Time Series Forecasting for Self-Aware Systems

André Bauer, Marwin Züfle, Nikolas Herbst, Albin Zehe, Andreas Hotho, Samuel Kounev

***Abstract*—Modern distributed systems and IoT applications are governed by fast living and changing requirements. Moreover, they have to struggle with huge amounts of data that they create or have to process. To improve the self-awareness of such systems and enable proactive and autonomous decisions, reliable time series forecasting methods are required. However, selecting a suitable forecasting method for a given scenario is a challenging task. According to the "No-Free-Lunch Theorem", there is no general forecasting method that always performs best. Thus, manual feature engineering remains to be a mandatory expert task to avoid trial and error. Furthermore, determining the expected time-to-result of existing forecasting methods is a challenge.**

In this article, we extensively assess the state-of-the-art in time series forecasting. We compare existing methods and discuss the issues that have to be addressed to enable their use in a self-aware computing context. To address these issues, we present a step-by-step approach to fully automate the feature engineering and forecasting process. Then, following principles from benchmarking, we establish a level-playing field for evaluating the accuracy and time-to-result of automated forecasting methods for a broad set of application scenarios. We provide results of a benchmarking competition to guide in selecting and appropriately using existing forecasting methods for a given self-aware computing context. Finally, we present a case study in the area of self-aware data-center resource management to exemplify the benefits of fully automated learning and reasoning processes on time series data.

***Index Terms*—Time series forecasting, self-aware computing, feature engineering, forecasting competition, time series analysis.**

I. Introduction

During the past decade, many different research communities have explored the aspects of self-awareness in computing systems, each from their own perspective. To the artificial intelligence community, the natural unit of self-awareness is the software agent; to those who study autonomic computing, it is the autonomic element. One can identify at least a dozen other research communities for which the self-awareness of a computing system is a central issue. However, the underlying commonalities in these notions of self-awareness are often obscured by the differences in the nomenclature and the lack of precise definitions.

Our notion of self-aware computing is in line with the consensus at the 2015 Dagstuhl Seminar 15041[1] (see Section II). A deployed self-aware system is continuously monitoring itself and its environment; this typically involves building series of various kinds of sensor data over time. The collected time series data can then be leveraged to enable proactive and autonomous decisions.

All authors are affiliated to the University of Würzburg, Germany {andre.bauer,marwin.zuefle,firstname.lastname}@uni-wuerzburg.de
Manuscript received Mai 31, 2019.
[1]http://www.dagstuhl.de/15041

Self-aware computing systems often implement complex prediction methods to reason about future states of the environment and how they affect the system behavior as time progresses. In nature, animals or humans have the ability called intuition to "foresee" upcoming events. In science, time series forecasting based on statistical analysis is established as an active field of research. We consider forecasting as prediction when there is an explicit time component involved. In 1970, Box and Jenkins achieved a milestone in the field by publishing their book "Time Series Analysis Forecasting and Control" [1]. Nowadays, forecasting is an established and essential pillar in many disciplines that require means to "foresee" the future by examining past observations.

Time series forecasting aims to forecast how a time series develops as time progresses. The time series may represent some monitoring data provided by a sensor, where each measurement is labeled with a time stamp. Many real-life scenarios where forecasting is needed have stringent requirements on the speed of the forecasting mechanism and the reliability (i.e., accuracy) of the provided forecasts. Also, the end-to-end process of forecast execution from feature and method selection, to data pre-processing, model building and prediction, needs to be fully automated as no human expert can be involved in the internals of a self-aware computing system. However, selecting a suitable forecasting method for a given scenario is a challenging task. According to the "No-Free-Lunch Theorem", there is no general forecasting method that always performs best. Thus, manual feature engineering remains to be a mandatory expert task to avoid trial and error. Furthermore, determining the expected time-to-result of existing forecasting methods is a challenge. We consider the challenge of constructing an automatic and generic workflow for time series forecasting that achieves robustly accurate results with reliable time-to-result as crucial for a self-aware system that relies on forecasting.

In this article, we present the major building blocks for fully-automated time series forecasting and assess state-of-the-art methods in the area. We summarize and compare existing methods, while discussing issues that have to be addressed to enable their use in a self-aware computing context. While covering a broad spectrum of relevant methods for feature engineering, feature selection, and forecasting, our goal here is not to present a general and exhaustive survey of these areas. We instead focus on outlining a step-by-step approach to fully automate the feature engineering and forecasting process. Finally, following principles from benchmarking, we establish a level-playing field for evaluating the accuracy and time-to-result of forecasting methods for a broad set of application areas, including in particular established reference scenarios for self-aware computing. We provide results of a broad benchmarking competition to guide in selecting and appropriately using existing forecasting methods.

The paper is structured as follows: We start by introducing the basic terms and definitions in the context of self-aware computing (Section II) and time series forecasting before reviewing existing forecasting methods based on statistical analysis and machine learning (Section III). We discuss the strengths and weaknesses of the individual methods and provide a brief tutorial on their practical use including R code snippets and example plots. Following this, in Section IV, we address the question of how to automatically extract and select the important features for forecasting, while applying appropriate transformations for increasing the forecast accuracy. Building on this, in Section V, we address the question of how to build a generic forecasting approach that delivers robust forecasting accuracy with a reliable time-to-result. Finally, in Section VI, we address the question of how to compare different forecasting methods in a fair manner by using suitable/reliable measures for quantifying the forecast quality. We conclude the paper by presenting an excerpt of results of a comprehensive forecast method competition and illustrate the benefits of time series forecasting in the reference scenario of a self-aware data center (Section VII).

II. Forecasting for Self-Aware Systems

"Cogito, ergo sum" (in English: "I think , therefore I am") is a philosophical proposition by René Descartes. Inspired by self-awareness in humans, there has been a lot of research on aspects of self-awareness in computing systems. Consequently, different notions and approaches have been proposed [2], [3]. In this section, we introduce the notion of self-awareness as defined by the 2015 Dagstuhl Seminar 15041[2]. Moreover, we discuss which aspects of self-awareness are prerequisites for time series forecasting and how time series forecasting can can be leveraged in the context of established reference scenarios of self-aware systems.

A. Definition of Self-Awareness

The consensus at the 2015 Dagstuhl Seminar 15041 "Model-driven Algorithms and Architectures for Self-Aware Computing Systems" was that *self-aware computing systems* have two main properties [4], [5]. They

- *learn models*, capturing *knowledge* about themselves and their environment (such as their structure, design, state, possible actions, and runtime behavior) on an *ongoing basis*; and
- *reason* using the models (to predict, analyze, consider, or plan), which enables them to act based on their knowledge and reasoning (for example, to explore, explain, report, suggest, self-adapt, or impact their environment)

and do so in accordance with *high-level goals*, which can change.

In other words, these properties enable self-aware systems to act autonomously based on their knowledge and reasoning (e.g., to explore, explain, report, suggest, self-adapt, or impact their environment). More precisely, self-aware systems are designed from the ground up to gather and maintain information about their state and their environment, and they use this information to learn models at run time and reason about their behavior. Consequently, a self-aware system is more than only applying simple rules or heuristics defined during the design phase. Further, a self-aware system is expected to not only observe and react, but also to learn, reason, and act. The term "model" in this context is used in a generic manner and includes knowledge about the self, its environment, and its goals. The goals consist of higher-level goals that are not under the direct control of the system and lower-level goals that may be generated during the learning or reasoning.

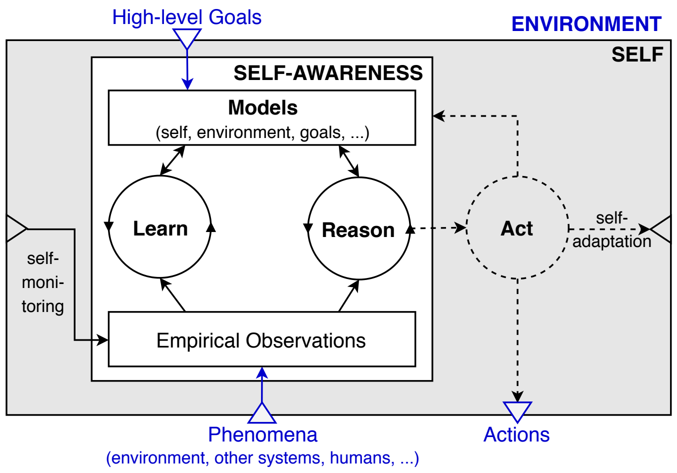

Fig. 1. Model-based learning and reasoning loop (LRA-M) [6].

Similar to autonomic systems that use the MAPE-K loop [7], the principle and structure of a self-aware system can be represented by a *LRA-M loop* (model-based learning and reasoning loop) as depicted in Figure 1. The figure shows the environment in which the self-aware system (self) is operating. Based on the empirical observations of the environment and the higher-level goals, different activities may be performed within the self. More precisely, the empirical data are used for the ongoing learning process, and the reasoning process may trigger actions affecting both the self as well as possibly the environment. Moreover, these actions can also affect the learning and reasoning activities. Although the learning and reasoning are mentioned separately, these two processes may be intertwined with each other. To sum up, self-aware systems must be able to gather necessary knowledge of what is relevant to their goals, learn models on an ongoing basis, and use the learned models to reason about this knowledge and act appropriately.

The basic idea of time series forecasting is to learn from historical data a suitable model that allows one to forecast how the data will evolve. While considering the notation of self-aware systems, the forecasting tasks can be mapped to elements of the LRA-M loop: The historical data consists of empirical observations of phenomena obtained by the sensors. In the learning process, the observations are used to train or update the parameters used for building and maintaining the forecasting model. In the reasoning process, the forecasting

[2]http://www.dagstuhl.de/15041

model can then be used to obtain a forecast on how the observations will develop as time progresses. The system may act based on the forecast and/or the current observed data.

B. Levels of Self-Awareness

As self-aware systems are inspired by the self-awareness of human beings, Lewis et al. [8] introduced in a previous work five levels of self-awareness derived from the self-knowledge of a human. In a more recent work, Lewis et al. [9] introduced three overarching levels of self-awareness that are considered in this paper: (i) *pre-reflective self-awareness*, (ii) *reflective self-awareness*, and (iii) *meta-reflective self-awareness*.

A pre-reflective self-awareness is the lowest level of self-awareness and it is a prerequisite for the higher levels. A system at this level is only aware of its sensors and is thus able to make observations. That is, a pre-reflective self-aware system lacks on *time awareness* [9]. In other words, such a system is not aware of the existence, the interaction, and the causality of historical data. Consequently, time series forecasting is not possible at this level. Therefore, pre-reflective self-aware systems are limited to trigger-based rules and can only adapt based on the current observation [9].

Reflective self-aware systems deal mainly with a relationship between the entity conducting the reflection and the entity being reflected upon [9]. More precisely, a reflective self-aware system is able to create a conceptual model of the knowledge gathered and experiences perceived. A system at this level begins as a pre-reflective self-aware system, but then starts building models of the data collected by its sensors during operation. Thus, this level allows building models on how the observations evolve over time or how observations as well as actions affect the environment. Consequently, reflective self-awareness supports time series forecasting.

The highest level is the meta-reflective self-awareness, also called *meta-self-awareness*. A system at this level possesses reflective self-awareness and beyond that, it is also aware of its self-awareness. In other words, a meta-self-aware system creates a conceptualization of the underlying reflective self-awareness and/or its associated output (i.e., the reflective model). That is, such a system can judge its behavior and can, for instance, change the learning mechanisms to achieve better quality. At this level, it is possible to change the forecasting method during run-time as the system knows due to its meta-self-awareness how well a particular method performs.

C. Reference Scenarios

The idea of self-aware computing can be applied in various domains. Kephart et al. [10] define three reference scenarios that cover a broad set of characteristics and issues that one may encounter in self-aware computing systems, while spanning different domains and a variety of scales and levels of complexity. Please note that self-aware computing systems are not limited to the enumerated reference scenarios and may be deployed in various further use cases. The first scenario focuses on an *adaptive sorting algorithm*; it exemplifies how a self-aware individual system element may adapt to changes in the data on which it operates, the environment in which it executes, or the requirements or performance criteria to which it manages itself. The second scenario is a *self-aware data center*; it exemplifies the issues of coordination, cooperation, and competition that arise within self-aware applications composed of multiple interacting self-aware elements or components. For instance, in a data center, service providers aim to optimize the application performance while minimizing the costs for using the data center infrastructure; the data center owner aims to maximize the resource efficiency and minimize energy consumption by efficiently sharing resources among different applications and services. In this context, time series forecasting can be leveraged to predict how application workloads change over time and thus make it possible to allocate and schedule resources efficiently. Indeed, forecasting provides the basis for enabling elastic resource provisioning where the system can *proactively* adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible. In Section VII, we show how time series forecasting can be employed in a self-aware data center scenario to enable proactive resource provisioning. The last scenario focuses on cyber-physical systems. In this scenario, different use cases such as a *thermostat*, *smart home*, *smart micro-grid*, and *system of autonomous shuttles* are introduced. In all these cases, time series forecasting may be beneficial. For instance, while forecasting the electricity consumption and price, the smart home or smart grid can buy the required electricity cheaply in advance and accumulate it in a battery. In the context of the autonomous shuttles, if the number of expected passengers can be forecast, either a suitably-sized shuttle or the required amount of shuttles can be planned to be available for the transport at the right time.

III. Forecasting Basics

In 2010, Peter Sondergaard (Gartner Research) stated "Information is the oil of the 21st century, and analytics is the combustion engine". To better understand how this "treasure" can be efficiently used, we introduce the basics of time series analysis and forecasting. Moreover, we provide the foundation for understanding the rest of the paper.

A. Terms and Definitions

A *univariate time series* is an ordered collection of values of a quantity obtained over a specific period or since a certain point in time. In general, observations are recorded in successive and equidistant time steps (e.g., hours). Typically, internal patterns exist such as auto-correlation, trend, or seasonal variation. Mathematically, if $y_t \in \mathbb{R}$ is the observation at time t, a univariate time series is defined by

$$Y := \{y_t : t \in T\}\,. \tag{1}$$

Especially in the context of self-aware computing systems, there are typically multiple sensors. The observations of each sensor can either be stored separately forming a univariate time series or they can be stored together to form a *multivariate time series*. In other words, a time series can also be multivariate, that is, it may have multiple observations for each point in time. Indeed, while using a multivariate time series more information can be used for the forecasting task, but most state-of-the-art methods can only forecast/predict one variable while taking the other information into account. To this end, we focus on univariate time series forecasting. This way, we do not narrow the spectrum of methods and can cover classical statistical frameworks.

1) Components of a Time Series: A time series can also be seen as a composition of trend, seasonality, cycle, and irregular components. The long-term development in a time series (i.e., upwards, downwards, or stagnate) is called *trend*. Usually, the trend is a monotone function unless external events trigger a break and cause a change in the direction. The presence of recurring patterns within a regular period in the time series is called *seasonality*. These patterns are caused by climate, customs, or traditional habits such as night and day phases. Rises and falls within a time series without a fixed frequency are called *cycles*. In contrast to seasonality, the amplitude and duration of the cycles vary over time. The unpredictable part of a time series is called *irregular component* possibly following a certain statistical noise distribution. It is also considered as the residual time series after all other components have been removed. Methods for decomposing a time series are presented in Section IV-A1.

Although not supported in many time series modeling frameworks, *bursts* [11] are considered part of the irregular component. Bursts are time-periods limited in duration with (usually positive) extreme values. As bursts can stem from planned events or be derived from known correlations, they could in some cases be predicted. On the other hand, we consider an unforeseeable burst as anomaly [12] that may be detected and removed from the time series data in order not to deflect predictions.

2) Stationarity: One of the most important characteristics of a time series is the stationarity. Hence, most statistical forecasting methods have the assumption that the time series is either stationary or can be "stationarized" through a transformation. The statistical properties (such as mean, variance, auto-correlation) of a stationary time series do not change over time. Therefore, a stationary time series is easier to model and forecast. In practice, however, time series are usually showing a mix of trend or/and seasonal patterns and are thus non-stationary [13]. To this end, time series are transformed, seasonally adjusted, made trend-stationary by removing the trend, or made difference-stationary by possibly repeated differencing (i.e., computing the differences between consecutive observations).

3) Forecasting a Time Series: While performing a forecast, one can distinguish between *one-step-ahead* and *multi-step-ahead* forecasting. As the name indicates, when performing a one-step-ahead forecast, only the next value is forecast, whereas when performing a multi-step-ahead forecast, several values are forecast at a time. The term *forecast horizon* represents the number of values that are forecast at a time.

The forecasting task itself can usually be divided into eight basic steps: (i) problem definition, (ii) data analysis, (iii) data pre-processing, (iv) feature engineering, (v) method selection, (vi) model fitting, (vii) forecasting, and (viii) evaluation. In general, the steps (iv) and (v) are done manually. However, the decisions in both steps form a crucial part. In a self-aware computing system, the last step can be leveraged to provide feedback for improving all previous steps. Indeed, at the meta-self-awareness level, the system can continuously optimize and self-improve the previous steps, reflecting changes in the observed data patterns or in the forecasting requirements. For example, an input feature may increase in relevance over time, which may trigger the selection of a different method or a combination of multiple methods as part of an ensemble forecast.

B. Forecasting Methods

As the forecasting accuracy heavily depends on the forecasting method, the choice of the most suitable forecasting method is a crucial part. According to the "No-Free-Lunch Theorem" [14], there is no forecasting method that performs best for all time series. To this end, we introduce different state-of-the-art forecasting methods.

1) sNaïve: The Naïve forecast is the simplest way of predicting future values based on historical data. More precisely, the naïve forecast repeats the latest observation of the training part for the entire forecasting horizon:

$$y_t = y_{t-1}. \tag{2}$$

In this equation, y_t is the forecast value and y_{t-1} is the value of the latest observation. *sNaïve* is an extension of *Naïve* that integrates the seasonal pattern for the forecast. That is, instead of the latest observation, the observation at the point of time exactly one period ($freq$) ago is chosen as forecast:

$$y_t = y_{t-freq}. \tag{3}$$

Due to their simplicity, these methods are typically only used as baseline methods.

2) Theta: Theta is a forecasting method that decomposes the de-seasonalized time series into short and long term components [15]. V. Assimakopoulos and K. Nikolopoulos proposed this approach in 2000 [15]. To adjust the curvatures of the time series, the authors introduce the θ coefficient. This coefficient is multiplied with the second derivative of the time series. More formally, the Theta model can be formulated as:

$$\hat{y}''_t = \theta y''_t = \theta \cdot (y_t - 2y_{t-1} + y_{t-2}). \tag{4}$$

In this equation, y_t are the observed values and $\hat{y}$ the adjusted values. A small value of θ ($\theta < 1$), which represents the long term component, flattens the time series and a high value ($\theta > 1$), which stands for the short term component, increases the curvature. V. Assimakopoulos and K. Nikolopoulos suggest creating two θ lines, that is, applying $\theta = 0$, which results in a linear regression line, and $\theta = 2$. Each θ line needs to be forecast using a proper forecast method. The linear regression line can simply be continued and exponential smoothing is applied on the $\theta = 2$ line. Then, a simple average is used to combine the two forecasts. Finally, the forecast is re-seasonalized. Although according to R. Hyndman and B. Billah [16], the Theta model is similar to simple exponential smoothing with drift, R. Hyndman provides the Theta model as a separate method in his forecast package [17].

3) ETS: One approach to forecast time series without trend component and seasonal pattern is the Simple Exponential Smoothing (SES). In 1956, R. Brown introduced this method [18]. The typical form for SES is the component form:

$$\hat{y}_t = \alpha \cdot y_t + (1-\alpha) \cdot \hat{y}_{t-1}. \quad (5)$$

In this equation, $0 \leq \alpha \leq 1$ denotes the smoothing factor, y_t the recent observation, and $\hat{y}_{t-1}$ the most recent forecast. In 1957, C. Holt extended SES with an estimate of the trend [19]. In 1960, P. Winters improved Holt's method to capture seasonality [20]. The resulting exponential methods use different ways to combine the seasonal pattern with the trend component: either additive, multiplicative, or not present. The components and the observation itself can be seen as a state. Hence, exponential methods can be seen as state space models. In 2002, Hyndman et al. [21] introduced a state framework for automatically using the most suitable method. To this end, a third component called *error* is introduced. The resulting state space model is labeled as *ETS* for Error, Trend, Season or *ExponenTial Smoothing*. Each component can be either additive, multiplicative, or not present, whereas their weightings are adjustable.

4) sARIMA: In 1938, H. Wold laid the groundwork for using ARMA models for time series [22]. An *ARMA model* is a combination of *autoregressive* $AR(p)$ model and *moving-average* $MA(q)$ model. As the term autoregressive indicates, the observed variable is represented as a linear combination of its past values. The $AR(p)$ model, where the order p determines the number of past values, is formulated as:

$$y_t = c + \epsilon_t + \sum_{i=1}^{p} \varphi_i \cdot y_{t-i}. \quad (6)$$

In this equation, y_{t-i} are the past values, φ_i the weights of the combination, ϵ_t is white noise (i.e., random vector with zero mean, finite variance, and statistically independent), and c is a constant term. In contrast to the autoregressive model, the moving-average $MA(q)$ model uses the past forecast errors to represent the observed value as a linear combination. Here, the order q determines the number of past errors. The $MA(q)$ can be formulated as:

$$y_t = c + \epsilon_t + \sum_{j=1}^{q} \Theta_j \cdot \epsilon_{t-j}. \quad (7)$$

In this equation, ϵ_{t-j} are past forecast (white noise) errors, Θ_j the weights of the combination, and c is a constant term. *ARIMA models* are a variant of ARMA models relaxing the requirement for stationary time series through differencing[3]. The latter is represented by the 'I' (for integrated) in ARIMA. The parameters that determine the ARIMA model are the order p of the AR model, the order q of the MA model, and the degree d of the differencing. The ARIMA model can be formulated as:

$$(1 - \sum_{i=1}^{p} \varphi_i B^i)(1-B)^d y_t = c + (1 + \sum_{j=1}^{q} \Theta_j B^j)\epsilon_t. \quad (8)$$

[3]Time series with trends or with seasonality are not stationary.

In this equation, the variables are the same as in the ARMA model and B is the backshift notation with $B^i y_t = y_{t-i}$. *sARIMA* is a variant of an ARIMA model that is capable of modeling seasonal patterns. To this end, each non-seasonal component of the ARIMA model is extended with its seasonal counterpart. In addition to the parameters of the ARIMA model, the *sARIMA* model [23] is specified by the parameters of the seasonal components (P, Q, and D) and the number of periods per season m.

5) tBATS: In 2011, De Livera et. al [24] introduced an extension of the exponential smoothing state space model (see Section III-B3). This extension was developed to overcome the bad performance in detecting complex seasonal patterns achieved, for instance, by ETS. For modeling complex seasonal patterns, a trigonometric representation based on a Fourier series is used. Further, the data is transformed with a Box-Cox transformation (see Section IV-C) and the error component is represented as ARMA model. This extension is called *tBATS* where the acronym stands for the key features: trigonometric, Box-Cox transformed, ARMA errors using Trend and Seasonal components.

6) ANN: There are different kinds of *ANNs* (Artificial Neural Networks) that can be used for time series forecasting.

a) Feed-forward Neural Networks: A simple ANN is a feed-forward neural network, trained with lagged values (i.e., back-shifted values) of a time series. The network consists of one hidden layer. The number of lags and the number of nodes in the hidden layer are automatically selected [17]. The model of the time series can be written as:

$$y_t = f(\Phi_l(y_{t-1})) + \epsilon_t. \quad (9)$$

In this equation, ϵ_t is the error series, f is the neural network, and $\Phi_l(y_{t-1}) = (y_{t-1}, \ldots, y_{t-l})$ is a vector containing l lagged values. This network is able to detect non-linear patterns and it may thus outperform statistical methods.

b) Other Neural Network Variations: Recently, other types of neural networks have become more and more popular in machine learning. These networks have the advantage of being able to work with most kinds of data without requiring much manual pre-processing and feature engineering. They extract features directly from the raw data. Of particular interest for time series forecasting are long short-term memory (LSTMs) [25] and convolutional neural networks (CNNs) [26]. Models based on these neural networks have shown promising results [27], [28] and specialized architectures have been proposed [29]. However, their reliance on very large amounts of training data and rather long training time makes them hard to apply in the context of self-aware computing systems.

7) XGBoost: In 2014, a scalable end-to-end tree boosting system called *XGBoost* [30] (eXtreme Gradient Boosting) was released. For a data set of n examples and m features, a tree ensemble model with k additive functions is created where each function corresponds to an independent tree structure. Depending on the target value, decision or regression trees are used. For time series forecasting, the leaves of regression trees are summed up to predict the output. To learn the functions, a regularized object is used that selects a model with simple and predictive functions. As the functions cannot be optimized with traditional methods, the model is learned in an additive greedy manner. This approach is also known as *gradient tree boosting*. To reduce overfitting, two additional techniques besides the regularized objective are used. The first technique is shrinkage, which reduces the influence of each individual tree to leave space for future trees. The second technique is feature subsampling.

8) Random Forest: Random forest is an ensemble method that consists of multiple decision trees. The construction of each decision tree is based on a certain degree of randomness. The first method towards random forests, as typically used today, was developed by T. K. Ho in 1995 [31]. He introduced the concept of using random subspaces of the feature space for decision tree generation. The so-called *random decision forests* were extended by L. Breiman in 1999 [32] who combined his version of bagging with the random feature subspace selection of T. K. Ho. It was Breiman who coined the term *random forest*.

9) SVM: In 1995, V. Vapnik et al. developed the *SVM* model (Support Vector Machines) [33]. Typical use cases for *SVM*s are classification and pattern recognition. The basic idea of using an SVM for binary classification (e.g., a positive and a negative class) is to find a linear separation line that partitions the training data into the two classes maximizing the margin between the line and the borderline cases. That is, training samples are represented by their feature vectors in a high-dimensional space and the SVM is trained to find a line where all positive training samples are on one side and all negative samples on the other side. In order to enable fitting a hyper-plane even when the points are not perfectly linearly separable, SVMs allow for some leeway by the use of *slack variables*. Analogous to SVMs, Support Vector Regression (SVR) has been proposed [34], which is based on the same principle but allows the prediction of numerical values. In this case, a threshold of ϵ is given, which defines a margin of acceptable errors for numerical predictions. The slack variables are then used to regulate the punishment for errors above this threshold.

Since it is not always possible to linearly separate the two classes, the *Kernel Trick* [35] can be used to implicitly transform the feature vector to a higher-dimensional space, where the data is linearly separable. The Kernel Trick is based on the fact that the calculations done during the optimization of the SVM only rely on the dot product between two vectors, not on the vectors themselves. Thus, if the dot product between the transformed, higher-dimensional vectors can be written in terms of the original vectors, calculating the transformation becomes unnecessary.

C. Comparison of the Forecasting Methods

Looking at the different forecasting methods with their inherent advantages and limitations, we can conclude that each method has its use cases and sweet-spots. For a given scenario, the selection of a suitable method may be determined by the following criteria [36]: (i) is the seasonal component: (a) non-existent, (b) multiplicative, or (c) additive?; (ii) is the trend component: (a) non-existent, (b) multiplicative, or (c) additive?; (iii) is the time series dominated by trend or season?; (iv) is there an exponential or linear growth?

Based on our experience and in accordance with other scientific work [37], [23], we outline the advantages and drawbacks of the described forecasting methods. The methods can be grouped into two classes: (i) statistical methods (sNaïve, Theta, ETS, sARIMA, and tBATS) and (ii) machine learning-based methods (ANN, XGBoost, Random Forest, and SVM). In general, machine learning-based methods are rather fast compared to the statistical methods, but achieve lower accuracy when trend data comes into play. Indeed, if there is enough data available the trend can be modeled, but usually time series do not cover the required time span. If a measure of the uncertainty of a forecast is required, statistical methods can be used as they provide a prediction interval (i.e., an estimate of an interval in which a future observation is expected to fall with a certain probability). Further, the statistical methods cannot handle covariates while machine learning-based methods become more accurate by leveraging this additional information. A summary of the strengths and weaknesses of all individual forecasting methods presented in this section is shown in Table I [37], [23].

1) sNaïve: As the naïve and sNaïve do not create a forecasting model, both methods provide the forecast instantly. However, both of these methods only produce static repeating patterns for long multi-step-ahead forecasting. That is, they cannot capture changes in the trend.

2) Theta: The *Theta* model is good for time series with a strong trend, but struggles with long or multiple seasonal patterns.

3) ETS: ETS is good for time series with strong trend and exhibits good performance in detecting sinus-like seasonal patterns, but is rather bad in detecting long and complex seasonal patterns. In addition, this method requires positive values.

4) sARIMA: sARIMA is used as black box-model and offers a family of models to provide accurate forecasts. Moreover, this method can handle non-stationary time series. However, the selection and identification of a model are time-consuming. That is, the runtime may be quite high and unpredictable.

5) tBATS: In contrast to ETS, *tBATS* has an improved capability to handle complex seasonality, but also requires positive values.

6) ANN: Neuronal networks are able to detect non-linear patterns. A wide variety of architectures can be used to build a model. Consequently, the networks are difficult to design. Due to their inherent black-box design, there is no direct explainability. Moreover, these networks may tend to overfitting and training is often computationally expensive.

7) XGBoost: Although *XGBoost* is a fast and accurate method, it requires many hyper-parameter settings and is sensitive to overfitting if the data is noisy.

8) Random Forest: Random Forest integrates overfitting prevention and is also able to identify complex correlations between input features and forecasting performance. However, it provides a poor explainability of the result.

9) SVM: This method is robust to small training sets and employs mathematical methods to avoid over-fitting. However, *SVM* (with an active kernel) is very sensitive to hyper-parameter settings and thus, hyper-parameter optimization is

TABLE I
STRENGTHS AND WEAKNESSES OF FORECASTING METHODS.

Method	Strengths	Weaknesses
sNaïve	+ almost no run-time + very easy to use and intuitive forecast	– provides no useful values for multi-step-ahead forecasting – captures no trend
Theta	+ good for time series with a strong trend	– cannot handle long or multiple seasonalities very well
ETS	+ good for time series with a strong trend + good for detecting sinus-like seasonal patterns	– cannot handle long or multiple seasonalities very well – requires positive values
sARIMA	+ can handle non-stationary time series + option to automatically estimate parameters	– unpredictable and high run-time for model training – insights are limited to parameters
tBATS	+ can handle complex seasonal patterns	– requires positive values
ANN	+ can detect non-linear patterns + data-driven approach	– tends to overfitting of training data – training often computationally expensive
XGBoost	+ fast run-time + accurate method	– cannot handle trend data very well – requires many hyper-parameter settings
Random Forest	+ identifies correlations between features and performance + integrates overfitting prevention	– has poor explainability of the result – cannot extrapolate trend data very well
SVM	+ use mathematical models to prevent overfitting + is robust to small data sets	– is highly sensitive to hyper-parameter settings – training often computationally expensive

essential. In addition, training is often computationally expensive.

D. Forecasting Methods R Code Snippets

After providing a brief introduction of the most common forecasting methods, we present some R^4 code snippets. The function calls shown here are explicitly held simple without many parameter settings. Before presenting the forecasting function calls, some basic code has to be executed where `train` is a vector of the original time series values and `freq` is the frequency of the time series.

```
# required packages
library(forecast)
library(xgboost)
library(randomForest)
library(e1071)

# used for statistical method training
history <- ts(train, frequency = freq)

# used for ML method training
ind <- seq(1,length(train))
period <- seq(1,length(train)) %% freq
features <- as.matrix(cbind(ind, period))

# used for ML method prediction
len <- length(train)
ind <- seq(len+1,len+horizon)
period <- seq(len+1,len+horizon) %% freq
future <- as.matrix(cbind(ind, period))
```

The following function calls are very generic and simple. Domain-specific parameter optimization can be performed for almost all methods. In contrast to the other methods, XGBoost does not offer default settings. Therefore, we only report for this method the parameters.

[4]We use R version 3.4.0.

```
# sNaive
fc <- snaive(history, h = horizon)
# sARIMA
fit <- auto.arima(history, stepwise = TRUE)
fc <- forecast(fit, h = horizon)
# ETS
fit <- ets(history)
fc <- forecast(fit, h = horizon)
# tBATS
fit <- tbats(history)
fc <- forecast(fit, h = horizon)
# Theta
fc <- thetaf(history, h = horizon)
# ANN
fit <- nnetar(history)
fc <- forecast(fit, h = horizon)
# XGBoost
fit <- xgboost(label = train, data = features,
       max.depth = 4, eta = 1, nround = 10,
       objective = "reg:linear", nthread = 2)
fc <- predict(fit, future)
# Random Forest
fit <- randomForest(y = train, x = features)
fc <- predict(fit, future)
# SVM
fit <- svm(y = train, x = features)
fc <- predict(fit, future)
```

The resulting forecasts of two example time series are shown in Figure 2 (*AirPassengers* from *R* package *datasets*) and Figure 3 (*taylor* form *R* package *forecast*). For both time series, there is no forecasting method that performs best. The machine learning methods struggle with the first time series as it has only 120 observations and has a multiplicative trend. ETS cannot handle the period length of the second time series. Random Forest, SVM, tBATS, and XGBoost have problems with the seasonal pattern of the second time series.

IV. Time Series Feature Engineering

"At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily

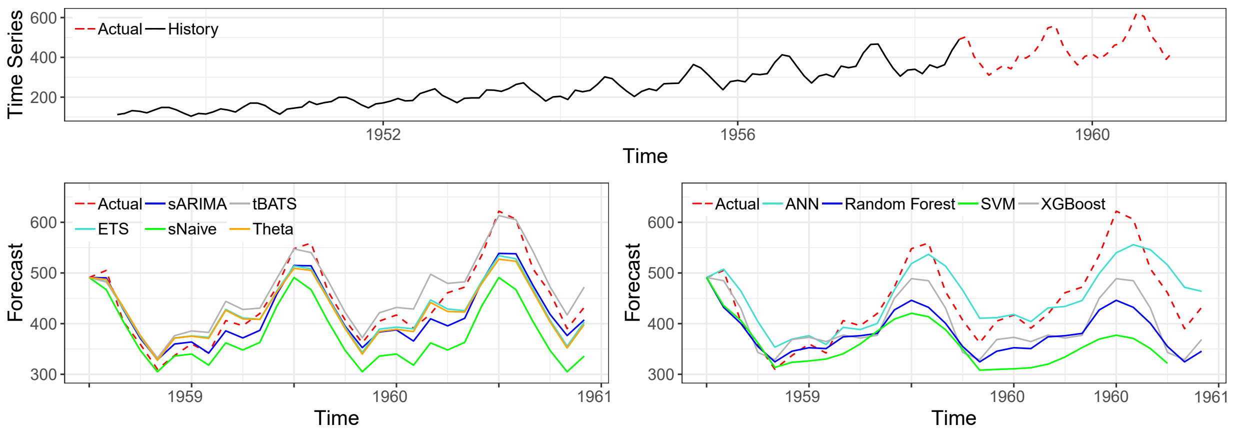

Fig. 2. Forecasts of all individual methods on an example time series with strong seasonal and trend patterns.

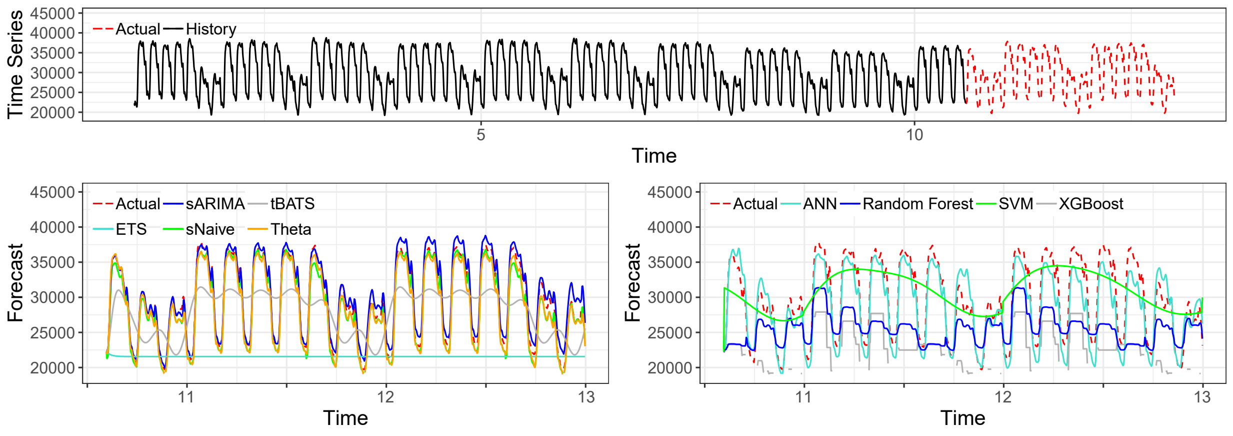

Fig. 3. Forecasts of all individual methods on a seasonal example time series with no trend.

the most important factor is the features used" [38, p. 5]. The essence of this quote from Pedro Domingos, University of Washington, is that the key to success in both forecasting and classification is solid feature engineering.

A *feature* is an individual measurable property or characteristic of an observed or extracted phenomenon that can be used for forecasting as long as it is useful for the model. Simply spoken, a feature is information (e.g., the seasonal pattern of a time series) that can help in forecasting. The choice of informative, discriminatory, and independent characteristics is a crucial step. To this end, in this section, we look at approaches to automatically extract and select the important features for forecasting, while applying appropriate transformations for increasing the forecast accuracy.

A. Intrinsic Time Series Feature Extraction

Besides the raw time series data, extrinsic and intrinsic data may help to increase the forecast accuracy. That is, more information (features) can be extracted from external correlated data sources (e.g., weather information) or from the given time series (e.g., seasonal effects).

1) Time Series Decomposition: As a time series consists of different components (see Section III-A1), a common approach is to break down the time series into its components. The parts can either be used for modifying the data (e.g., removing the trend or seasonality) or they can be used as intrinsic features. The classical *decomposition* originated in the first half of the last century. There are basically two forms of decomposition: (i) additive and (ii) multiplicative. Simply spoken, for an additive time series, the amplitude of the seasonality stays roughly the same, whereas, for multiplicative time series, the amplitude evolves with the trend. That is, if there is an increasing trend, the amplitude of the seasonal pattern increases.

The classical approach uses moving averages with the window size of the seasonality to smooth the seasonal influences for calculating the trend. However, while using this method, the first and last $m/2$ observations for the trend would be unavailable when using moving average with a window size of m. Further, classical decomposition assumes that the seasonal component does not change over time; usually, the seasonal component does not change, but in long time series changes in the seasonal pattern may occur. An improvement of the classical decomposition is the *X-11 method* [39]. This method allows to access all observations from the trend and can handle slow changes in the seasonal pattern. Based on this method, the *X-12* and *X-13* (also known as Seasonal Extraction in ARIMA Time Series) were developed [40]. However, these methods can only handle time series with monthly or quarterly observations. Nowadays, there exist different decomposition methods (such as the one from Facebook [41], which decomposes a time series into trend, season, holiday effects, and irregular) leading to diverse components (season, trend, cycle, events, linear part & non-linear part, ...) and different forms of decomposition (e.g., a multiplicative trend and an additive season). A common method that overcomes the drawbacks of the aforementioned methods is *STL* (Seasonal and Trend decomposition using Loess) [42]. Figure 4 depicts an exemplary decomposition based on this method. STL was established 1990, it can handle any type of seasonality, allows the seasonal pattern to change over time, and dissembles the given time series into the components trend T, season S, and irregular I (also called remainder):

$$Y(t) = T(t) + S(t) + I(t). \tag{10}$$

Although STL uses an additive decomposition, it can also be used for multiplicative time series by applying the logarithm:

$$\begin{aligned} Y(t) &= T(t) \cdot S(t) \cdot I(t) \text{ equals to} \\ \log Y(t) &= \log T(t) + \log S(t) + \log I(t). \end{aligned} \tag{11}$$

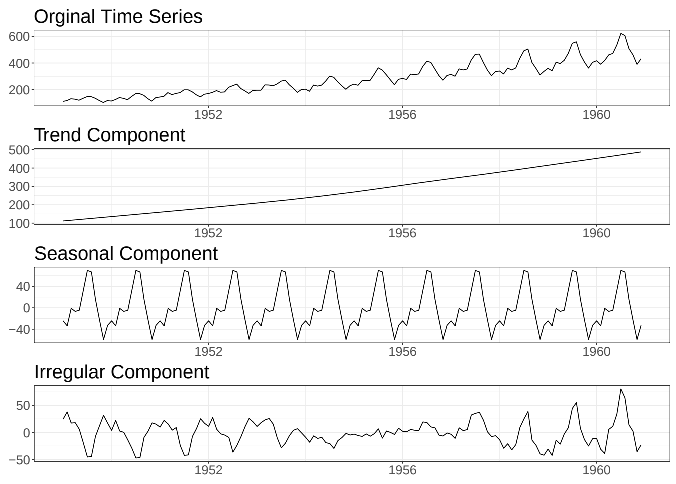

Fig. 4. Example decomposition.

2) Fourier Terms & Frequency Detection: In many fields, especially for forecasting, it is helpful to know the frequencies, i.e., the lengths of the seasonal patterns. For instance, if the most dominant frequency is unknown for a given time series, the time series cannot be decomposed (see Section IV-A1). By dominant frequencies we refer to the most frequent period or seasonal pattern such as days in the year. Also, if the dominant frequency is available, the information on the next dominant frequency (e.g., the week within the year) is helpful. Consequently and following the basic idea in mathematics to break down complex objects into more simpler parts, a time series can be represented as a weighted sum of sinusoidal components. Each component, also referred to as Fourier term, can be characterized by its coefficient and frequency.

A powerful mathematical tool for frequency analysis is the Fourier transformation. More generally, the *Fourier transformation* decomposes a function of time into its compound frequencies. That is, the Fourier transformation gives insight into the time series at different frequencies, but does not provide information on how the time series evolves over time. When using the Fourier transformation, the distribution of the frequencies in a time series or the *spectral density* of the time series can be determined. The spectral density assigns significance of different frequencies in time series data to identify any intrinsic periodic signals.

Based on information from the Fourier transformation, we can: (i) derive (at least) the dominant seasonal patterns from the time series (see Section IV-A1) and (ii) extract for each frequency its Fourier term. These terms allow, on the one hand, to approximate the original time series with the sum of the relevant terms. On the other hand, the relevant Fourier terms can be used as additional information to fit a better forecasting model.

However, there are time series that cannot be "nicely" decomposed into sinusoidal components. Consequently, an estimator for the spectral density is required. To this end, A. Schuster [43] introduced the mathematical tool *periodogram* in 1899. An example outcome is depicted in Figure 5, where the horizontal axis shows the frequencies and the vertical axis the spectrum. The most dominant frequency in this example is 12 and every multiple of 12 is dominant. The periodogram calculates the spectrum for many different possible frequencies by using Fourier transformations. Theoretically, it is possible to find sines and cosines with the frequencies $\nu_k = k/T$ in a time series with the length $T/2$ and $k \in 1, \dots, T$. While using the frequencies ν_k, the spectral density and periodogram are identical [44]. The first step is to calculate the discrete Fourier transformations $X(\nu_1), \dots, X(\nu_T)$:

$$\begin{aligned} X(\nu_k) &= \frac{1}{\sqrt{T}} \sum_{t=1}^{T} e^{-2\pi i t \nu_k} y_t \\ &= \frac{1}{\sqrt{T}} \left(\sum_{t=1}^{T} \cos(2\pi t \nu_k) y_t - i \sum_{t=1}^{T} \sin(2\pi t \nu_k) y_t \right). \end{aligned} \tag{12}$$

Then, the periodogram calculates the squared absolute value of the discrete Fourier transformation as follows:

$$\begin{aligned} I(\nu_k) &= |X(\nu_k)|^2 \\ &= \frac{1}{T} \left| \sum_{t=1}^{T} e^{-2\pi i t \nu_k} y_t \right|^2 \\ &= \frac{1}{T} \left(\left(\sum_{t=1}^{T} \cos(2\pi t \nu_k) y_t \right)^2 + \left(\sum_{t=1}^{T} \sin(2\pi t \nu_k) y_t \right)^2 \right). \end{aligned} \tag{13}$$

B. Feature Selection

Besides the actual time series values, additional information (e.g., information derived by the intrinsic feature extraction, see Section IV, or external features) can be considered to build a forecasting model. However, in forecasting, as well as in classification, it is often desirable to reduce the number of

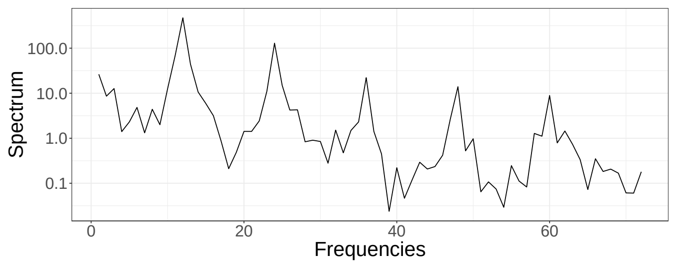

Fig. 5. Example periodogram with dominant frequency 12.

features that a model receives as input. Focusing on the most important features has several advantages: First, it can prevent the model from overfitting by removing features that help to predict single examples very well, but fail to generalize to others. Second, reducing the number of input variables/features can significantly speed up the training and prediction of the model leading to better time-to-result. Finally, knowing which features are most important can improve the understanding of the underlying data and can thus help to derive a better forecasting model. There exists a multitude of methods for assessing the importance of individual features and selecting the most important ones. Our goal in this section is not to provide a complete overview over the field of feature selection (which can easily fill a separate paper), but to convey the main ideas and provide some commonly used examples, which are relevant for time series forecasting. For a more detailed overview, we refer to [45]. The following methods can be roughly categorized into three groups:

1) Statistical Feature Selection: This group uses statistical analysis to determine the influence of features on the target value, that is, the observed values recorded in a time series. To this end, an arbitrary measure for the relatedness of two or more variables can be used. The general procedure is to calculate the relatedness of all variables to the target value and select the n most important ones. To the best of our knowledge, there is no general rule of thumb for selecting the "best" value of n, that is, the number of features for building the forecast model. In practice, the number of features can be determined by evaluating different sizes of feature sets and then deciding on a number based on the specific requirements for training time and model accuracy. To assess the influence of a feature, there exist different methods. In the following, we describe the Coefficient of Determination R^2 as an example and give a brief overview of other measures that can be used for statistical feature selection.

a) Coefficient of Determination: One common measure is the *Coefficient of Determination* R^2 that describes how much of the variance in an output variable can be explained from an input variable by a linear function:

$$R^2 = 1 - \frac{\sum_i (x_i - y_i)^2}{\sum_i (y_i - \overline{y})^2}. \tag{14}$$

In this equation, x_i are the values of the input variable, y_i the values of the target variable and $\overline{y}$ is the average of all y_i. The R^2 measure typically lies in the range between 0 and 1, where 1 means that the input explains all variance in the output and 0 means that it cannot explain any of the variance. Technically, it is possible that the R^2 measure is negative. However, this would indicate that a model using variable x_i for prediction would perform worse than a model that always predicts the mean value of y, which is unlikely. Thus, higher values of the R^2 measure for a feature indicate a more important feature.

b) Other Statistical Measures: Simple alternatives are correlation measures such as Pearson correlation [46], Spearman rank-correlation [47] or others [48]. A more sophisticated measure comes from information theory and is called *Mutual Information* [49]. This measure captures the joint probability distribution and the product of the marginal probability distributions of two variables. Similarly, Gini Impurity can be used to select important features, but requires discretization of the variables [50]. Another method that is able to deal with multiple input variables is the *Analysis of Variance* (ANOVA). The ANOVA was developed by Ronald Fisher [51] and calculates for a set of variables their individual as well as joint importance for the output.

2) Model-internal Feature Selection: In contrast to the first group, this feature selection is made a-posteriori. That is, we train one of the following models that allow us to extract the importance of individual features from their internal parameters once they have been trained.

a) Linear Models: In linear classifiers like Support Vector Machines without kernels and the related regression methods (see Section III-B9), the absolute value of the weight w_i for feature i can be used directly as a measure of the importance [52]. Consequently, it is possible to rank features by their weights w_i and select the most relevant features according to this ranking. Note that, to use the weights of the features as a measure for the importance of a feature, the input variables have to be normalized. Otherwise, the scale of the variables could greatly influence the weights that are assigned by the model.

b) Tree-based Models: Tree-based methods provide an explicit ranking of the input features, as they rely on some of the aforementioned statistical feature weighting methods, or similar methods [53]. That is, a decision tree is constructed by repeatedly splitting the data set according to the value of a variable. To select the variable for the next split, features are weighted by their importance according to a statistical measure of "impurity" for the resulting split (e.g., Entropy or Gini Impurity [54]). The criterion for the split is the variable that brings the largest reduction in impurity. Intuitively, this can be thought of as follows: Given the dataset D and the subsets D_1 and D_2 ($D_1 \cup D_2 = D$) that result from splitting the dataset according to one variable, how much better would a majority classification, that is, assigning the most frequent label in the (sub-)dataset, work on D_1 and D_2 compared to D? Formally, this is defined as the difference I_{diff} between the impurity I_D in D and the sum of impurities in D_1 and D_2:

$$I_{\text{diff}} = I_D - I_{D_1} - I_{D_2}. \tag{15}$$

The splitting is repeated until a predefined threshold is met; that is, the remaining features are not important enough to help much in the prediction. Although this method is described for classification, it can be easily adapted to regression tasks by dividing the values of the output variable into a set of "buckets". The features can then be selected by taking the variables that are used in the n layers of the decision tree closest to the root. As a more fine-grained alternative, the reduction of the impurity I_{diff} in the data set achieved by splitting it according to a variable can be directly used to rank the features. Similar to decision trees, this procedure can also be applied to random forests [55].

3) Wrapper Methods: This last group is sometimes referred to as *wrapper methods*, as they are based on wrapping other models in a feature selection loop [56]. In other words, an inner model is repeatedly trained and evaluated, varying the subset of features selected as input. That is, the best feature set can be selected that leads to the best accuracy when trained on the training data and evaluated on a separate validation data (that is disjoint from both the training and test data). When having only a small feature set, an exhaustive search can be applied. However, for a large number of features, sophisticated methods are required. In general, there exist two different approaches: (i) Starting from an empty subset and adding features (*Forward Selection*) or (ii) starting from the full set and removing features (*Backward Elimination*). Both approaches work iteratively until some stopping criterion (e.g., a maximum training time, a minimum prediction quality or a maximum/minimum number of features) are met.

a) Greedy Best First Search: One simple forward selection strategy is the Greedy Best First Search (GBFS) [57], [58]. It is based on repeatedly adding the single best feature to an initially empty subset of features until a stopping criterion is met. Since GBFS is a greedy algorithm, it may not find the best overall feature subset, but can often be used as an estimation of a good feature set.

b) Simulated Annealing: A more sophisticated strategy is *simulated annealing* initially used as a method for optimizing arbitrary complex functions [59]. Generally, it can be used as both forward selection and backward elimination method or even starting from a random subset of features. Since training on fewer features is faster, we use this method for forward selection. This allows the method to evaluate more feature subsets in the same time compared to using it as a backward elimination method. The high-level idea of simulated annealing is to choose an initial element from the search space and then select a "neighbor" of the current element until some criterion is met. In the context of feature selection, the search space is the set of all possible feature subsets $S \subseteq F$, where F is the set of all available features. The quality of S is then evaluated by training the specific method on this subset of features and evaluating its prediction quality. At each step, the algorithm randomly decides whether or not to select the currently evaluated neighbor S' based on the prediction quality of S and S', where a higher prediction quality of S' leads to a higher probability of selecting S'. The probability of selecting a neighbor also depends on the current progress of the optimization, usually making the model less likely to select a neighbor that decreases the performance towards the end of the optimization. This is sometimes implemented using a *temperature* parameter[5] or the current iteration count. Note that simulated annealing has a chance to select a neighbor that performs *worse* than S. This is intended, as it can prevent the algorithm from getting stuck in local maxima. More formally, simulated annealing can be expressed in pseudo-code as shown in Algorithm 1.

```
Data: Set of all features $F = \{f_1, ..., f_n\}$, method $m$
Result: A subset of features $S$
Initialize the feature set as $S = \emptyset$;
for $k \in 1, ..., k_{max}$ do
    S' = neighbor(S);
    select_prob = P(score$_m$(S), score$_m$(S'), k);
    if select_prob $\geq$ random(0,1) then
        S = S';
    end
end
return $S$;
```

Algorithm 1: Simulated Annealing

A neighbor of a state S is any state $S' \subseteq F$ where only one feature is removed or added. The function `neighbor(S)` returns one randomly selected neighbor of S. `score`$_m$(S) returns the prediction quality of method m when trained and evaluated on S and $P(\texttt{old}, \texttt{new}, k)$ is defined as follows:

$$P(\texttt{old}, \texttt{new}, k) = \begin{cases} 1, & \texttt{new} > \texttt{old} \\ e^{k * \frac{\texttt{new}-\texttt{old}}{|\texttt{old}|}} & \text{else.} \end{cases} \tag{16}$$

c) Recursive Feature Elimination: This method is a backward elimination strategy and is basically a reversed version of the Greedy Best First Search described in Paragraph IV-B3a. *Recursive Feature Elimination* (RFE) starts from the full feature set and, in each iteration, removes the least important features. This procedure requires a regressor that provides an estimate of a feature's importance, for example, linear regression or decision trees.

4) Comparison of Feature Selection Techniques: One advantage of the statistical methods is that they do not rely on training a method first, but can be directly calculated from the data. On the other hand, the features selected by these methods may not be optimal for all models, as they only capture how well the model that is the basis of the statistical measure can predict the target value. Thus, a feature subset can be selected that does not perform well for the final forecasting model.

The model-internal methods can partially address this issue. Since they extract features considered particularly important directly from a model (e.g., linear regression or random forests), it can be assumed that these features are indeed the most helpful for this kind of model. However, the selected features are only helpful when the same type of model is used for feature selection and prediction: for instance, the features weighted highly by a linear regression may not be the most helpful for a random forest model. As these methods are not applicable for all forecasting methods and it may be quite time-consuming, they may not be suitable in all situations.

[5]The temperature provides the analogy to annealing in metal work, explaining the name of the method.

The forward search wrapper methods have the advantage of being applicable to all types of forecasting methods and, therefore, being able to extract the most relevant feature subsets in any situation. However, the usage of these methods is associated with a potentially very high extraction time: The wrapper methods rely on training a forecasting method repeatedly on varying subsets of features, which may not be computationally feasible. The same issues apply to backward elimination strategies, which additionally require models that provide an estimate of their features' importances.

Therefore, when applying feature selection techniques, a trade-off between the additional time-wise cost of wrapper methods against the potentially worse, but faster, statistical methods has to be considered.

C. Feature Transformation

Data may be quite complex, e.g., having high variance and/or multiplicity within a time series, an adjustment or simplification of this data can improve the forecasting model. To this end, there exist different methods that transform time series. Daily life examples are currency exchange rates (e.g., Euro into US dollar). However, this example is a linear transformation and does not affect the data complexity. More precisely, the data distribution is not changed. In practice, non-linear transformations are used. For instance, a common and useful transformation is to apply the logarithm as it reduces both variance and multiplicative effects. Although this method may improve the forecasting model, the transformed data may not be normally distributed, so the improvement may not reach its full potential. The Box-Cox transformation [60] tries to transform the data into "normal shape". To this end, this transformation offers logarithm and power transformations. The Box-Cox transformation is defined as follows:

$$w_t = \begin{cases} \ln(y_t) & \text{if } \lambda = 0 \\ (y_t^{\lambda} - 1)/\lambda & \text{otherwise} \end{cases} \tag{17}$$

where w_t is the transformed time series, y_t the original time series, and λ the transformation parameter that determines the function.

Based on this transformation, forecasts can be conducted. Note that the forecasts have to be re-transformed to be in the right scale. To re-transform the time series, the same λ is required:

$$y_t = \begin{cases} \exp(w_t) & \lambda = 0 \\ (\lambda w_t + 1)^{1/\lambda} & \text{otherwise.} \end{cases} \tag{18}$$

As the transformation and therefore the accuracy of the forecasting model depends on the the transformation parameter λ, V. M. Guerrero [61] proposes a method that estimates the best λ by minimizing the coefficient of variation for the time series.

D. Feature Engineering R Code Snippets

After providing a brief introduction of feature engineering, we present some R^6 code snippets for extracting intrinsic features and a wrapper method with an exhausting search. Before presenting the snippet, some basic code has to be executed where `train` is a vector of the original time series values, `freqs` is a list of all the frequencies sorted descending regarding their dominance of the time series, `ind` is an index for dividing the time series into training and validation set, and `end` is the length of the time series.

```
# required packages
library(forecast)
library(ggm)

# used for statistical method training
his <- msts(train, seasonal.periods = freqs)
# get the season, trend, and remainder
his.stl <- stl(his, s.window = 'periodic',
                robust = TRUE)$time.series
# get the Fourier term for each frequency
fou <- fourier(his, K = rep(1,length(freqs)))

# build the feature combinations
features <- cbind(his,fou,his.stl)
s.feat <- powerset(1:ncol(features))

acc <- c()

# wrapper with exhausting search
for(i in 1:length(s.feat)){
  feat <- as.matrix(features[,s.feat[[i]]])
  model <- nnetar(his[1:ind],
                  xreg = feat[1:ind,])
  fc <- forecast(model,
                 xreg = feat[(ind+1):end,])
  # get MASE based on validation data
  acc[i] <- accuracy(fc,
                     his[(ind+1):end])[12]
}

# get features with lowest MASE
best.set <- s.feat[[which(acc == min(acc))]]
```

V. Advanced Forecasting

In 1997, the "No-Free-Lunch Theorem" for optimization algorithms [14] was formulated. In short, there is not a single algorithm that performs best for all scenarios since improving the performance of one aspect normally leads to a degradation in performance for some other aspect. In fact, this theorem also applies to time series forecasting. Through the diversity of time series, the tuning of one forecasting method on a given time series would result in a worse forecast on another time series. Taking the inherent drawbacks and limitations of forecasting methods into account (see Section III-C)) and keeping the "No-Free-Lunch Theorem" in mind, it can be concluded that there is no monolithic forecasting method that performs best for all kinds of time series.

In this section, we discuss the challenge: How to build/find a generic forecasting approach that delivers robust forecasting accuracy? Robust here means that the variance in forecasting results should be reduced, not necessarily improving the forecasting accuracy itself. In academia, many hybrid or ensemble forecasting mechanisms and forecasting method recommendation systems have been proposed to face the "No-Free-Lunch Theorem", that is, minimizing the variance

[6]We use R version 3.4.0.

of monolithic forecasting methods. As the combination of different forecasting methods may increase variance in time-to-result, we pose ourselves the challenge: How to build a generic forecasting approach that has a reliable time-to-result? To this end, we design an automatic forecasting workflow aiming to have robustly accurate forecasts with reliable time-to-result.

A. Hybrid and Ensemble Forecasting

To overcome the drawbacks of individual forecasting methods, many hybrid and ensemble forecasting methods have been developed. These approaches combine at least two individual forecasting methods. In terms of ensemble forecasting [62], [63], [64], n individual forecasting methods m_i, $i = 1, \ldots, n$, are applied on the same time series resulting in n individual forecasts F_{m_i}. Each of these forecasting methods m_i has a weight w_i assigned. By definition, the weights sum up to one:

$$\sum_{i=1}^{n} w_i = 1. \tag{19}$$

There exist many different approaches for weight assignment, ranging from simply equal weights to highly sophisticated algorithms. In the case of unequal weights, they can either be learned in an offline pre-processing phase or based on the training error of each specific time series. The final forecast F is calculated as a weighted sum of all individual forecasts F_{m_i}:

$$F = \sum_{i=1}^{n} w_i \cdot F_{m_i}. \tag{20}$$

Using ensemble forecasting reduces the variance in forecasting performance and thus, the risk in trusting one individual forecasting method. This characteristic is often called *robustness*. However, the quality of the forecast highly depends on the weight assignment algorithm. Also, the run-time of such ensemble methods is comparably high since all individual forecasting methods m_i need to be applied.

In contrast to ensemble forecasting, hybrid forecasting methods aim at explicitly compensating for disadvantages of specific individual forecasting methods by adding other methods that are strong in the respective aspects. There exist two common hybrid forecasting strategies: (i) applying multiple individual forecasting methods sequentially on the remainder of the previous method [65], [66] and (ii) decomposing the time series and applying a certain individual forecasting method for each component [67], [68], [69]. Let Y be the time series used to train the forecasting method m. Further, let m consist of n individual forecasting methods, m_1 to m_n. The trained forecasting model of m_i on Y is formulated as $T_{m_i}(Y)$. Now, strategy (i) can be described as a step-by-step modeling of Y:

$$\begin{aligned} Y &= T_{m_1}(Y) + \hat{Y}_1 \\ \hat{Y}_i &= T_{m_{i+1}}(\hat{Y}) + \hat{Y}_{i+1}. \end{aligned} \tag{21}$$

In this equation, $\hat{Y}_i$ represents the remainder, that is, the part of the time series that was not captured by the forecasting methods m_1 to m_i. The final forecast is calculated as a simple sum of the forecasts F_{m_i} of all individual methods m_i:

$$F = \sum_{i=1}^{n} F_{m_i}. \tag{22}$$

Of course, using many forecasting methods (i.e, a high n), can lead to overfitting of the training data. Thus, a trade-off between modeling the training data perfectly and keeping the set of models robust is required. A typical selection of forecasting methods is to use one statistical method first (e.g., sARIMA), followed by one machine learning technique (e.g., ANN). On the contrary, strategy (ii) can be formulated as:

$$Y = \sum_{i=1}^{n} \tilde{Y}_i \tag{23}$$

where $\tilde{Y}_i$ represents a component (e.g., trend) of the time series. The decomposition does not necessarily need to be additive. For instance, multiplicative decompositions are possible too. Each component $\tilde{Y}_i$ is then modeled using a particular forecasting method m_i resulting in a trained model for each time series component. Again, the final forecast is a simple sum of the forecasts F_{m_i} of all individual methods m_i:

$$F = \sum_{i=1}^{n} F_{m_i}. \tag{24}$$

Typical decomposition methods are seasonal and trend decomposition using Loess (STL) [42], wavelet decomposition [67], and empirical mode decomposition [68].

B. Forecasting Method Recommendation

Another approach to avoid the downsides of individual forecasting methods is to use recommendation systems. A variety of recommendation systems for time series forecasting methods exists ranging from manually created expert systems [70] to dynamically and automatically learned rules [71], [72], [73]. For automatic rule learning, typically various time series characteristics, referred to as *meta-features*, are calculated. Then, rule generation algorithms (e.g., decision trees or classifiers) are applied to capture dependencies between the meta-features and the best performing forecasting method. To recommend a forecasting method for a new time series, the meta-features are calculated and the rules or classifiers are applied to them. However, some challenges for forecasting method recommendation are the representativeness of the data set used for learning, the choice of meta-features, and the choice of rule generation algorithm or classifier. A less representative data set leads to a poor generalization of the rules, while a bad selection of meta-features and rule learners results in the relationship between time series characteristics and the performance of the forecasting method being missed.

C. Telescope: Automated Forecasting Workflow

The assumption of data stationarity is an inherent limitation for time series forecasting. Any time series property that eludes stationarity, such as non-constant mean (trend), seasonality, non-constant variance, or multiplicative effect, poses a challenge for the proper model building. Consequently, we take all the techniques and information discussed in this article into account to design an automated forecasting workflow called Telescope that addresses these issues. Telescope automatically transforms the time series, derives intrinsic features from the time series, selects a suitable set of features, and handles each feature separately. Figure 6 shows the high-level Telescope forecasting workflow. The white rounded rectangles represent data or information, the blue boxes are actions, the green parallelograms mark features, and the grey rounded rectangles are the target variable. Although the diagram seems to be sophisticated, the procedure can be split into 5 phases: (i) pre-processing (*Frequencies Estimation*), (ii) feature engineering (*Box-Cox Transformation*, *Fourier Series*, *Decomposition*, and *Remove Trend*), (iii) model building (*XGBoost Train*), (iv) forecasting (*Continue Pattern*, *ARIMA Forecast Trend*, and *XGBoost Predict*), and (v) post-processing (*Add Trend* and *Inv. Box-Cox Transformation*).

In general, self-aware systems are governed by human interactions. That is, time series produced or observed by these systems are subjected to human habits and are thus seasonal. Therefore, most of the mentioned actions assume a seasonal time series. In the unlikely case where no seasonality exists within a time series, the forecasting method has a fallback and makes a forecast with the ARIMA model.

1) Pre-processing: The algorithm estimates the most dominant frequencies from the time series (i.e., the lengths of the most significant periods) by applying periodograms (see Section IV-A2) on the data. The Telescope forecasting workflow iterates over the found frequencies and matches each frequency with reasonable frequencies (e.g., daily, hourly, and yearly). If a frequency matches a reasonable frequency with a tolerance ($< 5\%$), this frequency is considered. We assume that reasonable frequencies match multiples of natural time units. The most dominant frequencies are derived by putting the likely (the threshold is greater or equal than 50% of the most dominant one) reasonable frequencies to a set.

2) Feature Engineering: Forecasting methods, especially machine learning methods, struggle with changing variance and multiplicity within a time series [74]. To this end, the pre-processed time series is adjusted by applying a Box-Cox transformation (see Section IV-C). This step reduces both variance and multiplicative effects of the time series. That is, the adjustment results in a simpler model, which leads to an improved forecast model. As the Box-Cox transformation depends on the transformation parameter, the parameter is estimated by the method proposed by Guerrero [61] and restricted to values greater than or equal to zero.

Although most forecasting methods assume stationary time series (i.e., the mean and variance of a time series do not change over time), many time series exhibit trend or/and seasonal patterns. That is, in practice, time series are usually non-stationary [13]. To tackle the non-stationarity, the Telescope forecasting workflow decomposes the time series and then handles each part separately. The trend is removed to make the time series trend-stationary and the seasonality is used as an intrinsic feature. To this end, the adjusted time series is split with STL (see Section IV-A1) and the most dominant frequency into its components: trend, seasonality, and remainder.

As time series may have multiple seasonal patterns (such as daily and weekly), Fourier terms (see Section IV-A2) for each dominant frequency found in the pre-processing step are extracted from the adjusted time series for modeling the different patterns.

3) Model Building: After the feature engineering step, several features are available that can be considered for building a forecasting model. As stated before in Section IV-B, there are several methods that propose the best subset of the features. To have a reliable and timely forecast, this approach avoids using such methods due to their inherent long and unpredictable run-times. That is, the approach uses semi-automatic rules for selecting the best feature set.

Firstly, as a strong trend both increases the variance and violates stationarity, the trend was removed during the feature engineering step and the time series is now trend-stationary. That is, the considered features include seasonality and Fourier terms. Consequently, the target value corresponds to the adjusted, de-trended time series. Although seasonality can also violate stationarity, time series models usually explicitly take seasonality into account. Also, machine learning methods are suitable for pattern recognition. To this end, we keep the seasonality as a feature. Secondly, the remainder of the time series is not explicitly considered a feature. That is, the machine learning method learns the remainder as the difference that is missing to fully recreate the target value. We apply XGBoost as machine learning method due to its success at Kaggle challenges[7]. A further motivation is that XGBoost supports a watchlist preventing overfitting and is time efficient.

4) Forecasting: To forecast the adjusted time series, each feature and the trend has to be forecast separately. As the seasonality and the Fourier terms are recurring patterns per definition, these features can merely be continued. Based on the trend component, an ARIMA model without seasonality is determined that forecasts the future trend of the time series. We select ARIMA as it is able to estimate the trend even from a few points and we use an automatic version that selects the most suited model [17].

To predict and assemble the future adjusted time series, the same features are required as used for the model building (see V-C3). That is, the forecast patterns of the seasonality and Fourier terms in combination with the model are used to predict the future de-trended, adjusted time series. Then, the forecast trend is appended to the prediction to create the future adjusted time series.

5) Post-Processing: As the time series was adjusted with the Box-Cox transformation, the forecast adjusted time series has to be re-transformed. To this end, the inverse Box-Cox transformation with the identical transformation parameter from the pre-processing is applied to the forecast adjusted time series. After this re-transformation, the forecast of the original time series is available.

[7]XGBoost winning challenges: https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

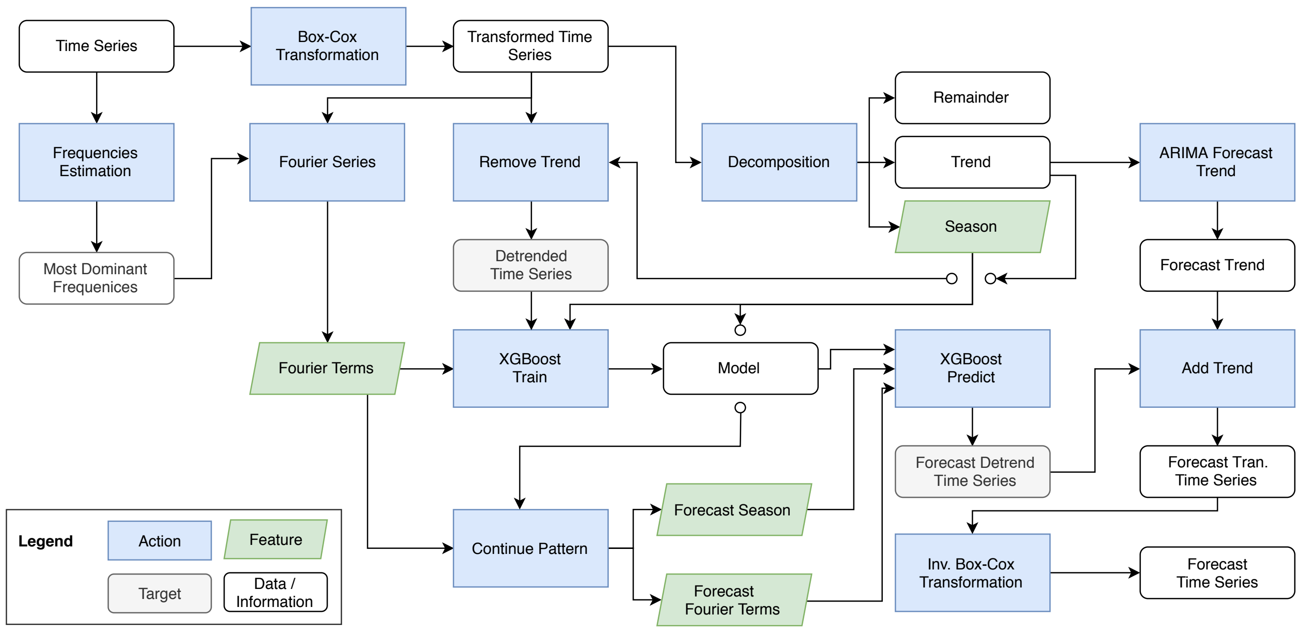

Fig. 6. The Telescope automated time series forecasting workflow.

VI. Forecasting Methods Competition

As time series forecasting is an essential pillar in many decision-making fields, there is a broad range of academic work concerning forecasting. To decide which method is the most suitable for a specific problem, one normally has to trust the performed evaluation of the authors. In order to investigate how trustworthy and meaningful these state-of-the-art evaluations are, in this section, we review scientific papers matching the terms *time series forecasting*, *time series analysis*, or *time series prediction*. The papers were collected from the search engines: Google Scholar, Mendely, IEEE Xplore, and Semantic Scholar. We filtered and considered only papers that were published during the last 40 years and contain an evaluation section. From these, we select papers that have received an average of at least 8.5 citations per year (Google Scholar) resulting in a data set containing 100 scientific papers.[8] The found papers were published between 1982–2019 and were cited between 29–2440 times. Moreover, the selected papers cover different topics, for example, supply chain management, river flow, tourism, traffic, stock prices, electric/power demand, and many more. In our view, automated time-series analysis is a central building block of self-aware systems as a general concept. To this end, our review is not limited to papers that are directly related to self-aware systems.

We are interested in the following questions: *(I) How many time series were used? (II) How many forecasting methods were evaluated? (III) How many measures were considered for the evaluation?* Figure 7 shows the distribution as a box plot of how many time series, forecasting methods, and measures were used in the reviewed papers. The median, that is, 50% of the reviewed papers used no more than three time series,

[8]The list of reviewed articles is available at https://doi.org/10.5281/zenodo.3716035

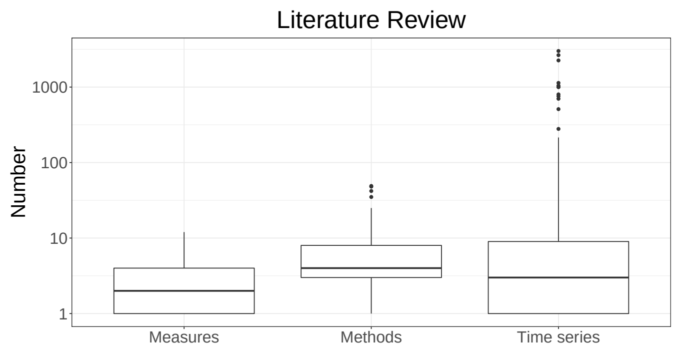

Fig. 7. Distribution measures, methods and times series in the evaluations of the selected TOP 100 scientific papers on time series analysis.

considered maximal four methods and used maximal two measures for formulating academic results. Even 75% of the studied papers used at maximum nine time series, maximal eight forecasting methods and maximal four measures. There are papers, which are outliers and depicted as points in the figure, that used more than 1000 time series. However, these articles used the M-competitions or the Watson macroeconomic database and therefore the considered time series have a high degree of similarity. For the ranking of the methods, almost all papers consider MAPE, RMSE, or related error measures, but do not investigate statistic measures of the error values like standard deviation. Moreover, none of the observed papers takes the time-to-result into account as part of their evaluation. No information is provided on the run-time of forecasting methods. Often in offline scenarios, predictions are not time-critical. However, especially when used in the context of self-aware computing systems, forecasts have strict deadlines to allow timely and reliable planning.

Based on our review, we can conclude that most papers only compare a small set (mostly related) methods in their empirical evaluation while considering only the forecast error and not the time-to-result. Consequently, these evaluations are not very helpful to compare/choose a suitable forecasting method. To this end, we pose ourselves the questions: How to compare different forecasting methods in a fair manner? and What are suitable/reliable measures for quantifying the forecast? Our idea is to establish a forecast benchmark that enables a fair, meaningful, and reliable comparison of forecasting methods.

A. *Benchmarking in a Nutshell*

Quoting the Scottish engineer, mathematician, and physicist William Thomson (1824-1907) - also known as Lord Kelvin: "To measure is to know... If you cannot measure it, you cannot improve it.", we consider benchmarks as a key instrument for improvement and competition. A benchmark is a tool for the evaluation and comparison of systems, components or methods with respect to specific characteristics, such as performance, reliability, or security [75]. In the context of forecast methods, key concerns among others are accuracy and time-to-result.

Benchmark designers must balance several, often conflicting, criteria to be fair and successful. Several factors must be taken into consideration, and trade-offs between various design choices will influence the strengths and weaknesses of a benchmark. When developing a new benchmark, the goals should be defined so that choices between competing design criteria can be made in accordance with those goals to achieve the desired balance.

The key characteristics can be organized in the following groups:

- **Relevance** How closely the benchmark behavior correlates to behaviors that are of interest to users.
- **Reproducibility** Producing consistent results when the benchmark is run with the same test configuration.
- **Fairness** Allowing different test configurations to compete on their merits without artificial limitations.
- **Verifiability** Providing confidence that a benchmark result is accurate.
- **Usability** Avoiding roadblocks for users to run the benchmark in their test environments.

Each benchmark is composed of three key elements: (I) metrics[9], (II) workload and data set definition, and (III) measurement methodology (also known as run rules). The measures determine what values should be derived based on measurements to produce the benchmark results. The workload and data set definition determine under which usage scenarios and conditions measurements should be performed to derive the measures. Finally, the measurement methodology defines the end-to-end process to execute the benchmark, collect measurements, and produce the benchmark results.

B. *Measures*

Basically, a forecast can either be evaluated a-priori or a-posteriori. As an a-posteriori evaluation requires the future values (typically data that is not available at the time of the forecast), the accuracy of a forecast can only be quantified afterward. In contrast, for a-priori evaluation, the forecasting method is evaluated before the real forecast is performed. Consequently, an estimator for the forecast error is required. One solution is to use the fitted model error as an indicator for the forecast error. However, this approach is not reliable because of, for instance, overfitting. That is, a method could perfectly match the historical data, but by doing so, the method loses its predictability and is not able to capture future values. To tackle such issues, V. Vapnik [33] introduces a complex theory that allows setting an upper bound for the forecast error. Expressed in a very simplified manner, this theory implies that:

$$\text{forecast error} < \text{model error} + \text{structural risk}. \qquad (25)$$

In other words, the forecast error is bounded by the model error of the fitted model and the theoretical risk that considers the model complexity due to the accompanying overfitting threat. Actually, the estimation of the risk is hard as methods are quite sophisticated or worked as black-box. A more common practice is to split the time series into a *train set* and *test set*. The train set is used to fit the model. Based on this model, a forecast is performed and then compared against the test set. As the test data is not used for the model fitting, this practice should deliver a reliable indicator. Typically, a time series is split into 80% train and 20% test. Note that, we use both forecast accuracy and forecast error as terms for quantifying forecasts: the lower the error or the higher the accuracy, the better the forecast. In general, there are three types of error measures [76], [13]:

1) Scale-dependent error measures: The forecast error is on the same scale as the data. That is, the interpretation of the results are intuitive when comparing different methods on time series with the same scale. However, these measures cannot be used to compare forecast across time series that have different scales. Examples are the mean forecast error (MFE) or root mean squared error (RMSE), where n is the forecast length, y_t the actual value, and f_t the forecast value:

$$\text{MFE} := \frac{1}{n}\sum_{t=1}^{n} y_t - f_t, \qquad (26)$$

$$\text{RMSE} := \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - f_t)^2}. \qquad (27)$$

2) Percentage error measures: On the one hand, this kind of measure is scale-independent and it can thus be used to compare forecasting methods across different time series. On the other hand, the forecast error is infinite or undefined if the actual value is zero. Further, outliers have a significant impact on the forecast error. Prominent examples are the mean absolute percentage error (MAPE) and symmetric mean absolute percentage error (sMAPE), where n is the forecast

[9]In the context of benchmarking the term metric is often used instead of measure. Not considered as metrics in the mathematical sense, both terms can be seen equivalent for the following.

length, y_t the actual value, and f_t the forecast value:

$$\text{MAPE} := \frac{100\%}{n}\sum_{t=1}^{n}|\frac{y_t - f_t}{y_t}|, \quad (28)$$

$$\text{sMAPE} := \frac{200\%}{n}\sum_{t=1}^{n}|\frac{y_t - f_t}{y_t + f_t}|. \quad (29)$$

3) Scaled error measures: To approach the problem of dividing by the actual value, scaled errors normalize the forecast error by a baseline. This normalization makes the forecast scale-independent and thus, the measures can be used to compare forecasts across time series that have different scales. However, if the baseline has values that are equal to each other, the forecast error is undefined as a division by zero has to be done. Examples are the mean absolute scaled error (MASE) or root mean square scaled error (RMSSE), where n is the forecast length, y_t the actual value, f_t the forecast value, p the length of the period (if there is no seasonality, $p = 1$), $\tilde{h}$ the length of the history, and h_i the historical values:

$$\text{MASE} := \frac{\frac{1}{n}\sum_{t=1}^{n}|y_t - f_t|}{\frac{1}{\tilde{h}-p}\sum_{i=p+1}^{\tilde{h}}|h_i - h_{i-p}|}, \quad (30)$$

$$\text{RMSSE} := \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(\frac{|y_t - f_t|}{\frac{1}{\tilde{h}-1}\sum_{i=p}^{\tilde{h}}|h_i - h_{i-1}|}\right)^2}. \quad (31)$$

4) Discussion of the measures: Several error measures can be used to evaluate a forecasting method. Each measure has its use cases, benefits, and drawbacks. For instance, the MFE shows the direction of the error while the RMSE does not, or the MAPE and sMAPE do not penalize extremes and deviations but are scale-independent. A finer distinction between different error measures is done in the works of M. V. Shcherbakov et. al [77], R. Adhikari and R. K. Agrawal [13], or R. J. Hyndman and A. B. Koehler [76]. In general, it is impossible to prove the correctness of a measure; it is more a common agreement on how to quantify the given property. To counter the weaknesses of a specific error measure, it is better to consider more than one of these measures when evaluating forecasts. Also, different measures provide different insights and thus, a better understanding of the forecast can be obtained. In the following, we include pivotal measure characteristics:

- **Definition** A measure should come along with a precise clear mathematical expression to assure consistent application and interpretation.
- **Interpretation** A measure should be intuitively understandable. Furthermore, it is important to specify: (i) if a measure has a physical unit or is unit-free, (ii) if it is normalized and if yes how, and (iii) clear information on the value range and the optimal point.
- **Repeatability** Repeatability implies that if the measure is computed multiple times using the same forecast method and time series, the same value is obtained.

C. Benchmarking Forecasting Methods

Based on our survey, we have found that the degree of quality of the evaluations suffers, on the one hand, due to the lack of commonly used and representative data sets, on the other hand, due to issues with the used methodology, for example, restriction to only a few competing methods or only few measures. To this end, we design a forecasting benchmark that allows a systematic, reproducible, comparable, and automated comparison of forecasting methods.

The forecasting benchmark allows to compare a specific forecasting method against state-of-the-art methods based on different measures (see Section III-B). In more detail, the benchmark offers a broad data set with a high degree of diversity (predicted one time series at a time and passed on to the benchmark for quantification), different quality measures, and a comparison to standard methods. The time series data set is split into four domains: (i) economics, (ii) finance, (iii) human access, and (iv) nature and demographics. Each domain contains 100 time series from different sources and different characteristics. The designed procedure for investigating how well a specific forecasting method performs is as follows:

1) The application domain is selected in which the forecasting method has to be investigated.
2) The time series from the domain are transferred to the forecasting method in a random order one after another.
3) For each time series, the method conducts a multi-step-ahead forecast and passes it to the benchmark.
4) The measures for each forecast time series are calculated.
5) A detailed overview and ranking compared to the state-of-the-art methods is shown.

From a statistical point of view, only the next value is important, but especially in the context of self-aware computing, where a fine granularity leads to several data points in a short time, planning requires several values in advance. Thus, the method performs a multi-step-ahead forecast and both the one-step-ahead and the multi-step-ahead forecast are quantified.

1) A Representative Set of Time Series: To have a sound and broad evaluation of forecasting methods, a highly heterogeneous data set that covers different aspects is required. Indeed, there are numerous data sets available online: competitions (e.g., NN3[10], M3[11], and M4[12]), kaggle, R packages, and many more. For instance, the M3 competition contains 3003 time series from different domains. However, most time series have a high degree of similarity and a length below 100 data points. Although, for instance, the M4 competition set contains 100,000 time series, these time series have low frequencies (1 $\sim$ yearly, 4 $\sim$ quarterly, 12 $\sim$ monthly, and 24 $\sim$ daily) and short forecasting horizons (6 to 48 data points). Further, the median length of a time series is 106. That is, we assume that if both data set are used alone, they are not suitable for benchmarking forecasting methods for self-aware systems as time series in this domain are generally larger and/or have a higher frequency. For instance, when sampling data each second, actions that are planned hourly have to take 3600 data points into account.

[10]NN3 competition: http://www.neural-forecasting-competition.com/NN3/
[11]M3 competition: https://forecasters.org/resources/time-series-data/m3-competition/
[12]M4 competition: https://www.mcompetitions.unic.ac.cy/the-dataset/

To this end, we assemble a data set containing 400 publicly available time series that are divided into 4 different domains: (i) economics (gas, electricity, sales, unemployment, ...), (iii) human access (calls, SMS, Internet, requests, ...) , (ii) finance (stocks, sales prices, gold, exchange rate, ...), and (iv) nature and demographics (rain, birth rate, solar hours, temperature, ...). The time series are publicly available and originate from 50 different sources, including also time series from M3 and M4. We group the sources into authorities from different countries, M3 competition, M4 competition, other competitions (e.g., kaggle), different universities, R packages, and other publicly available data sets (not assignable to the other groups). Further, our data set covers different frequencies (1 $\sim$ yearly to 3600 $\sim$ every second) and lengths (20 to 372,864). Both distributions are depicted in Figure 8. We transform the data set because each time series is available online and therefore the forecast is already known. To this end, each time series Y is linearly mapped to $u \cdot Y + v$ where u is a normally distributed random variable and v is a uniformly distributed variable.

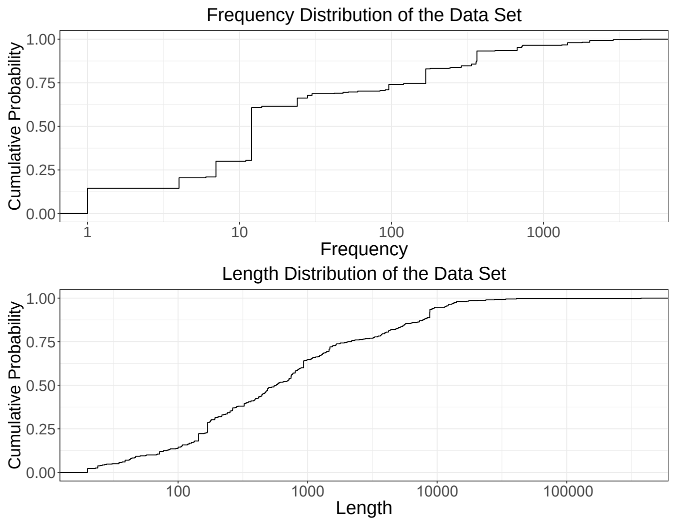

Fig. 8. Distribution of the time series lengths used in the data set.

2) Further Measures: Based on our survey and in order to use common measures, the benchmark uses sMAPE and MASE as error measures (see Section VI-B) due to their scale independence. In addition to the error measures, the benchmark also records the time-to-result. As the time-to-result depends on the operating system and hardware, the benchmark forecasts all time series with sNaïve beforehand. Then, the time-to-result of the specific method is normalized by the time-to-result of sNaïve that also has to be conducted on the test system. We choose those measures as all of them come with a deterministic mathematical expression, are simplistic as they can be described each in a compact sentence or formula, and are either unit-less (or have the unit of the time series) or a normalized ratios (percentage) with the values lying in the interval $(-\infty; \infty)$ where 0 is the optimal value. Besides the average values of each measure, the benchmark also reports the distribution and standard deviation.

Besides the commonly used measures, the benchmark also proposes new measures that are easy to interpret and scale-independent. That is, these measures give useful insights into the forecasting method and allow to compare them across different time series.

a) Mean Wrong-Estimation Shares: These measures capture the tendency of the forecasting method to whether under- or over-estimate actual values. That is, the mean under-estimation share (MUES) is the number of forecast values relative to the whole forecast where the forecast value is below the actual value. Analogously, the mean over-estimation share (MOES) is the relative number of values where the forecast value lies over the actual value. Values of this measure lie in the interval $[0, 1]$. The best value 0 is achieved when the forecasting method does not under-estimate or not over-estimate the actual values. Both measures can be defined while n is the forecast length, y_t the actual value and f_t the forecast value:

$$\text{MUES} := \frac{1}{n} \cdot \sum_{t=1}^{n} max(sgn(y_t - f_t), 0), \tag{32}$$

$$\text{MOES} := \frac{1}{n} \cdot \sum_{t=1}^{n} max(sgn(f_t - y_t), 0). \tag{33}$$

b) Mean Wrong-Accuracy Shares: These measures describe how much the forecasting method under- or over-estimate the actual values on average. That is, the mean under-accuracy share (MUAS) is the mean absolute percentage error between the forecast values and the actual values where the forecasting method under-estimates the actual values. Analogously, the mean over-accuracy share (MOAS) is the mean absolute percentage error where the forecasting method over-estimates the actual values. Values of this measure lie in the interval $[0, \infty)$, where 0 is the best value and indicates that there is no under- or over-estimation. Both measures can be defined while n is the forecast length, y_t the actual value and f_t the forecast value:

$$\begin{aligned} \text{MUAS} &:= \begin{cases} \alpha, & \exists t : f_t < y_t \\ 0, & \forall t : f_t \geq y_t \end{cases} \\ \text{with } \alpha &:= \frac{1}{n \cdot \text{MUES}} \cdot \sum_{t=1}^{n} \frac{max(y_t - f_t, 0)}{|y_t|}, \end{aligned} \tag{34}$$

$$\begin{aligned} \text{MOAS} &:= \begin{cases} \beta, & \exists t : y_t < f_t \\ 0, & \forall t : y_t \geq f_t \end{cases} \\ \text{with } \beta &:= \frac{1}{n \cdot \text{MOES}} \cdot \sum_{t=1}^{n} \frac{max(f_t - y_t, 0)}{|y_t|}. \end{aligned} \tag{35}$$

D. Excerpt of the Competition Results

To investigate how well the summarized standalone forecasting methods (see Section III-B) perform in comparison to the presented automated forecasting workflow, we use the aforementioned forecasting benchmark. Due to space limitations, we are not able to report detailed results (i.e., all statistical measures and for each domain). To this end, we

TABLE II
EXCERPT OF THE BENCHMARKING RESULTS.

Measure	Telescope	ANN	ETS	Random Forest	sARIMA	sNaïve	SVM	tBATS	Theta	XGBoost
avg. sMAPE [%]	19.97	24.82	28.95	37.43	20.63	21.82	58.63	23.62	21.10	23.80
std. sMAPE [%]	31.29	40.13	64.58	361.12	35.64	30.08	541.36	76.01	48.96	34.68
avg. MASE	0.77	1.12	1.20	1.67	0.73	1.02	2.13	0.88	0.94	1.01
std. MASE	2.23	3.72	3.43	6.56	2.17	3.27	9.46	2.83	2.97	3.27
avg. MUES	0.48	0.54	0.53	0.59	0.52	0.61	0.60	0.55	0.58	0.61
avg. MOES	0.52	0.46	0.47	0.41	0.48	0.39	0.40	0.45	0.42	0.39
avg. MUAS	0.64	0.16	0.70	0.16	0.22	0.15	0.19	0.15	0.14	0.32
avg. MOAS	3.35	7.43	20.19	10.22	6.07	5.46	9.47	3.48	16.52	7.14
avg. time_{snaive}	143.27	625.56	429.09	$1.43 \cdot 10^3$	$8.13 \cdot 10^5$	1.00	646.99	$7.66 \cdot 10^3$	10.66	5.46
avg. time [s]	0.59	2.56	1.34	6.41	$2.50 \cdot 10^3$	$5.41 \cdot 10^{-3}$	42.90	27.70	0.04	0.01
std. time [s]	4.34	10.63	7.50	25.61	$2.43 \cdot 10^4$	0.03	815.14	92.63	0.16	0.03

show only an excerpt of the results that the benchmark offers. The following results are the data that will be stored in the benchmark itself as reference for future methods. That is, each method is executed ten times on each time series and all values are stored within the benchmark.

Table II shows for each forecasting method while performing a multi-step-ahead forecast the averaged value over ten runs for each measure that the forecasting benchmark report. Note that, the lower the measures, the better. Consequently, the best value for each measure is 0. The best average sMAPE exhibits the Telescope method closely followed by sARIMA. In contrast, the best average MASE and standard deviation are shown by sARIMA followed both by Telescope. SVM exhibits the highest standard deviation for MASE. The lowest standard deviation for sMAPE is shown by sNaïve closely followed by Telescope. Although sARIMA achieves a good forecasting accuracy, it is 1,000 times slower than most methods, is 800,000 times slower than sNaïve, and has a time-to-result standard deviation of almost 25,000 seconds. In other words, sARIMA may achieve a good forecast, but the time-to-result may be unpractical for the reliable and timely planning of self-aware systems (the maximal time-to-result of sARIMA is 465,574 seconds that is almost 5.5 days).

As there are different superior methods for sMAPE and MASE, we take further metrics (see Section VI-C2) into account. Since MOES or MUES either reflects whether the forecasting method over-estimates or under-estimates the future time series, we can investigate the tendency of the forecasting methods: all state-of-the-art forecasting methods tend to under-estimate the actual values (MUES $>$ 50%). Especially, sNaïve under-estimates on average almost 2/3 of a time series. However, during the under-estimation, the forecasting methods are more accurate than over-estimation the time series. In contrast, the Telescope approach has the lowest difference between MOES and MUES, i.e., there is almost no tendency for either over- or under-estimation. Moreover, it has also the lowest error during the over-estimation, but the second worst error during the under-estimation.

As self-aware systems might have strict deadlines to allow timely and reliable planning, the time-to-result is as important as the accuracy. Due to its simple procedure, sNaïve has the lowest time-to-result followed by XGBoost, Theta, and Telescope. Methods like sARIMA, random forest, and tBATS are at least 1,000 times slower than sNaïve. In other words, sNaïve has an average time-to-result of 0.005 seconds, whereas sARIMA has on average 25,000 seconds. Our approach takes on average 0.6 seconds to produce a forecast.

Although the mean and standard deviation are practical statistical measures, we also investigate the distribution of the forecast errors (sMAPE) and the time-to-result. Figure 9 shows on the top the time-to-result distribution and at the bottom the error distribution. Both horizontal axes show the associated measure (seconds and percentage in log scale); the vertical axes depict the forecasting methods. For the distribution, both the values and a boxplot is shown for each method. In terms of the time-to-result, there are methods like sNaïve, XGBoost, and our approach that have a compact distribution, i.e., a reliable time-to-result. In contrast, SVM and sARIMA exhibits a wide distribution of their time-to-result. Thus, these methods have an unpredictable run-time. For the error distribution, the forecasting methods have comparable distribution between the interquartile range (i.e., between the 25% and 75% quantile). That is, all methods have a robust forecasting accuracy, whereas having an outlier with a small forecasting error is more likely than having a high forecasting error. Figure 9 further supports the "No-Free-Lunch Theorem" stating that there is no method that outperforms the other methods.

In summary, sARIMA shows a good forecasting accuracy, but has a highly unpredictable time-to-result. However, several methods have comparable forecasting accuracy. The Telescope method is the most balanced method as MOES and MOAS are almost equal and is, according to sMAPE, the best method on the benchmark data set and, according to MASE, the second best method. Moreover, the automated forecasting workflow Telescope has a comparable small stand deviation in time-to-result and error and is on average 1,000 times faster than sARIMA.

E. Focus on Time Series related to Self-Aware Systems

While considering the reference scenarios for self-aware systems (see Section II-C), we can identify 95 time series in the collected data set mentioned above that are related to the reference scenarios (see Section II-C). More particular, the time series are related to smart homes or smart micro-grid (gas, oil, and electricity prices, weather data, and electricity produc-

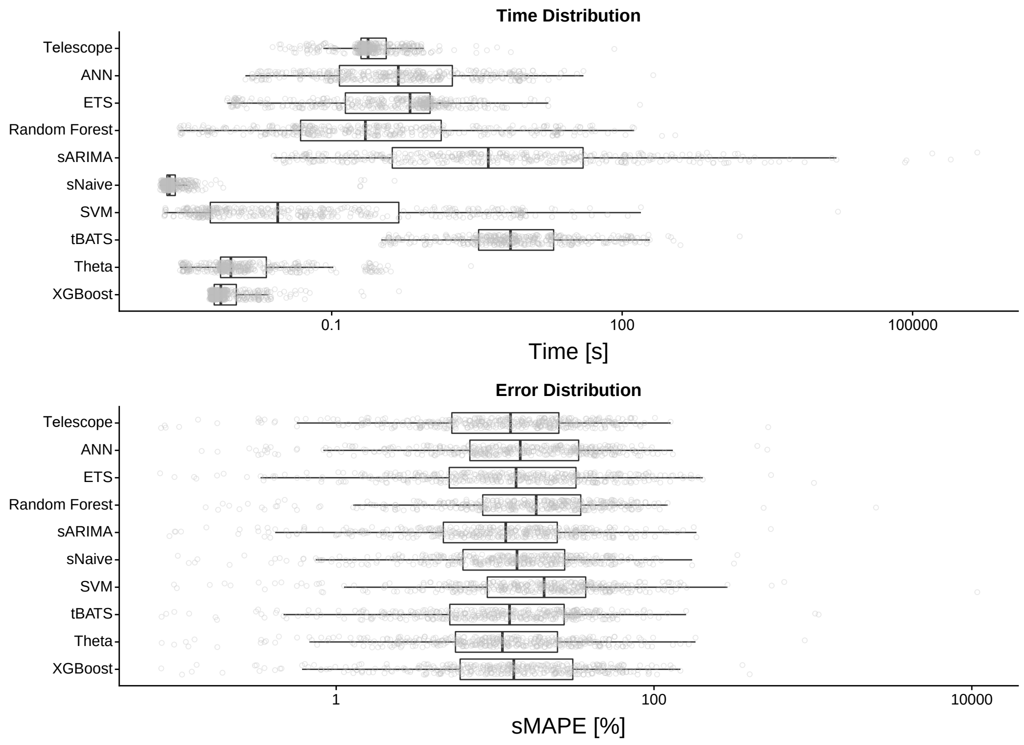

Fig. 9. Error (sMAPE) and time-to-result distribution of all forecasting methods on the benchmark competition data set.

tion & consumption), self-aware data center (web traces and workloads), and systems of autonomous shuttles (passenger and tourism data). Similar to Section VI-D, we compare and summarize the performance of the forecasting methods on this filtered data set containing only time series directly related to the reference scenarios for self-aware systems.

Table III shows the performance of the forecasting methods on the filtered data set over ten runs. The ranking for this reduced data set is almost the same as for the entire data set. That is, for instance, Telescope exhibits the best sMAPE on both the complete and reduced data set, closely followed by sARIMA. Also, on the reduced data set, sARIMA shows an unpredictable time-to-result despite its good accuracy.

VII. Case Study: Self-Aware Data Center

In this section, we wrap up the knowledge and the findings presented in Sections II-VI. Moreover, we illustrate benefits of time series forecasting for self-aware systems. More precisely, we investigate how predictive reasoning can improve the performance of a self-aware computing system. To this end, we select one of the reference scenarios for self-aware computing (see Section II-C)—the self-aware data center—and present a case study where a self-aware data center has to handle contradictory requirements for an application: (i) the application owner wants to minimize the resource usage and thus reduce the operating costs, and (ii) the users of the application expect certain performance criteria, for instance, response time and availability should not fall below a given threshold, referred to as Service Level Objectives (SLOs).

A. *Self-Aware Resource Management*

To provide a representative case study, an authentic workload was chosen stressing an exemplary cloud application. More precisely, we use an *IBM* customer information control system (CICS) transactions trace capturing four weeks of recorded transactions on an IBM z10 mainframe CICS installation. The incoming requests were sampled every 15 minutes, that is, one day consists of 96 data points. In this scenario, the self-aware data center had already observed and managed the application for three weeks. That is, the data center has been able to build a conceptual model, containing information on how many requests a single resource can handle to maintain the SLOs as well as how the scaling actions affect the different requirements for the application. Also, the workload has been recorded as historical time series data. To ensure a feasible experiment run duration, we accelerate the replay time by a factor of 15 during the experiments. As a result, the request rate now changes every minute. Consequently, the self-aware data center also adapts the amount of resources every minute. The timing of the adaptation and the amount of required resources are planned during the reasoning process of the self-aware data center. For estimating the amount of resources needed, the scaling logic of a representative auto-scaling mechanism is used.

We consider *React* [78] as representative example, since it is a simple and straight-forward approach. React provisions resources based on a threshold or a certain scaling indicator of an application. The considered indicators include, inter alia, the number of active connections and the number of requests per second. React gathers these indicators for each

TABLE III
PERFORMANCE COMPARISON OF THE FORECASTING METHODS BASED ON TIME SERIES RELATED TO POSSIBLE FUTURE SELF-AWARE SYSTEMS.

Measure	Telescope	ANN	ETS	Random Forest	sARIMA	sNaïve	SVM	tBATS	Theta	XGBoost
avg. sMAPE [%]	19.17	24.92	41.80	27.01	20.37	20.44	31.73	32.69	20.88	24.50
std. sMAPE [%]	15.46	17.74	45.38	20.00	17.53	15.15	35.92	110.29	23.74	22.50
avg. time [s]	0.52	3.58	1.31	11.56	$3.81 \cdot 10^3$	$6.67 \cdot 10^{-3}$	5.00	52.67	0.05	0.02
std. time [s]	0.82	4.76	2.52	30.63	$1.96 \cdot 10^4$	0.03	13.31	69.69	0.08	0.04

resource and calculates the moving average. Afterwards, the current resources with active sessions that are above or below the given threshold are determined. Then, if all resources have active sessions above the threshold, new resources are provisioned. If there are resources with active sessions below the threshold and with at least one resource that has no active session, idle resources are removed.

In this work, we use the *React* implementation by Papadopoulos et al. [79] that is available online[13]. React neither monitors nor adapts the application itself, but only estimates the amount of required resources upon receiving an input. In the first experiment, React gets the measured values such as the current request rate. That is, after React estimates the required amount of resources, the self-aware data center immediately adapts itself. However, it takes an inherent delay to provide the required amount of resources (e.g., provisioning time). To this end, in the second and third experiment, React receives the forecast request rate from either sARIMA or Telescope as arrival rate forecasts. We consider both forecasting methods due to their good performance shown in Section VI-D. The expected benefit of using these forecasts is that the reasoning can plan the adaptation in time so that the resources are available when they are needed.

B. Quantifying the Reasoning Strategies

To compare and quantify the performance of the different reasoning strategies (i.e., here the auto-scaling quality), we use user-, system-oriented, and time-based metrics. For the user-oriented metrics, we consider SLO violations. In the literature, there are different approaches on how to measure the auto-scaling quality at the system level. As we aim for intuitive metrics that can be precisely described using mathematical formulas, we consider metrics quantifing the system elasticity[14], which is commonly considered to be a central characteristic of the cloud paradigm [81]. To measure how timely the reasoning takes place, we record the proportion of learning and reasoning activities that exceed the specified time interval for each adaptation (i.e., auto-scaling) period.

To quantify the elasticity of the self-aware data center, we leverage the metrics introduced by Herbst et al. [80], which are also endorsed by the Research Group of SPEC[15]. More precisely, we use the wrong provisioning time share and the provisioning accuracy metrics. As we only aim to quantify how accurate the reasoning of the self-aware data center performs, we do not distinguish whether the system is in an over- or under-provisioned state as done by Herbst et al [80]. Therefore, we have to redefine both metrics: The *wrong provisioning time share* metric τ captures the share of time in which the system is under-provisioned or over-provisioned during the measurement interval. The *provisioning accuracy* metric θ reflects the relative amount of resources that are under-provisioned or over-provisioned during the measurement interval. Mathematically, both metrics can be described as:

$$\tau[\%] := \frac{100}{T} \cdot \int_{t=0}^{T} sgn|d_t - s_t|\, dt, \tag{36}$$

$$\theta[\%] := \frac{100}{T} \cdot \int_{t=0}^{T} \frac{|d_t - s_t|}{d_t}\, dt, \tag{37}$$

where T is the experiment duration and $t \in [0, T]$ the current time, s_t the resource supply at time t, and d_t the demanded resource units at time t. The resource demand d_t is the minimum amount of resources required to meet the SLOs under the load at time t. The values of both metrics lie in the interval $[0, \infty)$ where 0 is the best value, indicating that there is no under-provisioning or over-provisioning during the entire measurement interval.

C. Discussion of the different Reasoning Strategies

For good visibility and comprehensibility, only the auto-scaling behavior for Thursday and Friday is shown in Figure 10 for each of the three experiments. The first sub-figure shows the load profile and the requests per second over the experiment duration. The remaining three sub-figures each show the self-aware reasoning, that is, the resource adaptation based on React. For each scaling, the horizontal axis shows the time of the measurement in minutes; the vertical axis shows the number of running/required resources. The solid red curve represents the provisioned resources; the black dashed curve represents the required amount of resources. While using React only with the current request rate, the self-aware data center can only react to workload changes. Thus, the scaling shows an inherent delay: During the increasing load, the system has insufficient resources. In contrast, while the load decreases, the system has excessive resources. The proactive scaling based on sARIMA is able to closely follow the trend of the workload. However, as the time-to-result of sARIMA is highly unpredictable, some adaptations (e.g., the spike at Minute 250) are delayed. The proactive scaling based on Telescope manages to closely follow the demand. Indeed, there

[13]React Auto-Scaler Implementation: https://github.com/ahmedaley/Autoscalers

[14]Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible [80]

[15]Standard Performance Evaluation Corporation (SPEC)

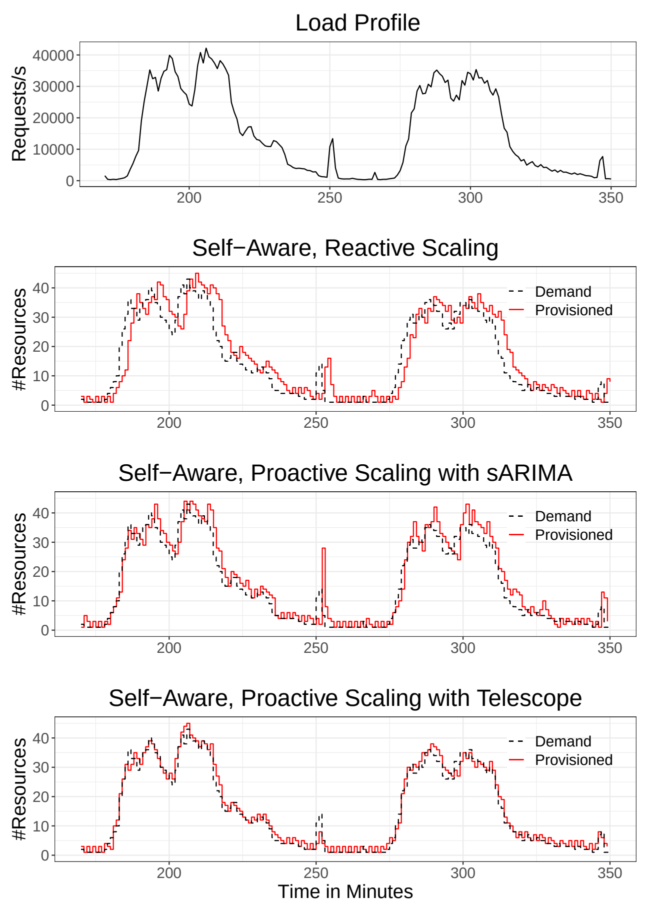

Fig. 10. Self-aware auto-scaling with different reasoning strategies.

are some outliers, such as the spike at Minute 250, that are under- or overestimated; however, the proactive scaling with Telescope is the most accurate one compared to the other reasoning strategies. To provide a quantitative comparison,

TABLE IV
COMPARISON OF DIFFERENT REASONING STRATEGIES FOR AUTO-SCALING

Reasoning Strategy	Wrong Prov. Time Share τ	Provisioning Accuracy θ	SLO Violations	Delayed Reasoning
reactive	86.11%	76.70%	7.35%	0%
sARIMA	41.07%	35.18%	6.73%	20.04%
Telescope	38.33%	23.15%	4.15%	0%

the wrong provisioning time share metric τ, the provisioning accuracy metric θ, the SLO violations, and the share of delayed reasonings are listed in Table IV. Note that the lower the values, the better. While reactive reasoning exhibits the worst values for the elasticity metrics and SLO violations, proactive reasoning based on Telescope exhibits the best values. That is, the reasoning based on Telescope leads in 38.33% of the experiment duration the system in an over- or under-provisioned state. Further, there are either 23.15% too many or few resources allocated. Due to its highly unpredictable time-to-result, sARIMA takes longer than the specified time interval for each adaptation period in 20.04% of the reasoning activities. Thus, this strategy exhibits only the second-best scaling quality and SLO violations.

VIII. Conclusion

As time series forecasting is an essential pillar in autonomic decision making of self-aware systems, we examine and review the state of research in time series forecasting. That is, on the one hand, we present and discuss the basics of time series analysis, state-of-the-art forecasting methods, and techniques from feature engineering. To become applicable for self-aware systems, we formulate explicit challenges and present, on the other hand, a step-by-step walk-through for fully automated feature engineering and forecasting. Following principles from benchmarking, we design a level-playing field for assessing and comparing the accuracy and time-to-result of automated forecasting methods for a broad set of time series data. Finally, we include the benchmark results of a forecasting method competition to guide in selecting and appropriately using existing forecasting methods and illustrate the benefits of time series forecasting in the reference scenario of a self-aware data center.

Acknowledgment

The authors would like to thank Valentin Curtef, Max-Con DataScience GmbH, for his constant support. This work was co-funded by the German Research Foundation (DFG) under grants No. KO 3445/11-1 & KO 3445/18-1 and the IHK (Industrie- und Handelskammer) Würzburg-Schweinfurt.

References

[1] G. E. Box and G. M. Jenkins, “Time series analysis forecasting and control,” Wisconsin University Madison Department of Statistics, Tech. Rep., 1970.
[2] P. R. Lewis, A. Chandra, S. Parsons, E. Robinson, K. Glette, R. Bahsoon, J. Torresen, and X. Yao, “A survey of self-awareness and its application in computing systems,” in *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*. IEEE, 2011, pp. 102–107.
[3] J. Pitt, *The computer after me: awareness and self-awareness in autonomic systems*. World Scientific, 2014.
[4] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska, Eds., *Model-driven Algorithms and Architectures for Self-Aware Computing Systems*, ser. Dagstuhl Reports, Dagstuhl, Germany, 2015.
[5] S. Kounev, N. Huber, F. Brosig, and X. Zhu, “A Model-Based Approach to Designing Self-Aware IT Systems and Infrastructures,” *IEEE Computer*, vol. 49, no. 7, pp. 53–61, 2016.
[6] S. Kounev, P. Lewis, K. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz, P. Inverardi, J. Kephart, and A. Zisman, “The Notion of Self-Aware Computing,” in *Self-Aware Computing Systems*, S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, Eds. Berlin Heidelberg, Germany: Springer Verlag, 2017.
[7] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, no. 1, pp. 41–50, 2003.
[8] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen, and X. Yao, “Architectural aspects of self-aware and self-expressive computing systems: From psychology to engineering,” *Computer*, vol. 48, no. 8, pp. 62–70, 2015.
[9] P. Lewis, K. L. Bellman, C. Landauer, L. Esterle, K. Glette, A. Diaconescu, and H. Giese, “Towards a framework for the levels and aspects of self-aware computing systems,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 51–85.
[10] J. O. Kephart, M. Maggio, A. Diaconescu, H. Giese, H. Hoffmann, S. Kounev, A. Koziolek, P. Lewis, A. Robertsson, and S. Spinner, “Reference scenarios for self-aware computing,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 87–106.

[11] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Burstiness in multi-tier applications: Symptoms, causes, and new models," in *Middleware 2008*, V. Issarny and R. Schantz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 265–286.

[12] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[13] R. Adhikari and R. K. Agrawal, "An introductory study on time series modeling and forecasting," *CoRR*, vol. abs/1302.6613, 2013.

[14] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, Apr 1997.

[15] V. Assimakopoulos and K. Nikolopoulos, "The theta model: a decomposition approach to forecasting," *International journal of forecasting*, vol. 16, no. 4, pp. 521–530, 2000.

[16] R. J. Hyndman and B. Billah, "Unmasking the theta method," *International Journal of Forecasting*, vol. 19, no. 2, pp. 287–290, 2003.

[17] R. Hyndman, G. Athanasopoulos, C. Bergmeir, G. Caceres, L. Chhay, M. O'Hara-Wild, F. Petropoulos, S. Razbash, E. Wang, and F. Yasmeen, *forecast: Forecasting functions for time series and linear models*, 2018, r package version 8.4. [Online]. Available: http://pkg.robjhyndman.com/forecast

[18] R. G. Brown, "Exponential smoothing for predicting demand. cambridge, mass., arthur d. little," 1956.

[19] C. Holt, "Forecasting trends and seasonal by exponentially weighted moving averages," *ONR Memorandum*, vol. 52, 1957.

[20] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[21] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting*, vol. 18, no. 3, pp. 439 – 454, 2002.

[22] H. Wold, "A study in the analysis of stationary time series," Ph.D. dissertation, Almqvist & Wiksell, 1938.

[23] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. Melbourne, Australia: OTexts, 2014. [Online]. Available: OTexts.org/fpp

[24] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[26] Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Back-propagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[27] M. A. Zaytar and C. El Amrani, "Sequence to sequence weather forecasting with long short-term memory recurrent neural networks," *International Journal of Computer Applications*, vol. 143, no. 11, pp. 7–11, 2016.

[28] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," in *International Conference on Machine Learning*, no. 34, 2017, pp. 1–5.

[29] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016. [Online]. Available: http://arxiv.org/abs/1609.03499

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *ACM SIGKDD 2016*. ACM, 2016, pp. 785–794.

[31] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[32] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[33] V. Vapnik, "The nature of statistical learning," 1995.

[34] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in neural information processing systems*, 1997, pp. 155–161.

[35] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[36] C. Chatfield, "What is the bestmethod of forecasting?" *Journal of Applied Statistics*, vol. 15, no. 1, pp. 19–38, 1988.

[37] N. Herbst, A. Amin, A. Andrzejak, L. Grunske, S. Kounev, O. J. Mengshoel, and P. Sundararajan, "Online Workload Forecasting," in *Self-Aware Computing Systems*, S. Kounev, J. O. Kephart, X. Zhu, and A. Milenkoski, Eds. Berlin Heidelberg, Germany: Springer Verlag, 2017.

[38] P. M. Domingos, "A few useful things to know about machine learning." *Commun. acm*, vol. 55, no. 10, pp. 78–87, 2012.

[39] W. R. Bell and S. C. Hillmer, "Issues involved with the seasonal adjustment of economic time series," *Journal of Business & Economic Statistics*, vol. 2, no. 4, pp. 291–320, 1984.

[40] E. B. Dagum and S. Bianconcini, *Seasonal adjustment methods and real time Trend-Cycle estimation*. Springer, 2016.

[41] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[42] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "Stl: A seasonal-trend decomposition procedure based on loess," *Journal of Official Statistics*, vol. 6, no. 1, pp. 3–73, 1990.

[43] A. Schuster, "The periodgram of magnetic declination as obtained from the records of the greenwich observatory during the years 1871-1895," *Trans. Cambridge Philosophical Soc.*, vol. 18, pp. 107–135, 1899.

[44] R. H. Shumway and D. S. Stoffer, "Time series analysis and its applications," *Studies In Informatics And Control*, vol. 9, no. 4, pp. 375–376, 2000.

[45] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[46] J. Benesty, J. Chen, Y. Huang, and I. Cohen, *Pearson Correlation Coefficient*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–4.

[47] L. Myers and M. J. Sirois, "Spearman correlation coefficients, differences between," *Encyclopedia of statistical sciences*, vol. 12, 2004.

[48] M. G. Kendall, "Rank correlation methods." 1948.

[49] A. Kraskov, H. Stoegbauer, and P. Grassberger, "Estimating mutual information," 2003. [Online]. Available: http://arxiv.org/abs/cond-mat/0305641

[50] W. Li, R. Yu, and X. Wang, "Discretization of continuous-valued attributes in decision tree generation," in *2010 International Conference on Machine Learning and Cybernetics*, vol. 1, July 2010, pp. 194–198.

[51] R. A. Fisher, "Statistical methods for research workers," in *Breakthroughs in statistics*. Springer, 1992, pp. 66–70.

[52] J. Brank, M. Grobelnik, N. Milic-Frayling, and D. Mladenic, "Feature selection using support vector machines," *WIT Transactions on Information and Communication Technologies*, vol. 28, 2002.

[53] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression trees," 1984.

[54] D. Coppersmith, S. J. Hong, and J. R. Hosking, "Partitioning nominal attributes in decision trees," *Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 197–217, Jun 1999.

[55] S. Nembrini, I. R. Knig, and M. N. Wright, "The revival of the Gini importance?" *Bioinformatics*, vol. 34, no. 21, pp. 3711–3718, 05 2018.

[56] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[57] J. E. Doran and D. Michie, "Experiments with the graph traverser program," *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, vol. 294, no. 1437, pp. 235–259, 1966.

[58] C. Wilt, J. Tyler Thayer, and W. Ruml, "A comparison of greedy search algorithms," 07 2010.

[59] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[60] G. E. Box and D. R. Cox, "An analysis of transformations," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 211–252, 1964.

[61] V. M. Guerrero, "Time-series analysis supported by power transformations," *Journal of Forecasting*, vol. 12, no. 1, pp. 37–48, 1993.

[62] J. M. Bates and C. W. Granger, "The combination of forecasts," *Journal of the Operational Research Society*, vol. 20, no. 4, pp. 451–468, 1969.

[63] L. M. De Menezes, D. W. Bunn, and J. W. Taylor, "Review of guidelines for the use of combined forecasts," *European Journal of Operational Research*, vol. 120, no. 1, pp. 190–204, 2000.

[64] M. Sommer, A. Stein, and J. Hähner, "Local ensemble weighting in the context of time series forecasting using xcsf," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8.

[65] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[66] P.-F. Pai and C.-S. Lin, "A hybrid arima and support vector machines model in stock price forecasting," *Omega*, vol. 33, no. 6, pp. 497–505, 2005.

[67] S. Soltani, "On the use of the wavelet decomposition for time series prediction," *Neurocomputing*, vol. 48, no. 1-4, pp. 267–277, 2002.

[68] N. Liu, Q. Tang, J. Zhang, W. Fan, and J. Liu, "A hybrid forecasting model with parameter optimization for short-term load forecasting of micro-grids," *Applied Energy*, vol. 129, pp. 336–345, 2014.

[69] M. Züfle, A. Bauer, N. Herbst, V. Curtef, and S. Kounev, "Telescope: A Hybrid Forecast Method for Univariate Time Series," in *Proceedings of the International work-conference on Time Series (ITISE 2017)*, September 2017.

[70] F. Collopy and J. S. Armstrong, "Rule-based forecasting: Development and validation of an expert systems approach to combining time series extrapolations," *Management Science*, vol. 38, no. 10, pp. 1394–1414, 1992.

[71] X. Wang, K. Smith-Miles, and R. Hyndman, "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series," *Neurocomputing*, vol. 72, no. 10-12, pp. 2581–2594, 2009.

[72] C. Lemke and B. Gabrys, "Meta-learning for time series forecasting and forecast combination," *Neurocomputing*, vol. 73, no. 10-12, pp. 2006–2016, 2010.

[73] M. Züfle, A. Bauer, V. Lesch, C. Krupitzer, N. Herbst, S. Kounev, and V. Curtef, "Autonomic Forecasting Method Selection: Examination and Ways Ahead," in *16th IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, June 2019.

[74] M. Sugiyama and M. Kawanabe, *Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation*. The MIT Press, 2012.

[75] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao, "How to build a benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '15. New York, NY, USA: ACM, 2015, pp. 333–336.

[76] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[77] M. V. Shcherbakov, A. Brebels, N. L. Shcherbakova, A. P. Tyukov, T. A. Janovsky, and V. A. Kamaev, "A survey of forecast error measures," *World Applied Sciences Journal*, vol. 24, no. 24, pp. 171–176, 2013.

[78] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *E-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE, 2009, pp. 281–286.

[79] A. V. Papadopoulos, A. Ali-Eldin, K.-E. Årzén, J. Tordsson, and E. Elmroth, "PEAS: A Performance Evaluation Framework for Auto-Scaling Strategies in Cloud Applications," *ACM ToMPECS*, vol. 1, no. 4, pp. 1–31, 8 2016.

[80] N. Herbst, A. Bauer, S. Kounev, G. Oikonomou, E. van Eyk, G. Kousiouris, A. Evangelinou, R. Krebs, T. Brecht, C. L. Abad, and A. Iosup, "Quantifying Cloud Performance and Dependability: Taxonomy, Metric Design, and Emerging Challenges," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (ToMPECS)*, vol. 3, no. 4, pp. 19:1–19:36, 2018.

[81] D. C. Plummer, D. Smith, T. Bittman, D. W. Cear-Ley, D. Cappuccio, D. Scott, R. Kumar, and B. Robertson, "Study: Five Refining Attributes of Public and Private Cloud Computing," Gartner, Tech. Rep., 2009.

André Bauer is a PhD student at the chair of software engineering at the University of Würzburg. He serves as elected newsletter editor of the SPEC Research Group. His research topics include elasticity in cloud computing, auto-scaling and resource management, autonomic and self-aware computing, and forecasting.

Marwin Züfle is a PhD student at the chair of software engineering at the University of Würzburg. His research topics include time series forecasting, data analytics, and critical event prediction. He received the University Award of the Main-Franconian Economy 2018.

Nikolas Herbst is a research group leader at the chair of software engineering at the University of Würzburg. He received a PhD from the University of Würzburg in 2018 and serves as elected vice-chair of the SPEC Research Cloud Group. His research topics include predictive data analysis, elasticity in cloud computing, auto-scaling and resource management, performance evaluation of virtualized environments, autonomic and self-aware computing.

Albin Zehe is a PhD student at the chair for Data Science at the University of Würzburg. His research is mostly focused on machine learning for natural language processing, analyzing literary novels and the development of their plot over time using deep learning.

Andreas Hotho is a professor at the University of Würzburg. He holds a Ph.D. from the University of Karlsruhe (AIFB). He has published over 100 articles in journals and at conferences, co-edited several special issues and books, and co-chaired several workshops. His general research area is on Data Science with focuses on the combination of data mining, natural languages processing and knowledge graphs.

Samuel Kounev is a professor and chair of software engineering at the University of Würzburg. His research is focused on the engineering of dependable and efficient software systems, systems benchmarking and experimental analysis; as well as autonomic and self-aware computing. He received a PhD in computer science from TU Darmstadt. He is a member of ACM, IEEE, and the German Computer Science Society.