TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Resource Forecasting for Satellite Operations using Multivariate Time Series Data

Authors	Maria Hanna, Felix Mujkanovic, Guido Sasahara, Maresa Schröder, Oscar Axel Vargas Salomón
Mentors	Dr. Clemens Schefels, M.Sc. Leonard Schlag German Aerospace Center (DLR)
Co-Mentor	M.Sc. Philippe Sünnen
Project Lead	Dr. Ricardo Acevedo Cabra (Dept. of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Dept. of Mathematics)

Feb 2021

Abstract

In this project, we investigate multivariate time series forecasting methods for the purpose of estimating the future consumption of resources onboard satellites of the German Space Operations Center, with the ultimate goal of easing resource planning for satellite operators. This forecasting task requires the incorporation of past data, as well as commands sent to the satellites during the forecasting time frame. Multivariate time series forecasting is a continuously evolving area of research. While the field was traditionally dominated by purely statistical methods, a variety of modern machine learning methods have gained popularity in the past few years. However, only few of them are versatile enough to fulfill our three key requirements: 1) using multivariate series, 2) forecasting over a longer time horizon, and 3) leveraging known future inputs. We perform an in-depth literature survey of commonly employed as well as very specialized methods, resulting in the further examination of four promising approaches from different research domains. A comparison between the models and a statistical baseline method shows that two of the investigated models employing neural network architectures – Temporal Fusion Transformers and ANFIS – achieve the lowest forecast errors across different time series, history sizes and prediction lengths. Thus, we conclude that these two methods are best suited to support resource planning for the German Space Operations Center.

Contents

1 Introduction

1.1 Problem Definition and Goals of the Project

The German Space Operations Center (GSOC) at the German Aerospace Center (DLR) operates a number of low Earth orbit and geostationary satellites for Earth observation. These satellites fulfill tasks for both scientific and commercial stakeholders.
Satellites are characterized by limited resources shared among numerous instruments and tasks. In addition, resource depletion, for example an empty battery, can lead to loss of the satellite. Therefore, resources have to be carefully managed. However, real-time resource control is not feasible, as contact to the ground station is only possible for a few minutes every time the satellite passes over the same location for most of these satellites flying in low Earth orbit. This means that resource usage between those communication time windows has to be determined in advance. Furthermore, manual resource control by human operators would require significant manpower and attention. In this complex context, automated resource planning is crucial for smooth operation. Currently, the resource planning system deployed at GSOC is based mainly on physical simulations set up by experts. While this approach has proven to be successful in operation, it requires expert knowledge and careful tuning based on past experience with each particular type of satellite.
GSOC's satellites record more than 70,000 telemetry parameters, i.e. the values recorded by various instruments onboard the satellites, such as attitude determination components, allowing the opportunity to leverage this data. The main purpose of this project is the exploration of various data analysis techniques to investigate which methods may have potential to optimize resource forecasting for GSOC's satellites.
More specifically, we will use multivariate time series data from both telemetry parameters and command parameters, i.e. commands sent from the ground station that modify the satellites' behavior, to forecast the behaviour of individual parameters (resources) relevant for the satellites' operations. The minimum forecast horizon of three hours is given by the shortest possible time span between real-time communication slots with the satellite. We will research both traditional statistical and modern machine learning (ML) methods for multivariate input / univariate output time series prediction and evaluate them using task-specific benchmark data. The data we will operate on is retrospective data from one GSOC satellite collected over the span of around 3 years.
Major challenges we will have to overcome are meeting the requirement of conditioning the forecast on known future commands, which will be sent to the satellite during real-time communication, as well as the unusual representation of the command parameters as time series. Further challenges include our relative lack of insight into GSOC's satellite operations amplified by data secrecy requirements, the large number of possible algorithms and input parameters, the absence of a clear evaluation metric, and the relatively long time for which the forecast has to be sufficiently accurate.

1.2 Approach

As with any data project, we will first analyze the structure, shape, and statistics of the data. How does the data look? How are the samples distributed in time? How typical are

periods of missing data? Are all features available at all times? Are there outliers? Are there obvious correlations between features? Do the statistics already unveil important characteristics of the data?
Our answers to questions like these will in the next stage guide our comprehensive literature research and help us in deciding which time series forecasting methods are viable regarding the data, which performance evaluation metric is right for our task, whether we need additional data imputation, and so on. In the end, we aim to choose around five state-of-the-art techniques or combinations of techniques from various distinct sub-fields of ML for further investigation. By spreading our options and, for example, not only including narrow domain-specific approaches tailored to time series, but also general learners such as neural networks, we aim to maximize our chances in finding a well-performing predictor.
In the next stage, each of us will implement and evaluate one or two methods. This work will certainly not be independent, as many challenges in ML tend to consistently arise, including the need for data cleansing, data transformation to purely metric form, data imputation, and so on. We may also go back to the research stage in case we find our approaches are severely lacking performance. Based on this, we will document each method's general suitability to the task, our approach to its implementation, challenges that arose and our solutions to them, and finally its overall forecasting performance.

2 Data Exploration

In this section, we take a first look at the data and its structure. The full results can be found in the `data_exploration` notebook.

2.1 Data Set

We are provided with data from 1094 consecutive days from one GSOC satellite. While the satellites record more than 70,000 parameters, our data set contains only a small subset of 11 telemetry parameters and 13 command parameters, which have been identified by DLR experts as most relevant. In total, this results in around 150 million unique `(parameter value, time stamp)` pairs. Due to security reasons, the data set was anonymized by the DLR in advance.
Telemetry parameters are denoted by the prefix `Param`, while command parameters are denoted by `CParam`. The parameters are divided into logically related groups, denoted by the letter `T`.

2.2 Individual Parameter Behavior

From Figures 1 and 2 as well as basic statistical measures (number of samples, mean, range, standard deviation) and plotting of spectrograms, we can see that most of the non-command parameters share the same periodicity (likely the duration of an orbit) and are mostly stationary. The nominal ranges, means, and variances of all non-command parameters differ yet are close-by, even though some series contain singular outlier samples. Most the command parameters do not exhibit periodicity. Some of the command parameters have metric values, while others are of categorical nature, sometimes with few and

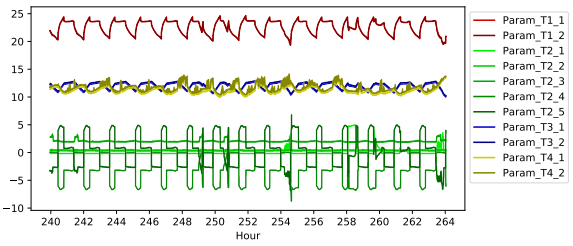


Figure 1: Plot of all non-command parameters over a single day.

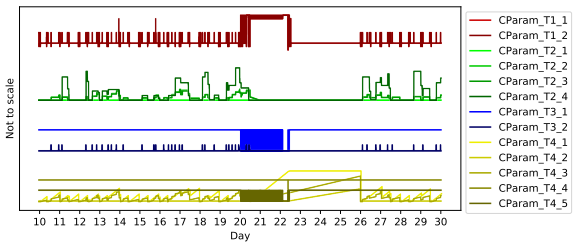


Figure 2: Plot of all command parameters over 20 days.

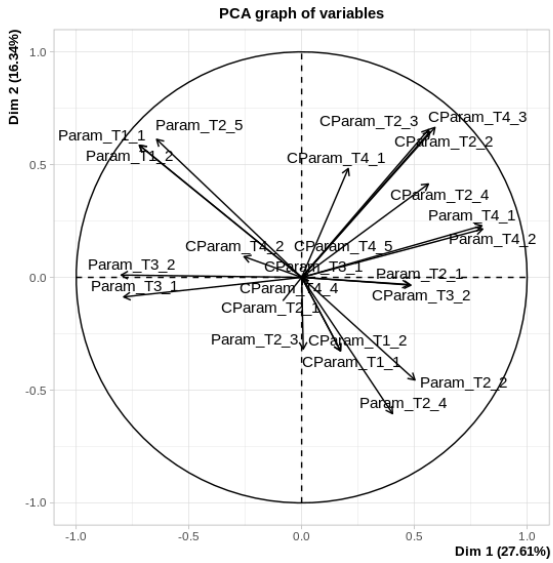


Figure 3: Principal Component Analysis of all parameters and commands.

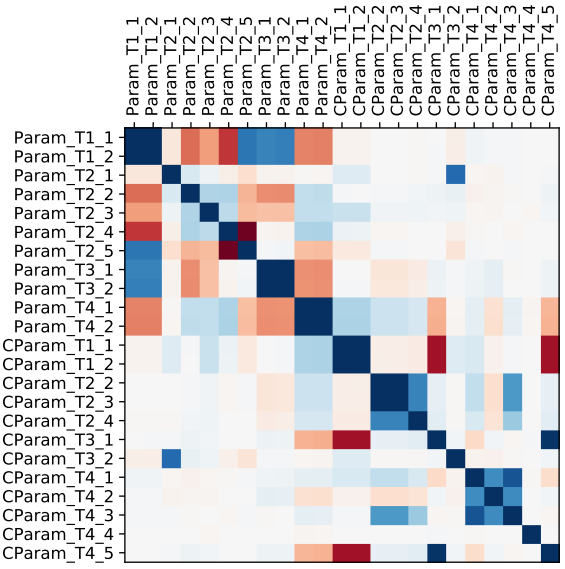


Figure 4: Pearson Correlations between the samples of pairs of time series. Positive correlations are blue, negative ones are red.

sometimes with a large number of states, whose ordinals sometimes also extend to very high numbers. As such, the nominal ranges, means, and variances of the command parameters differ greatly. Additionally, there are large differences in the number of samples among the command parameters. Finally, CParam_T2_1 is always 0 and can therefore be removed from our data set.

2.3 Parameter Interdependence

2.3.1 Principal Component Analysis

Figure 3 shows the results of a Principal Component Analysis performed jointly for both command and non-command parameters. The variance captured by the first two PCA dimensions is highly significant, indicating a meaningful result. We observe that some sharp cliques form, while other parameters are only loosely connected. We may assume that the following sets of parameters are strongly connected: non-command parameters T3_1 and T2_2; non-command parameters T1_1, T1_2, and T2_5; non-command parame-

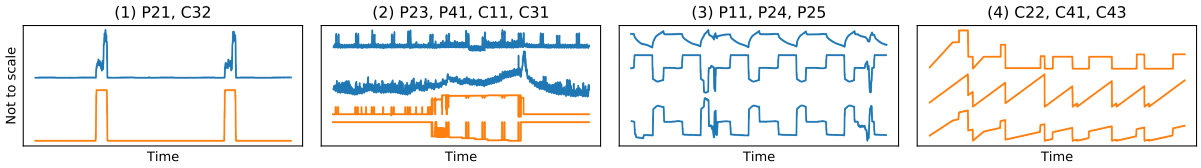


Figure 5: Exemplary correlated series. Non-command parameters are blue and command parameters are orange. The parameter names are listed from top to bottom.

ters T2_2 and T2_4 and command parameters T1_1 and T1_2; non-command parameters T4_1 and T4_2 and command parameters T2_2, T2_3, T2_4, and T4_3.

2.3.2 Pearson Correlation

Next, we computed the Pearson Correlations of a 100-day sub-series and plotted them (Figure 4). We find that a lot of parameters are redundant, visualized by the dark blue 2×2 squares along the diagonal. Param_T2_4/5 are even the inverse of each other. Apart from that, there are various correlations within the non-command and the command parameters, but there are only few strong correlations between the two groups.
We illustrate the correlations by plotting some of the correlated time series in Figure 5. In plot (1), we can see a command parameter and a non-command parameter that are highly correlated. Plot (2) shows two command parameters that are inversely correlated, and two non-command parameters that are correlated with both of the command parameters. In plot (3), the lower two non-command parameters are nearly inversely equivalent to each other. The top parameter is highly correlated, but it seems that there is no causation between the top two parameters, and instead the correlation is just caused by both parameters depending on the orbit seasonality. In plot (4), we see three command parameters that are somewhat correlated.

2.3.3 Auto-Correlation and Cross-Correlation

In the final step, we examined auto-correlations of the parameters and cross-correlations between the parameters and the commands. These statistics are an important tool to understand interdependence between time series.
We employ auto-correlation to detect periodicity. Non-command parameters T1_1, T1_2 and T2_1 do not show any auto-correlation. All other non-command parameters show auto-correlations on various lags. Especially non-command parameters T3_1 and T3_2 are continuously auto-correlated with decreasing trend up to lag 31. The non-command parameters T4_1 and T4_2 show a continuously decreasing auto-correlation up to lag 7.
As depicted in Figures 6-9, command parameter T2_4 exhibits strong periodic cross-correlation with non-command parameters T2_1, T2_3, T2_5, T4_1, and T4_2, which is highly correlated to T4_1. Other parameter pairs do exhibit only slight or no periodic cross-correlation with varying period lengths.
Summing up, we see that there are indeed immediate correlations in the data that should be quite easy for a machine learning model to learn. Some of them may be a result of underlying causation, while others are most definitely only correlations. We stop the

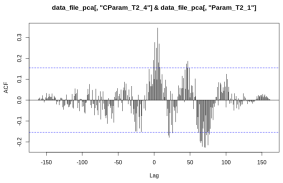


Figure 6: P21, C24

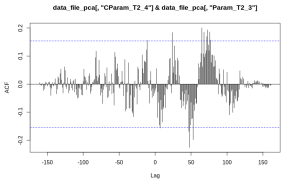


Figure 7: P23, C24

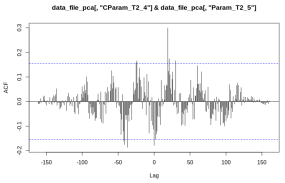


Figure 8: P25, C24

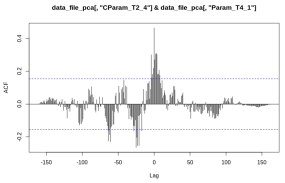


Figure 9: P41, C24

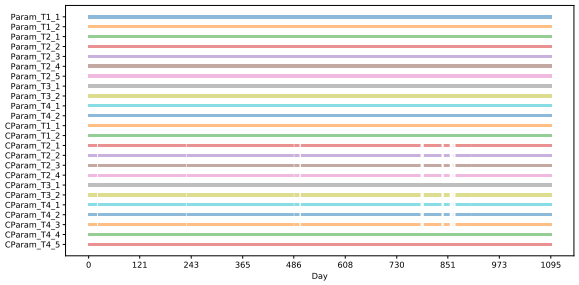


Figure 10: Sampled data points of all parameters over the whole time span.

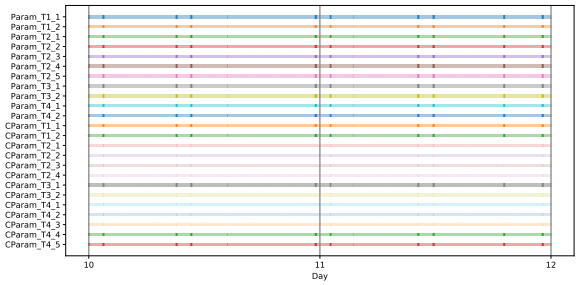


Figure 11: Sampled data points of all parameters over a span of two days.

correlation analysis at this point because further insights should be generated by the machine learning models, rather than manually acquired.

2.4 Data Sampling

Sampling rate is a key hyperparameter in time series analysis, and therefore requires careful selection. In order to make an informed decision, we first have to analyze how the data set is sampled, i.e., which intervals lie between consecutive observations.

2.4.1 Overview

To gain an overview of possible gaps in data, we visualize the sampled data points over the whole data set. As can be seen in Figure 10, there is a group of command parameters that has several larger gaps in the data.
A more detailed look at a shorter range of two days (Figure 11) clearly demonstrates that while all parameters vary in their sampling rate over time, the sampling rates across different parameters peak at similar times. Furthermore, we observe that the command parameters with the missing data are sampled less frequently in general than the other parameters.

2.4.2 Sampling Rates

In general, we observe that time stamps (and therefore the sample intervals) are characterized by some measuring variance, so discretization may be useful.
We can assign the parameters to four distinct groups regarding their sample frequency. We used these groups to gain a more detailed view on how the sampling intervals vary within this data set:

- Group 1 (all non-command parameters, CParam_T1, CParam_T4_4 and T4_5): Through discretization over the intervals, we clearly see that three main sampling rates are used: 30 seconds, 1 second, 0.5 seconds. Only 157 of around 50.000 intervals between consecutive samples differ.
- Group 2 (CParam_T2 and CParam_T3_2 through CParam_T4_3): The standard sampling interval for this group seems to be 78 seconds. A few times each day, data is sampled every 6 seconds.
- Group 3 (CParam_T3_1): The sampling times are largely similar to group 1. However, this parameter is sometimes sampled in very short succession, which differentiates it from the others.

From this analysis, we can derive some possible relations between the sample intervals and the commands:

- CParam_T3_1 only occasionally takes on values other than 4, and is the parameter that is occasionally sampled in rapid succession; this may suggest that the exact timing of the switch is of crucial interest, which may affect our choice of sample rate.
- The control parameters that can take many different values are all in the group that gets sampled less frequently (however, that sampling group also contains the binary CParam_T3_2).

In order to be able to make an informed decision on an appropriate choice of sampling rate, two questions may demand further exploration: First, we may have to further look into the relation between increased sample frequency and the telemetry parameters. Furthermore, deeper investigation may be required to confirm or disprove our assumed relation between sample frequency and command parameters.

2.5 Data Preparation

As discussed in Section 2.4, two main issues had to be resolved before we could start applying forecasting models to the data set: 1) the varying sampling rate and 2) gaps in the data. Furthermore, we had to decide on a history size (i.e., how much past data to use for a forecast) and a forecasting horizon (i.e., how long the forecast should be).
Different models require different history sizes for optimal performance, so we chose to try our models on two history sizes: 3 hours and 12 hours. Due to the relatively short periodic behavior of the command parameters, we expect that even longer history sizes would not benefit the forecasting performance and may instead lead to worse predictions due to overloading the models with irrelevant data.
The minimal acceptable forecasting horizon according to DLR requirements is 3 hours, which roughly equates to two orbits. Therefore, this was our chosen forecasting horizon.
We used a sliding window approach to determine how to partition the data into samples. We slide the window by the length of the forecasting horizon in each step: for a forecasting horizon of 3 hours, each window starts 3 hours apart. This guarantees that no time point is included in the test data more than once.

As some of our chosen models require the intervals between time steps to be constant, we had to resample the data to a constant sampling rate. After discussions with the DLR, we decided that no deeper investigation into the questions raised in Section 2.4.2 was necessary; selecting a reasonable constant sampling rate was sufficient.
As shown in Section 2.4.2, every parameter is commonly sampled at a rate of once every 30 seconds or more. Therefore, the lowest reasonable interval is 30 seconds, which is the first resolution we investigated. This rate requires forecasting 360 steps to obtain a forecast of 3 hours, which may be too difficult for some models. To account for this issue, we additionally work with sampling intervals of 180 seconds and 600 seconds, which greatly reduces problem complexity.
We applied linear interpolation to the real-valued parameters during resampling; however, categorical parameters required a different approach. As we assume any changes in those parameters to be relevant to the forecast, we apply the following strategy: If the value changes between two of the resampled time steps, we set the new value in the already first step to make sure that the models can recognize that this change happened before the second step. As we are not able to account for multiple changes between two time steps without introducing additional categories, we do not handle this case, but instead only use the first change.
Gaps in the data can cause inaccurate interpolation and therefore poor prediction performance. Therefore, they have to be removed. We decided to use 180 seconds as a threshold to determine a gap: if two samples are more than 180 seconds apart for any parameter, we do not use any samples from that time frame. Therefore, we obtain data that is not continuous, but has no gaps within each continuous (i.e., gap-less) interval.
We use the first 80% of the data for training and validation; the final 20% are held out as test set that is only used for performance evaluation at the end.

3 Survey of Forecasting Methods

Most traditional time series forecasting methods do not directly lend themselves to our special challenges, that is, (a) the multivariate data, (b) the exogenous variables (command parameters are available during the forecasting horizon), and (c) the long forecasting horizon. Hence, we conducted an extensive literature survey to identify methods from various research fields that are most suitable to our challenge. In the following, we will first present the most promising approaches from the literature and put them into context. Concluding this chapter, we then summarize the strengths and weaknesses of each model in Section 3.6.

3.1 Classic Machine Learning With Prior Data Transformation

This approach is the most straightforward one. We just interpret a fixed-size history of the time series as a feature vector for a traditional machine learning (ML) model such as k-nearest neighbors, decision tree, neural network, or Support Vector Machine (SVM). The model then forecasts a small, fixed-size forecasting horizon. We may apply the model iteratively to achieve any forecasting horizon. A straightforward extension is to use one or more transformations to transform the history to a fixed-size vector, and

then use that vector as the feature vector. There is an abundance of time series-specific transformations available in the literature, including the Wavelet, Shapelet, WEASEL, Tsfresh, and Catch22 transformations.
This approach trivially supports all of our special challenges. It uses proven and highly optimized traditional ML models and allows to easily incorporate research on time series feature extraction. Also, if the ML model is easy to interpret, the whole forecaster will be mostly easy to interpret. However, all this comes at the cost of not considering the temporal structure of the data, which is crucial to prediction accuracy. Therefore, we will not pursue this approach.

3.2 Deep Learning

Deep neural networks (DNNs) are multi-layered artificial neural networks aimed at learning abstract representations from data through an iterative optimization process called backpropagation. Due to their flexibility and prediction strength, they have become popular tools for many applications, one of which is time series forecasting [12]. Deep learning approaches to multi-step time series forecasting can generally be classified into two types [12]:

- **Iterative** methods predict one step at a time and use each output as an input to the next step. Typically, they use the same model weights in every step. This reduces computational complexity, but limits their flexibility.
- **Direct** methods predict the full output vector at once. Therefore, they require large amounts of training data, increasing with the length of the prediction, but are more flexible in dealing with inputs of various structure.

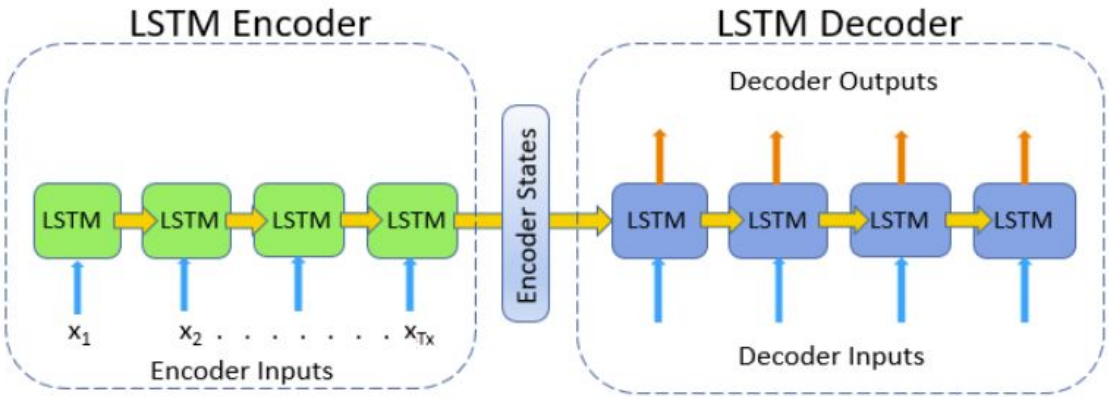


Figure 12: Exemplary overview of an Encoder-Decoder architecture.

We require model flexibility to include known command parameters during the forecasting horizon, which iterative methods cannot provide. For this reason, we have to use a direct method. Their general form is a sequence-to-sequence (Seq2Seq) network (as illustrated in Figure 12). In the first step, the whole input sequence is encoded using a neural network. The output of the encoder is then decoded using a second neural network that accepts additional inputs at each steps [12]. Typically, the network used in this

architecture is a type of recurrent neural network (RNN), as RNNs are by their design a natural fit for processing sequential data: each cell contains a memory of past information that is updated iteratively as the the inputs are processed. A commonly used RNN architecture is long short-term memory (LSTM), which is better at learning longer-range dependencies [7]. It is possible to use other types of units for the encoder-decoder design, such as convolutional neural networks (CNNs), which are useful for spatial prediction and classification. As we work with temporal data, which is sequential in nature, we expect an RNN-based architecture to be best suited for our task.

3.2.1 Advanced Seq2Seq Architectures

While simple Seq2Seq architectures have shown moderate success in time series forecasting, they tend to struggle to model dependencies between outputs and known future inputs, particularly if the forecasting horizon is long [4]. In the past few years, several additions to the architecture have been proposed to solve this challenge.
Attention mechanisms learn important temporal dynamics, allowing the network to directly focus on significant time steps in the past [20, 13]. Graph neural networks are networks that typically leverage known graph structures to more accurately learn dependencies. In the domain of time-series forecasting, a separate network can be trained to learn graph-like dependencies between variables, allowing the forecasting network to use this knowledge [26]. A popular method to improve interpretability of results is to generate quantile forecasts that show the uncertainty of predictions instead of single-point forecasts [4, 13].
The Temporal Fusion Transformer [13] is a Seq2Seq architecture that includes several of the additions outlined above, such as attention mechanisms and quantile forecasts. Furthermore, the architecture includes variable selection networks at each step, which further improves interpretability. For these reasons, we choose this as our Seq2Seq model.

3.2.2 Probabilistic Time Series Forecasting

Probabilistic time series forecasting aims at predicting the probability distribution of a time series. Since methods forecasting a single point instead of a distribution disregard the risk caused by high noise in the data, probabilistic forecasting is a very powerful tool when uncertainty is high. Normalizing flows map the complex distribution of the data to a commonly known and simple distribution by sequentially employing bijections and invertible functions. The original density of the data can then easily be expressed in terms of this simple distribution via the change of variable formula [18]. Deep learning models for probabilistic time series forecasting based on conditional normalizing flows [18] are capable of modelling conditional multivariate dynamics of time series, thus incorporating interaction effects and statistical dependencies in between the separate series. These Temporal Conditioned Normalizing Flows are suitable for our purposes, since they allow for estimating the distribution of the parameter of interest conditioned on the command parameters.

3.3 Classical Statistical Forecasting

Classical statistical methods serve as a standard approach towards data modeling. Although they are more and more often replaced by more powerful approaches capable of recognizing richer patterns, the statistical methods tend to offer better interpretability and require less data. Motivated by this, we first study the most widespread classical methods. Later, in Section 3.4.3, we augment them with non-statistical extensions to improve model performance and obtain hybrid models which are suitable to our task.

3.3.1 Autoregressive Models

Autoregressive models make predictions based on lagged data, i.e., past information contained within the series. One of the most well-known statistical time series forecasting methods is the Autoregressive Integrated Moving Average (ARIMA), which consists of three main components. The autoregressive framework predicts the variable of interest employing a linear combination of lagged values. The integrated part of an ARIMA model removes non-stationarity by learning the derivative order needed such that the data shows constant variance[1]. Finally, the Moving Avarage scheme uses past forecast errors, similarly to the autoregressive technique, for predicting the variable of interest.
Further modifications can be applied in order to suit the data better, e.g., for data that is already stationary, the integrated part is not required, so an ARMA model is used. For forecasting multivariate data, as in our project, an extension such as ARIMAX or Regression with ARIMA errors is appropriate. Both methods employ features (covariates) in order to help to improve the model fit. The two methodologies differ in the way the exogenous variables are added into the model. The ARIMAX model simply adds the covariates. A key disadvantage is the difficulty of interpreting its coefficients: they can only be interpreted conditional on the value of previous values of the response variable. On the other hand, a regression with ARIMA errors models a standard ARIMA over the data; it tries to explain the forecasting errors in terms of the covariates. Intuitively speaking, its objective is to capture the patterns that the lagged values could not find while predicting. Here, the features' parameters can be directly interpreted, like in regression modeling. Lastly, we studied seasonality ARIMA models. This approach considers a seasonality component over all the terms of the normal ARIMA model. The seasonal part of the model consists of elements that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period.

3.3.2 Exponential Smoothing

Exponential smoothing enhances the concept of autoregression by employing weighted averages of past observations. The weights decay exponentially as the observation lags increase. In other words: The more recent the observation, the higher its impact on the forecast. The Holt-Winters method, an adaptation of the exponential smoothing concept, is one of the most widely used forecasting methods. This technique captures seasonality, building the forecast equation out of three smoothing equations for level, trend and one seasonal component respectively, with its corresponding smoothing parameters.

[1] A time series is stationary if it does not depend on the time at which the series is observed. If trends or seasonality exist, it is not stationary.

One drawback is that the method requires positive values. Thus, it is necessary to apply a transformation, typically a Box-Cox or Yeo-Johnson transformation. The need for transformation and the usage of an iterative forecasting process in this technique increase the error when predicting over a long forecast horizon.

3.3.3 Spectral Analysis

Spectral analysis is used to detect cycles and trends in the data. This results in a measure of the relative contribution of cycles in a band of frequencies. In other words, its objective is to decompose a time series into simpler structures based on a frequencies decomposition framework. We looked into two decomposition methods: the Fourier and Wavelet Transform. Since the Fourier Transform lacks a time frame interpretability, we focused our studies on wavelets. A wavelet is a function that may be described as a localized wavelike function. In contrast to the techniques explained in Section 3.3.1 and 3.3.2, a stationarity assumption can be avoided. This is mainly because the wavelets are split in a translation and a scale phase. The translation part refers to movements along the time axis, while the scale refers to the spreading out of the wavelet. The idea is to perform a multi-resolution decomposition procedure which consists of a *high-pass* and *low-pass* filtering process. The high-pass filter allows high-signal frequencies corresponding to the signal details to go through, while the low-pass filter allows low-signal frequencies and hence a smoothed version of the signal to go through. In an iterative process, the low-pass filter is applied over the information retained by the high-pass filter. This procedure ends when the low-pass filter cannot find new patterns on the remaining part.

3.4 Hybrid Models

Hybrid models have become more and more prominent in the literature in the past few years due to their ability to counter limitations of other existing approaches. Time series usually consist of linear and nonlinear as well as stationary and non-stationary components, which in many cases cannot accurately be characterized by just one model, but rather requires a combination of at least two models [27]. Furthermore, pure machine learning models are very sensitive to preprocessing and prone to overfitting, whereas pure statistical models require the data to be stationary. Hybrid models combine statistical methods with artificial neural networks to address these issues [12], and have been proven to be more effective than individual models on a variety of data sets. In particular, they demonstrate accurate performance even with little training data [12].
Hybrid models can be classified into three different groups of approaches. The preprocessing-based hybrid approach incorporates the preprocessing step to the model itself, thereby reducing the information loss due to preprocessing. The optimization-based hybrid approach learns parameters of a model using metaheuristic algorithms. Finally, the component-based hybrid approach combines multiple prediction models into one in a way that they compensate for each other's disadvantages [6].

For our project, the preprocessing-based as well as the component-based hybrid approach turned out to be of greater interest. Four models with different characteristics, including the widely used ARIMA-ANN architecture [16], methods based on a combination of

ordinary differential equations and RNNs [3, 19], as well as ES-RNN [23] were analyzed further.
Approaches based on ordinary differential equations are well-suited for irregularly sampled or sporadically-observed time series [3, 19]. Therefore, they seem well-suited for the satellite data. However, as explained in Section 2.5, we decided to train all models on gapless, regularly resampled data. Thus, the models based on ordinary differential equations are not optimal for our project, but are worth pursuing in future work.

3.4.1 Exponential Smoothing RNN

The Exponential Smoothing Recurrent Neural Network (ES-RNN) [24] uses Holt-Winters exponential smoothing to capture seasonality, trend and level of the individual univariate series. In a second step, the resulting series are fed into a stacked and dilated Long Short Term Memory (LSTM) architecture to analyze the non-linear trend and to perform cross-learning [23]. Although this method seems to be a promising hybrid approach for our forecasting problem, adapting the underlying dilated RNN model to include known future command parameters would be infeasible for the given time horizon and might result in bad performance of the model.
Generally, pre-implemented hybrid models are built to specifically suit only one or very few data sets and are difficult and time-consuming to adapt to fit our data. Furthermore, changing the structure of the models for incorporating future command parameters might violate the mathematical theory behind the models and thus might not only result in bad performance, but also in poor interpretability. Hence, it is not possible to simply use a pre-existing implementation, but it will rather be necessary to manually construct a method.
Deseasonalization techniques and exponential smoothing functions available in various Python libraries cannot be automatized to our data, since we neither know the periodicity beforehand nor do the time stamps of our data follow any common frequency, so the available Python methods cannot detect the periodicity. Thus, building our own model for deseasonalizing and detrending all different time series as the first step in the preprocessing-based hybrid approach would require substantial efforts and is therefore out of the scope of this project. Nevertheless, it seems worth pursuing a hybrid model following the idea of the ES-RNN for future work.

3.4.2 Introduction to Wavelet Hybrid Models

Hybrid models based on a wavelet transformation are follow a multi-step process. In the first step, the target time series is transformed to different high- and low-signal frequencies. Next, each wavelet is forecast via an arbitrary estimation method before an inverse wavelet transformation is performed to obtain the final forecast.

3.4.3 Wavelet-ARIMA

As discussed in Section 3.3, we aim to augment a classic statistical method to improve prediction performance. We decided on the ARIMA process in conjugation with the wavelet transformation. The Wavelet-ARMA model is planned to be implemented via the procedure shown in Figure 13.

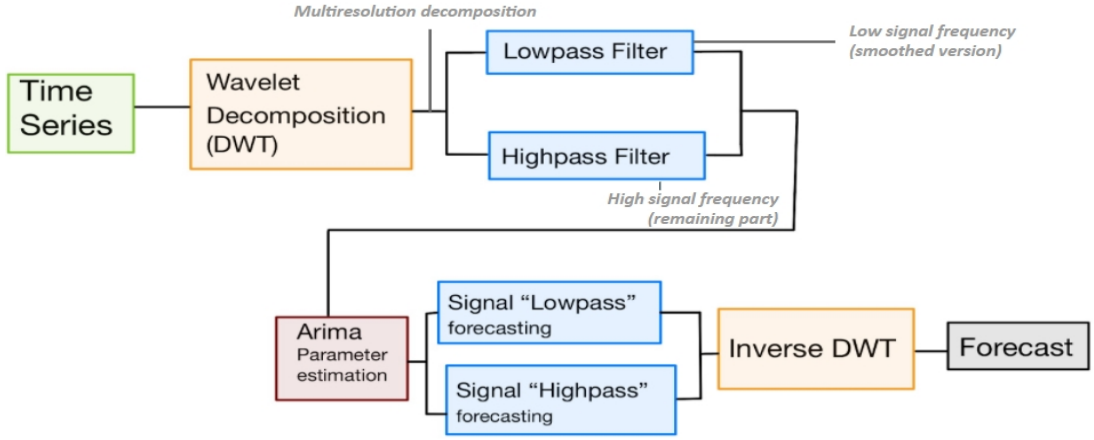


Figure 13: Wavelet-ARIMA model diagram

The traditional ARIMA model seems to be slightly superior in prediction to the basic wavelet approach [5]. In order to improve our individual ARIMA predictions, we decided to modify the traditional ARIMA model by including the command parameters as exogenous variables. A regression model with ARMA errors is defined as follows [8]:

$$\begin{aligned} y_t &= \beta x_{CP_t} + \eta_t \\ \eta_t &= \phi_1 \eta_{t-1} + ... + \phi_p \eta_{t-p} - \theta_1 z_{t-1} - ... - \theta_q z_{t-q} + z_t. \end{aligned} \tag{1}$$

In our case, the y represents the split time series from the parameter of interest and $X = \{x_{CP_1}, ..., x_{CP_n}\}$ the information matrix from the command parameters. We use an ARIMA model to forecast the residuals η, i.e., the information that the regression model could not capture. The results can be directly interpreted: The regression coefficients have their usual interpretation, as have the *AR, I, MA* of the ARIMA models. Finally, this technique merges the individual forecasts via an inverse stationary wavelet transformation.

3.4.4 Advanced Wavelet Approaches

Instead of ARIMA, more advanced models can be used for prediction after applying the wavelet decomposition. For instance, Wavelet-ANN [15] retains the wavelet approach outlined in Section 3.4.2, but replaces the ARIMA model in the forecasting step with an artificial neural network, which is typically better at modeling non-linear relations. While the proposed approach uses a simple neural network architecture, more complex architectures are also feasible, providing an interesting field for further research.

3.4.5 Wavelet Temporal Conditioned Normalizing Flow

The second variant of a hybrid method based on wavelet transformation employs the temporal conditioned normalizing flow described in Section 3.2.2 for forecasting the respective wavelets. We expect the temporal conditioned normalizing flow to be able to forecast the separate frequencies as univariate targets conditioned on the multivariate command parameter space more accurately than the original more complicated time-series. A further

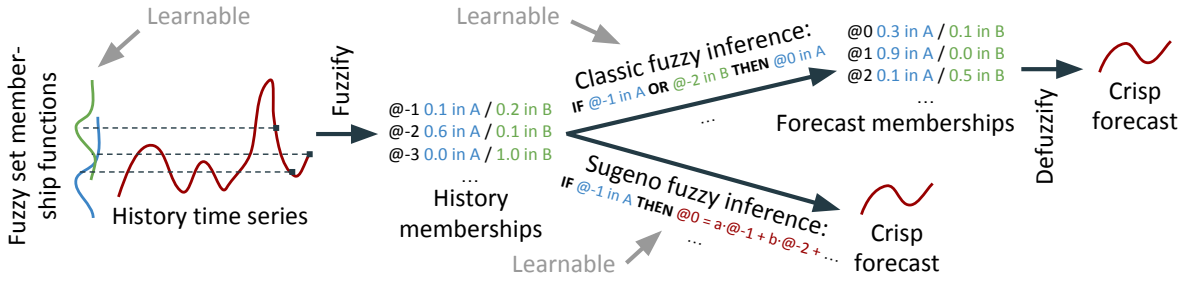


Figure 14: Illustration of the fuzzy time series forecasting framework.

advantage of creating a hybrid model based on these two methods is the capability of the temporal conditioned normalizing flow to learn a multivariate conditional distribution of all wavelets conditioned on the command parameters, which allows for cross-learning.

3.5 Fuzzy Time Series Forecasting

Methods based on fuzzy logic are surprisingly popular in the literature. All of them are more or less based on the same framework [21, 2], which is illustrated in Figure 14. In this framework, a time series history to be forecast is first sampled at a fixed number of time stamps. These samples are then fuzzified, that is, their memberships (between 0 and 1) in several (learnable) fuzzy sets over the value range of the time series are determined. Next, a (learnable) fuzzy inference system (FIS) is applied, which either directly yields a crisp forecast (Takagi & Sugeno rules) or memberships of the forecast samples in the fuzzy sets (classic rules), which are then defuzzified in a second step. Note that because this framework is in its core feature-based, the exogenous command parameters are supported out of the box. A concrete implementation entails three choices:

- Choice of the fuzzy sets covering the data domain. This is called partitioning. Typically, each variable of the time series is partitioned separately. The sets may be chosen uniformly spread, but better performance can be achieved by either choosing them cleverly (e.g., by prior clustering) or by optimizing them (e.g., by using genetic algorithms, where the fitness of a partitioning is evaluated by learning and testing a model with that partitioning). There are lots of techniques for this.

- Choice of the fuzzy logic rules that infer the consequents (forecast) from the antecedents (history memberships). With classic rules, the consequents are fuzzy sets. With Takagi & Sugeno rules, the consequents are linear combinations of the raw time series data. There are various non-neural and various neural approaches for this.

- If necessary, choice of a defuzzification technique to defuzzify the consequent memberships into a crisp forecast. There are several methods for this, some of which are more intuitive, while others are computationally cheaper. Some of them also yield confidence intervals or even probability distributions instead of point estimates.

Even though the literature offers an abundance of methods for these three choices [21, 2], an experimental comparison is lacking, impeding model selection. In addition, many

methods may prove difficult to implement because of a lack of clear documentation. However, there is a python library (`pyFTS` [22]) that implements some of the non-neural methods which we assume to be the most popular ones. As a starting point, one could try combinations of these pre-made methods and some easy-to-implement non-premade ones, especially simple neural approaches. Then, one could dig deeper into the best performing methods.

3.5.1 ANFIS

Instead of considering each component of the framework separately, it is also possible to design a neural network that mimics the whole framework end-to-end. This yields a powerful neural network that is quite interpretable due to the fuzzy structure. The most prominent of these methods is ANFIS, whose publication is one of the most cited ones in fuzzy inference [9]. Most of the time, Takagi & Sugeno rules with a linear combination consequent or a "pseudo"-Gaussian consequent are used, since applying classical rules and then defuzzifying has proven to not be worth the additional computational effort. Each rule has its own fuzzy sets over the input, usually one for each scalar in the input; the memberships of the input in those are AND-combined to yield the rule's antecedent. Training is often not done entirely with gradient descent, but with a both more efficient and more effective hybrid combination of gradient descent for the fuzzy sets and another method [10] for the rule consequents, such as least squares. All this is visually illustrated in Section 4.5.1. Various extensions to ANFIS have been proposed, including straightforward support for multivariate time series [25], CON-ANFIS, and using ANFIS as a downstream ensemble combination approach.
All in all, ANFIS seems to be the most promising fuzzy time series forecasting approach. Hence, we choose it as our primary fuzzy model.

3.6 Summary

Since this project aims at exploring various time series forecasting methods for the purpose of resource forecasting under the special requirements of GSOC's satellite fleet, we will implement models from different data analysis fields. As a baseline method, we will employ the popular ARIMA method described in Section 3.3.1. Our deep learning model of choice will be the Temporal Fusion Transformer introduced in 3.2.1. Further, we will analyze two approaches based on wavelet transformations, namely the statistical method Wavelet-ARIMA explained in Section 3.4.3 and the hybrid Wavelet-Normalizing-Flow described in Section 3.4.2. The final implementation will be the fuzzy time series forecasting approach ANFIS outlined in Section 3.5.1. Table 1 shows an overview of the strengths and weaknesses of the advanced models we will investigate further. The explanation of the categories along with a more detailed overview over all discussed models can be found in Appendix A.

Model	Suitability	Implement.	Tuning	Performance	Training data	Runtime	Explainability
TFT	2	2	0.5	1	-1	0	0
W-ARIMA	1	0	1	2	0	-1	1
W-NF	0	0	1	0	2	0	-1
ANFIS	2	0	1	1	1	2	1

Table 1: Qualitative assessment of selected forecasting methods. This is to be understood as a subjective overview and not as a direct model-to-model comparison between models of different categories. The models were rated on a scale from -2 to 2.

4 Experiments

4.1 Baseline ARIMA

ARIMA is a relatively simple model that makes a forecast based on the history of the target parameter only. Therefore, it serves as a benchmark for other models: if another model performs worse than this, we can discard it. As the standard ARIMA requires manual configuration of hyperparameters, we use the AutoARIMA model from the `sktime` Python framework [14], which automatically selects appropriate hyperparameters.

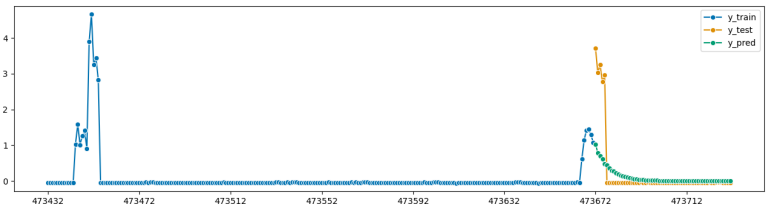


Figure 15: AutoARIMA prediction for Parameter T2-1 with 12 hours history size and a resolution of 180s.

We observe that the model is able to capture periodic behavior, but fails to forecast sudden changes, as shown in Figure 15. In further experiments, we saw that ARIMA does not perform well with a high resolution of 30 seconds, which suggests that the model gets overwhelmed with both the amount of input data and the amount of time steps that have to be predicted. It can capture the behaviour of the first few points and tries to follow that trend but fails afterwards, resulting in a straight-line prediction. On the other hand, when it is trained with 180 seconds resolution, it performs better since it can understand the periodicity of the data.
Another observation is that the model is very sensitive to small changes, behaving vastly differently depending on the exact time point at which the prediction starts. This behaviour is illustrated in figure 18.

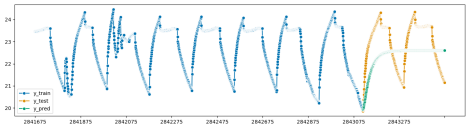


Figure 16: Prediction of Parameter T1_1 on 30s resolution

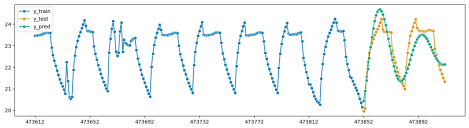
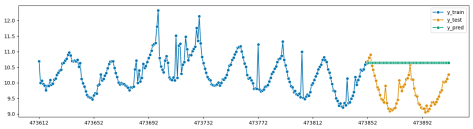


Figure 17: Prediction of Parameter T1_1 on 180s resolution



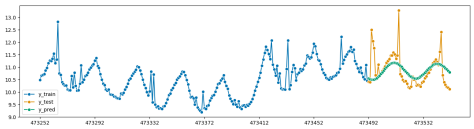


Figure 18: Two AutoARIMA predictions of Param_T4_1 with vastly different results

4.2 Temporal Fusion Transformer

4.2.1 Implementation and Training

For the Temporal Fusion Transformer (TFT), we use the existing implementation in the `python-forecasting` package [1]. The package does not include native support of our sliding window approach, so we extended it by a class that loads data based on a provided list of window start indices.
As training the models is computationally expensive and can take several hours, we used Linux virtual machines provided through the LRZ Compute Cloud for training.

4.2.2 Results

The power to be generalizable to various problems is one of the biggest strengths of neural networks, which can also be seen in the results of our forecasting task (see table 2). While there is some difference in performance between different sampling configurations, the network is able to adapt to and generate fairly accurate forecasts for all of them. Overall, the model does not seem to profit from increasing the history size from 3 hours to 12 hours, suggesting that it may be able to learn accurate predictions with limited data.
Figure 19 displays one prediction for Param_T1_1 (with history size 3 hours, forecasting horizon 3 hours, and sampling rate 180 seconds) made by the model. We observe that it accurately predicts the general behavior of the parameter, but fails to predict the severity of the spike at around time index 40. We also see that the model is less certain of its prediction at around that time.

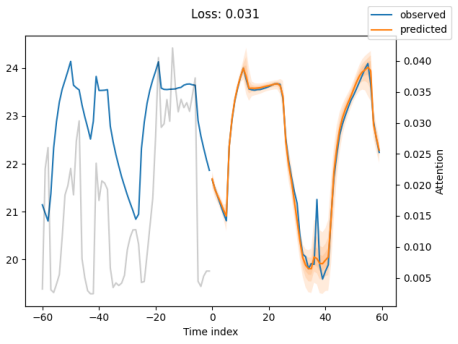


Figure 19: Example prediction made by the TFT. The grey line shows the attention, i.e., which importance the model assigns to each step in the history. The shaded orange areas display confidence intervals: the larger the area, the less confident the prediction.

Figures 20 and 21 show a key interpretability feature of the TFT: We can see which

variables were most important for the predictions of each model. The example shows the variable importance for predicting Param_T1_1 with the configuration from above. We can see that the past values of the variable itself were most impactful, but some of the control parameters also have a strong impact. Such analyses help verify that the model learns reasonable relationships.

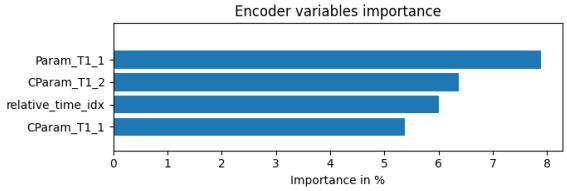


Figure 20: Example of learned encoder variable importance.

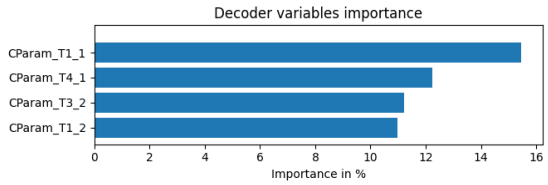


Figure 21: Example of learned decoder variable importance.

4.2.3 Future Work

Hyperparameter tuning is an important step when training neural networks. In our case, the default parameters were sufficient to demonstrate the potential of the model for the forecasting task. Due to this, there was no need to spend extended efforts trying to maximize prediction performance. However, we would expect the model to perform even better with carefully selected hyperparameters.
Another important consideration when training neural networks is the amount of data required. We have not tested model performance for different amounts of data, using the whole provided time range for every experiment. It is worth investigating how much data is required to generate predictions of acceptable accuracy. One possible way of obtaining more training batches without having to wait until more data is generated would be to reduce the amount of time by which the sliding window is moved. Moving the window by 9 minutes instead of 3 hours, for instance, would increase the number of batches by a factor of 20. It may be worth assessing this trade-off between increasing the number of training batches and increasing redundancy in the training data.

4.3 Wavelet-ARIMA and Wavelet-ANN

The wavelet approach decomposes each of the time series generated from the parameters $\{T_1, ..., T_4\}$ into a time-frequency space. This enables us to identify the different structures that compose the time series, i.e., frequencies. As the whole forecast is based on this step, we require the wavelet transform to capture the key characteristics of our data. We address this challenge by testing decompositions based on different resolutions. In this way, we were able to identify the specter in which the Wavelet performs better on our data. For the implementation, we employed the Stationary Wavelet Transform of the Python wavelet transform module `PyWavelets` [11]. We applied the models not only to the different sampling rates explained in Section 2.5, but also to different forecast horizons: 3 hours, 6 hours, and 12 hours. This allows us to gain a better understanding of the limitations of this approach.

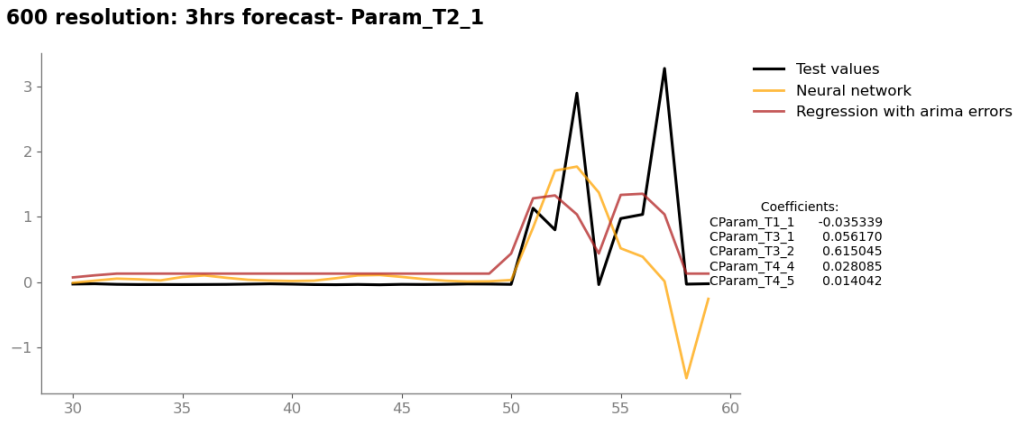


Figure 22: Example prediction of parameter T2_1 using a wavelet approach.

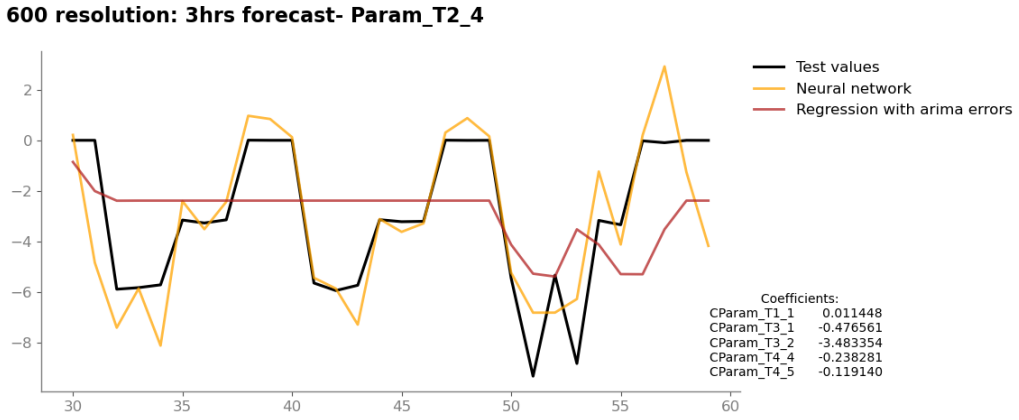


Figure 23: Example prediction of parameter T2_4 using a wavelet approach.

4.3.1 Regression With ARIMA Errors

The main motivation behind combining the Wavelet Decomposition with an ARIMA-based model is to boost the performance of the basic ARIMA approach. We expect the ARIMA model to be able to explain the characteristics of the target parameter that the regression model alone cannot capture. In combination, this should improve prediction performance. However, while the prediction seemed to be less prone to extreme errors, the regression with ARIMA errors modeling only showed a minor improvement on some parameters and resolutions compared to the baseline model (see Table 3. It is possible that the ARIMA model is just too simplistic to leverage the additional insights embedded in the wavelets. Exploring the combination of the wavelet transformation with more powerful prediction methods may be a worthwhile endeavor.

4.3.2 ANN

Due to the underwhelming results of the Wavelet-ARIMA approach, we additionally ran some small experiments using the Wavelet-ANN architecture. Indeed, we observe that this seems to have the potential to be better at forecasting non-regular behavior (see Figure 23), although it is still prone to severe prediction errors, as shown in Figure 22.

4.3.3 Next steps

The Wavelet decomposition allows our models to identify behavioral patterns. However, producing accurate forecasts from these intermediate results proved to be a challenging task. In small-scale tests, we could see that a hybrid neural-network model such as Wavelet-ANN may be able to improve on the statistical ARIMA prediction. This approach could be investigated further. Expanding on this, the simple ANN could be replaced with a more complex architecture to generate a prediction in conjunction with the Wavelet transformation. A comparison between this combined approach and the purely neural-network-based forecasts may provide interesting insights.

4.4 Wavelet Normalizing Flow

For the implementation of a hybrid model consisting of a wavelet transform and a temporal conditioned normalizing flow, we employ the Python libraries `PyWavelets` [11] as described in Section 4.3 and the "PyTorch probabilistic time series forecasting framework" `PyTorchTS` [17]. We focus on the `DeepAR` package for forecasting the univariate wavelets and the `TempFlow` package for estimating the multivariate distribution of all wavelets conditioned on the command parameters. The estimators learn density parameters of a given distribution by maximizing the log-likelihood of the fed time series at every time step [18].
Since `PyTorchTS` is not supported by a Microsoft Windows operating system, we need to implement the model on a Virtual Machine simulating a Linux distribution.

4.4.1 Implementation

We implement two different versions of the hybrid Wavelet Normalizing Flow model. Both models apply a stationary wavelet transformation to the parameter of interest. For the univariate model, multiple `DeepAREstimator`s are trained, such that each wavelet can be forecast as a univariate time series based on the multivariate command parameters. The multivariate variant only employs one `TempFlowEstimator` modelling the conditional multivariate distribution of all wavelets on the command parameters. The final prediction of the parameter of interest is obtained via an inverse wavelet transform of the median forecasts of the separate wavelets.
Although the estimators already perform acceptable on forecasting the wavelets based on their unconditional distributions, adding multiple command parameters to the model easily confuses the estimators. Hence, the influence of the command parameters on the respective wavelets needs to be analyzed prior to adding further covariates to the models. Figures 24, 25, 26 depict the effect of including different quantities of command parameters in the `DeepAREstimator`s.

4.4.2 Results and Future Work

Training and hyperparameter tuning of the two models could not be finalized due to unforeseen complications in the implementation process and an in-depth investigation of the hybrid ES-RNN model prior to the implementation of normalizing flows. These steps are therefore left for future work. In addition, we suggest to apply a method to identify which

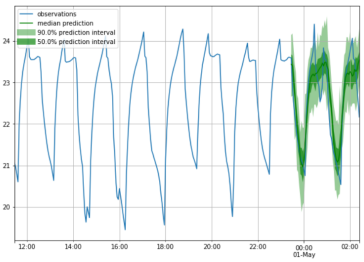


Figure 24: Distribution of parameter T1_1.

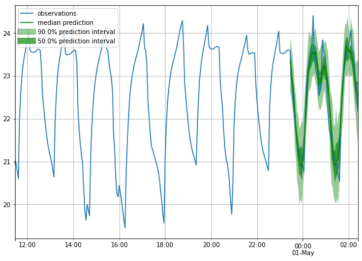


Figure 25: Conditional distribution of param. T1_1 on few command parameters.

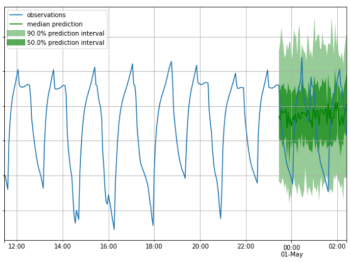


Figure 26: Conditional distribution of parameter T1_1 on all command parameters.

covariates are best suited for predicting a certain target parameter. This could be done by performing a regression predicting the target parameter using the command parameters, and defining that those commands whose coefficients exceed a certain threshold are included in the model.
In comparison to the univariate approach, the multivariate Wavelet-Normalizing-Flow shows better forecasts in a significantly shorter training time, even without hyperparameter optimization. This suggests that the multivariate approach should be the preferred option in future applications. Both variations of a Wavelet Normalizing Flow create worse forecasts with increasing prediction length. Observing a low accuracy of the forecast of high frequency wavelets and considering the better performance of the `DeepAREstimator` directly applied to the parameter of interest than the performance of the proposed hybrid framework, we conclude that the hybrid wavelet normalizing flow might not be the model of choice for our purposes. However, a temporal conditioned normalizing flow directly applied to the target time series without a prior wavelet transform seems to be a promising alternative.

4.5 ANFIS

Using a fuzzy time series forecasting method like ANFIS first requires us to fill in some deliberate holes in the design:

- The structure of the learnable antecedent membership functions over the input time series. For each continuous input value, we decide on using one (per rule) independent 1-dimensional Gaussian membership function. For each categorical input value, we use one (per rule) membership function that assigns some learnable membership to each category. The overall membership of an input in the antecedent fuzzy set of a rule is obtained by multiplying the membership function outputs for that rule of each input value[2].

[2]Say that we use: 3 rules; 30 steps of history; a 20 steps big forecasting horizon; 15 continuous input parameters known during the history; 5 continuous input parameters known during the forecasting horizon; and 2 categorical input parameters known during both the history and the forecasting horizon and take 7 respectively 9 categories. Our model then has $3 \cdot (30 \cdot 15 + 20 \cdot 5)$ independent Gaussian membership functions (one for each rule and continuous input value) and $3 \cdot ((30+20) \cdot (7+9))$ independent categorical membership functions (one for each rule and categorical input value).

- The structure of the learnable inference rules. We use Takagi & Sugeno rules with a linear combination consequent.
- How to represent the forecast in the ANFIS model. We decide on creating a model that individually predicts each time point for each target parameter over the entire forecasting horizon at once[3].

Further details regarding the model structure can be inferred from the well-documented ANFIS implementation that we wrote specifically for our use case.
Note that ANFIS is not a deep neural network architecture, but rather a shallow one. This makes it quite interpretable as discussed in Section 4.5.1. However, shallow networks tend to require more weights to achieve the same performance compared to deep networks. Even worse, as ANIFS is a feature-based model mostly ignoring the temporal structure, it is bound to scale poorly with increasing input and output size. For example, the number of weights grows quadratically with the sampling rate s.

4.5.1 Interpretability

Due to their structure, ANFIS models are quite interpretable, which we illustrate through the univariate forecasting toy example depicted in Figure 27. We have trained two ANFIS models on this task. Model A was trained using only stochastic gradient descent (SGD), while model B used the hybrid learning scheme.
In Figures 28 and 29, we visualize the learned 1-dimensional Gaussian membership functions for each rule and history time point; they are stacked horizontally. You can see which shape a history time series must have in order to reach high membership in the fuzzy set of each rule. In Figures 30 and 31, we visualize the weights of the linear Takagi

[3]Say we have a 20 step forecasting horizon and forecast 5 params. Our model then has $20 \cdot 5$ outputs.

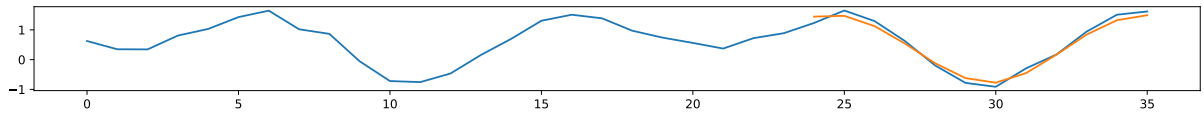


Figure 27: Forecasting toy example.

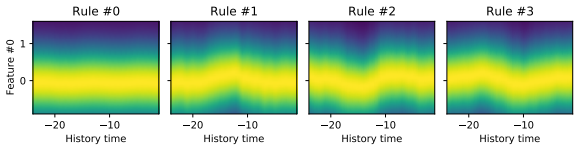


Figure 28: Membership functions learned using only gradient descent. (A)

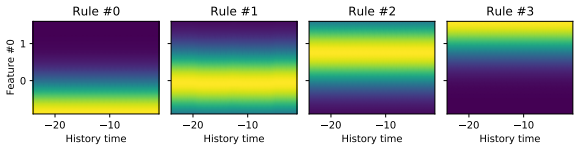


Figure 29: Membership functions learned using the hybrid learning scheme. (B)

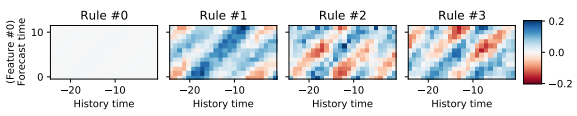


Figure 30: Matching rule consequents. (A)

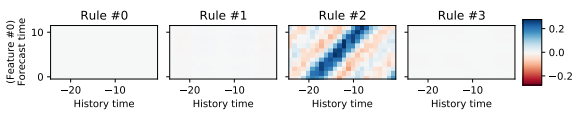


Figure 31: Matching rule consequents. (B)

& Sugeno rule consequents, which are just regression weights over the input values. Note that the plots (almost) completely characterize the two models. All learnable parameters (apart from the consequent bias) are visualized.
We can now visually follow what happens when forecasting a time series. First, its membership in each rule's antecedent fuzzy set is computed. The vector of memberships is then L_1-normalized. Next, each rule's consequent is computed by running the regression. Finally, the consequents are multiplied with the normalized memberships and summed up to yield the final forecast.
Interpreting the plots gives an interesting insight. Because pure SGD training is not as effective, it leads to a more inefficient model (A) that matches specific shapes of the history series and uses different weights depending on the matched shape. The hybrid-trained model (B) on the other hand discovers after only one epoch that a single rule without any antecedent structure is already sufficient to represent the data, which means that in this case, ANFIS mostly degrades to a linear regression model. The non-hybrid training was unable to discover this. However, note that the fast hybrid training got stuck after one epoch, while the slow pure gradient descent training managed to achieve a lower validation loss in the end.

4.5.2 Training, Results, and Future Work

While training various ANFIS models, we gained the following insights:

- All training data has to be on the same scale, ideally z-normalized. Otherwise, the scale of the gradients for the antecedent membership functions becomes unstable.
- If possible, the training data should not contain extreme outliers. They have an overproportionally large impact on the L_2 loss. Even worse, they can make the rule consequent gradients very large, while the antecedent gradients stay comparatively small. If this happens and one does not compensate for it, the antecedents stay mostly constant during training, so ANFIS degenerates to linear regression.
- Training performs best when the Gaussian antecedent membership functions are initialized as seen in Figure 29. Other schemes have proven inferior.
- During training, the validation loss sometimes jumps up unexpectedly; however, this is no reason for concern. Usually, after some time, the validation loss will drop back down to somewhere even below the validation loss before the jump. Of course, if the validation loss continues to go up for some time, the training can be stopped, as the model is most likely overfitting in that case.
- Our ANFIS models trained best with a learning rate (LR) of 10^{-3} for the antecedent membership functions and 10^{-4} for the consequents. If a lower LR is used for the antecedents, they stay constant during training, and so ANFIS degenerates to linear regression. Note that the required ratio between the two LRs can vary, so it is key to always confirm that training actually moves the membership functions after a couple of epochs. After convergence, dropping the LR once by a factor of 10 can also slightly improve the validation loss.

- Hybrid learning is a double-edged sword. Employing it directly after initialization immediately reduces the loss drastically, but often puts the model into a position where it has little incentive to adjust the antecedent membership functions. When we instead apply it once after gradient descent-only training has converged, the validation loss increases. However, when we then continue with gradient-descent training, we often converge to a model with a lower validation loss than the first converged model. In summary, hybrid learning can be helpful to "unstuck" the training.

- While we could train most ANFIS models on our own computers, for the biggest one (sampling rate of 30s), we had to turn to a more powerful server that provides the amount of RAM required for computing the gradients of that model. This is due to the surprisingly quick parameter explosion.

- Increasing the number of rules from 16 to 32 did not substantially improve performance. If we visualize the antecedents and consequents, we observe that the model seems to mostly ignore the added ones. Maybe bigger numbers of rules could improve performance? Maybe smaller numbers could have a regularizing effect and thereby improve performance? These are still open questions.

The remaining details of the training procedure can once again be inferred from our code.

4.6 Model Comparison

We compare the different methods based on the scaled mean squared error (MSE) evaluation metric. We prefer it over other metrics for its simplicity and widespread adoption as a universal regression evaluation measure. By focusing on the parameters T1_1, T2_1, T2_4, T3_1, and T4_1 representing voltage, two different currents, battery status and temperature, we ensure to include one parameter of every inter-correlated group of parameters as investigated in our correlation analysis in Section 2.3.2. The final score depicted in Table 2 is given by the average MSE over the parameters after manual adjustment of the scale of each time series variable. More detailed results on the performance of all models on the five parameters separately can be found in Table 3 in Appendix B.

Sampling			ARIMA	TFT	Wav-ARIMA	ANFIS
3h	3h	30s	5.644	0.117	0.726	**0.101**
		180s	0.839	**0.067**	0.681	0.097
		600s	0.557	**0.091**	0.760	0.114
12h	3h	30s	0.799	**0.123**	0.662	-
		180s	0.517	**0.084**	0.685	-
		600s	0.475	**0.095**	0.509	-

Table 2: Scaled mean squared error achieved by the models on various samplings of the test data averaged over the parameters T1_1, T2_1, T2_4, T3_1, and T4_1. A sampling is identified by a history size, a forecasting horizon, and a sampling rate.

5 Conclusion

We presented a general overview of traditional statistical and modern machine learning methods for multivariate time series forecasting using both past observations and known future covariates. Especially well-suited methods based on a wavelet transformation, fuzzy logic and a deep neural network with attention mechanisms were analyzed further. The performance of the models was evaluated based on the mean squared error metric against a baseline error achieved by an ARIMA method.
Our results show a very strong and robust overall performance of the Temporal Fusion Transformer. Against our expectations, the Wavelet-ARIMA does not clearly outperform the benchmark ARIMA model, although it seems to be less prone to extreme errors. The performance of the fuzzy method ANFIS is almost comparable with the performance of the Temporal Fusion Transformer. For a sampling rate of 30 seconds, ANFIS even outperforms the deep neural network architecture on average.
Nevertheless, we regard the Temporal Fusion Transformer as the most suitable method for the purpose of resource forecasting for satellites, also considering the poor scalability of ANFIS for longer forecasting horizons and a larger number of input parameters.
Since the error achieved by the Temporal Fusion Transformer does not significantly decrease for increasing history size, we suggest to employ the model on a three hour history size as an optimization tool for resource forecasting of GSOC's satellites. Extending the forecast horizon to a longer time frame as well as hyperparameter optimization of the Temporal Fusion Transformer and the Temporal Conditioned Normalizing Flow is left for future work.

Bibliography

[1] J. Beitner. *PyTorch Forecasting*. 2020. URL: https://github.com/jdb78/pytorch-forecasting.

[2] M. Bose and K. Mali. "Designing fuzzy time series forecasting models: A survey". In: *International Journal of Approximate Reasoning* 111 (2019), pp. 78–99.

[3] E. De Brouwer et al. "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series". In: *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 7377–7388.

[4] C. Fan et al. "Multi-Horizon Time Series Forecasting with Temporal Attention Learning". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 2527–2535.

[5] A. Fonseca Lemus. "Wavelet analysis on financial time series". PhD thesis. Universidad del Rosario, 2018.

[6] Z. Hajirahimi and M. Khashei. "Hybrid structures in time series modeling and forecasting: A review". In: *Engineering Applications of Artificial Intelligence* 86 (2019), pp. 83–106.

[7] H. Hewamalage, C. Bergmeir, and K. Bandara. "Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions". In: *International Journal of Forecasting* 37.1 (2021), pp. 388–427.

[8] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts.com, Heathmont, Victoria, 2014.

[9] J.-S. R. Jang. "ANFIS: Adaptive-Network-Based Fuzzy Inference System". In: *IEEE Transactions on Systems, Man and Cybernetics* 23.3 (1993), pp. 665–685.

[10] D. Karaboga and E. Kaya. "Adaptive network based fuzzy inference system (ANFIS) training approaches: a comprehensive survey". In: *Artificial Intelligence Review* 52.4 (2019), pp. 2263–2293.

[11] G. R. Lee et al. *PyWavelets: A Python package for wavelet analysis*. 2019.

[12] B. Lim and S. Zohren. "Time Series Forecasting With Deep Learning: A Survey". In: (2020). arXiv: 2004.13408 [stat.ML].

[13] B. Lim et al. *Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting*. 2020. arXiv: 1912.09363 [stat.ML].

[14] M. Löning et al. *sktime: A Unified Interface for Machine Learning with Time Series*. 2019. arXiv: 1909.07872 [cs.LG].

[15] A. H. Nury, K. Hasan, and M. J. B. Alam. "Comparative study of wavelet-ARIMA and wavelet-ANN models for temperature time series data in northeastern Bangladesh". In: *Journal of King Saud University - Science* 29.1 (2017), pp. 47–61.

[16] A. R. S. Parmezan, V. M. A. Souza, and G. E. A. P. A. Batista. "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model". In: *Information Sciences* 484 (2019), pp. 302–337.

[17] K. Rasul. *PyTorchTS: PyTorch Probabilistic Time Series forecasting*. 2020. URL: https://github.com/zalandoresearch/pytorch-ts.

[18] K. Rasul et al. "Multi-variate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows". In: (2020). arXiv: 2002.06103 [cs.LG].

[19] Y. Rubanova, R. T. Q. Chen, and D. K. Duvenaud. "Latent ODEs for Irregularly-Sampled Time Series". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 5320–5330.

[20] S. Shih, F. Sun, and H.-Y. Lee. "Temporal pattern attention for multivariate time series forecasting". In: *Machine Learning* 108.8 (2019), pp. 1421–1441.

[21] P. C. L. Silva. "Scalable Models for Probabilistic Forecasting with Fuzzy Time Series". PhD thesis. 2019.

[22] P. C. L. Silva et al. *pyFTS: Fuzzy Time Series for Python*. 2018. URL: https://github.com/PYFTS/pyFTS.

[23] S. Smyl. "A hybrid method of Exponential Smoothing and Recurrent Neural Networks for time series forecasting". In: *International Journal of Forecasting* 36.1 (2020), pp. 75–85.

[24] S. Smyl, J. Ranganathan, and A. Pasqua. *M4 Forecasting Competition: Introducing a New Hybrid ES-RNN Model*. 2018. URL: https://eng.uber.com/m4-forecasting-competition/.

[25] A. Vlasenko et al. "A Novel Neuro-Fuzzy Model for Multivariate Time-Series Prediction". In: *Data* 3.4 (2018), p. 62.

[26] Z. Wu et al. "Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 753–763.

[27] W. Xu et al. "A hybrid modelling method for time series forecasting based on a linear regression model and deep learning". In: *Applied Intelligence* 49.8 (2019), pp. 3002–3015.

Appendix A – Pre-Implementation Approach Evaluation

Approach resp. Model	Suitability Adaptable to the command parameters being available during the forecasting horizon? −2 Approach not usable at all. −1 Heavy customization needed, which is intellectually challenging. 0 Medium customization needed. +1 Customization is straightforward. +2 Built-in support for our special case.	Implementation −2 Impl. considered impossible for us. −1 Impl. from scratch necessary & hard. 0 Impl. from scratch necessary & doable (e.g., because of available parts or reference impl.). +1 Impl. available, some modifications necessary. +2 Ready-to-use impl. available.	Tuning How much tuning of hyperparameters of switching out of different parts (e.g., NN architectures) will be necessary? −2 Infeasible to find a good configuration. −1 Feasible, but very time-consuming. 0 Requires manual or time-consuming tuning. +1 Automatic & concise search possible. +2 Virtually no tuning required.	Performance Is the model proven in practice? Choose the best result that is reported somewhere. −2 Only works on the most simple data. −1 Only works on somewhat simple data. 0 Okaish, could potentially work for us. +1 Good performance even on complex data. +2 Performance is excellent even on the most complex data.	Training data How much training data is necessary? −2 Definitely not feasible for us. −1 Unsure whether feasible. 0 Feasible, but still a lot. +1 Okayish requirement. +2 Requires surprisingly little.	Runtime How long do training as well as prediction take? −2 Definitely not feasible for us. −1 Unsure whether feasible for us. 0 Feasible, but still a lot (10–60 minutes). +1 Okayish performance (few minutes). +2 Fast performance (seconds).	Explainability How well is the model inherently interpretable? How well is it explainable via external means? −2 No way of ever understanding anything. −1 Somewhat accurate local explanations (i.e., explanations of a single prediction) possible. 0 Pretty good local explanations possible. +1 Globally interpretable with some effort. +2 Globally interpretable with little effort.
Feature-based Classic ML	2	2	1	0	1	2	0
Transforms + Classic ML	2	1	0	-	1	1	0
Feature-based Neural Net	2	1	0	-	−1	0	−1
Seq-to-seq architecture	1	1	0	0	−1	0	−2
Stacked architecture	−1	0	0	−1	0	1	−2
Vanilla CNN	0	1	0	0	−1	0	−2
Vanilla LSTM/GRU	1	1	0	0	−1	0	−2
Graph Neural Nets	0	1	0	0	−1	0	−2
Temp. Fusion Transformer	2	2	0.5	1	−1	0	0
Exponential Smoothing	1	1	1	0	1	1	1
VAR	1	1	1	−1	1	1	2
Wavelet-ARIMA	1	0	1	2	0	−1	1
LSTM-ARIMA	1	0	0	0	−1	0	−1
ES-RNN	0	1	1	1	−1	0	−1
GRU-ODE-Bayes	−1	1	1	1	−1	-	−1
Latent ODE	−1	1	1	1	−1	-	−1
Wavelet-Normalizing-Flow	0	0	1	0	2	0	−1
Classic Fuzzy Forecasting	1	2	−1	1	1	2	1
ANFIS	2	0	1	1	1	2	1

Appendix B – Breakdown of the MSE Results

Sampling			Param	ARIMA	TFT	Wav-ARIMA	ANFIS
3h	3h	30s	T1_1	5.831	**0.100**	0.990	0.108
			T2_1	2.992	**0.023**	0.514	0.046
			T2_4	1.159	0.204	1.071	**0.175**
			T3_1	17.709	**0.088**	0.860	0.095
			T4_1	0.531	0.167	0.192	**0.084**
3h	3h	180s	T1_1	0.995	**0.059**	0.922	0.101
			T2_1	0.938	**0.024**	0.620	0.049
			T2_4	0.758	**0.125**	0.857	0.168
			T3_1	1.246	**0.041**	0.813	0.085
			T4_1	0.257	0.089	0.192	**0.083**
3h	3h	600s	T1_1	0.921	**0.087**	1.078	0.120
			T2_1	0.260	**0.034**	0.142	0.056
			T2_4	0.739	**0.162**	1.018	0.185
			T3_1	0.670	**0.078**	0.593	0.118
			T4_1	0.197	**0.092**	0.969	**0.092**
12h	3h	30s	T1_1	1.098	**0.077**	1.205	-
			T2_1	0.392	**0.024**	0.144	-
			T2_4	0.719	**0.265**	1.11	-
			T3_1	1.513	**0.087**	0.624	-
			T4_1	0.271	**0.163**	0.216	-
12h	3h	180s	T1_1	0.774	**0.061**	1.36	-
			T2_1	0.340	**0.024**	0.170	-
			T2_4	0.681	**0.193**	1.171	-
			T3_1	0.577	**0.050**	0.591	-
			T4_1	0.214	**0.090**	0.129	-
12h	3h	600s	T1_1	0.790	**0.087**	0.964	-
			T2_1	0.196	**0.028**	0.240	-
			T2_4	0.658	**0.182**	0.350	-
			T3_1	0.538	**0.085**	0.783	-
			T4_1	0.195	**0.092**	0.208	-

Table 3: Scaled mean squared error per parameter on various samplings of the test data. A sampling is identified by a history size, a forecasting horizon, and a sampling rate (in that order). "TFT" is short for "Temporal Fusion Transformer".