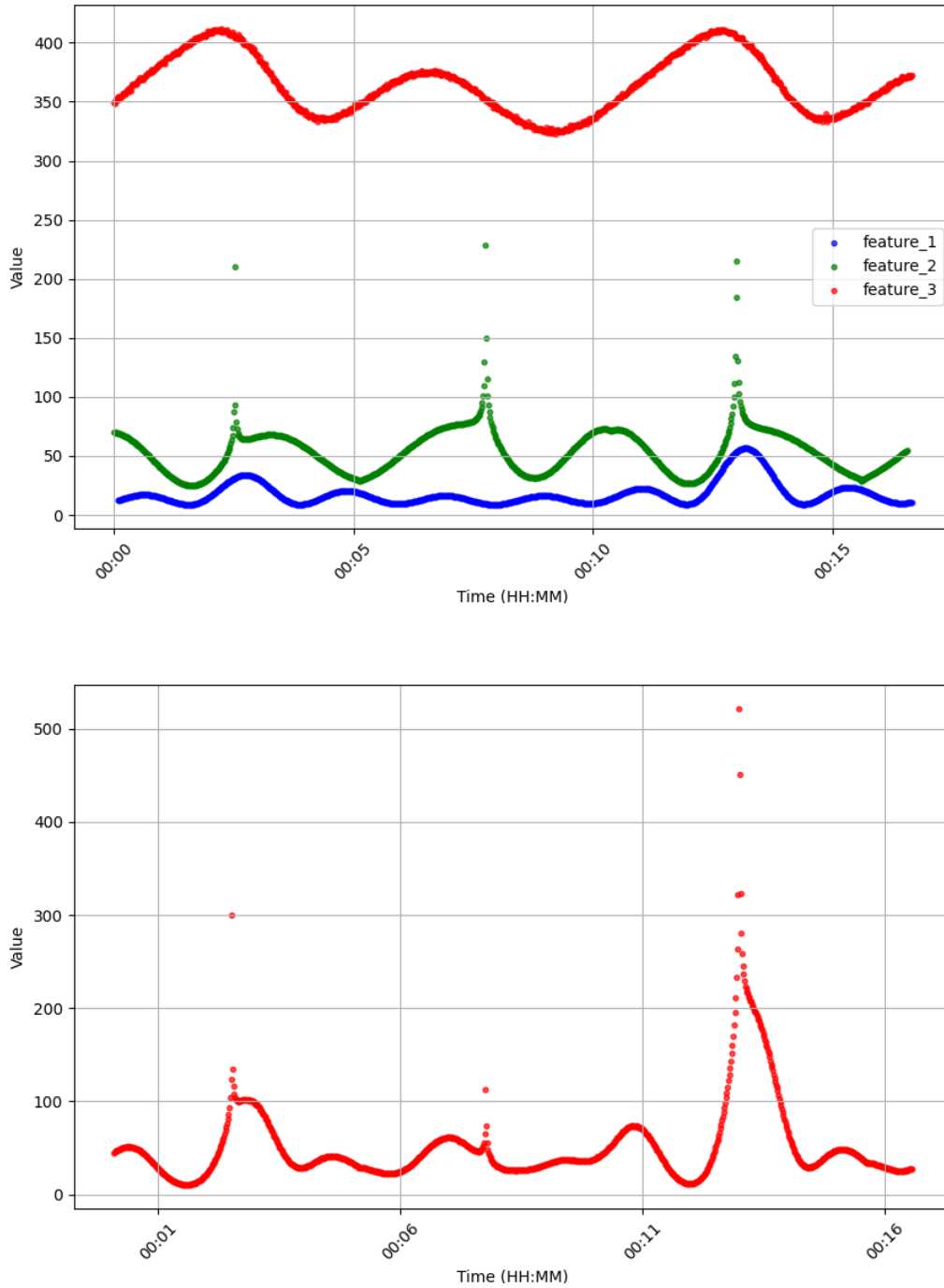


Figure 5.6: The output of the transfer function combines transformed input features to create complex synthetic time series patterns using the example values in Table 5.1.

By setting this configuration, we demonstrate the system’s capability to process complex details of input features, time shifting, and nonlinear transformation, which is appropriate for simulating

realistic data with multiple dimensions.

5.4 Conclusion

A synthetic time series generator has been designed and developed to help investigate the assessment of DL interpretability approaches. The system described in this chapter facilitates the creation of complex temporal behaviour simulation along with analysis, by enabling the generation of high-definition datasets that possess adjustable characteristics. An extensive range of data patterns, including basic trends and complex non-linear interactions, can be produced through the system with supporting examples. The testing scope for model explanations becomes possible through this method, which exposes them to diverse conditions.

In the next chapter, we will use the framework developed here to generate synthetic data. This data will be used in case studies as we develop a technique to evaluate model explanations.

Chapter 6

Evaluating interpretations

‘Without data, you’re just another person with an opinion.’ - W. Edwards Deming

6.1 Overview

In this chapter, we design a metric to evaluate explanations. This allows us to evaluate every step of the process, from synthetic data generation to training from the synthetic data to interpreting feature attribution, in a controlled environment where the ASV is known.

Section 6.2 starts the chapter with a demonstration of the process from data generation to model interpretation. The interpretation of feature attributions uses G-DeepSHAP on approximated attributions, aligning interpretability methods with theoretical expectations illustrated by comparing exact Shapley values. Section 6.3 discusses important factors to consider when evaluating interpretation performance. Lastly, Section 6.4 presents the evaluation metric that evaluates the accuracy of the feature attributions with MSE.

6.2 Demonstration: TS-MLP on simple polynomial

In this section, we work on a simple example with a straightforward polynomial function with no temporal dependencies on the test bench system. This example demonstrates the whole end-to-end process from data generation to model training and feature attribution generation. This is a starting point for more intricate relationships and time-dependent models handled in the following chapters.

6.2.1 Goal

This experiment aims to validate the test bench by performing all the steps in the process — data generation, model training, and attribution evaluation — successfully on a simple function. We evaluate our computed feature attributions against known ASV to see that the model correctly identifies the contributions of each of the input features in the feature attributions.

6.2.2 Polynomial description

We construct a dataset with the known features and target values. The features and the target variable have an intentionally intuitive relationship, making evaluation straightforward. We start by using a polynomial regression model given in Equation 6.1:

$$y = x_1 + x_2^2 + x_3^3 \tag{6.1}$$

where $\{x_1, x_2, x_3\} \in \mathbb{R}$ represent the input features, and $y \in \mathbb{R}$ is the target variable.

6.2.3 Expected Shapley values

As part of this demonstration, we use G-DeepSHAP, a variant of the SHAP method, to compute feature attributions for the synthetic dataset. (Refer to Section 3.3.4 for a detailed explanation of G-DeepSHAP). We will generally, when referring to DeepSHAP, mean the implementation of G-DeepSHAP. The synthetic data constructed in this manner provides a basis for evaluating feature attribution methods since the values of the feature terms in the mapping function should directly align with the G-DeepSHAP attribution values when plotted against the corresponding

input features. Since G-DeepSHAP follows the principles of SHAP, being an additive attribution approach, the attributions for each feature should aggregate to the difference-from-reference for each output sample, as shown in Equation 6.2.

This implies that the attributions for each feature aggregate to the difference-from-reference for each output sample due to Equation 3.9. Using the mean target value as a reference value, and simplifying the symbol for the contribution of feature i to the output y as ϕ_i , Equation 3.9 can be restated as:

$$y(x) - \phi_0 = \sum_{i=1}^N \phi_i \quad (6.2)$$

Here, N denotes the number of features in the feature space, ϕ_i represents the contribution of input feature i , and $y - \phi_0$ stands for the output target ‘difference-from-reference.’

Considering Equation 6.1 and Equation 6.2, it follows that for a specific x_1, x_2 and x_3 :

$$\phi_1 + \phi_2 + \phi_3 = x_1 + x_2^2 + x_3^3 - \phi_0 \quad (6.3)$$

$$= x_1 + x_2^2 + x_3^3 - (\mu_1 + \mu_2^2 + \mu_3^3) \quad (6.4)$$

where $\mu_i = E(x_i)$ and $\phi_0 = E(y)$. Since there is no interaction between the different input variables (they are simply summed), we then expect the G-DeepSHAP values to be given by:

$$\phi_1 \approx x_1 - \mu_1 \quad (6.5)$$

$$\phi_2 \approx x_2^2 - \mu_2^2 \quad (6.6)$$

$$\phi_3 \approx x_3^3 - \mu_3^3 \quad (6.7)$$

We apply the concept of ‘difference-from-reference’ to compute attribution values, where each feature’s contribution is determined based on deviations from the mean reference. The polynomial model serves as a fundamental test case for evaluating feature attribution techniques, as its straightforward structure allows us to compare computed attributions directly to known functional contributions. This framework also enables us to analyse how attribution methods perform under different transformations of the mapping function. In future work, temporal dependencies can be introduced to assess attribution techniques in dynamic settings, extending beyond static feature interactions.

6.2.4 Data generation

Although this dataset does not have temporal dependencies, it is treated as a time series in line with the approach used for all experiments in this study; the data is indexed as if it is a time series. We generate a data set of 10,800 samples, with 60% allocated for training, 20% for validation, and 20% for evaluation. To accomplish this, we produced uniformly distributed data points as inputs for a polynomial function, such as Equation 6.1, for 10,800 samples (one sample per second equaling 3 hours of data).

Figure 6.1 displays the resulting output variable, sorted from smallest to largest according to the y -value in Equation 6.1.

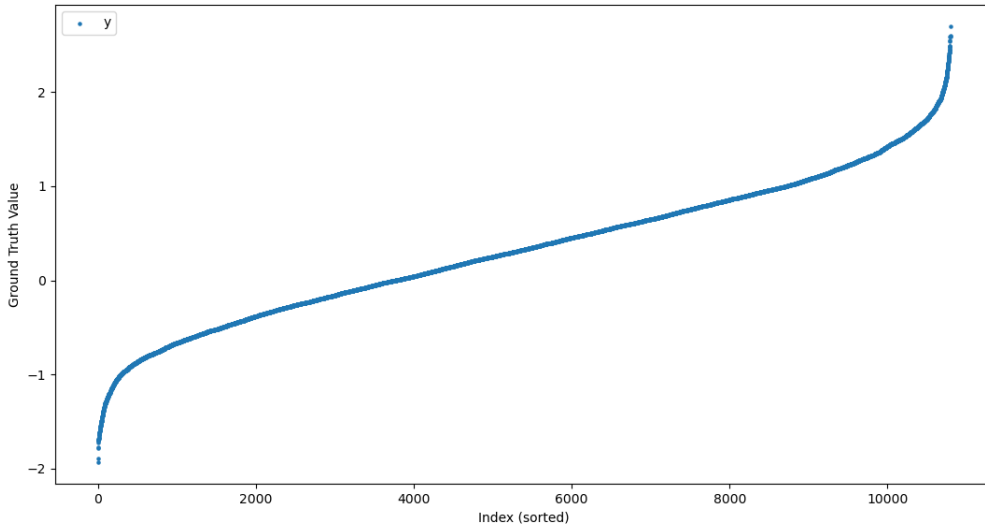


Figure 6.1: Output feature of the polynomial from the smallest to largest value of y .

6.2.5 Trained TS-MLPs

In this section, we evaluate the capacity of the TS-MLP model of the polynomial function with no temporal dependencies to capture the known underlying relationship expressed by Equation 6.1 and explored in 6.2.4. The primary objective is to train a TS-MLP using the previously generated uniformly distributed dataset based on the transfer function shown in Equation 6.1 implemented within the Knowit framework. We focus on testing the ability of the model to

learn and interpret the polynomial function under a temporal factor-free context.

The methodology of the model training is as follows:

- The uniformly distributed dataset, without any temporal dependencies, is utilized to train a TS-MLP within the Knowit framework. The dataset is split 60% for training and 20% for both testing and evaluation sets.
- A manual hyperparameter search is used to find relevant training hyperparameters. These only include learning rate, batch size, and model architecture. We systematically tested the combinations of these hyperparameters until adequate accuracy was obtained. The final hyperparameters are as follows:

Table 6.1: Hyperparameter values selected through manual tuning for optimal model performance of a TS-MLP for no temporal dependencies

Hyperparameter	Value
Batch Size	64
Learning Rate	0.001
Model Architecture	TS-MLP
Width	512
Dropout	0

- In addition, a fourth input feature is defined as a constant value of 0. This is done to the model’s ability to discriminate relevant from irrelevant input variables in a simplified setting.
- The input chunk is set to current feature values; the output value is then predicted by these chunk values using all other training hyperparameters explicitly defined as the default values from the Knowit toolkit.

The model obtains a train mean absolute error of 0.0423. In the validation performance, the mean absolute error is below 0.0417, and the evaluation mean absolute error is 0.0422. The model predicts the current target value $y(t)$ based on the previous five time steps of the three input features (x_1, x_2, x_3) , with an additional dummy feature x_4 fixed at zero. The model performs well according to the evaluation dataset, which implies that we can extract attributions to determine whether the true relationship was modelled.

6.2.6 Interpretations

In our analysis of a single sample prediction, as detailed in Section 6.2.5, we select one representative instance from the validation set. For this instance, the input values at the current time step are $x_1 = 0.92$, $x_2 = 0.61$, and $x_3 = 0.59$. In addition, we include the dummy feature $x_4 = 0$, which is fixed at zero throughout the dataset to test whether the model correctly distinguishes between relevant and irrelevant features. These values form part of the input window that the TS-MLP processes in order to predict the target at the next time step.. An interpretation matrix was generated using the G-DeepSHAP method, quantifying the contribution of each feature to the model’s prediction. For this specific instance, the attributions are computed based on the selected reference values, with 100 subset samples used to establish the baseline. The explanation focuses on a single local interpretation, meaning only the attribution values for this particular sample evaluation set are considered.

The resulting matrix, visualized as a heatmap in the KnowIt toolkit, is shown in Figure 6.2. For this heatmap, the warmer colors represent higher contributions to the feature value, so there is a higher magnitude of feature attributions.

The feature that affected the output the most is x_1 . While x_3 would be expected to have a more impactful attribution due to its cubic nature, this is not visible here as we are only looking at a local interpretation of a random sample (and $0.61^2 > 0.59^3$). The global feature effect may be different, as we will return to later. Feature x_4 has no effect on the prediction. It is important to note that x_4 is simply a zero input feature, included to assess the model’s ability to differentiate between relevant and irrelevant input variables; as can be seen, it indeed assigns the x_4 feature an attribution score of zero seen by the value estimation on the right of Figure 6.2.

The trained TS-MLP model’s interpretation of the underlying polynomial relationship is examined, focusing on feature importance. Figure 6.3 depicts the values of the x_1 feature, which corresponds to the linear term in the polynomial equation: $y = x_1 + x_2^2 + x_3^3$. Here, the values are sorted from the smallest input value x_1 to the largest. The red data points represent Exact Explainer Shapley values (ASVs) as discussed in Section 6.2.3. These ASVs are slightly lower than the corresponding feature attributions estimated by G-DeepSHAP indicated as blue points. These attributions are derived from the deep learning model trained on the polynomial function. The green points, on the other hand, indicate the relevant component of the polynomial function

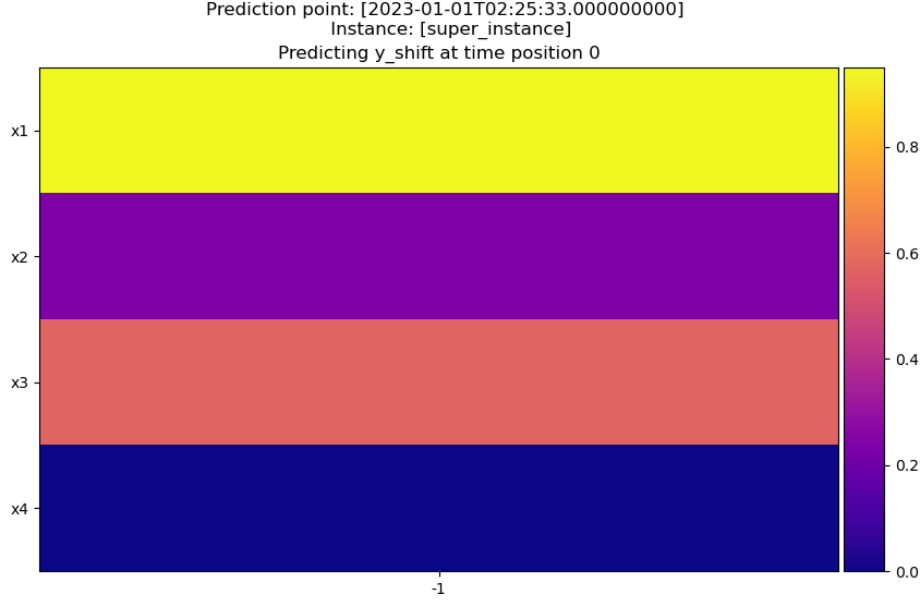


Figure 6.2: Interpretation heatmap of a random representative single sample, showing us how these values might look as a matrix, important to understand for the sections to follow.

directly.

This difference arises between ASVs and DSVs because the baseline values are subtracted from the Exact Shapley values, as shown in Equation 6.5. In this experiment, we compare three sets of attributions: The ASVs are calculated directly from the polynomial equation, Exact Shapley values are derived by calculating through the Shapley value formula, giving us the expected attribution values described in Section 6.2.3, and finally, approximations calculated using G-DeepSHAP are calculated. Through these comparisons, we gain a framework to determine the extent to which these interpretations are aligned with the theoretically correct Shapley values and, therefore, gain insight into the interpretability of the model.

Similar analyses are conducted for the higher-order polynomial terms, x_2^2 and x_3^3 ; the model's ability to accurately capture the known relationships between the input features and the target variable is further evaluated. Figures 6.4 and 6.5 illustrate these analyses.

Figure 6.4 further illustrates the subtraction of the baseline for the individual feature term, as described in Equation 6.5. Because of the nature of its input feature, the observed difference is more pronounced. It can be seen that G-DeepSHAP is very close to the ASVs, but the

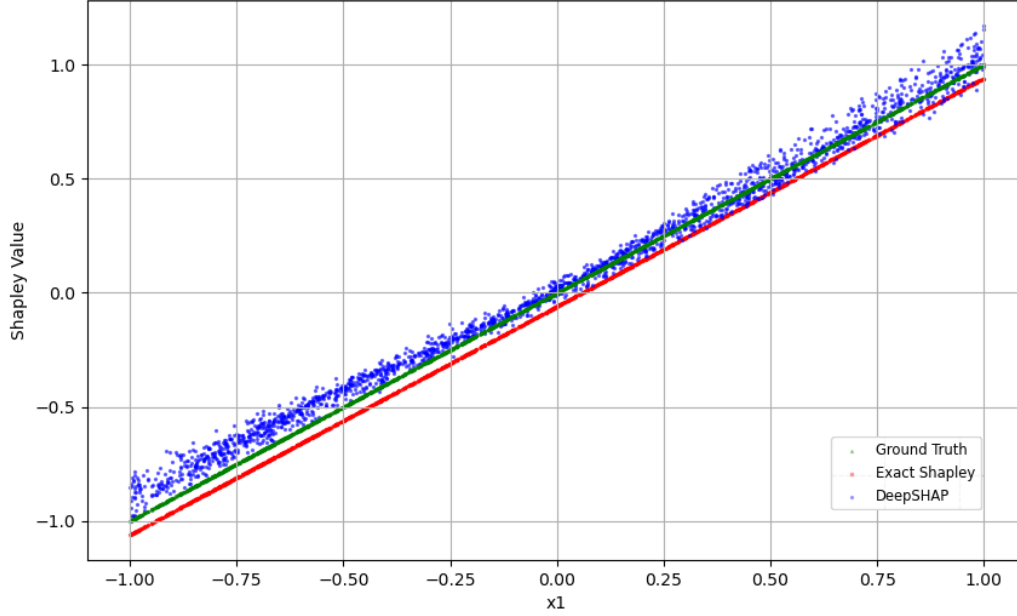


Figure 6.3: A comparison of the G-DeepSHAP Values (DSVs)('DeepSHAP') and ASVs ('Exact Shapley') for x_1 . The 'Ground Truth' corresponds to the polynomial component

$$f(x_1) = x_1.$$

exact Shapley value is noticeably lower than the ground truth value, and the G-DeepSHAP value behaves in a similar, but much less clear, pattern due to approximation and accuracy loss of the trained model. Since the squared feature term converts all negative values to positive, the following definite integral calculation approximates the baseline value using an average calculation:

$$x_b \approx \int_{-1}^1 \left(\frac{x^2}{2} \right) dx = \frac{1}{3}$$

where x_b represents an approximate baseline value. As noted, this equation provides an estimated baseline value, which is not necessarily equal to the true baseline due to the distribution of the values and the inherent manner in which subsets are averaged by the G-DeepSHAP process described in Section 3.3.5. (This will be revisited in Section 6.3.2).

The results shown in Figure 6.5 further validate that the model has successfully learned to approximate the underlying polynomial relationship. In this case, the baseline value, also observed in Figure 6.3 for the x_1 feature, is not as distinct from the feature attributions due to the offsetting effects of positive and negative uniformly distributed values within the polynomial-specific

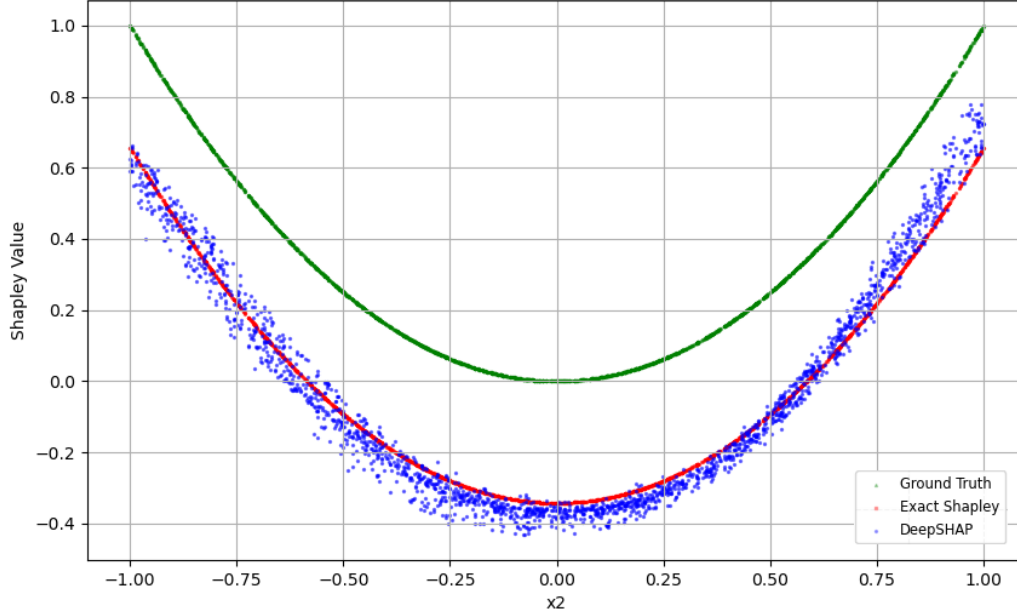


Figure 6.4: A comparison of the DSV(‘DeepSHAP’) and ASVs (‘Exact Shapley’) for x_2 . The ‘Ground Truth’ corresponds to the polynomial component $f(x_2) = x_2^2$.

transfer function operation.

Being able to generate a ASVs explanation analytically and then measure it against the explanation generated by an interpretability technique allows us to evaluate different evaluation metrics quantitatively. We return to the process of constructing such an evaluation metric in Section 6.4. First, we note important considerations when trying to match a ground truth to a generated explanation.

6.3 Important considerations

6.3.1 Scaling

An essential step in machine learning pipelines is to scale or standardize feature values to allow a fair comparison between features and increase the model’s performance and stability [52]. However, it cannot be ignored that feature scaling can considerably change the exact mathematical equation that portrays the relationship between the input features and the output variable. In

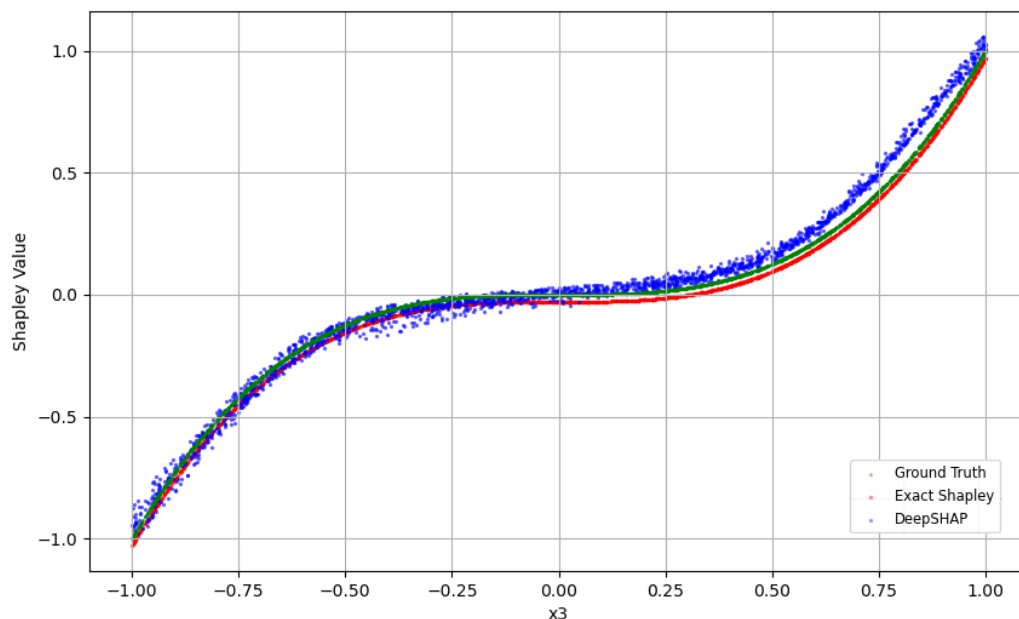


Figure 6.5: A comparison of the DSV(‘DeepSHAP’) and ASVs (‘Exact Shapley’) for x_3 . The ‘Ground Truth’ corresponds to the polynomial component $f(x_3) = x_3^3$.

Section 6.2.6, the amplitude of the features was chosen to lie between 0 and 1 so that explicit scaling was not needed. We now see features that do have these complexities.

Z-score normalisation

For example, take equation $y = ax_1 + bx_2^2 + cx_3^3$. Z-score normalization is the process of removing the mean and dividing it by the standard deviation for each feature. This does feature normalization, which means transforming features to have a mean of zero and a variance of one, and it eliminates inconsistency in scaling provided by feature values at more or less the same scale. Standardizing is an advantage when training machine learning models, which comes at the cost of modifying the coefficients and exponents of the mathematical equation. Because of this, the equation generally strays away from the inputs of each feature, reaching the outputs of the dependent variable.

In the original equation discussed in Section 5.3, Equation 6.1, the exponents indicate the relative importance or impact of each feature x_1 , x_2 , x_3 on the output variable y . However, if the model

training precedes the standardization of features, the equation will vary because of rescaling. The standardized equation would take the form $y = x'_1 + x'^2_2 + x'^3_3 + K$, where x'_1 , x'_2 , and x'_3 represent the standardized versions of the features. Additionally, the coefficients a , b , and c are multiplied by the standard deviations of x_1 , x_2 , and x_3 , respectively, and a constant term K is introduced. These mathematical transformations play an important role in how well the model actually represents this connection between the input features and the output variable of the problem.

$$x'_1 = \frac{x_1 - \mu_1}{\sigma_1} \quad (6.8)$$

$$x'_2 = \frac{x_2 - \mu_2}{\sigma_2} \quad (6.9)$$

$$x'_3 = \frac{x_3 - \mu_3}{\sigma_3} \quad (6.10)$$

Here μ_i and σ_i are the mean and standard deviation of x_i , respectively. Substituting these values yields,

$$\begin{aligned} y &= a(\sigma_1 x'_1 + \mu_1) + b(\sigma_2 x'_2 + \mu_2)^2 + c(\sigma_3 x'_3 + \mu_3)^3 \\ &= a(\sigma_1 x'_1 + \mu_1) + b(\sigma_2^2 x'^2_2 + 2\sigma_2 \mu_2 x'_2 + \mu_2^2) + c(\sigma_3^3 x'^3_3 + 3\sigma_3^2 \mu_3 x'^2_3 + 3\sigma_3 \mu_3^2 x'_3 + \mu_3^3) \\ &= a\sigma_1 x'_1 + a\mu_1 + b\sigma_2^2 x'^2_2 + 2b\sigma_2 \mu_2 x'_2 + b\mu_2^2 + c\sigma_3^3 x'^3_3 + 3c\sigma_3^2 \mu_3 x'^2_3 + 3c\sigma_3 \mu_3^2 x'_3 + c\mu_3^3 \\ &= \alpha_s x'_1 + \beta_s x'^2_2 + \gamma_s x'_2 + \delta_s x'^3_3 + \epsilon_s x'^2_3 + \zeta_s 2x'_3 + \eta_s \end{aligned} \quad (6.11)$$

As we can change the exponents or coefficients for the standardization process, the relationship between the input features and the output variable thus changes. Thus, the resulting equation is derived from the model against the standardized features, except the original equation cannot fully match the feature importance and its impact on the prediction. In addition to this, if the target variable is itself also standardized, then,

$$y'_s = \frac{y - \mu_y}{\sigma_y} \quad (6.12)$$

$$\begin{aligned} &= \frac{1}{\sigma_y} (a\sigma_1 x'_1 + a\mu_1 + b\sigma_2^2 x'^2_2 + 2b\sigma_2 \mu_2 x'_2 + b\mu_2^2 + c\sigma_3^3 x'^3_3 + 3c\sigma_3^2 \mu_3 x'^2_3 + 3c\sigma_3 \mu_3^2 x'_3 + c\mu_3^3 - \mu_y) \\ &= \frac{1}{\sigma_y} (\alpha_s x'_1 + \beta_s x'^2_2 + \gamma_s x'_2 + \delta_s x'^3_3 + \epsilon_s x'^2_3 + \zeta_s 2x'_3 + \eta_s - \mu_y) \\ &= \frac{1}{\sigma_y} (\alpha_s x'_1 + \beta_s x'^2_2 + \gamma_s x'_2 + \delta_s x'^3_3 + \epsilon_s x'^2_3 + \zeta_s 2x'_3 + \eta'_s) \end{aligned} \quad (6.13)$$

The model results should be interpreted and presented in a way that reflects the effects of standardization. The consequence of standardization needs to be understood in order to fully comprehend the meaning of what you chose to standardize. With the correctly applied equation

and taking into consideration the effects of standardization, it is possible to tell what the input variables do to the output variable.

Min-max normalisation

The other method is min-max scaling, which scales the value of all features to the same range, typically between 0 and 1.

For an illustration of min-max normalization, let us assume $y = ax_1 + bx_2^2 + cx_3^3$. The latter is scaled by subtracting the minimum value and dividing by the range, that is, the maximum value for every feature. This method addresses the problem of scaling and compacting all feature values of the same range.

When features are scaled using min-max normalization, their distributions and the coefficients in mathematical equations change. These changes influence how features contribute to the model's predictions, which can alter the computed attribution values.

In the original equation discussed in Section 5.3, Equation 6.1, the exponents indicate the relative importance or impact of each feature x_1, x_2, x_3 on the output variable y . Nevertheless, in the event of normalization before model training, the same equation will experience variations owing to the normalization procedure. The normalized model would be represented by the equation $y = x'_1 + x'^2_2 + x'^3_3 + K$, where x'_1, x'_2 , and x'_3 represent the normalized versions of the features. Additionally, the coefficients a, b , and c (eqn. 6.17) are scaled by the range of x_1, x_2 , and x_3 , respectively, and a constant term K is introduced. However, although this data normalization process makes the above mathematical transformations on the input features, it may come to distort the model's capacity to learn the true relationship between input features and output variables. This is because,

$$x'_1 = \frac{x_1 - \min(x_1)}{\max(x_1) - \min(x_1)} = \frac{x_1 - \omega_1}{\varrho_1} \quad (6.14)$$

$$x'_2 = \frac{x_2 - \min(x_2)}{\max(x_2) - \min(x_2)} = \frac{x_2 - \omega_2}{\varrho_2} \quad (6.15)$$

$$x'_3 = \frac{x_3 - \min(x_3)}{\max(x_3) - \min(x_3)} = \frac{x_3 - \omega_3}{\varrho_3} \quad (6.16)$$

Note that we have replaced the minimum and maximum operations of the set of samples above

with symbols. Substituting these values yields,

$$y = a(\varrho_1 x'_1 + \omega_1) + b(\varrho_2 x'_2 + \omega_2)^2 + c(\varrho_3 x'_3 + \omega_3)^3 \quad (6.17)$$

Using the same simplification variables as in Equation 6.11, we obtain,

$$y = \alpha_n x'_1 + \beta_n x'^2_2 + \gamma_n x'_2 + \delta_n x'^3_3 + \epsilon_n x'^2_3 + \zeta_n 2x'_3 + \eta_n \quad (6.18)$$

The min-max normalization procedure of the coefficients allows the connection between the input and output variables to be changed. Thus, the model behavior derived is the equation obtained based on normalized attributes and may, in terms of feature relevance and the same output, differ from the original equation. Also, assuming the target variable is also normalized,

$$y'_n = \frac{y - \min(y)}{\max(y) - \min(y)} = \frac{y - \omega_y}{\varrho_y} \quad (6.19)$$

Following the same process as in Equation 6.20, we obtain,

$$y'_n = \frac{1}{\sigma_y} (\alpha_n x'_1 + \beta_n x'^2_2 + \gamma_n x'_2 + \delta_n x'^3_3 + \epsilon_n x'^2_3 + \zeta_n 2x'_3 + \eta'_n) \quad (6.20)$$

In order to understand the effect of min max normalization on mathematical representation, it is important to find out the underlying values and final results of the equation and model. If the right equation is applied and we understand the meaning of min-max normalization, researchers will be able to learn something good about the input features that are related to the target variable.

Scaling demonstration

In this section, we generate data with the results to train a model. The study aims to assess the extent to which the feature attributions from the trained model reflect the known dependencies between the feature inputs and the target variable while working with both scaled and unscaled input data representations.

We will compare results from models trained on min-max normalized data versus those trained on unscaled data to see how data scaling affects model performance and interpretability, as shown in Equation 6.1.

Scaled versus unscaled data generation and TS-MLP training

MLP's have become one of the most effective methods in time series prediction. This work describes a practical method for selecting optimal hyperparameters for an MLP model to make reliable predictions on a large dataset.

Thus, input features are chosen to be uniformly distributed across the input sequence for the sake of model training by presenting the model with an input sequence as diverse as possible. These input points have no temporal effects, and a random state has been set for the replicability of the experiment. The specific features used are described as follows:

$$x_1 \sim \mathcal{U}(-0.1, 0.1)$$

$$x_2 \sim \mathcal{U}(-1, 1)$$

$$x_3 \sim \mathcal{U}(-30, 10)$$

where $\mathcal{U}(a, b)$ represents a uniform distribution of samples between the two limits a and b . Here, again, we select Equation 6.1 as the preferred transfer function. Note that for this choice of x_i , μ_1 and μ_3 are approximately 0 but $\mu_2 \approx 0.33$.

A dataset of 10,800 samples is partitioned into 60% training, 20% validation, 20% evaluation, and an accurate TS-MLP model is trained. We train the model on 300 epochs with the Adam optimiser and the Mean Squared Error loss function. For this dataset, training used a 666 random state, a width of 4092, a dropout rate of 0, an input chunk size of 1, a prediction horizon of 1, and a patience of 100 epochs before training is stopped.

The impact of these primary differences is assessed by comparing model performance before and after scaling. We translate the tuned hyperparameters back to scalars and use unscaled data to identify the best model. We then scale the dataset with the same optimal model parameters and retrain the model to highlight the difference in performance. To accommodate negative numbers in our interpretation, we apply the min-max norm, the details of which are presented in Section 6.3.1, to normalize the data from -1 to 1 and shift and scale the data. Thus, the normalization equation becomes,

$$x' = 2 \cdot \frac{x - \min(x)}{\max(x) - \min(x)} - 1 = 2 \cdot \frac{x - \omega_1}{\varrho_1} - 1 \quad (6.21)$$

This ensures the input data's min-max normalization is scaled between -1 and 1 .

To optimize the model’s performance, we conducted a thorough grid search over two key hyperparameters: We tested the learning rate and a batch size ranging from 8 to 64. We vary the batch size between 8 and 64 and vary the learning rate from 0.1 to 0.00001. The results of the five best grid search models for the batch size and learning rate hyperparameters are presented in Table 6.2 below:

Table 6.2: Model training configurations and corresponding MSE values for a MLP with unscaled data

Model Trained	Batch Size	Learning Rate	Epoch	MSE
1	512	0.001	270	23.3
2	256	0.001	290	28.77
3	1024	0.001	295	32.03
4	128	0.001	84	40.77
5	64	0.01	82	79.81

Employing the parameters of Model 1 from Table 6.2 for the input-scaled data only, not scaling the output, leads to a mean squared error of 0.083. This level of performance is adequate to derive interpretations from both models for comparative analysis, with the first noticeable difference demonstrated through the model MSE ranking. Thus, we have an unscaled data model to compare to a scaled data model with the same hyperparameters.

Scaled versus unscaled MLP interpretation

In this section, we present the findings of our experiments with min-max normalization-scaled and unscaled data, underscoring the importance of data scaling in enhancing a model’s capability to accurately discern the relationships between features and the target variable. The values of each of the separate features are sorted in ascending order. For the G-DeepSHAP, a reference sample of 100 is chosen; this concept is discussed in detail in Section 6.3.2, and the evaluation set is used to plot interpretations. Our results, shown in Figure 6.6, demonstrate that using scaled data—motivated by our discussion at the start of Section 6.3.1 and the analytical formulation in Equation 6.18—yields improved model performance, as seen in the left column of Figure 6.6. These findings show that the model can capture the correct correlations between the features and the target variable, and the obtained feature attributions are in line with the actual connections. Data scaling affects the natural structure of the generated data in the right column and changes it to a new form because of the scaling operation. Conversely, with unscaled data (right column

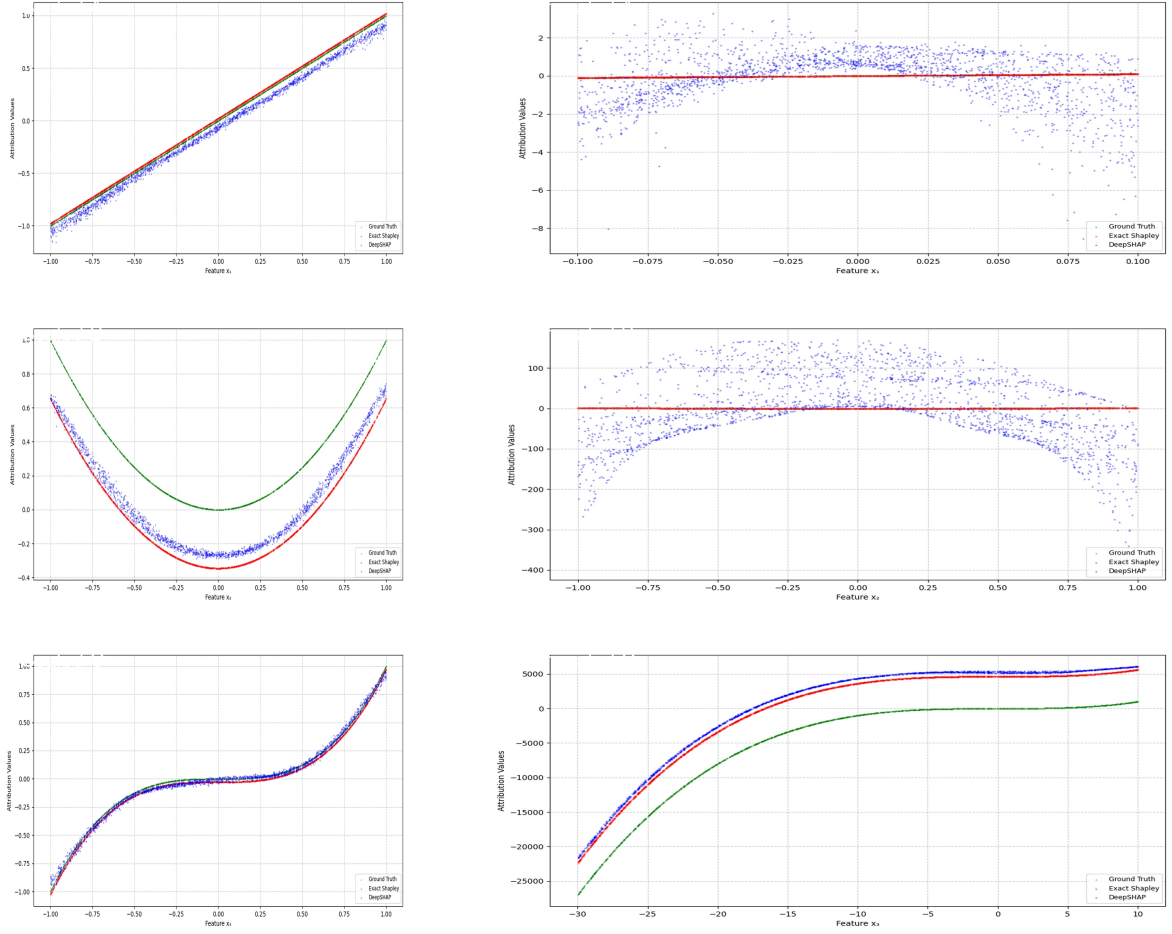


Figure 6.6: Comparison of the TS-MLP interpretations when using scaled (left column) versus unscaled data (right column). The blue points depict the G-DeepSHAP approximations, the red points represent the Exact Shapley estimations, and the green line indicates the ground truth equation (covered by the red dots). The first row shows the attribution values for x_1 , the second row for x_2 , and the third for x_3 features. Note the green line is hidden behind the red line in the upper right figures

of Figure 6.6), the model accurately identifies the relationship only for the most prominent feature, x_3 , while struggling to precisely interpret the relationships involving the less significant features, x_1 and x_2 . Notice the axis illustrating the difference in scaled and unscaled data.

This study shows how data scaling affects model performance and interpretability. To this end, by comparing models trained on scaled vs. unscaled data, we show that scaling dramatically increases the model’s capability to accurately interpret feature relationships in data. Finally, we find that the model trained with scaled data learned and assigned appropriate importance

to all relevant features, while the model trained on unscaled data was more difficult, especially for less important features. The results show how preprocessing techniques like data scaling can help in developing robust as well as interpretable machine learning models.

6.3.2 Reference values

In this section, we examine how different baselines (mean, zero, and G-DeepSHAP) affect the interpretation of feature contributions in the SHAP-based attribution analysis. We constructed a 3-feature polynomial function for this experiment: $y = x_1 + x_2^2 + x_3^3$. We randomly sampled 10,000 values for each feature, and each feature value is randomly sampled from the uniform distribution over $[-1,1]$. These samples cover a wide range of possible values for x_1 , x_2 , and x_3 , drawn uniformly from $[-1,1]$.

Experimental setup

We further derive Shapley values for a subset of 100 samples from the generated data distribution. We discovered that we could perform SHAP value calculations on a subset and clarify and visualize the feature attributions in a computationally efficient fashion while capturing the representative patterns in the feature contributions. Shapley values are computed for each feature under three baseline conditions: First, the mean baseline (using the average value of each feature across the dataset), zero baseline (setting all features to 0), and a G-DeepSHAP-like baseline sampling 100 representative data samples from the generated dataset to approximate realistic values so the baseline can capture typical feature interactions.

Visualisation and analysis

Figure 6.7 illustrates baseline comparisons of Shapley values: input values for each feature are plotted against Shapley values for each feature under each baseline condition. The color-coded legend differentiates between baseline methods: Shapley values are represented in blue for the mean baseline, red for the zero baseline, and green for the G-DeepSHAP-like baseline, with each color indicating how the baseline choice affects the distribution and range of feature attributions. We visualize how each baseline changes the distribution and range of the Shapley values and

demonstrate the sensitivity of feature attributions to the chosen baseline.

Background on G-DeepSHAP’s popularity

We include G-DeepSHAP as its relevance in the study only becomes apparent in the study here. G-DeepSHAP is developed to work on more complex data structures, like images, time series, and text embeddings. G-DeepSHAP is designed to improve SHAP’s explanations on networks with structured data using the theoretical foundation of Shapley values, with the reference values computed based on actual samples. In contrast to the mean baseline used in Lundberg’s original SHAP formulation, where features are assumed to be independent, this approach relies on a self-consistent baseline. For linear models or features with very little interdependency, the mean reference performs well. However, they may wreak havoc with the Shapley values of complex models and grossly simplify interactions among a myriad of different features. Refer to Section 3.3.5 for a more detailed discussion on G-DeepSHAP.

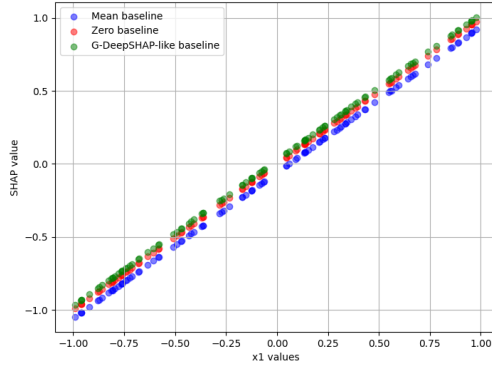
Using standard baselines (e.g., the mean of the feature or 0) simplifies the analysis of the attributions. The drawback of this approach is the introduction of a potential bias that is dependent on the set of sample values chosen to construct the baseline.

In general, G-DeepSHAP is a data-driven method of aligning Shapley values with real data settings, which increases the interpretability and trustworthiness of feature attributions.

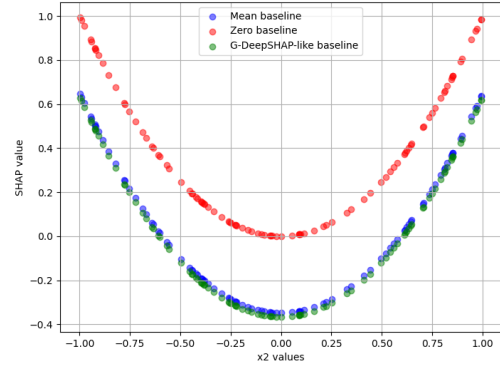
6.3.3 Local and global attribution values

One of the central ideas of machine learning interpretability is the differentiation between local and global attribution methods. It becomes relevant in experiments here as we shall prove. These approaches provide insights into how individual features affect model predictions over regions of analysis varying from individual data points to multiple data points [53], [54]. We will briefly illustrate the key differences in this topic.

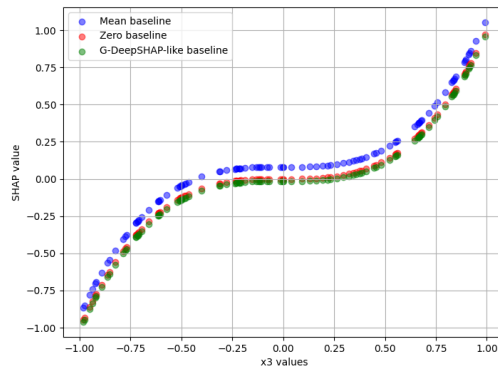
Explanation techniques that explain a model’s predictions for a particular instance in a dataset are called explanation techniques at local attribution. Feature contributions are computed for each instance, and local explanations are provided for each prediction. For example, a local



(a) x_1 feature attributions



(b) x_2 feature attributions



(c) x_3 feature attributions

Figure 6.7: Comparative scatter plots of model interpretations across baseline types. The first column represents SHAP values calculated with the mean baseline, the second column with a zero baseline, and the third column with the G-DeepSHAP-like baseline. Blue, red, and green dots denote the SHAP values for each baseline condition.

Shapley value would be the estimated contribution of each feature in the examples, substituting a model prediction of a particular instance with respect to a reference output (e.g., a model average prediction or a baseline value). The local attributions allow us to reason whether or not each feature affects the prediction at a fine enough and acceptable level for our needs (e.g., visibility decisions such as loan approvals, medical diagnoses, etc.) and in cases where we do not already have an interpretable algorithm for the prediction.

Global attribution techniques summarize feature influence for the whole dataset, thus giving a big picture of the importance of features. Contrary to localized predictions, which express

each prediction regarding feature contributions, global explanations aggregate those feature contributions across instances to pinpoint which features consistently affect model behaviour. This kind of attribution shows users patterns of feature importance, allowing them to learn the general dependencies of the model. For example, we show that, in many cases, the absolute global Shapley values can be approximated using the average of absolute local Shapley values over many samples to show which features impact the overall result. Such insight helps validate model behaviour, debugging, and convincing ourselves that the model’s behaviour is aligned with domain expectations.

Both perspectives are essential for understanding model behaviour: The former local explanations enable users to explain individual predictions, and the latter global explanations give a complete view of how a given feature influences a large variety of situations. Together, these two provide a full framework of interpretability for further machine learning models using more transparent and robust methods.

6.3.4 Sample size

This section investigates the influence of varying sample sizes on the accuracy and reliability of G-DeepSHAP attribution interpretations. We explored baseline sample sizes ranging from 2 samples to 1,000 samples in total, and clarified how the sample size affects the stability and trustworthiness of the attributions. The findings clarify the tradeoffs between computational efficiency and the precision of the model interpretations. Generally, the larger the sample set for the baseline computation, the longer the computational time.

The choice of baseline samples is rather important. In Figure 6.8, we show how different numbers of baseline samples affect the comparison of model interpretations. This figure comprises four-by-three images where each row corresponds to a different number of baseline samples (2, 10, 100, and 1,000). The first column represents x_1 , second column represents x_2 and third column represents x_3 . The color-coded side legend provides a clear interpretation of the lines: blue color stands for G-DeepSHAP values of the best TS-MLP model described in 6.2.5, green is used for the ground truth of our analytical Equation 6.1, and red color for exact SHAP values of the same. This visualization focuses on the effects of the number of baseline samples on the attribution values and the comparison between the methods.

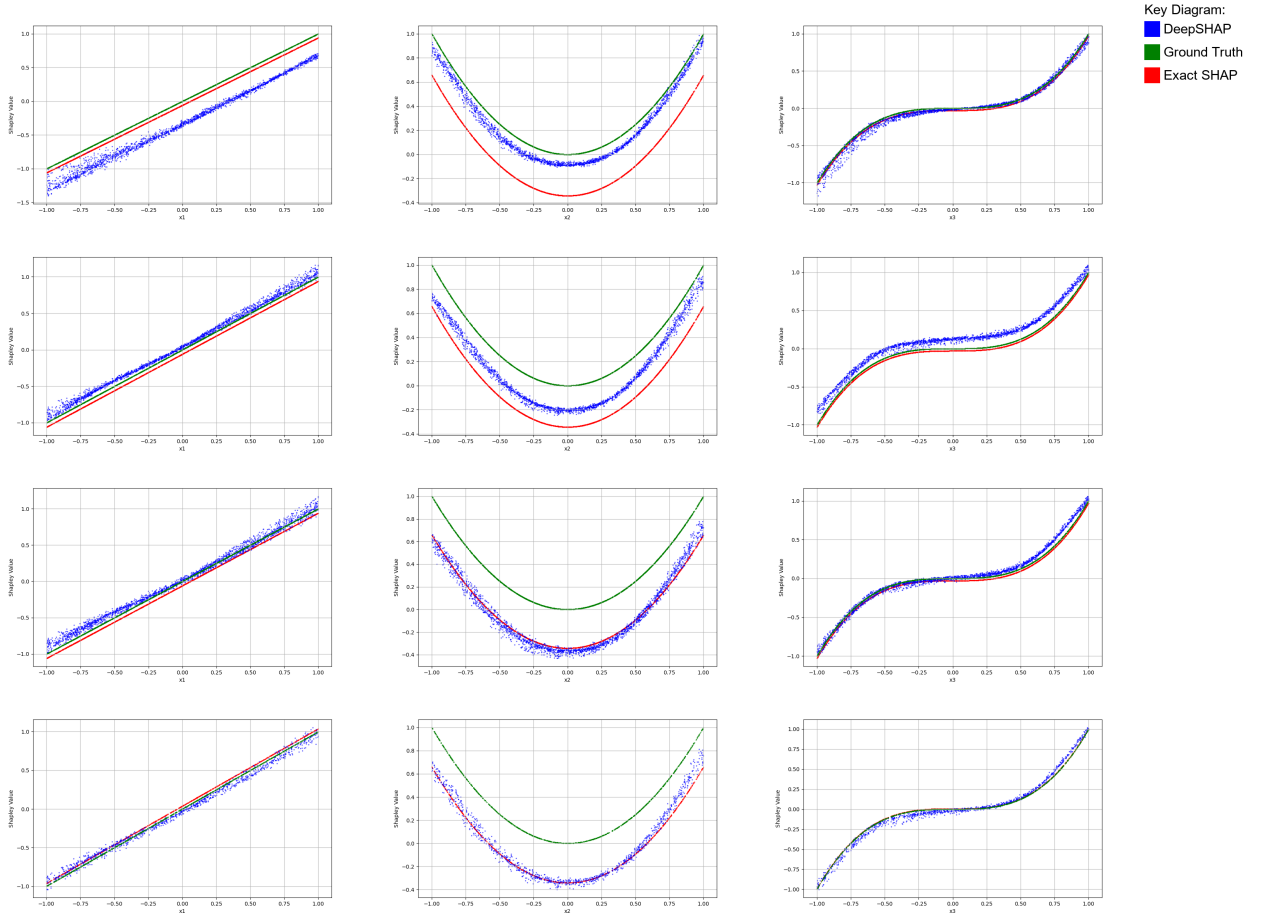


Figure 6.8: A 4x3 grid of images displaying model interpretations. The first row represents results with two baseline samples, the second with 10 baseline samples, the third with 100 baseline samples, and the fourth with 1,000 baseline samples. The first column corresponds to the x_1 , the second column to x_2 , and the third column to the x_3 feature. The side legend indicates colour codes: blue for G-DeepSHAP, green for ground truth, and red for Exact SHAP.

The results presented in Figure 6.8 show that when the number of baseline samples is small (e.g., 2), the G-DeepSHAP values differ from the Exact SHAP values, meaning there is a large discrepancy with the ASVs. Nevertheless, the approximation is better if the number of baseline samples is large. For 100 baseline samples, the G-DeepSHAP values are relatively closer to the Exact SHAP values. The same pattern is observed with 1,000 baseline samples; the G-DeepSHAP values are again very close to the Exact SHAP values. This suggests that a reasonable number of baseline samples should be chosen for a more accurate attribution analysis.

In particular, it is shown that when the number of baseline samples is restricted (for example, to

2), the G-DeepSHAP values deviate from the Exact SHAP values, which means that the deviation from the ASVs is substantial. However, as the size of the baseline set increases, the approximation becomes better. G-DeepSHAP values are close to the Exact SHAP when 100 baseline samples are compared. This continues with 1,000 baseline samples and the G-DeepSHAP values. This study aims to use the exact SHAP values. This result supports the need to choose an appropriate number of baseline samples to the extent that attribution analysis will be more accurate.

6.4 Evaluation metric

We introduce a measurement that we developed in order to measure the accuracy of a set of explanations, and consequently, the technique to extract the explanations. In this study, we compare analytical Shapley values (ASVs) derived from a known function with feature attributions generated by G-DeepSHAP for deep learning models applied to time series data. To systematically quantify the difference between these two attributions, we compute the Mean Square Error (MSE) of the attribution matrices generated by both techniques. MSE is chosen because it provides a precise, quantitative measure of how closely the computed feature attributions align with ASVs. Unlike ranking-based or qualitative metrics, MSE captures the magnitude of attribution errors, making it particularly useful for evaluating interpretability methods in a controlled and comparative manner.

6.4.1 Methodology

The methodology for obtaining this metric involves the following steps:

1. Extract the DSV from the DNN described in Section 3.3.5 along with the sample input values.
2. Using the same input sample values, compute the Exact SHAP Attribution (ASVs) values described in Section 3.3.1 for the transfer function, in this case, $y = x_1 + x_2^2 + x_3^3$.
3. Convert ASVs values into an ASVs matrix, structured analogously to the DSVs matrix. In this experiment, where we consider only one-time step and three features, the ASVs

matrix has dimensions 3×1 , each row representing a distinct feature, and the single column corresponding to the ASVs value for that feature at the given time step. In this step, ASVs values are assigned to their corresponding time features, while time features with no importance are set to zero. This ensures that the resulting matrix reflects time-dependent attributions.

4. Calculate the Mean Square Error between the matrices, sample-by-sample, similar to the sample heatmap matrix visualised in Figure 6.2.
5. Find a single metric value by averaging the MSE values.

Thus, we create two matrices for comparison:

1. **Shapley matrix:** This matrix represents the contributions of features at different time steps for a specific synthetic equation, with the features being 'switched off' by setting their values to zero. It is structured as an $M \times N$ matrix, where $Y_{i,j}^{(s)}$ contains the Shapley value indicating the contribution of feature j at time step i to the model's prediction. The G-DeepSHAP matrices of Know-it are used as references, and this matrix is constructed to align with the same time steps and features.
2. **Time-dependent G-DeepSHAP matrix:** This matrix mirrors the Shapley matrix by indicating feature contributions over time, with noncontributing features having estimates close to zero. It is structured as an $M \times N$ matrix, where $\widehat{Y}_{i,j}^{(s)}$ contains the G-DeepSHAP value representing the contribution of feature j at time step i to the model's prediction. Unlike the Shapley matrix, no additional construction is performed here; the G-DeepSHAP values are interpreted directly.

We can quantify explanations by comparing this single numeric output (this mean MSE) for the two attribution matrices, with ASV being the ground truth attribution and G-DeepSHAP being the approximation to be measured.

The MSE is calculated using the following formula for a single sample s :

$$\mathcal{L}_{MSE}^{(s)} = \frac{1}{|A|} \sum_{i=1}^F \sum_{j=1}^T (Y_{i,j}^{(s)} - \widehat{Y}_{i,j}^{(s)})^2 \quad (6.22)$$

Where:

-
- $Y_{i,j}^{(s)}$ is the ASV for feature m and timestep n in sample s .
 - $\widehat{Y_{i,j}^{(s)}}$ is the estimated attribution value for the same feature m and timestep n , generated by the model.
 - F is the number of features.
 - T is the number of timesteps.
 - $|A| = F \times T$ is the total number of elements in each ASVs and DSVs matrix.

The overall mean MSE across all samples is computed as follows.

$$\mathcal{L}_{MSE}^{\text{mean}} = \frac{1}{N} \sum_{s=1}^N \mathcal{L}_{MSE}^{(s)} \quad (6.23)$$

Where:

- N is the number of samples.
- $\mathcal{L}_{MSE}^{(s)}$ is the per-sample MSE computed in Equation 6.22

6.4.2 Results

Applying this methodology in Section 6.4.1 to our G-DeepSHAP reference value, as detailed in Section 6.3.2, we obtain numerical values for our matrices, described in Table 6.3, for the best model discussed in Section 6.2.5.

Table 6.3: $\mathcal{L}_{MSE}^{\text{mean}}$ for different sample sizes of a MLP model trained on simple polynomial

Number of Samples	$\mathcal{L}_{MSE}^{\text{mean}}$
2	0.0466105
10	0.0213995
100	0.0065428
1,000	0.0031386

Figure 6.9 depicts the relationship between the mean MSE and the number of baseline samples chosen in the G-DeepSHAP calculations. The x-axis, on a logarithmic scale, represents the varying number of baseline samples, ranging from 2 to 1,000, similar to Section 6.3.4, while the y-axis

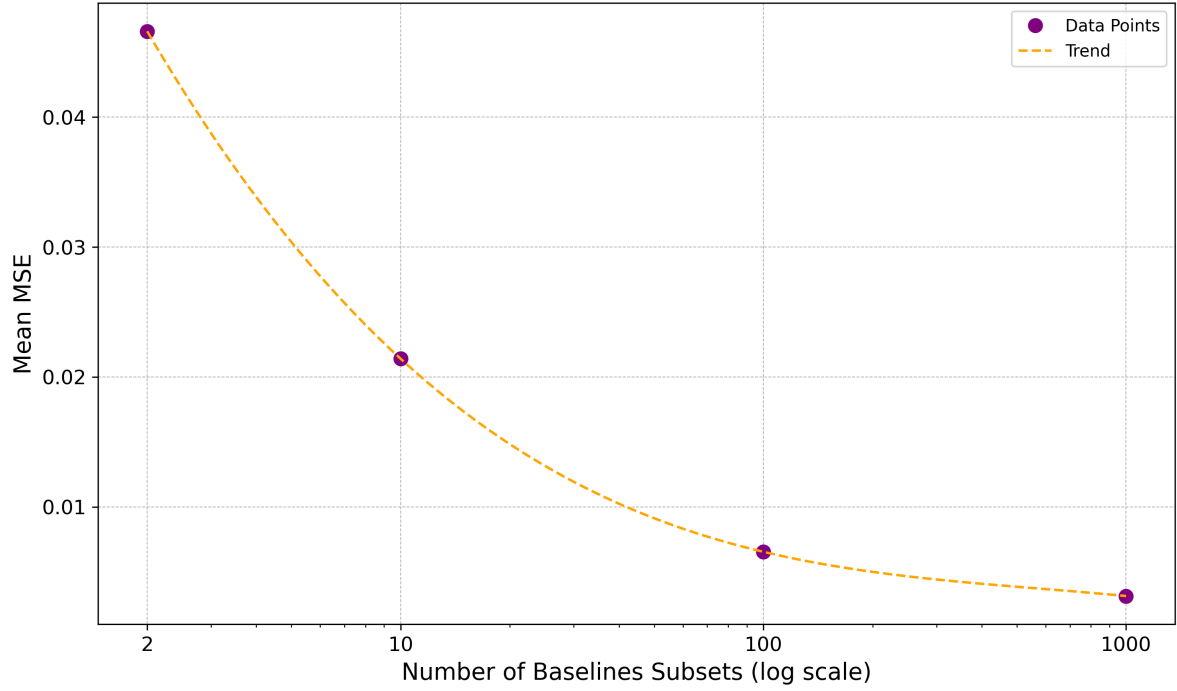


Figure 6.9: The plot shows the relationship between the mean squared error (MSE) and the number of baseline samples used in calculating G-DeepSHAP. The mean MSE is plotted on the y-axis, with the number of baseline samples represented on a logarithmic x-axis (from 2 up to 1,000). The baseline sample size trend line shows that interpretability performance becomes better as the baseline sample size increases. Note further iterations have shown this plot line to not be as smooth, however follow the same trend

shows the corresponding mean MSE values. The trend line demonstrates how interpretability performance evolves as the baseline sample size increases.

Typically, increasing the sample subset size, as shown in Figure 6.9, leads to a lower Mean MSE but at greater computational power. Consequently, the use of larger sample sizes results in a trend toward obtaining more accurate, more robust, and more generalizable interpretations.

6.4.3 Discussion

The accuracy of interpretability methods is measured quantitatively by the MSE. Different models and techniques can be used to test the way DL models interpret time series data best. It was found that the assumption of zero reference points could improve the performance of this

dataset by a good amount.

6.5 Conclusion

In the above, we have used a simple polynomial function to validate the test bench system; the system is shown as valid in synthetic data generation, model training, and feature attribution in a given environment. The data generation process successfully generated a dataset with known relationships, such that a learned TS-MLP could learn the polynomial mapping function with a minimal mean absolute error. This was confirmed by the model being able to capture the underlying function well.

Feature attributions computed by G-DeepSHAP validated the test bench’s capability to produce credible interpretations. Although computed attributions matched closely with theoretical ASV and exact Shapley values in some cases, differences highlighted the need for a well-chosen baseline and the model’s approximation quality. These results are useful in that they demonstrate the system’s reliability in estimating feature contributions.

The results of scaling and normalization experiments demonstrate the importance of preprocessing data for both performance and interpretability. We demonstrated that scaling improves attribution accuracy and enables the model to learn better feature relationships, as evidenced by comparing results on scaled and unscaled datasets. At the same time, they illustrate the importance of incorporating preprocessing techniques in interpretability pipelines.

Attribution results were greatly impacted by baseline selection, too. For structured data, using larger sample sizes, the G-DeepSHAP baseline was able to effectively reduce biases and capture complex feature interactions. Additionally, the analysis was used to show that increasing the sample size of the baseline improves the accuracy and stability of attributions at the cost of increased computation.

We used mean squared error (MSE) for an evaluation metric to quantify interpretability accuracy. Finally, we found that larger baseline sample sizes increased the accuracy of feature attribution, making the tradeoffs between computational cost and feasible interpretability straightforward.

Finally, the evaluation of the test bench system shows its adaptability and robustness, and that it is reliable for evaluating machine learning models. The test bench is based on this foundation, and this can be used to extend the test bench to support more complex relationships and temporal dependencies in future chapters.

Chapter 7

Test bench demonstrations

‘Quality is never an accident; it is always the result of intelligent effort.’ - John Ruskin

7.1 Overview

In this chapter, we explore the interpretability of TS-MLP and TCN models when used to predict polynomial functions with time dependencies. We evaluate the ability of the models to learn and explain time-dependent patterns using time-lagged features. We first experiment with applying a TS-MLP to a polynomial equation with time-shifted terms. Feature attribution is measured using G-DeepSHAP, and results are compared to the ASVs which are our true attribution values. We perform a similar experiment using a TCN to investigate interpretability. Finally, we add a more complex time-dependent equation and test the models’ ability to handle dynamic time shifts, as well as the accuracy of generated explanations.

7.2 TS-MLP on time-dependent polynomial

We briefly review the goal of the TS-MLP time-dependent polynomial experiment before describing the method used and discussing the results.

7.2.1 Goal

We incorporate a time dependency into a polynomial function. The aim is to demonstrate the ability of a TS-MLP to model such dependencies. We then analyse the explanations to determine their accuracy. The accuracy is obtained by comparing G-DeepSHAP to ASVs. The accuracy of this explanation will also establish a baseline model and explanation for later comparisons using a TCN.

7.2.2 Methodology

Dataset generation

In order to extend the dataset described in Section 6.2.4, we add time shifts; in particular, we include the time-shifted features for x_1 , x_2 , and x_3 . The output equation is still conditioned on these random points. The random point generation allows the model to learn what inputs from previous steps contribute to the output of the current time step. However, since the input features are generated independently at each time step, they contain no true temporal correlations—each value is drawn from a uniform distribution over $[-1, 1]$. As a result, plotting these features as time series would simply show random fluctuations with no meaningful trends. For this reason, we omit such a visualisation and instead describe their properties analytically: the uniform sampling ensures that the features are stationary with zero mean and unit variance, and that any temporal dependencies captured by the model arise solely from the imposed lag structure rather than from correlations in the data itself. This design provides a controlled baseline for evaluating whether models can exploit lagged inputs even in the absence of genuine time-series structure.

The transfer function is a multivariate polynomial function of the set of input features $x_1, x_2, \dots, x_k$. There is no change in the input features, but time shifts are introduced in the transfer function. The introduction of these time shifts results in the following equation:

$$y(t) = x_1(t-1) + (x_2(t-3))^2 + (x_3(t-4))^3 \quad (7.1)$$

At time step t , we have y as a function of x_1 , x_2 , and x_3 , at previous time steps $t-1$, $t-3$, and $t-4$, respectively. This structure allows for an evaluation of how well the model can learn and adapt to time-lapsed dependencies in the transfer function.

Model training

A TS-MLP model is used to learn the transfer function in Equation 7.1. Table 7.1 details the model architecture and training setup. These hyperparameters were determined through a grid search over learning rates, batch sizes, and network configuration, picking the model with the minimum validation loss. The Mean Square Error (MSE) loss function, the Adam optimiser, and a maximum of 120 epochs were used for all trials. Mean Absolute Error (MAE) is used during evaluation to measure the prediction accuracy of unseen data (The loss function and performance measure should not be confused).

To preserve the time sequence, data is temporally partitioned into a training set(60%), validation set(20%), and testing set(20%) by splitting the series along the time axis in chronological order. This means that earlier observations are assigned to the training set, the next block to validation, and the latest block to testing, ensuring that no future information leaks into past data. For all sets, the first few targets are padded with zeros as inputs so that the models can predict without discarding data at the start. An ‘input chunk’ refers to the number of time steps the model uses to predict the target. Here, 5 input (time) steps with 4 variables (features) are used as input chunks. Note that an additional feature, x_4 , is set to zero just to confirm that the model differentiates between relevant and irrelevant features.

Table 7.1: TS-MLP model training hyperparameters for polynomial with temporal effects

Hyperparameter	Value
Architecture	MLP
Width	512
Dropout rate	0.0
Task	Regression
Batch size	64
Optimiser	Adam
Learning rate	0.001
Loss function	MSE
Maximum epochs	120
Performance metric	Mean Absolute Error (MAE)
Input chunk	[-4, 0]
Input components	x_1, x_2, x_3, x_4
Output chunk	[0, 0]
Output component	transfer1
Data split method	Chronological
Training/Validation/Test split	60% / 20% / 20%

All model settings not specified in Table 7.1 are set to the Know-it default values as described in Appendix A.4. The data split method is chronological to prevent information leakage from future to past and to mimic real forecasting conditions.

Explanation analysis

To understand the model’s predictions, we use the ASVs and DSVs as described in Sections 3.3.1 and 3.3.4:

- **ASVs:** These values are calculated directly from the polynomial equation using the Exact Explainer, telling us how each feature contributed to the output.
- **DSVs:** These are calculated from the TS-MLP predictions and provide insight into what and when the model considers the essential features.

The DSVs values are then evaluated against the ASVs using the metric developed in Section 6.4, which we shall call the Average Exact Difference (AED).

7.2.3 Results

The TS-MLP was trained on input sequences of the three relevant features (x_1, x_2, x_3) over five past time steps, with an additional dummy feature x_4 set to zero to test feature relevance. The target output is $y(t)$, which is defined as a polynomial function of the lagged inputs (Equation 7.1). Training was performed on chronologically split data (60% train, 20% validation, 20% test) using the hyperparameters given in Table 7.1.

When evaluated on the test set, the TS-MLP achieved an MAE of 0.04. For interpretability, G-DeepSHAP produced explanations with an average explanation difference (AED) of 0.01×10^{-1} . Taken together, these metrics confirm that the model both predicts accurately and provides explanations that are closely aligned with the defined ground truth.

Figure 7.1 illustrates an explanation of one representative sample point (selected from the validation set) extracted from the TS-MLP model at epoch 93. This point is shown purely for illustration, and at this epoch the model achieved its lowest validation MAE of 0.04. The

coloured squares in the figure represent the contributions of the input features x_1 , x_2 , and x_3 at their respective time-lagged positions ($x_1 : t - 1$, $x_2 : t - 3$, and $x_3 : t - 4$) as defined in Equation 7.1. The random sample displayed in Figure 7.1 illustrates the model’s capability to identify relevant time steps.

More specifically, $y(t)$ is explicitly a function of the lagged inputs. This means that the G-DeepSHAP interpretation captures the same time dependencies as the expected model, attributing importance to features at the corresponding lagged time steps. While this is illustrated on an individual example for clarity, the same behaviour is consistently observed across many sampled points, confirming that the interpretation generalises rather than being sample-specific. By visualising these time-dependent contributions, we can assess how well the TS-MLP model captures and relies on temporal dependencies.

Mean absolute attribution heatmap

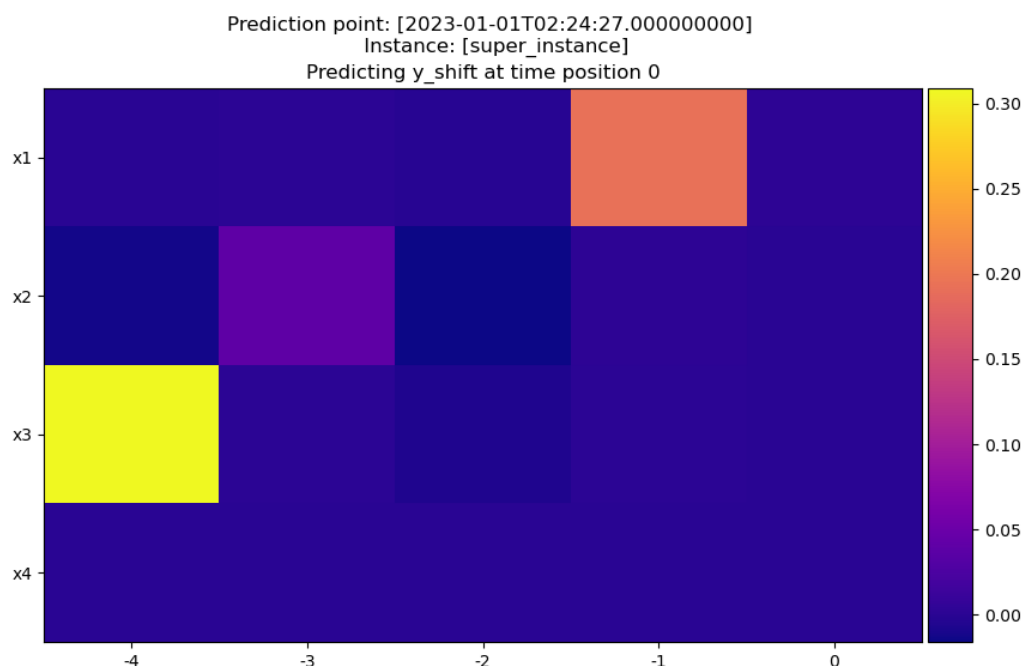


Figure 7.1: Mean absolute attribution heatmap for time-dependent feature contributions of the TS-MLP.

We can further confirm the way in which the model has identified the time shifts in Equation 7.1 by visualising the mean absolute attribution heatmap of features across all samples, as

shown in Figure 7.2. The absolute is taken for each sample element; the mean is computed and

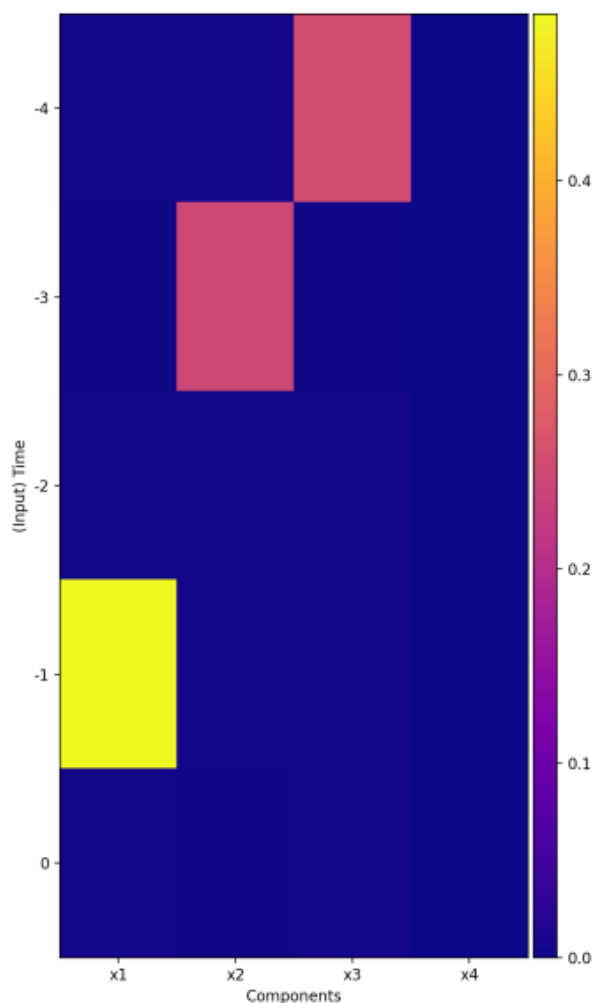


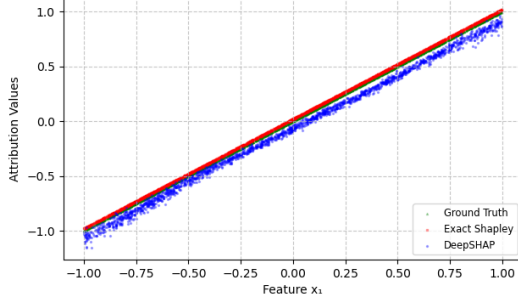
Figure 7.2: Average absolute effect of each feature over time.

displayed on the heatmap shown in Figure 7.2. The mean absolute heatmap helps us confirm that G-DeepSHAP estimates the non-relevant features of certain time steps near zero and can identify constant time shifts.

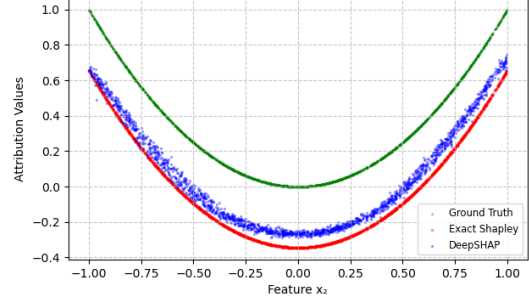
Feature attribution comparisons

Figure 7.3a shows the behaviour of x_1 , its target ground truth, (DSVs) and ASVs. The figures also serve as evidence of alignment between feature attributions and ASVs. Figure 7.3a shows the consistency between attribution methods applied to x_1 and validates the ability to capture its direct influence on the output. Likewise, in x_2 in **Figure 7.3b**, we focus on the squared term in

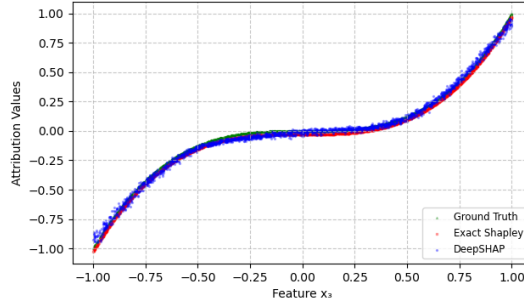
the predictions. The comparison of ASVs and DSVs with the ground truth demonstrates how the methods capture nonlinear contributions. In the end, **Figure 7.3c** shows the predictions made by x_3 and how its cubic term is important in the model. The comparison of these attribution methods to the visual analysis highlights their explanations and closely captures higher-order feature effects.



(a) Comparison of x_1 feature contributions



(b) Comparison of x_2 feature contributions



(c) Comparison of x_3 feature contributions

Figure 7.3: A comparison of feature contributions for x_1 , x_2 , and x_3 using ASV and DSV from the TS-MLP model.

Conclusion

These results show that the TS-MLP effectively learns the time-dependent polynomial function and that G-DeepSHAP provides faithful explanations of the temporal dependencies. The close agreement between ASVs and DSVs further verifies the reliability of the interpretability approach.

7.3 TCN on time-dependent polynomial

7.3.1 Goal

We repeat the same experiment with a TCN and uses the same polynomial function as defined in Equation 7.1. We again assess whether the explanations obtained are accurate, but in this case, when the time-lagged data is modelled with a TCN rather than a TS-MLP. We repeat this simple experiment as a first step to using a TCN to model and interpret a more complex dataset, as is done in the subsequent section.

7.3.2 Methodology

The experiment with a TCN and time-dependent polynomial exactly follows the same steps of the TS-MLP experiment discussed in Section 7.2 but uses a TCN model, which is better suited to handling time-dependent data. The TCN architecture is adapted to recognise patterns across time shifts, as discussed in Section 3.2.3. The only difference is in the architectural hyperparameters searched over. While the TS-MLP searches over hyperparameters such as layer width and the number of hidden layers to model sequential data, the TCN searches over architectural hyperparameters such as kernel size, dilation rate, and the number of convolutional filters to capture temporal dependencies. The final set of hyperparameters is shown in Table 7.2.

Again, we set feature four to zero to confirm that the model differentiates between relevant features. All the model settings not specified in Table 7.2 are then set to the Know-it default values, which can be found in Appendix A.4.

7.3.3 Results

We see only the important attribution values extracted for the equation in Figures 7.4a, 7.4b, and 7.4c, where the model reached its best validation performance (MAE of 0.41×10^{-1}) at epoch 99. These figures show that the attributions derived from the model correspond with the ASVs, as extracted by the exact explainer discussed in Section 3.3.2.

Table 7.2: TCN model training hyperparameters for polynomial with temporal effects

Hyperparameter	Value
Architecture	TCN
Dropout rate	0.0
Task	Regression
Batch size	64
Optimizer	Adam
Learning rate	0.001
Loss function	MSE
Maximum epochs	100
Performance metric	Mean Absolute Error (MAE)
Input chunk	[-4, 0]
Input components	x_1, x_2, x_3, x_4
Output chunk	[0, 0]
Output component	transfer1
Data split method	Chronological
Training/Validation/Test split	60% / 20% / 20%

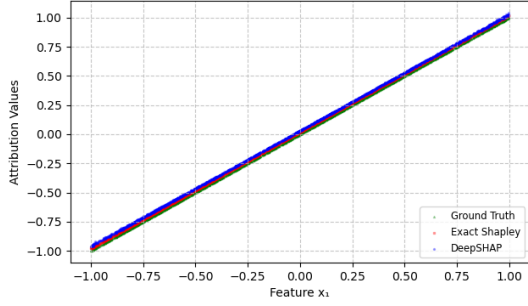
In particular, the attribution values for the TCN model agree with the ASVs, which confirms that the feature attributions are relatively well captured by the model. In contrast to the TS-MLP, the attributions in the TCN exhibit less curved line approximations when ordered to a polynomial term such as x_2 . These straighter line approximations contrast with the TS-MLP, whose attributions tend to follow smoother and more rounded approximations, as seen in Figure 7.4b compared to Figure 7.3b.

More importantly, while the two models (TS-MLPs in Section 7.2 and TCN here) learn and approximate feature attributions, these results suggest that one model might still be preferred over the other due to its explanation style. In other words, the TCN model obtains less sensitive attributions with respect to nonlinear interactions than the TS-MLP. The distinction highlights the role a model architecture plays in explanations.

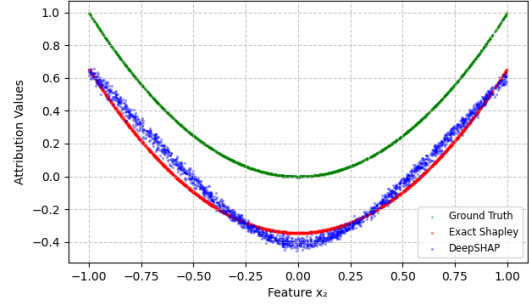
Figures 7.4 illustrate the explanations from the TCN model to be approximations of the ASVs and illustrate the close alignment between the explanations.

7.3.4 Conclusion

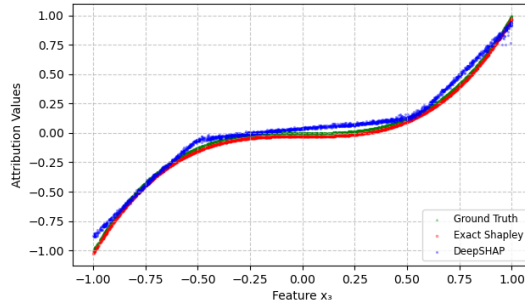
When evaluated on the test set the TCN model successfully learns the time-dependent polynomial function with a relatively low AED of 0.81×10^{-3} using the interpretation metric developed



(a) Comparative analysis of x_1 feature contributions.



(b) Comparative analysis of x_2 feature contributions.



(c) Comparative analysis of x_3 feature contributions.

Figure 7.4: A comparison of x_1 , x_2 , and x_3 feature inputs with their ASVs (from the polynomial equation), DSVs (DeepSHAP, from the TCN model), and true equation values.

Feature contributions are extracted from the model’s interpretation matrix and sorted by input magnitude.

in Section 6.4. Since the model is trained on the same data and considers the same time steps, one might assume that it produces more reliable explanations. However, a lower mean MSE does not necessarily indicate better interpretability—rather, it may simply reflect that the TS-MLP model is making more accurate predictions overall. The result shows that the TCN captures temporal dependencies well since the ASVs and DSVs closely align. We analyse the TCN’s interpretability across input features x_1 , x_2 , and x_3 by exploring their feature attributions in Figures 7.4a, 7.4b, and 7.4c. Although there are some distinct differences between these explanations and those of Figures 7.3a, 7.3b, and 7.3c, these are to be expected, since they use different architectures.

These findings confirm that the TCN model effectively captures time-lagged relationships, pro-

ducing feature attributions that closely align with ASVs. While both the TCN and TS-MLP generate meaningful explanations, their different architectural properties lead to distinct attribution behaviours. The TCN appears to smooth over nonlinear interactions, whereas the TS-MLP produces more curved approximations. This highlights the importance of selecting model architectures based on both predictive performance and interpretability needs in time-series analysis.

7.4 Solar wind – magnetopause standoff distance analysis

This section extends our analysis of model interpretability in time-dependent polynomial functions from Section 7.3 to a real-world solar wind dataset collected during storm events. The new complexities introduced by the solar wind dataset allow us to explore the use of these methods for practical interpretation. We apply insights gained from our experiments on time-dependent polynomial functions to the solar wind dataset, demonstrating how these findings inform our interpretation of solar wind interactions during storm events.

We describe two experiments: the first, where a model is provided with all necessary features for accurate predictions, and the second, where the model must estimate a feature value to generate a prediction.

7.4.1 Background

Interactions between the solar wind and planetary magnetospheres are a fundamental problem in space physics. These interactions define how a magnetized planet, like Earth, responds to a continuous stream of charged particles from the sun [55]. We investigate the key modelling parameters associated with the solar wind-Earth interaction in the framework developed by Walker and Russell (1995) [56]. We overview essential parameters such as magnetopause standoff distance (R_{mp}), solar wind speed (V_{sw}), proton number density ($\mathbf{N_p}$), and time delays.

We follow the notation convention outlined in Section 6.2.3, where physical parameters such as $\mathbf{v}$, $\mathbf{n_p}$, and $\mathbf{r_{mp}}$ are written in lowercase boldface to indicate their vector representation.

Magnetopause standoff distance (r_{mp})

The distance from the Earth's centre to the point where magnetic pressure inside the Earth's magnetosphere is balanced by the dynamic pressure of the incoming solar wind is called the magnetopause standoff distance (r_{mp}). The magnetopause is the outer boundary of the magnetosphere, and it serves as a protective shield, deflecting charged solar particles that could otherwise strip away the planet's atmosphere [57].

An empirical formula allowing the estimation of r_{mp} is proposed by Walker and Russell [55]: the solar wind pressure is equated to the planetary magnetic pressure. The approximation for the magnetopause distance is given by:

$$r_{mp} = k \left(\frac{b_0^2}{\mu_0 \rho v_{sw}^2} \right)^{\frac{1}{6}} \quad (7.2)$$

where k is a constant, b_0 represents the magnetic field strength at the planet's surface, μ_0 is the permeability of free space, ρ is the solar wind mass density, and v_{sw} is the solar wind speed. To simplify Equation 7.2, it is commonly rewritten in terms of the proton number density n_p and speed v_{sw} as follows:

$$r_{mp} \propto (v_{sw} \cdot n_p^2)^{-\frac{1}{6}} \quad (7.3)$$

As a case in point, the relationship in expression 7.3 shows that when the solar wind speed or density increases (thereby creating greater solar wind pressure), the magnetosphere compresses, decreasing the associated r_{mp} distance.

Solar wind speed (v_{sw})

The solar wind speed, v_{sw} , is the velocity at which charged particles travel from the Sun through space, primarily protons and electrons. This velocity is critical in measuring the dynamic pressure exerted on Earth's magnetosphere. The magnetosphere is particularly responsive to the solar wind with higher speeds, as increased v_{sw} enhances the compressive force on the magnetopause boundary.

The value of v_{sw} fluctuates in response to solar activity and the complex configuration of the interplanetary magnetic field. Near Earth, v_{sw} typically ranges from 300 to 800 km/s. However, it may increase significantly during solar storms or periods of high solar activity.

Proton number density (n_p)

The proton number density, n_p , represents the density of protons within the solar wind and is measured in particles per cubic centimetre. Since protons constitute the majority of the solar wind's mass, n_p is a primary factor in determining the total mass density of the solar wind.

Together with v_{sw} , the proton density controls the dynamic pressure of the solar wind, which in turn influences the extent of planetary magnetospheres. An increase in n_p results in higher dynamic pressure on the magnetosphere, compressing it and reducing the standoff distance r_{mp} .

Time delay estimation

Solar wind observations are taken some distance from the Earth (for example, at the L1 Lagrange point used by the current solar wind observation spacecraft). The L1 point is roughly 1.5 million km from Earth or just 1 percent of the distance to the Sun. The distance of the spacecraft from Earth introduces a time delay between observing solar wind at the spacecraft and its interaction with Earth's magnetosphere. In order for solar wind-magnetosphere interactions to be analysed accurately, it is necessary to synchronise solar wind conditions with geomagnetic responses so as to determine the time delay.

The time delay, or propagation time, Δt , is determined by dividing the distance d between the observation point and Earth's magnetosphere by the solar wind speed v_{sw} :

$$\Delta t = \frac{d}{v_{sw}} \quad (7.4)$$

Here, d represents the Euclidean distance from the spacecraft's position to Earth's magnetopause in a three-dimensional space:

$$d = \sqrt{x^2 + y^2 + z^2} \quad (7.5)$$

Let x , y , and z be the spacecraft's position coordinates (i.e., offset from Earth's centre) in kilometres. For ease of calculation, the distance is often normalized by Earth's radius ($R_{\text{Earth}} = 6,378 \text{ km}$).

Simplified time delay estimation

Instead of being reported at the spacecraft’s position, the solar wind parameters in the OMNI¹ dataset is propagated to a reference point near Earth’s magnetosphere [58]. Because the OMNI data are pre-time-shifted to Earth, we explicitly reverse this shift to restore upstream timing and ensure that the models learn genuine sequential dependencies rather than a static regression on lagged inputs. The bow-shock nose is the location on Earth’s dayside where the supersonic solar wind is first slowed and deflected by the magnetosphere. In our analysis, we deliberately *undo* this alignment using a simplified de-shifting method that assumes radial solar wind propagation and ignores transverse deviations. This restores an upstream time series that is only approximately aligned but is more suitable for training models to learn temporal dependencies.

The Euclidean distance of the spacecraft (d) from Earth’s centre was calculated directly from the position data (x, y, z) , and the delay was determined using the observed solar wind speed ($\mathbf{v}_{sw}$):

$$d = \sqrt{x^2 + y^2 + z^2}, \quad \Delta t = \frac{d}{\mathbf{v}_{sw}}.$$

Equation 7.4 is a simple way to approximate the time differences and provides a point of comparison with OMNI time shifts. At the same time, data points were converted to minutes for clarity and consistency, as we have been working with time delays.

Only points 40,344 to 41,364 are displayed in Figure 7.5, which shows solar wind data over a 17-hour period. The values obtained by undoing the OMNI time shift using our simplified method are displayed in solid lines, while the original OMNI measurements (which were already pre-shifted to Earth) are plotted in dashed lines.

Applications and importance

The accurate estimation of time delays is required for the time series synchronizing of solar wind data with geomagnetic responses. Real-time space weather forecasting means correcting the time delays and an improved the understanding of the coupling of magnetospheric solar

¹OMNI data are propagated to the bow-shock nose and thus already represent Earth-aligned conditions; see OMNIWeb documentation at <https://omniweb.gsfc.nasa.gov/>.

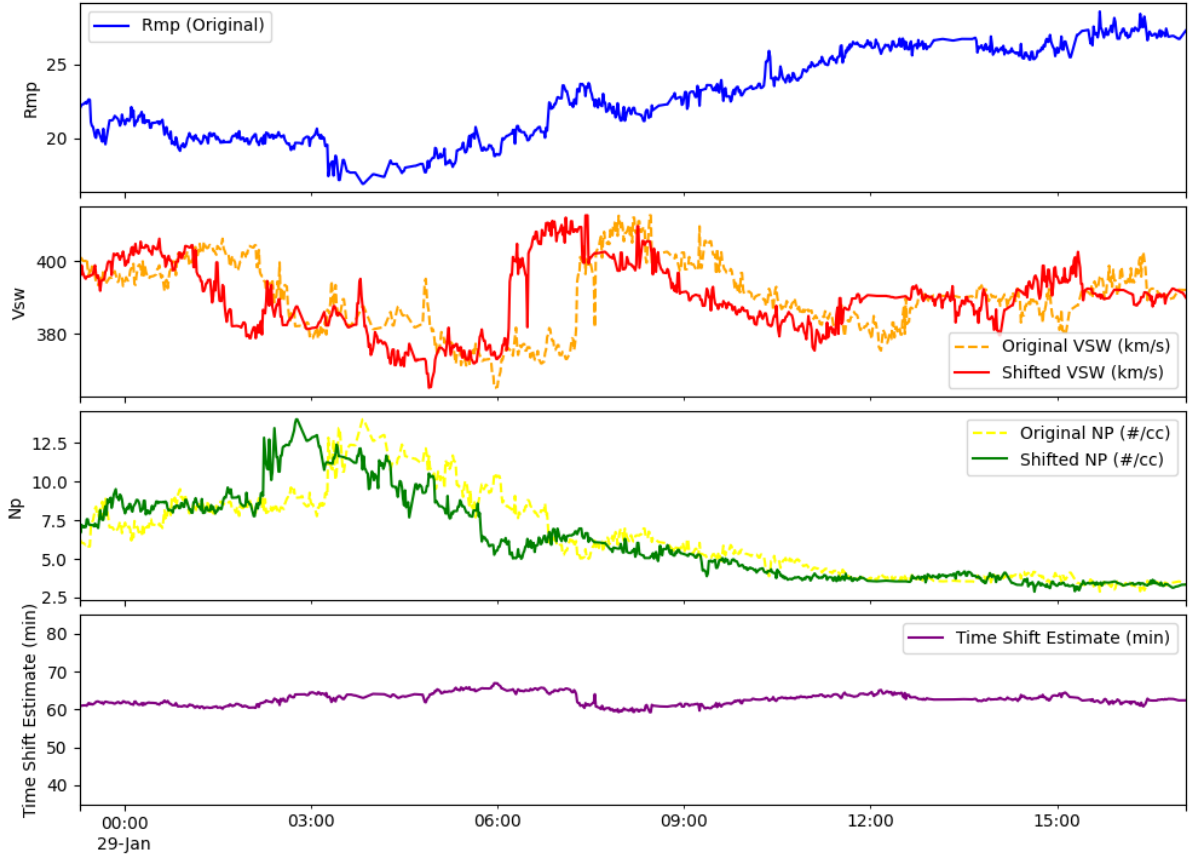


Figure 7.5: OMNI values for v_{sw} and n_p , with r_{mp} calculated from the (dashed) original values using Equation 7.3. The shifted v_{sw} and n_p features are time-lagged using the simplified time shift into past time steps.

wind interaction. Such predictions are needed to forecast the levels of geomagnetic storms and mitigate the possible effects of these storms on technological systems within the Earth's space environment.

We put emphasis on the approach employed in this study: We used the simplified calculation to target specific experiments. We explore the evaluation of time delay techniques during solar wind data analysis.

7.4.2 Magnetopause standoff distance

The magnetopause standoff distance, represented as $\mathbf{r}_{\text{mp}}$, is estimated using an empirical function derived from Walker et al. [56]:

$$\mathbf{r}_{\text{mp}} = 110.2(\mathbf{v}_{sw}\mathbf{n}_p^2)^{-1/6} \quad (7.6)$$

In our approach, as outlined in Section 6.3.1, we apply scaling to these values to enhance learning accuracy in the DNN. Specifically, we employ zero-one scaling, as detailed in Section 6.3.1, to rescale all values between 0 and 1 using the standard formula:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \quad (7.7)$$

The transformation shown in Equation 7.7 ensures that $x_{\min}$ is mapped to 0 and $x_{\max}$ to 1. After scaling, Equation 7.6 is reformulated as follows:

$$\mathbf{v}'_{sw} = \frac{\mathbf{v}_{sw} - v_{sw\min}}{v_{sw\max} - v_{sw\min}} = \frac{\mathbf{v}_{sw} - v_{sw\min}}{\sigma_{\mathbf{v}_{sw}}} \quad (7.8)$$

$$\mathbf{n}_p' = \frac{\mathbf{n}_p - \mathbf{n}_{p\min}}{\mathbf{n}_{p\max} - \mathbf{n}_{p\min}} = \frac{\mathbf{n}_p - \mathbf{n}_{p\min}}{\sigma_{\mathbf{n}_p}} \quad (7.9)$$

$$\mathbf{r}_{\text{mp}} = 110.2 \left((\sigma_{\mathbf{v}_{sw}} \mathbf{v}'_{sw} + v_{sw\min})(\sigma_{\mathbf{n}_p} \mathbf{n}_p' + \mathbf{n}_{p\min})^2 \right)^{-\frac{1}{6}} \quad (7.10)$$

If we scale the target values further, Equation 7.10 can be rewritten as:

$$\mathbf{r}_{\text{mp}}' = \frac{110.2 \left((\sigma_{\mathbf{v}_{sw}} \mathbf{v}'_{sw} + v_{sw\min})(\sigma_{\mathbf{n}_p} \mathbf{n}_p' + \mathbf{n}_{p\min})^2 \right)^{-\frac{1}{6}} - \mathbf{r}_{mp\min}}{\mathbf{r}_{mp\max} - \mathbf{r}_{mp\min}} \quad (7.11)$$

Notably, the model scales values based on the minimum and maximum of the training data, which means all scaling parameters are derived solely from the training dataset.

Furthermore, if we consider a time-shifted variant, Equation 7.6 becomes:

$$\mathbf{r}_{\text{mp}}(t)' = \frac{110.2 \left((\sigma_{\mathbf{v}_{sw}} \mathbf{v}'_{sw}(\tau) + v_{sw\min})(\sigma_{\mathbf{n}_p} \mathbf{n}_p'(\tau) + \mathbf{n}_{p\min})^2 \right)^{-\frac{1}{6}} - \mathbf{r}_{mp\min}}{\mathbf{r}_{mp\max} - \mathbf{r}_{mp\min}} \quad (7.12)$$

where τ represents the time shift, defined as $\tau = t - d$, with d calculated as a simplified dynamic time delay based on the spacecraft's position and the observed solar wind speed, rather than the full OMNI time-shifting methodology.

If τ is provided to the model as an input feature, it is scaled similarly, so the model receives τ' , the scaled version. The model can learn the original time shift effect from the scaled input. The scaling of τ follows:

$$\tau' = \frac{\tau - \tau_{\min}}{\tau_{\max} - \tau_{\min}} \quad (7.13)$$

During model inference, to identify relevant time steps, the model needs to reverse the scaling, expressed as:

$$\tau = \tau'(\tau_{\max} - \tau_{\min}) + \tau_{\min} \quad (7.14)$$

In addition, as the data is indexed in minutes while the original input data is in seconds, we further refine the calculation:

$$\tau = \text{round}(\tau'(\tau_{\max} - \tau_{\min}) + \tau_{\min}) \quad (7.15)$$

Equation 7.15 rounds τ to the nearest integer minute, ensuring consistency in time indexing across data resolutions.

Theoretical solar Shapley values

To compute the Shapley values for a solar wind model, we express the output as follows:

$$\mathbf{r}'(\mathbf{v}_{sw}, \mathbf{n}_p) - \phi_{0_{\mathbf{r}'(\text{baseline})}} = \phi_{\mathbf{v}_{sw}} + \phi_{\mathbf{n}_p} \quad (7.16)$$

where:

- $\mathbf{r}'(\mathbf{v}_{sw}, \mathbf{n}_p)$ represents the output of the function for specific values of $\mathbf{v}_{sw}$ and $\mathbf{n}_p$.
- $\phi_{0_{\mathbf{r}'(\text{baseline})}}$ is a reference output, typically determined using baseline values of $\mathbf{v}_{sw}$ and $\mathbf{n}_p$ (e.g., the mean values or calculated using the G-DeepSHAP approach).
- $\phi_{\mathbf{v}_{sw}}$ and $\phi_{\mathbf{n}_p}$ denote the Shapley values, quantifying the contributions of $\mathbf{v}_{sw}$ and $\mathbf{n}_p$ to the deviation of the model output from the reference value.

In addition, the DSVs are computed for the input features, which include the time shift (**tshift**) as one of the features. However, the neural network does not inherently recognise **tshift** as a

time shift signal and thus cannot directly attribute or explain its effects as such. As measured by the impact on the output, feature relevance is captured by G-DeepSHAP, while time itself, as a unit, does not provide any contribution as a predictive feature. Rather, time adds context by indicating at specific intervals which feature is most important towards the final output.

Measurements of $\mathbf{n_p}$ and $\mathbf{v_{sw}}$, as well as the simplified estimates of *time_shift_estimate*, are utilized in this study. Unlike the precomputed time shifts in the OMNI dataset, which represent the actual number of seconds by which each data point has already been shifted, our approach undoes the shift using a simplified methodology based on spacecraft position and observed solar wind speed.

7.4.3 Informed magnetopause standoff distance system

Goal

The goal of this experiment is to train a predictive model for the magnetopause standoff distance, $\mathbf{r_{mp}}$, using real measured solar-wind data as input features. Specifically, the model takes solar wind speed ($\mathbf{v_{sw}}$) and solar wind pressure ($\mathbf{n_p}$) as inputs and predicts $\mathbf{r_{mp}}$ as the target variable.

We evaluate whether the model can generate explanations that align with the ground truth ASVs, which are derived from the analytical equation governing $\mathbf{r_{mp}}$. This equation serves as a reference for assessing whether the feature attributions produced by the model correspond to known physical relationships.

Methodology

A TCN model is configured and trained using scaled input data following the same protocol as described in Section 7.2. The experimental setup includes the key hyperparameters and model configurations as summarised in Table 7.3. An ideal set of hyperparameters was chosen to optimize model performance. Model configurations not specified in Table 7.3 default to Know-it's standard settings.

Table 7.3: TCN model training hyperparameters for storm data with temporal effects

Parameter	Value
Architecture	TCN
Dilation base	2
Dropout rate	0.0
Task	Regression
Input components	np_shifted, vsw_shifted, tshift_est_minutes
Input chunk	[-100:0]
Output components	rmp
Output chunk	[0:0]
Data split method	Chronological
Split portions (Train/Validation/Test)	60% / 20% / 20%
Scaling method	Zero-One
Scaling tag	Full (input and output scaled)
Batch size	32
Learning rate	0.001
Loss function	MSE
Optimizer	Adam
Maximum epochs	150
Performance metric	Mean Absolute Error (MAE)
Train/Validation/Test sizes	30,000 / 10,000 / 10,000

Results

The trained model configuration shown in Table 7.3 achieves a mean absolute error (MAE) of 0.014, indicating a high level of accuracy in predicting $\mathbf{r}_{\mathbf{mp}}$.

Interpretation

Figure 7.6 provides a random sample interpretation heatmap, showing DSV per sample and time step. We can clearly see that the model takes a few different time steps as important and not just one, as would be expected from ASVs for a known equation.

Overall, the results align closely with expectations derived from the underlying equations. The model correctly highlights *np_shifted* as the dominant feature, reflecting its squared role in the formulation, while *v_{sw}* contributes less but is still in line with its expected influence. The *time_shift_est* feature shows a limited direct effect on predictions, which is consistent with its artificial inclusion for experimentation. The distribution of maximum attribution points also follows the expected lag structure, with the model capturing fixed time dependencies reliably.

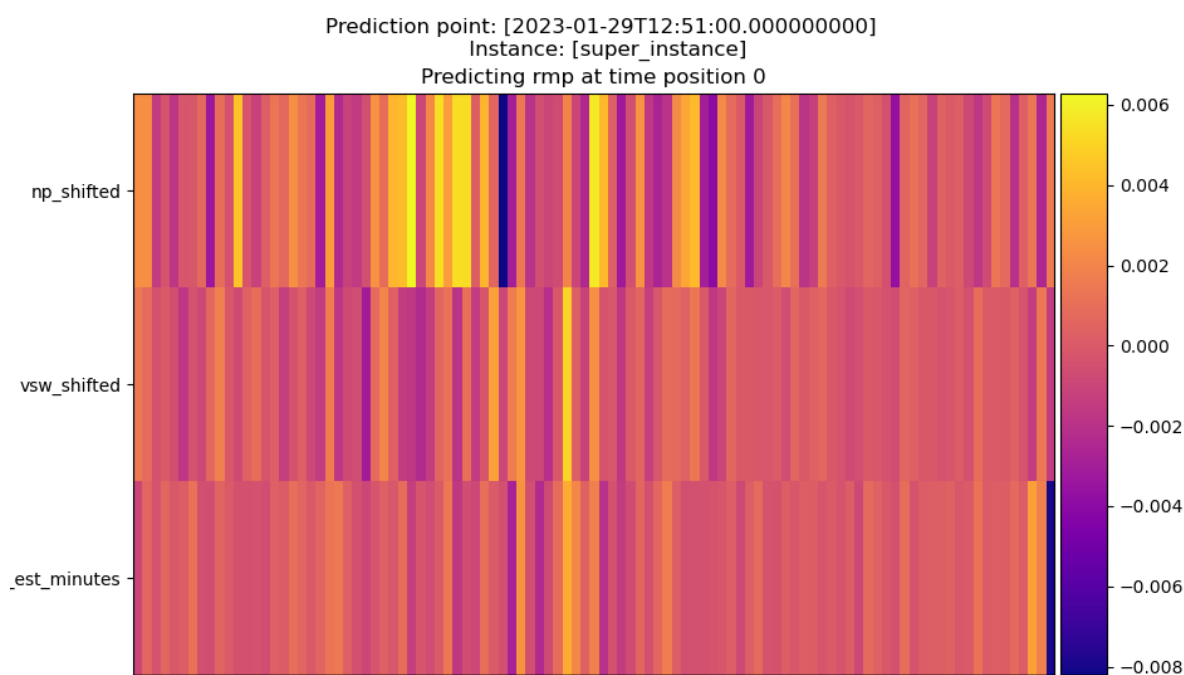


Figure 7.6: Mean absolute heatmap of interpretations, showing the significance of each feature over time (x-axis starting from one second ago on the right-most tick).

Some deviations appear in cases with more dynamic time effects, but even here the trends correspond to the intended behaviour, confirming that the interpretability results are largely as anticipated.

We also extract the mean absolute interpretations from the model. Figure 7.7 presents the mean absolute interpretation heatmap across the evaluation sample set, calculated over 1,000 points. Previously, we focused on the importance of features and time. Now, we also consider their significance to the prediction point—how much each feature and time step contributed to the model’s prediction at a specific instance. The leftmost pane highlights the importance of each time step for a specific feature input, with color intensity representing the mean absolute DSVs value across 1,000 samples from the evaluation set. The middle pane illustrates how much the input feature contributes to a specific prediction point, again using color to indicate the magnitude of its mean absolute value. The rightmost pane captures the importance of each time step to the prediction itself, with color showing the overall significance across the evaluation set. Together, these panes provide a complete view of the relationships between feature and time, feature and prediction, and time and prediction.

The left pane in Figure 7.7 displays the mean absolute DSV for each feature over time, indicating

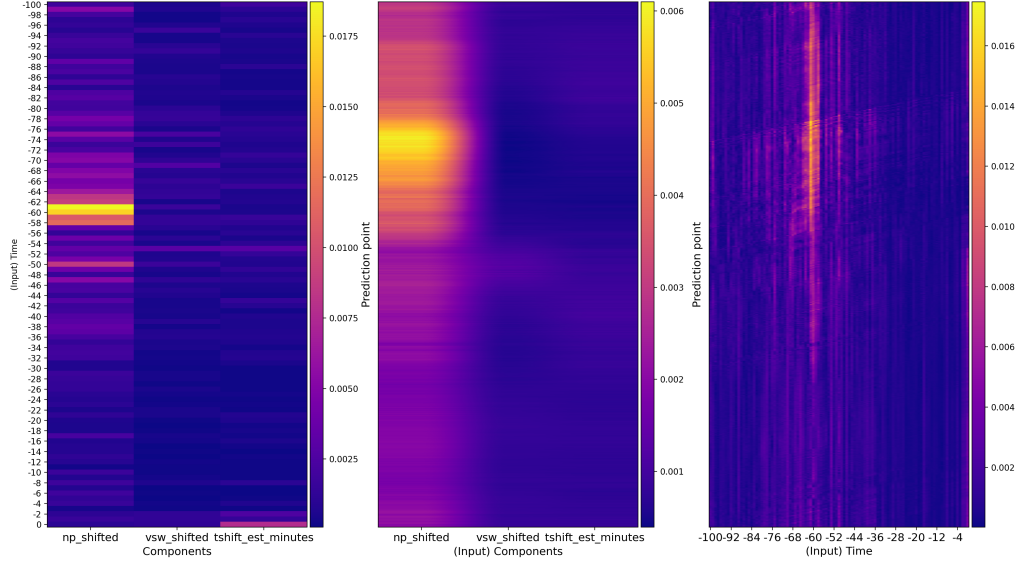
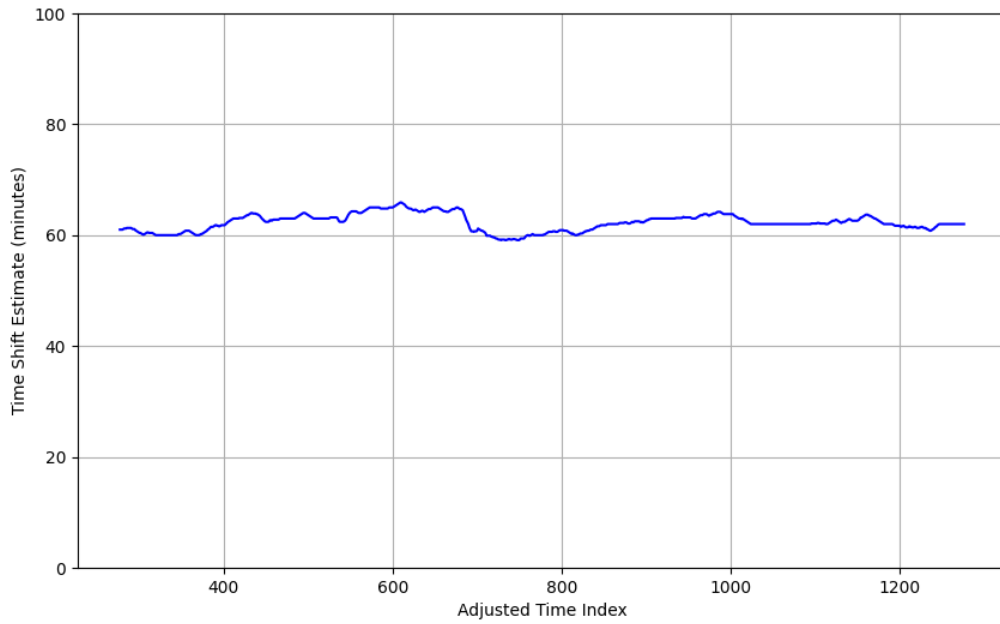


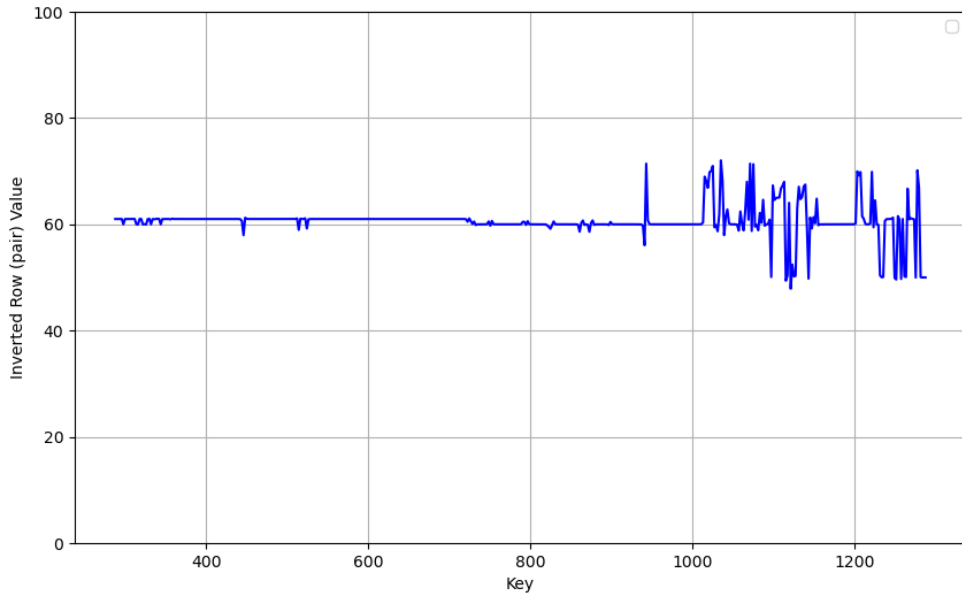
Figure 7.7: Mean absolute interpretation heatmap of the DSV from the ground truth, evaluated over 1,000 samples of the validation set. The left pane shows feature importance over time, the middle pane highlights feature contributions to specific prediction points, and the right pane captures time step importance to the final prediction.

that *np_shifted* is the most influential feature, especially around the 60-minute mark. The $\mathbf{v}_{sw}$ feature is less significant but still contributes at specific time steps. This aligns with Equation 7.12, where *np_shifted* is squared, amplifying its influence when its value is greater than 1, while reducing its contribution when it is less than 1.

The *time_shift_est* feature’s most recent value also appears crucial, as shown in the lower-right corner of the left pane in Figure 7.7. While its inclusion may seem unconventional – since the time delay is not typically known a priori in input-output mappings – it was included here explicitly for experimental purposes to evaluate its impact on model performance and interpretability. The middle panel suggests that this feature has almost no influence on the prediction point, while the right panel highlights the most influential time step for each prediction point. The slanted lines show that as time moves forwards or backwards, the most important past time step also shifts forwards or backwards, as the model predicts in a time-based trend.



(a) Gaussian smoothing time series absolute feature time points of true ASVs



(b) Model predictions G-DeepSHAP time series absolute points for time points.

Figure 7.8: Evaluation of storm data using G-DeepSHAP: (a) Gaussian smoothing time series absolute feature time points of true ASVs, (b) Model predictions G-DeepSHAP time series absolute points for time points. Both figures use 1,000 sample points for reference values as outlined in Section 6.3.4.

If we plot the time steps corresponding to each sample’s absolute maximum feature values, we obtain Figure 7.8b, showing the distribution of these key time steps across samples vs. the true extracted time steps in Figure 7.8a. We can observe here that the model does, however, find a mean for the time shift.

Additionally, Figure 7.8 highlights the maximum absolute feature’s time shift step per sample, providing insight into which time steps are most relevant for predictions. The linear trends in the graph correspond to the *time_shift_est* values at the respective sample points, illustrating how this feature evolves dynamically across the dataset. This graph offers a clear view of predominant time-shift values per feature, but does not account for all feature time effects. This approach contrasts with the fixed time effects observed in Section 7.2 and demonstrates the TCN’s ability to capture consistent time dependencies, though it struggles with dynamic time effects.

Conclusion

We can successfully fit the model to predict the magnetopause standoff distance based on the given input features. Although the raw time index is not included as a separate feature, the inputs are shifted in time, so the model does capture temporal dependencies. This means that the learned attributions highlight lagged inputs ($x_1 : t - 1$, $x_2 : t - 3$, $x_3 : t - 4$) rather than a direct time variable. In the case of the $\mathbf{r}_{\text{mp}}$ equation, the dependence arises from direct relationships between the solar wind parameters and $\mathbf{r}_{\text{mp}}$, expressed at these shifted positions. As a result, there is no explicit attribution to “time” itself, but the time-shifted features do play an important role.

The lack of growth limits, meaning that the model does not exhibit a clear feature attribution stability, suggests an area for future work requiring a new performance evaluation method. In particular, we do not obtain ASVs for a raw time feature, even though time-shifted contributions are non-zero and important for accurate explanations. We investigate this limitation further in the following section by considering the implications of omitting a direct time feature.

7.4.4 Ill-informed magnetopause standoff distance system

To establish a benchmark for model interpretation, we first extract the mean absolute interpretations from the ASVs of Equation 7.12, which does not take into account time as a feature for attribution value calculation as in Section 7.4.3 and which represents the expected ASVs the model should closely approximate if predicting correctly. Figure 7.9 presents the mean absolute interpretation heatmap across the evaluation sample set, calculated over 1,000 points.

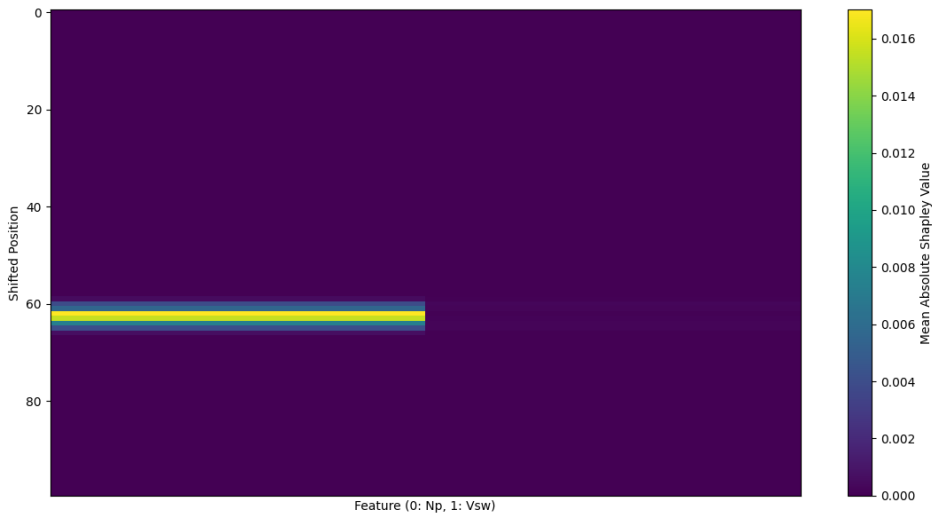


Figure 7.9: Mean absolute interpretation heatmap of the ASVs from the ground truth, evaluated over 1,000 sample points.

If we plot the time steps corresponding to each sample’s absolute maximum feature values, we obtain Figure 7.8a, showing the distribution of these key time steps across samples.

As seen in Figure 7.8a, which has been Gaussian smoothed, there is quite a resemblance compared to the time shift data in Figure 7.5’s time shift estimate. This ground truth analysis provides an expected interpretation pattern for the model’s performance. In the ideal scenario, we would anticipate similar patterns in the model’s interpretations. However, this section explores the TCN’s potential shortcomings in achieving this ideal. Expanding upon the previous experiment, this section examines the effect of excluding the `tshift_est_minute` feature from the model. Removing this feature makes the model more compatible with the Shapley value

interpretability metric, although this exclusion inherently reduces the information available to the model. Consequently, the model must independently learn the dynamic temporal effects.

This approach highlights the challenge of providing the model with time-shift information as an explicit input feature. While additional input information might intuitively seem helpful, the model lacks a mechanism to interpret a feature like time shift (X) as ‘the number of steps by which other features should be shifted.’ Instead, if the actual time shifts are available, it would be more practical to apply them directly to the data before training, avoiding unnecessary complexity.

Goal

The goal of this experiment is to train the model using only those feature variables directly impacting the ASVs calculations, namely $\mathbf{v}_{sw}$ and $\mathbf{n}_p$, excluding the explicit time shift. This setup aims to reveal valuable explanatory insights derived solely from the feature variables for our metric analysis.

Method

The model configuration remains consistent with that in Table 7.3, except for reducing the input features to include only $\mathbf{v}_{sw}$ and $\mathbf{n}_p$. We follow the same training and evaluation processes as in the previous experiment.

Results

The results show that the model with a reduced input feature set has a MSE of 1.7×10^{-2} , slightly more than the previous configuration, as expected. In Figure 7.10, we present model predictions and true values in the evaluation set versus the sum of G-DeepSHAP attributions and the sum of ASVs using our baseline G-DeepSHAP from the same set.

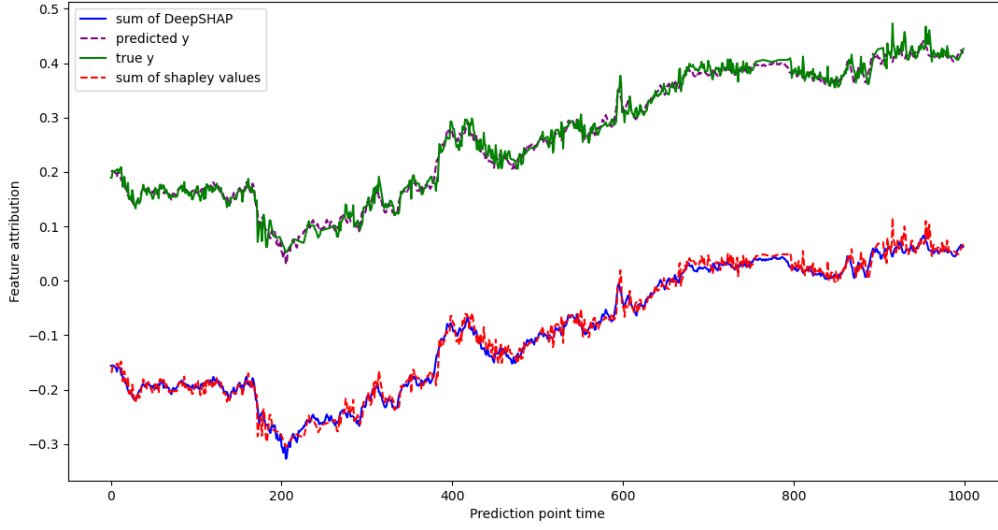


Figure 7.10: Evaluation of storm data: true and predicted values, alongside the sum of DSVs, using 1,000 sample points as discussed in Section 6.3.4. Only evaluation set points between sample indices 40,344 and 41,344 are plotted here.

Conclusion

We visualize the mean absolute Shapley values across interpretations in Figure 7.11 to further examine feature contributions.

From Figure 7.9, we observe a noticeable shift in the mean feature contributions, particularly for $np_shifted$, compared to Figure 7.11. Interestingly, the model continues to emphasize the most recent value of $\mathbf{v}_{sw}$, as seen in the lower right of the left panel. The dominant contribution to the prediction remains np , which is consistent with the $\mathbf{r}_{mp}$ equation where np enters quadratically and therefore exerts a stronger influence than the other features. The model’s learned time effects differ slightly from the previous configuration, even if we cannot explicitly capture the exact time function. Comparing Figures 7.7 and 7.11 reveals the model’s adaptation in identifying relevant time steps under varied setups.

As a final analysis, we plot the most relevant feature’s time step based on the maximum absolute feature value per sample. Figure 7.12 illustrates the relevant time graph for each sample, which once again achieves a mean but not the exact function compared to Figure 7.8a.

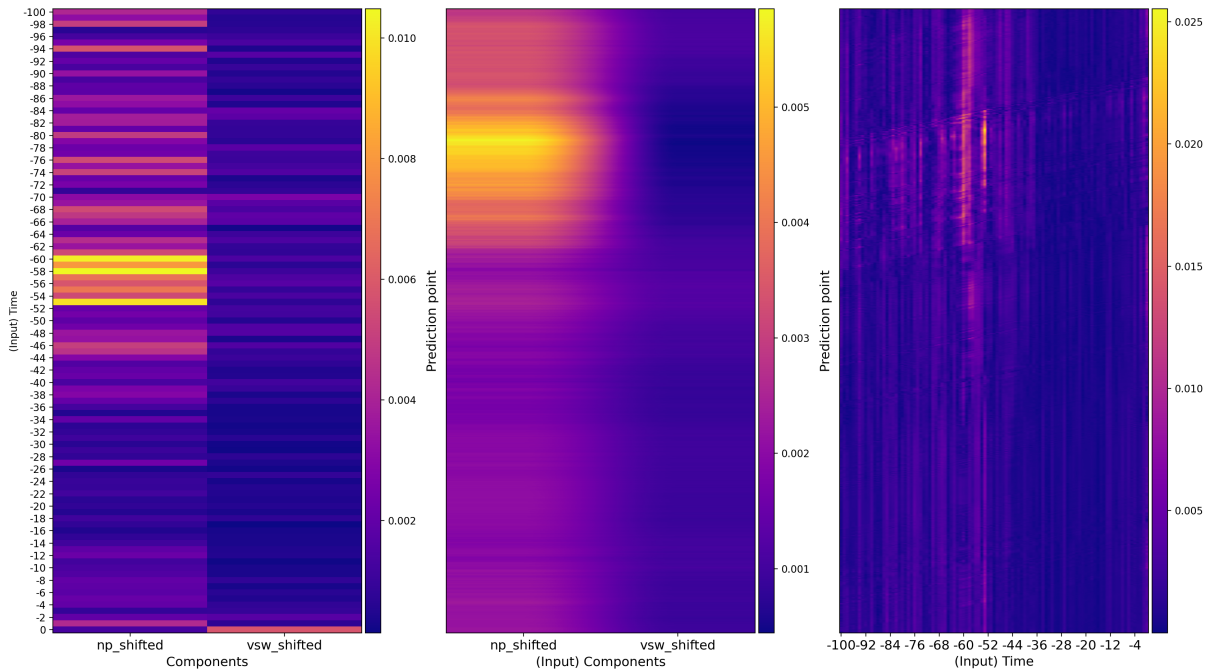


Figure 7.11: Mean absolute heatmap of feature contributions for the model without time shift information.

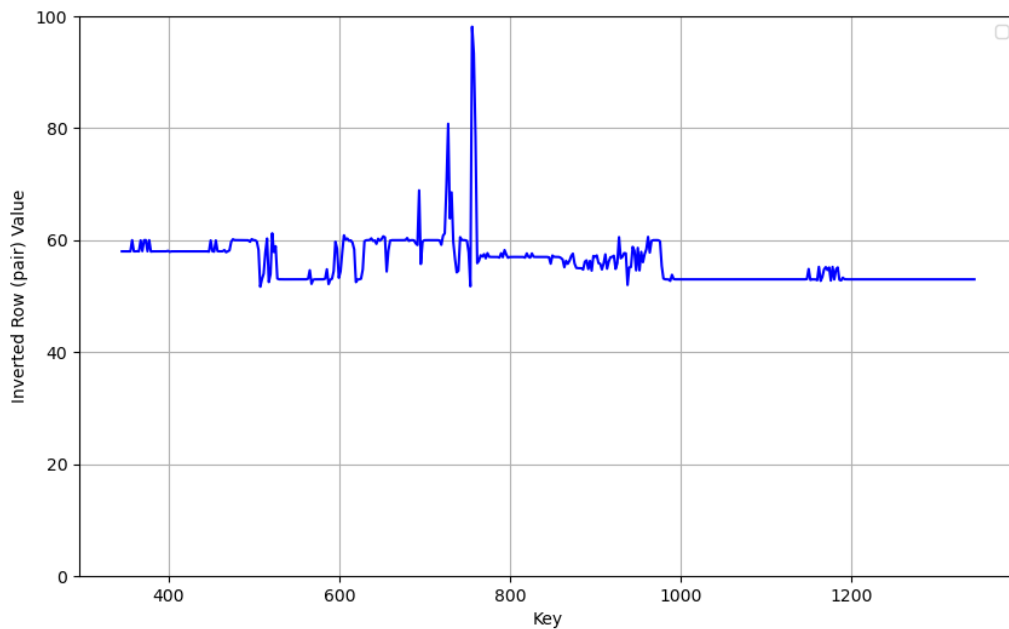


Figure 7.12: Time step distribution for the most relevant features across samples, showing the model's response to the lack of explicit time shift information.

Based on this visualization, the model has difficulty completely encapsulating the dynamic time effect without direct time-shift data, which may be a limitation in its explanations.

We conclude by computing the mean MSE over ASVs to get 2.94×10^{-5} . One might initially suspect that the matrices are similar since this is a low mean MSE. However, this value is dominated by the fact that many feature values at different time steps are close to zero, which is more pronounced in features that are not very important.

The mean MSE provides an effective way to analyse how well the models are interpreted. However, it does not necessarily allow us to isolate important differences between the top features. Therefore, a fairly general assessment of interpretability alignment allows for more subtle distinctions between features with higher contributions to be overlooked.

7.4.5 Evaluation metrics

In the earlier simplified experiments, we focused only on mean squared error (MSE) as a first proof of concept, since it provided a straightforward way to compare attribution matrices while we developed and validated the technique itself. At this stage, however, the evaluation is expanded to include additional metrics that capture more nuanced aspects of similarity and interpretability. These metrics allow us to distinguish not only overall alignment (via MSE) but also how models differ in their emphasis on key features or non-zero elements.

In this section, we evaluate the alignment, similarity, and interpretability of feature-level attributions using quantitative metrics. We analyse the matrices using the defined metrics. Each metric provides insight into how well two matrices align, their similarity, and their interpretability at the feature level. The reported metric values represent per-sample averages, with different metrics averaging over different aspects of the data. Table 7.4 outlines these metrics in detail.

MSE

In our evaluation, the AED across all compatible file pairs was 2.94×10^{-5} . This suggests that, overall, the matrices exhibit a high degree of similarity across most of their elements.

Table 7.4: Per-sample evaluation metric

Metric	Description
MSE	The averaged squared difference between matrix elements.
Weighted mean squared error	Calculates MSE in the same way with weights applied to each element, paying attention to the data points that are not zeros in both matrices and which may indicate higher variability in potentially more important features.
Top-5 feature difference	calculates the absolute differences with respect to the five largest feature values in each matrix, focusing on the features with the most effect sizes.
Cosine similarity	Calculated as the cosine similarity of non-zero features, it determines the directional consistency of non-zero components and whether matrices share that similarity in feature prioritization.

Weighted MSE

The weighted MSE down-weights the zeros and increases the weight of non-zero attribution values. This lessens the impact of unimportant features while emphasizing variation in larger, more informative ones. Compared to the AED, which averages across all entries, this makes it a more focused comparison.

$$\text{Weighted MSE} = \frac{\sum_{j=1}^n w_j (x_j - y_j)^2}{\sum_{j=1}^n w_j}$$

where w_j is the weight given to element j , and x_j and y_j are the attribution values for element j in the two matrices under comparison. In our instance, only non-zero attributions contribute to the error because $w_j = 1$ for non-zero entries and $w_j = 0$ otherwise.

The weighted MSE value for our matrices was 2.95×10^{-5} , which is very close to the mean MSE. This indicates that the non-zero attribution values in the two matrices are highly similar.

Top-5 feature difference

The top 5 feature difference metric is designed to capture the variation between the most influential features of two attribution matrices. To compute this measure, we first identify the

five elements with the largest absolute attribution values in each matrix, since these represent the dominant features. We then compare the corresponding values from the two matrices and calculate their mean absolute difference:

$$\text{Top-5 feature difference} = \frac{1}{5} \sum_{j \in \mathcal{I}_5} |x_j - y_j|$$

where $\mathcal{I}_5$ is the set of indices of the five largest absolute attributions (taken from one of the matrices or from their union, depending on the evaluation design), and x_j and y_j are the attribution values at those indices.

This metric is particularly useful for understanding differences in the most significant features. In our evaluation, the Top-5 Feature Difference was equal to 1.7×10^{-2} . Given that the dominant feature attributions typically range between 0 and 1 in magnitude, this small difference shows that the two matrices agree closely on their most important features. In other words, the variation among the Top-5 values is minor when compared to the scale of the attributions themselves and to the overall MSE values reported earlier.

Cosine similarity

The cosine similarity of matrices with corresponding non-zero values determines the directional relationship of their features and reveals the feature’s importance pattern. This is calculated as cosine similarity:

$$\text{Cosine similarity} = \frac{\sum_{j=1}^m x_j y_j}{\sqrt{\sum_{j=1}^m x_j^2} \cdot \sqrt{\sum_{j=1}^m y_j^2}}$$

In this context, the objects x_j and y_j correspond to the non-zero entries in the matrices, whereas the symbol m corresponds to the total number of efficient entries. A value closer to 1 means that the two directional patterns are quite similar, and vice versa; that is, a value closer to -1 means the two directional patterns will be in complete contrast.

In the experiments of this study, the directional impact is slightly misaligned, which is reflected in the cosine similarity of non-zero elements between the two matrices, which is approximately -0.21×10^{-1} . What this result tells us is that the magnitudes of the important features are fairly similar, but what they influence — whether it is positive or negative — differs slightly.

Conclusion

The MSE of 2.94×10^{-5} and the weighted MSE of 2.95×10^{-5} are on the order of 10^{-5} , far smaller than the 0-1 attribution scale, indicating almost identical metric behavior and minimal deviation between evaluation indicators. This implies that the matrices have their values aligned very closely. The Top-5 Feature Difference tells about the existence of rather small differences between the most important features, and the Cosine Similarity tells about small directional changes in non-zero elements. These metrics are used together to give an overall picture of how similar two matrices are and to look for subtle differences in which of the features within a matrix are more relevant.

7.5 Conclusion

We show that TS-MLP and TCN are effective models for modelling and interpreting time-dependent relationships in both synthetic polynomial data and a real-world solar wind dataset. We examine the ability of these models to learn time shifts and to offer explanations, as well as the accuracy of these explanations.

When applied to model simple synthetic polynomial equations, both the TS-MLP and TCN learned time-lagged dependencies very well, as attested by low mean squared errors of the interpretability metric. We assessed feature relevance over time by the use of ASVs and DSV. We then used them to quantify how individual features contribute to prediction, providing complementary local and global explanations of feature relevance. Specifically, the TCN was observed to provide better interpretability metric accuracy in understanding patterns of sequential data patterns than the TS-MLP.

Applying these models to real solar wind data, we extended their interpretability to practical problems, namely predicting magnetopause standoff distance ($\mathbf{r}_{\text{mp}}$) in dynamic solar wind. Timeshift adjustments were incorporated into temporal datasets using both simplified and OMNI techniques, where the need for synchronisation was emphasised. Despite the challenges involved in modelling dynamic time effects, these results demonstrate that TCN – based architectures are capable of capturing these complex temporal relationships accurately and can produce interpretable predictions in real-world scenarios. By using these models on actual solar wind

data, we were able to extend their interpretability to real-world issues, such as forecasting the magnetopause standoff distance (R_{mp}) in dynamic solar wind. Both OMNI and simplified methods were used to incorporate timeshift adjustments into temporal datasets, with a focus on synchronization. This guarantees that the experiments are authentic sequential learning tasks that require explicit modeling of temporal dependencies, rather than being reduced to static regression on pre-shifted inputs. These results show that TCN-based architectures can efficiently capture these intricate temporal relationships and generate interpretable predictions in real-world scenarios, despite the difficulties associated with modeling dynamic time effects. While our evaluation extended beyond basic error measures by incorporating AED, weighted MSE, top-k feature differences, and cosine similarity, additional metrics such as Kendall's tau or temporal correlation measures could further enrich the assessment of attribution ranking and temporal coherence in future work.

In general, this chapter presents evidence for the use and applicability of machine learning models in time-dependent tasks and stresses the importance of interpretability techniques for the understanding and improvement of such systems. This foundation can be built upon in future work that investigates more complex temporal structures and improves the interpretability of the metrics for increased insight.

Chapter 8

Conclusion

‘We are what we repeatedly do. Excellence, then, is not an act, but a habit.’ - Aristotle

8.1 Overview

In this dissertation, we investigated the challenges and methodologies related to the interpretability of DL models for time series data. We now summarise what we did (Section 8.2) and what we found (Section 8.3). In Section 8.4, we address which of our objectives were met and how, and finally, in Section 8.5, we discuss expansions on this work that may be useful.

8.2 Summary

The temporal dynamics typical of time series models introduce additional challenges to their interpretability. To better understand the application of interpretability techniques to time series data, an interpretability testbench was designed and created. The process included the following:

- A synthetic data generator was created to generate data for diverse scenarios, with control

of the time series characteristics, including variable interactions, noise levels, and temporal dependencies. Doing this allows the building of datasets for testing interpretability methods, which may be benchmarked.

- We developed a quantitative evaluation framework to evaluate the accuracy of feature attribution techniques, demonstrated using G-DeepSHAP. However, this framework applies to any feature attribution technique, as it compares the feature attribution results to the ASVs as ground truth attribution values. In this approach, an evaluation method is proposed for systematically evaluating attribution accuracy using a AED as described in Section 6.4.1. This method quantifies the discrepancy between computed feature attributions and the ground truth attribution values, providing a structured framework for evaluating interpretability methods. This approach enables a direct comparison between different attribution techniques under controlled conditions using the mean MSE.
- We implemented two architectures, a TCN and TS-MLP, and trained them with synthetic datasets to explore the interpretability of time series models under different conditions.
- The synthetic test bench generator was integrated into existing Know-It components, forming an end-to-end pipeline for data generation, model training, and feature attribution. An evaluation method was also included.
- We evaluated the feature attribution method G-DeepSHAP to assess its ability to provide feature interpretability and to identify some of the factors that influence results.
- Finally, the proposed method is demonstrated through a real-world application of forecasting solar storm events to indicate practical applicability.

Collectively, these contributions offer a systematic scheme to assess and improve the transparency and reliability of DL models for time series analysis.

While prior works have explored different interpretability test benches, as summarised in Table 2.1, our approach introduces a flexible data generation process, evaluates attribution-based techniques with a quantitative metric, and integrates the evaluation of explanations as part of an end-to-end pipeline. This design enables the evaluation of interpretability methods in time series models.

8.3 Key findings

- **Synthetic time series data:**

- The synthetic data generation process could successfully produce trending, seasonal, and non-linear interaction effects to model complex temporal phenomena, including phase shifts and lead-lag relationships.
- An important observation was that synthetic data properties must be aligned with the properties of real-world applications if a meaningful evaluation of interpretability techniques is to be performed. Greater differences in the evaluation of interpretability techniques are observed when comparing realistic data against overly simplistic data.

- **Interpretability techniques:**

- The choice of reference values had the largest influence on the reliability of Shapley values. Meaningful attributions depend on producing appropriate reference values.
- Many attribution techniques are computationally efficient but sensitive to feature scaling, as shown with G-DeepSHAP. G-DeepSHAP assigns attributions based on feature magnitudes, so rescaling features can distort attributions relative to the expected theoretical values. To ensure consistency, the corresponding equations should also be scaled accordingly. This draws attention to an essential question that is often neglected in the literature, namely whether the model learns the true theoretical equation or the scaled variation.
- The sample size was found to be another important factor that affects the stability and accuracy of Shapley value calculations. Particularly in datasets with complex dynamics, a larger sample size is needed.

- **Accuracy of interpretability techniques:**

- Inconsistencies between feature importance and dynamic time-dependent effects in complex datasets were revealed through interpretability analysis, leading us to believe that standard attribution methods will not capture nuanced temporal dependencies.

8.4 Evaluation against research goals

The research directly addressed the questions outlined in Section 1.4 and demonstrated progress in resolving the challenges posed by the problem statement (Section 1.2). The responses to each question are detailed below:

1. What aspects (such as temporal dependence, seasonality, or stationarity) should be considered when examining interactions in time series?
 - Key time series characteristics, such as temporal dependencies, trends, seasonality, and noise, were systematically modelled in the synthetic datasets as discussed in Section 5.2.3.
 - The inclusion of time delays between variables allowed the study of lagged interactions, while configurable noise levels provided insights into robustness under real-world conditions, as shown in Section 6.2 and Sections 7.2 to 7.4.
 - The synthetic data generator enables controlled experiments to analyse feature attribution and its contributing factors, as demonstrated in Section 5.3.
2. What criteria would a test bench need to adhere to in order to be valid for testing interpretability techniques?
 - The test bench incorporated modular components, such as a **Data Generator** (a module created for generating synthetic data), **Attribution Generator** (module used to generate the attribution values for features), and **Evaluator** (a module used to compare different attributions), ensuring flexibility and scalability for testing diverse DL architectures and interpretability methods, as described in Chapter 4.
 - Validity of attribution values was ensured using ASVs, allowing quantitative comparisons of attribution accuracy under different conditions, as shown through Sections 6.2, 7.2 and 7.3.
 - The system was designed with key considerations: it needed to process synthetic datasets with well-defined properties to allow controlled experiments, scale to different levels of time series complexity to support diverse use cases, and ensure reproducibility so that results could be reliably validated and compared, as illustrated in Section 6.3.

3. How can a platform be designed to test the accuracy of time series interpretability techniques?

- The test bench was designed as an end-to-end system encompassing data generation, model training, feature attribution, and feature attribution evaluation as described in Chapter 4.
- Accuracy testing used ASVs in order to compare the estimated attribution against the ground truth shown in Section 6.2.6. G-DeepSHAP was the chosen attribution method to demonstrate this process.
- Two key architectures commonly used for time series regression modelling, namely TCNs and TS-MLPs, were developed and integrated into an end-to-end pipeline to analyse interpretability under different temporal conditions discussed and illustrated in Sections 7.2 and 7.3.

4. What practical considerations influence the accuracy of the evaluation process?

- Noise levels and temporal dependencies in the datasets significantly influenced the performance of attribution methods shown in various experiments by comparing Sections 6.2, 7.2 and 7.3.
- Model complexity, choice of reference values, scaling of variables, and the number of sample points were critical factors influencing attribution accuracy and were discussed in Section 6.3.
- Previous studies did not emphasise the need to test datasets with varying characteristics, a gap that was addressed in this work. This means that previous approaches did not systematically evaluate how different dataset properties, such as noise levels or temporal dependencies, affect attribution methods. The proposed framework explicitly allows for these variations to be considered, ensuring a more robust evaluation as outlined in Section 8.2.

5. How is a time series test bench itself evaluated?

- The accuracy of the attributions estimated by the test bench was evaluated by its ability to provide reliable ASV comparisons, modular integration with existing frameworks, and reproducible results, as seen in Sections 6.2.6, 7.2, and 7.3.

-
- The application to solar wind data provided the preliminary validation of the utility of the test bench, as this is a more realistic application using real-world data, as shown in Section 7.4.
 - Potential limitations of the test bench include the way temporal patterns or dependencies are represented in the data, as predefined structures may not fully capture real-world complexity. While dataset size itself is not a direct limitation (as arbitrarily large datasets can be created), the effectiveness of evaluations may depend on having sufficient data to assess attribution accuracy reliably, and the computational resources to create ASVs.

8.5 Future Work

In this study, we provide preparatory work for the interpretation of time series model prediction, using synthetic data. There are many places in which to build upon this foundation.

Firstly, better generalisation to real-world applications would be achieved if the realism of the synthetic data generation is improved. In some cases, this may involve increasing the complexity of the noise model (for example, by introducing more complex noise models and irregular patterns to mimic an actual economy or biomedical signal acquisition environment). Dataset generation can be expanded to include spatiotemporal or multi-scale phenomena in order to determine how interpretability techniques perform on these datasets.

Another important direction is the improvement of the computational efficiency of the test bench. Evaluation of larger datasets and more complex models can be done in a computationally efficient manner if computing frameworks are implemented to be distributed. This would enhance the scalability of interpretability research for broader benchmarking and comparison.

This research can be extended to examine additional attribution methods. The research evaluated multiple Shapley-based strategies based on their precision features alongside system stability and processing speed. Evaluation of attribution results depended on which reference values researchers used: either average values or samples from random distributions. The research needs to investigate new methods for reference selection together with interpretability performance assessment in scenarios with varying dataset complexities. Additional research can

explore alternative interpretability approaches beyond Shapley methods, such as saliency maps, LIME, and attention mechanisms.

The testing process for the test bench could progress through real-world validation steps. Generalisation testing may involve applying the system to multiple data types, including climate, financial, and industrial monitoring datasets, to verify its practical use. Working with domain experts can help identify suitable interpretability approaches for specific applications. Using models that exhibit interpretability, rather than relying solely on post hoc methods, could further enhance the practical implementation of interpretability approaches in use cases.

New evaluation metrics can be designed to measure fairness together with uncertainty and trustworthiness in interpretability research. These techniques will gain broader usability in different sectors when assessment methods are aligned to human-focused targets of AI. Extending the test bench to include image and speech processing domains alongside time series data will help encourage research collaboration and openness within different fields.

Lastly, there is another avenue of improvement focused on further modelling techniques. Specific limitations to attribution accuracy, dataset scalability, and the ability to capture complex temporal dependencies require attention-based mechanisms and causal inference methods. Hybrid approaches that combine Shapley values with probabilistic or Bayesian methods may provide additional robustness and accuracy. It would also allow for a more comprehensive interpretability assessment for other types of model architectures.

This research strengthens the interpretability assessment framework in the Know-It toolkit by testing interpretability methods on real-world datasets, guided by specific research questions. The assessment methodology for feature attributions gets improved through the implementation of mean MSE in combination with other evaluation metrics. The ongoing work could further develop this framework to enhance the reliability and operational capabilities of interpretability methods within deep learning-based models.

8.6 Concluding remarks

Interpretability of DL models for time series was the critical challenge addressed in this dissertation. Through the construction of a synthetic test bench, the generation of controlled datasets,

and the evaluation of feature attribution techniques, this dissertation provides a framework to systematically evaluate interpretability methods.

We drew several important conclusions from our findings about the potential of synthetic test benches as a foundational tool for benchmarking and improving interpretability techniques. Although it was shown that a technique such as G-DeepSHAP works under controlled conditions; sensitivity to baseline values and noisy data can reduce accuracy. In addition, we applied the test bench to real-world solar wind data to demonstrate its relevance beyond synthetic datasets.

This dissertation provides contributions to the emerging field of interpretable AI and DL time series modelling in general. We presented methodologies and findings that can assist in making AI systems more understandable and trustworthy in different domains.

Appendix A

Supplemental content

We provide supplementary results related to evaluating feature attribution values and the use of KnowIt in the evaluation process

A.1 Shapley value calculation

Since Shapley's values are instrumental to this study, we provide an in-depth understanding of these values.

Calculation process

The calculation process is as follows:

1. **Permutations:** When calculating the Shapley value for a given player, all alternate player orderings or permutations are considered. This explains the many techniques for adding players to the group.
2. **Marginal contributions:** For each permutation, the marginal contribution of each participant is computed. This contribution depicts the additional value that a player brings to the group when they join, compared to the group's worth without them.

-
3. **Average contribution:** By analyzing all permutations, the average marginal contribution of each participant is computed. This average shows the player's overall contribution across all subgroups offered.
 4. **Shapley value:** An individual player's Shapley value is computed by summing their average marginal contributions across all permutations. The player should receive an appropriate part of the overall value.

The Shapley value is the only attribution technique that fits the efficiency, symmetry, dummy, and additivity requirements, all of which may be used to establish fair compensation.

- **Efficiency:** that the feature contributions add up to the difference of the forecast for x and the average.
- **Symmetry:** In the case that j, k contribute equally to all conceivable nondegenerate coalitions then their contributions should be the same.
- **Dummy:** To any coalition of feature values compatible with the projected value, we should include a feature j with a Shapley value of 0 if it is not included in any coalition which does not change the projected value.
- **Additivity:** It makes sure that, for a feature value, you can compute the Shapley value for each tree separately, averaging them, and obtain the Shapley value for the random forest's feature value.

The following graphic provides an intuitive understanding of the Shapley value: The random order of the feature values enter the room. Each feature value in the room plays the game (i.e., contributes toward the prediction). The Shapley value of a feature value is the average increase in prediction that the coalition that the feature value joined receives when the feature value joins them.

A.1.1 Case study: Shapley value applied to two variables

It is a story, in which you (Player 1) and your friend (Player 2) late into the night together trying to finish some coding for a contest. Eventually, you win the R10,000 first prize. Now you

want to settle the money fairly. Your friend says that you should just split it equally. So you have more hyperparameter tweaking abilities. You believe you are due more because you have put in more to the team. With this in mind, how do you get the money fairer?

Your friend has a time machine, conveniently. Odds are, each of you sends yourself back in time and re-does the coding contest, this time solo player 1) and a friend (Player 2) enter a coding contest. You end up winning the first prize of R10,000. Now, you want to split the money fairly. Your friend suggests that you just split it equally. Your hyperparameter tweaking abilities, on the other hand, are greater. You feel you are entitled to a greater share since you have contributed more to the team. How can you distribute the money fairly with this in mind?

Conveniently, your friend owns a time machine. You each go back in time and redo the coding contest – this time alone. So you come second and win R7,500. You come third and get R5,000. None of us would make any prize money (R0) if none of you played. Below we record the value of these 4 coalitions of the players. You clearly should get more of the prize money but not sure how to distribute it.

$$C_{12} = R10,000 \tag{A.1}$$

$$C_1 = R7,500 \tag{A.2}$$

$$C_2 = R5,000 \tag{A.3}$$

$$C_0 = R0 \tag{A.4}$$

A way of doing this is to find the expected 'marginal contribution' of each player. It is this weighted average of a player's contributions to all the underlying coalitions that a player could belong to.

Player 1 (P1) may join a coalition of Player 2 (P2). P2 moves up three places from third to first and the prize money is boosted by R5,000.

Alternatively, P1 may join a coalition of no players with no increment, and so stay at R7,500.

These are P1's marginal contributions to the project. The average of them yields a marginal contribution of R6 250.

$$C_{12} - C_2 = R10,000 - R5,000 = R5000 \quad (\text{A.5})$$

$$C_1 - C_0 = R7,500 - R0 = R7,500 \quad (\text{A.6})$$

$$(R5,000 + R7,500)/2 = R6,250 \quad (\text{A.7})$$

We may use a similar method to compute P2's expected marginal contribution. The value is R3,750 this time. P1 will receive R6,250, while P2 will receive R3,750. These are also referred to as Shapley values. Take note that the values add up to R10,000 in total reward money.

A.1.2 Case study: Shapley value with three features

With reference to the previous example, this time we have a team of 3 players. The calculations are complicated and we'll have to go back in time a few more times. There will now be 8 possible coalitions seen below. Together you win first prize (R10,000). There are now 3 coalitions of 2 players. For example, a coalition of P1 and P3 would come second (R7,500). There will also be coalitions of 1 Player. For example, if P3 played alone they would not make any prize money (R0). Perhaps they should have invested in a better GPU.

$$C_{123} = R10,000 \quad (\text{A.8})$$

$$C_0 = R0 \quad (\text{A.9})$$

$$C_{12} = R7,500 \quad (\text{A.10})$$

$$C_{13} = R7,500 \quad (\text{A.11})$$

$$C_{23} = R5,000 \quad (\text{A.12})$$

$$C_1 = R5,000 \quad (\text{A.13})$$

$$C_2 = R5,000 \quad (\text{A.14})$$

$$C_3 = R0 \quad (\text{A.15})$$

From the coalition values we can calculate the Shapley value of P1. P1 can join 4 coalitions now. P1 can form a coalition consisting of P2 as well as P3, of P2 alone, P3 alone, or of no

players. We repeat calculations of the marginal contribution of P1 to each of these coalitions. We finally take the weighted average. Thus our Shapley value for R5,000.

$$C_{123} - C_{23} = R10,000 - R5,000 = R5,000 \quad (\text{A.16})$$

$$C_{12} - C_2 = R7,500 - R5,000 = R2,500 \quad (\text{A.17})$$

$$C_{13} - C_3 = R7,500 - R0 = R7,500 \quad (\text{A.18})$$

$$C_1 - C_0 = R5,000 - R0 = R5,000 \quad (\text{A.19})$$

$$R5,000 * (1/3) + R2,500 * (1/6) + R7,500 * (1/6) + R5,000 * (1/3) = R5,000 \quad (\text{A.20})$$

The problem in this is, where do we get these weights from? That's why we give the first marginal contribution by 1/3, the second by 1/6, and so on. So these are the probabilities P1 makes that specific contribution. We get an expected marginal contribution by weighting by probabilities.

These probabilities aren't obvious where they come from. First, we wish to know how many ways 3 people can come together to form a coalition. Not only do you only win R10 000 if all 3 work together, but the actual R10 000 prize money means you can only win it if they all work together.

We make this assumption that each member enters sequentially with the same probability. For example, P1; then P3; then P2. Thus we have a total of $3! = 6$ ways of building the coalition. To put it generally, there are $n!$ ways that a team of n people can exist.

We noted that if P1 joins a coalition of P2 and P3, the marginal contribution would only be R5000. This can happen in two ways. P2 and P3 then P1 or P3 and P2 and P1. We can say, in other words, that for two of the 6 ways in which the team can form (P1 will do this marginal contribution). Now we have a 1/3 probability that P1 makes this contribution.

The only way P1 can make the second contribution (R2,500) is it. That is, if P2 joins, then P1. That means our probability is 1/6. The third (R7,500) contribution has a probability of 1/6 too. The fourth contribution (R5000) has a probability of 1/3. There are two ways P1 can make it, like with the first contribution. P1 joins first (or later P1 joins).

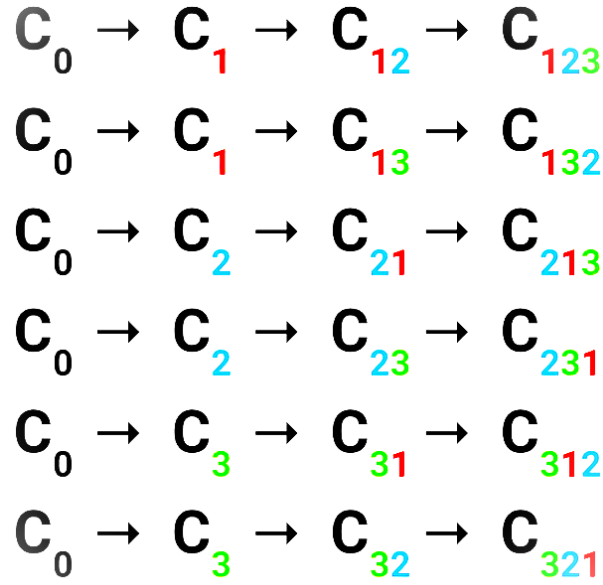


Figure A.1: Six ways of forming a three-player coalition

For P2 and P3 we can follow the same process. In fact, the Shapley values for these players come out to be R3,750 and R1,250 respectively. The Shapley values are again all equal to the total prize money. Because all the prize money will be divided fairly, Shapley values will always do so.

A.2 Model configuration

Essential parameters are included in a model configuration, which defines how to set up and structure the regression model. It extends to other architectures (We noted that if P1 joins a coalition of P2 and P3, the marginal contribution would only be R5000. This can happen in two ways. P2 and P3 then P1 or P3 and P2 and P1. We can say, in other words, that for two of the 6 ways in which the team can form (P1 will do this marginal contribution). Now we have a 1/3 probability that P1 makes this contribution.

The only way P1 can make the second contribution (R2,500) is it. That is, if P2 joins, then P1. That means our probability is 1/6. The third (R7,500) contribution has a probability of 1/6 too. The fourth contribution (R5000) has a probability of 1/3. There are two ways P1 can make it, like with the first contribution. P1 joins first (or later P1 joins).

For P2 and P3 we can follow the same process. In fact, the Shapley values for these players come out to be R3,750 and R1,250 respectively. The Shapley values are again all equal to the total prize money. Because all the prize money will be divided fairly, Shapley values will always do so (TS-MLP and TCN), while being adaptable to different setups.

A.2.1 Identification parameters

The necessary identifiers for each experiment instance and model instance are assigned to give easy reference and reproducibility. For example, `experiment_name` and `model_name` are used for attributing to each experimental run and type of model.

A.2.2 Data parameters

Key data handling parameters include:

- **Dataset and task:** Defines the dataset name, and task type (`regression`) for instance.
- **Input and output components:** Input and output features (e.g., `x_1`, `x_2`, `x_3`) and chunking over time sequenced data (input chunk `[-4, 0]` and output chunk `[0, 0]`) are specified.
- **Data splitting and scaling:** The data is split into training testing and validation portions, commonly at a 60-20-20 percent ratio. This is scaled like zero one normalization to make it consistent.

A.2.3 Architecture parameters

In addition, 'spec' is indicated using `arch`, and denotes the selected model (i.e., the TS-MLPs or TCNs) TS-MLP and TCN :

- **Architecture type:** It is specified with `arch` specifying which model is chosen (i.e., TS-MLP or TCN).

-
- **Hyperparameters:** Key settings: layer width, dropout rate, and architecture-specific parameter. Let us use `width` of 4092 and `dropout` of 0.0 in TS-MLP, or TCN with the corresponding settings.
 - **Architecture type:** Specified using `arch`, indicating the chosen model (e.g., TS-MLP or TCN).
 - **Hyperparameters:** Key settings include layer width, dropout rate, and architecture-specific parameters. In the example, TS-MLP is configured with a `width` of 4092 and a dropout rate of 0.0.

A.3 Training configuration

Parameters of the learning process that determine how well the model generalizes are controlled through the training configuration.

A.3.1 Optimisation parameters

- **Loss function:** For regression tasks `mse_loss` is used as the primary loss function by the model.
- **Optimiser:** We pick the Adam optimiser and decide on learning rates, that can adjust the pace of learning.
- **Learning rate and epochs:** The learning rate is stuck to 0.001, and the model is trained for 150 epochs, tolerating a certain level of training thoroughness along with computational efficiency.

A.3.2 Batch and data split parameters

- **Batch size:** Through 32 samples per training batch we take care of effective gradient updates and memory management.
- **Data split Ratios:** The data is split into parts of 60% for training, 20% for validation, and 20% for testing, so metric evaluation for model performance is complete.

-
- **Scaling method:** Specifically zero-one scaling, which we apply to the input data when we want them to be consistent before feeding into the model.

This configuration makes for a purely clear and repeatable setup for model training, including all parameters necessary for directing the model's training.

A.4 Synthetic time series data generation configuration

In the following appendix, we describe in detail how the configuration file that this tool generates for the complex time series data is structured. The configuration allows the user to control how each feature is characterised, the underlying mathematical characteristics that go into the Light field, and any time shifts or transformations.

- **num_features:** is this; it provides a glimpse of the number of features (or variables) to be generated in the time series. That is, three features are defined.
- **num_samples:** on the number of data samples (time steps) to set (on the training data generator). To fully analyse, we generate a 10,000 samples modest time series.
- **time_index_frequency:** Frequency defines how DateTime string formatting will be used to define the frequency of the time index. We have stated here that we care about our time in 1S intervals (once every second).
- **defaults:** It specifies per feature generation what mean (**mean**) and standard deviation (**std**) to us. These are applied by default on everything that is not uniquely referenced to the feature class.
- **mean:** set the default mean value; 0.
- **std:** Set to Default standard deviation = 1.
- **features:** Each of these features takes a mathematical expression, a lag value, and optionally a noise standard deviation (**noise_std**). Each feature is created using a custom expression: it can be made from a built-in mathematical function or contains random noise.

-
- **x_1:** A custom expression is used to produce those with low amplitude seasonality and noise. A 4 time step lag is used.
 - **x_2:** A periodic component, lag of 3-time steps, custom expression with noise.
 - **x_3:** They all include higher amplitude seasonality and noise, and another noise component with coefficient `noise_std`. A 1 time step lag is applied.
 - **equation:** It produces output data by building shifted versions of features via custom equations. Then we can use Python's `eval()` function to evaluate these equations in order to do dynamic feature manipulation.
 - **expression1:** It combines `x_1` with a shift of 4 time steps.
 - **expression2:** Moves `x_2` 3 time steps.
 - **expression3:** `x_3` is shifted 1 time steps.
 - **transfer:** Applicable further transformations after the `equation` creation. Each feature is used as output from an equation to construct a transfer expression, where the relationships between these features may be complex.
 - **transfer1:** Scale and exponentiate `expression1` and `expression2`.
 - **transfer2:** This implements a more complex transformation, doing a nonlinear divide of the `expression1` by both `expression2` and `expression3`.

It provides flexible and powerful synthetic time series generation with heterogeneous temporal dynamics, noise and interaction of features. Simple trends to advanced nonlinear behaviour can be investigated in customisable settings across the gamut and these can be simulated to generate synthetic datasets that meet researchers' exact experimental needs.

Default parameter values for TCN and TS-MLP architectures

In this section, the default parameter values shown in experiments for TCN and TS-MLP are described in detail. We specifically selected these values to standardise training and evaluation, such that training and evaluation behave identically. Extra details are given for scaling and related preprocessing techniques where applicable, as well as scaling and interpretability.

Model architectures

TCN

- **Kernel size:** 3
- **Number of layers:** 2
- **Number of filters:** 32
- **Dropout Rrte:** 0.1
- **Pooling:** Max pooling with a size of 2
- **Activation function:** ReLU
- **Weight initialisation:** Xavier Normal
- **Batch normalisation:** Applied after each convolutional layer
- **Optimiser:** Adam
 - **Learning rate:** 1×10^{-3}
 - **Weight decay:** 1×10^{-5}
- **Loss function:** Mean Squared Error (MSE)

TS-MLP

- **Hidden layers:** 2
- **Number of neurons per layer:** [64, 32]
- **Activation function:** ReLU
- **Dropout rate:** 0.2
- **Optimiser:** Adam
 - **Learning rate:** 5×10^{-4}
 - **Weight decay:** 1×10^{-5}
- **Loss function:** Mean Absolute Error (MAE)

Scaling

- **Scaling Method:** Z-Score Normalization
 - Formula: $z = \frac{x-\mu}{\sigma}$
- **Applied to:** Both input and output features by default.
- **Scaling tag:** full (scales input and output together).
- **Alternative scaling method:**
 - Min-Max Scaling (0, 1) — Supported but not used in default configuration.

Interpretation methods

Shapley values

- **Method:** DeepLiftShap
- **Number of baselines:** 90
- **Baseline generation:**
 - Drawn from the same dataset as the prediction points.
 - Excludes prediction point IDs to prevent overlap.

Reference baseline values

- **Mean:** The default reference for interpretation.
- **Alternative Reference Options:**
 - Zero baseline.
 - Random baseline, sampled from dataset features.

Training parameters

- **Max epochs:** 100

-
- **Batch size:** 64
 - **Gradient clipping:**
 - **Value:** 0.5
 - **Algorithm:** Norm-based clipping.
 - **Early stopping:**
 - **Patience:** 10 epochs (validation performance plateau).

Preprocessing and dataset preparation

- **Prediction point dynamics:**
 - **Input chunk:** $[-10, 0]$ for TCN, $[-5, 0]$ for MLP.
 - **Output chunk:** $[1, 5]$ for TCN, $[1, 1]$ for MLP.
- **Splitting method:**
 - Chronological (default).
 - Train/validation/evaluation Split: 60%/20%/20%.
- **Handling missing values:**
 - Default: Leave as is.
 - Alternative: Interpolate with the `linear` method.
- **Padding:**
 - **Default method:** Zero Padding.

This appendix ensures that the default values and configurations for the TCN and TS-MLP architectures, along with relevant preprocessing and interpretation techniques, are explicitly documented for reproducibility and clarity in experimental setups.

Bibliography

- [1] C. Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Lulu.com, 2020, ISBN: 978-0244768522.
- [2] J. Brownlee, “Multivariate time series forecasting with lstms in keras,” *Machine Learning Mastery*, 2017, Published: October 21, 2017. Accessed: June 26, 2024. [Online]. Available: <https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/>.
- [3] A. A. Ismail, M. Gunady, H. C. Bravo, and S. Feizi, “Benchmarking deep learning interpretability in time series predictions,” in *Advances in Neural Information Processing Systems (NeurIPS 2020)*, 2020. DOI: 10.48550/arXiv.2010.13924.
- [4] M. Joseph, *Modern Time Series Forecasting with Python: Explore industry-ready time series forecasting using modern machine learning and deep learning*, 1st. Birmingham, UK: Packt Publishing Ltd, 2022, ISBN: 978-1803246802.
- [5] S. J. Prince, *Understanding Deep Learning*, 1st. Cambridge, MA: MIT Press, 2023, ISBN: 9780262048644.
- [6] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Melbourne, Australia: OTexts, 2018, ISBN: 978-0987507112.
- [7] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023, ISBN: 9781009389433.
- [8] M. Robnik-Šikonja and M. Bohanec, “Perturbation-Based Explanations of Prediction Models,” pp. 159–175, 2018. DOI: 10.1007/978-3-319-90403-0_9.
- [9] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017, pp. 4765–4774. DOI: 10.48550/arXiv.1705.07874.

-
- [10] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, “Explainable AI: A review of machine learning interpretability methods,” *Entropy*, vol. 23, no. 1, p. 18, 2020. DOI: 10.3390/e23010018. [Online]. Available: <https://doi.org/10.3390/e23010018>.
- [11] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, *et al.*, “Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai,” *Information Fusion*, vol. 58, pp. 82–115, 2020. DOI: 10.1016/j.inffus.2019.12.012.
- [12] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021. DOI: 10.1016/j.ijforecast.2021.03.012.
- [13] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017, pp. 5998–6008. DOI: 10.48550/arXiv.1706.03762.
- [14] H. Chen, S. M. Lundberg, and S.-I. Lee, “Explaining a series of models by propagating shapley values,” *Nature Communications*, vol. 13, no. 1, p. 4512, 2022. DOI: 10.1038/s41467-022-31384-3.
- [15] D. Barić, P. Fumić, D. Horvatić, and T. Lipić, “Benchmarking attention-based interpretability of deep learning in multivariate time series predictions,” *Entropy*, vol. 23, no. 2, p. 143, 2021. DOI: 10.3390/e23020143.
- [16] Y. Kang, R. J. Hyndman, and F. Li, “Gratis: Generating time series with diverse and controllable characteristics,” *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 13, no. 4, pp. 354–376, 2020. DOI: 10.1002/sam.11461.
- [17] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*. Berlin, Heidelberg: Springer, 2005, ISBN: 978-3-540-27752-1.
- [18] X. Zhu, Y. Xiong, M. Wu, G. Nie, B. Zhang, and Z. Yang, “Weather2k: A multivariate spatio-temporal benchmark dataset for meteorological forecasting based on real-time observation data from ground weather stations,” in *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 206, PMLR, 2023, pp. 2704–2722. DOI: 10.48550/arXiv.2302.10493.
- [19] S. Feng, C. Miao, K. Xu, *et al.*, “Multi-scale attention flow for probabilistic time series forecasting,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 5, pp. 2056–2068, 2024. DOI: 10.1109/TKDE.2023.3319672.

-
- [20] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: A review,” *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019. DOI: 10.1007/s10618-019-00619-1.
- [21] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. DOI: 10.1207/s15516709cog1402_1.
- [22] A. Lapedes and R. Farber, “Nonlinear signal processing using neural networks: Prediction and system modelling,” Los Alamos National Lab, Tech. Rep. LA-UR-87-2662, 1987.
- [23] M. Shiblee, P. K. Kalra, and B. Chandra, “Time series prediction with multilayer perceptron (mlp): A new generalized error based approach,” in *Advances in Neuro-Information Processing: 15th International Conference on Neural Information Processing (ICONIP 2008), Auckland, New Zealand, November 25-28, 2008*, M. Köppen, N. Kasabov, and G. Coghill, Eds., ser. Lecture Notes in Computer Science, vol. 5507, Berlin, Heidelberg: Springer, 2009, pp. 37–44. DOI: 10.1007/978-3-642-03040-6_5.
- [24] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, vol. abs/1803.01271, 2018. DOI: 10.48550/arXiv.1803.01271.
- [25] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks: A unified approach to action segmentation,” in *European Conference on Computer Vision*, Springer, 2016, pp. 47–54. DOI: 10.48550/arXiv.1608.08242.
- [26] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, PMLR, 2017, pp. 3319–3328. DOI: 10.48550/arXiv.1703.01365.
- [27] A. Shrikumar, P. Greenside, and A. Kundaje, “Learning important features through propagating activation differences,” *arXiv preprint arXiv:1704.02685v2*, 2019. DOI: 10.48550/arXiv.1704.02685.
- [28] B. Kim, M. Wattenberg, J. Gilmer, *et al.*, “Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav),” in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 2668–2677. [Online]. Available: <https://proceedings.mlr.press/v80/kim18d.html>.
-

-
- [29] A. Binder, G. Montavon, S. Lapuschkin, W. Samek, and K. Müller, “On the quantitative analysis of deep convolutional neural network models,” *arXiv preprint arXiv:1608.08220*, vol. abs/1608.08220, 2016. DOI: 10.48550/arXiv.1608.08220.
- [30] A. Gevaert, A.-J. Rousseau, T. Becker, D. Valkenburg, T. De Bie, and Y. Saeys, “Evaluating feature attribution methods in the image domain,” *Machine Learning*, pp. 1–46, 2024.
- [31] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, “Sanity checks for saliency maps,” *Advances in neural information processing systems*, vol. 31, 2018.
- [32] L. Brocki and N. C. Chung, “Feature perturbation augmentation for reliable evaluation of importance estimators in neural networks,” *Pattern Recognition Letters*, vol. 176, pp. 131–139, 2023.
- [33] L. S. Shapley, “a Value for n-Person Games,” *Classics in Game Theory*, vol. 2, pp. 69–79, 2020. DOI: 10.2307/j.ctv173f1fh.12.
- [34] H. Turbé, M. Bjelogrić, C. Lovis, and G. Mengaldo, “Evaluation of post-hoc interpretability methods in time-series classification,” *Nature Machine Intelligence*, vol. 5, no. 3, pp. 250–260, 2023.
- [35] A. A. Ismail, M. Gunady, H. C. Bravo, and S. Feizi, “Supplementary material: Benchmarking deep learning interpretability in time series predictions,” *Supplementary Material*, 2020.
- [36] O. Koç, Ö. Uğur, and A. S. Kestel, *The impact of feature selection and transformation on machine learning methods in determining the credit scoring*, Jan. 2023. DOI: 10.48550/arxiv.2303.05427. [Online]. Available: <https://arxiv.org/abs/2303.05427>.
- [37] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, *Neural networks designing neural networks: Multi-objective hyper-parameter optimization*, Jan. 2016. DOI: 10.48550/arxiv.1611.02120. [Online]. Available: <https://arxiv.org/abs/1611.02120>.
- [38] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, “Temporal convolutional networks for action segmentation and detection,” Jul. 2017. DOI: 10.1109/cvpr.2017.113. [Online]. Available: <https://doi.org/10.1109/cvpr.2017.113>.
- [39] G. Varol, I. Laptev, and C. Schmid, *Long-term temporal convolutions for action recognition*, Jan. 2016. DOI: 10.48550/arxiv.1604.04494. [Online]. Available: <https://arxiv.org/abs/1604.04494>.
-

-
- [40] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, *A simple neural attentive meta-learner*, Jan. 2017. DOI: 10.48550/arxiv.1707.03141. [Online]. Available: <https://arxiv.org/abs/1707.03141>.
- [41] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018. [Online]. Available: <https://arxiv.org/abs/1803.01271>.
- [42] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, “Very deep convolutional neural networks for raw waveforms,” in *Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2017, pp. 421–425. DOI: 10.1109/ICASSP.2017.7952132.
- [43] C. Grant, “Knowledge discovery with additive attribution methods for geomagnetic index prediction,” Ph.D. dissertation, North West University, 2024. [Online]. Available: <https://repository.nwu.ac.za/handle/10394/42417>.
- [44] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017, pp. 4765–4774. DOI: 10.48550/arXiv.1705.07874.
- [45] W. A. Falcon, *Pytorch lightning*, <https://github.com/PyTorchLightning/pytorch-lightning>, Accessed: 2024-09-20, 2019.
- [46] N. Kokhlikyan, V. Miglani, M. Martin, *et al.*, “Captum: A unified and generic model interpretability library for pytorch,” *arXiv preprint arXiv:2009.07896*, vol. abs/2009.07896, 2020. DOI: 10.48550/arXiv.2009.07896.
- [47] R. J. Chen, M. Y. Lu, T. Chen, D. F. K. Williamson, and F. Mahmood, “Synthetic data in machine learning for medicine and healthcare,” *Nature Portfolio*, vol. 5, no. 6, pp. 493–497, Jun. 2021. DOI: 10.1038/s41551-021-00751-8. [Online]. Available: <https://doi.org/10.1038/s41551-021-00751-8>.
- [48] F. Bahrpeyma, M. Roantree, P. Cappellari, M. Scriney, and A. McCarren, “A methodology for validating diversity in synthetic time series generation,” *Elsevier BV*, vol. 8, pp. 101 459–101 459, Jan. 2021. DOI: 10.1016/j.mex.2021.101459. [Online]. Available: <https://doi.org/10.1016/j.mex.2021.101459>.
-

-
- [49] P. Srinivasan and W. J. Knottenbelt, *Time-series transformer generative adversarial networks*, Jan. 2022. DOI: 10.48550/arxiv.2205.11164. [Online]. Available: <https://arxiv.org/abs/2205.11164>.
- [50] J. E. Monogan, *Time series analysis*, Jan. 2015. DOI: 10.1007/978-3-319-23446-5_9. [Online]. Available: https://doi.org/10.1007/978-3-319-23446-5_9.
- [51] V. Cerqueira, L. Torgo, and C. Soares, *Machine learning vs statistical methods for time series forecasting: Size matters*, Jan. 2019. DOI: 10.48550/arxiv.1909.13316. [Online]. Available: <https://arxiv.org/abs/1909.13316>.
- [52] L. B. V. de Amorim, G. D. C. Cavalcanti, and R. M. O. Cruz, “The choice of scaling technique matters for classification performance,” *Elsevier BV*, vol. 133, pp. 109924–109924, Jan. 2023. DOI: 10.1016/j.asoc.2022.109924. [Online]. Available: <https://doi.org/10.1016/j.asoc.2022.109924>.
- [53] G. Plumb, D. Molitor, and A. Talwalkar, *Model agnostic supervised local explanations*, Jan. 2018. DOI: 10.48550/arxiv.1807.02910. [Online]. Available: <https://arxiv.org/abs/1807.02910>.
- [54] I. E. Nielsen, D. Dera, G. Rasool, N. Bouaynaya, and R. P. Ramachandran, “Robust explainability: A tutorial on gradient-based attribution methods for deep neural networks,” *Institute of Electrical and Electronics Engineers*, vol. 39, no. 4, pp. 73–84, Jul. 2022. DOI: 10.1109/msp.2022.3142719. [Online]. Available: <https://doi.org/10.1109/msp.2022.3142719>.
- [55] T. I. Gombosi, “The solar wind interaction with planetary magnetospheres,” in *Multiscale Coupling of Sun-Earth Processes*, P. Song, H. J. Singer, and G. L. Siscoe, Eds., Dordrecht: Springer, 2004, pp. 169–186. DOI: 10.1007/1-4020-2768-0_2.
- [56] R. J. Walker and C. T. Russell, “Solar-wind interactions with magnetized planets,” in *Introduction to Space Physics*, M. G. Kivelson and C. T. Russell, Eds., New York: Cambridge University Press, 1995, pp. 164–182, ISBN: 9780521451040.
- [57] N. Science, *Earth’s magnetosphere: Protecting our planet from harmful space energy*, <https://science.nasa.gov/science-research/earth-science/earths-magnetosphere-protecting-our-planet-from-harmful-space-energy>, Accessed: 2024-11-24.
- [58] N. G. S. F. Center, *Omni data documentation*, Available at https://omniweb.gsfc.nasa.gov/html/ow_data.html#time_shift, Accessed: 2024-11-24, 2023.
-