Time Series Forecasting of Runtime Software Metrics: An Empirical Study

Federico Di Menna
federico.dimenna@graduate.univaq.it
University of L'Aquila
Italy

Luca Traini
luca.traini@univaq.it
University of L'Aquila
Italy

Vittorio Cortellessa
vittorio.cortellessa@univaq.it
University of L'Aquila
Italy

ABSTRACT

Software applications can produce a wide range of runtime software metrics (*e.g.*, number of crashes, response times), which can be closely monitored to ensure operational efficiency and prevent significant software failures. These metrics are typically recorded as time series data. However, runtime software monitoring has become a high-effort task due to the growing complexity of today's software systems. In this context, time series forecasting (TSF) offers unique opportunities to enhance software monitoring and facilitate proactive issue resolution. While TSF methods have been widely studied in areas like economics and weather forecasting, our understanding of their effectiveness for software runtime metrics remains somewhat limited.

In this paper, we investigate the effectiveness of four TSF methods on 25 real-world runtime software metrics recorded over a period of one and a half years. These methods comprise three recurrent neural network (RNN) models and one traditional time series analysis technique (i.e., SARIMA). The metrics are gathered from a large-scale IT infrastructure involving tens of thousands of digital devices. Our results indicate that, in general, RNN models are very effective in the runtime software metrics prediction, although in some scenarios and for certain specific metrics (*e.g.*, waiting times) SARIMA proves to outperform RNN models. Additionally, our findings suggest that the advantages of using RNN models vanish when the prediction horizon becomes too wide, in our case when it exceeds one week.

CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**; **Software evolution**; **Extra-functional properties**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

time series forecasting, software monitoring, runtime software metrics

ACM Reference Format:
Federico Di Menna, Luca Traini, and Vittorio Cortellessa. 2024. Time Series Forecasting of Runtime Software Metrics: An Empirical Study. In *Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3629526.3645049

ICPE '24, May 7–11, 2024, London, United Kingdom

ACM ISBN 979-8-4007-0444-4/24/05
https://doi.org/10.1145/3629526.3645049

1 INTRODUCTION

As software systems grow in complexity, the task of ensuring software quality becomes increasingly challenging. Today software systems constantly evolve, with frequent daily software releases [46], and they operate under highly variable workloads [5], which make them susceptible to unforeseen software failures [5, 55, 58]. In such a dynamic environment, traditional proactive strategies, such as software testing, are often insufficient for ensuring consistent operational efficiency [45, 58]. For this reason, monitoring is emerging as a key activity for maintaining operational efficiency of software systems [12, 23, 32].

Modern software applications can produce large volumes of runtime metrics, which are typically stored as time series data in specialized databases [23], (*e.g.*, Prometheus [15]). Dedicated monitoring teams continuously analyze these time series to identify and mitigate potential software issues [12]. However, the vast volume of collected data can make manual analysis costly and potentially ineffective. To address this challenge, researchers started to develop automated techniques that can facilitate the identification of software issues or aid in the debugging process [1, 6, 16, 19, 25, 53].

Despite these advancements, significant opportunities in the realm of data analysis remain unexploited. Time series forecasting (TSF), in particular, presents a promising avenue for enhancing the current practices in software monitoring. Indeed, the ability to predict future trends in runtime software metrics could facilitate the adoption of proactive measures, potentially preventing significant software failures or optimizing resource allocation. Runtime software metrics are notoriously difficult to analyze, due to their inherent instability [5, 20, 38, 54]. However, recent advancements in TSF have demonstrated its successful application across diverse fields, including economics [7, 50], meteorology [11], and healthcare [48]. Moreover, TSF has recently begun to gain attention also in the software domain [3, 10, 31, 35]. For instance, Amin *et al.* [3] employed AutoRegressive Integrated Moving Average (ARIMA) models to predict reliability a software system based on testing results. Krishna *et al.* [35] used time series analysis to forecast bug reports and enhancement requests in software projects. Bauer *et al.* [10] proposed the use of TSF in self-aware systems.

Despite these efforts, to date there is still little knowledge about the effectiveness of TSF methods for the prediction of runtime software metrics. With this paper, we aim to fill this gap by presenting an empirical assessment of multiple TSF methods on runtime software metrics. The study aims at addressing the following research questions:

RQ$_1$ *How effective are TSF methods when applied to predict short-term runtime software metrics?*

RQ$_2$ *Do TSF methods exhibit diverse forecasting accuracy over different classes of runtime software metrics?*

RQ$_3$ *To what extent does forecasting accuracy degrade when applied to predict longer-term runtime software metrics?*

To answer our research questions, we investigate the forecasting accuracy of four TSF methods, including a seasonal autoregressive moving average model (SARIMA) and three different recurrent neural networks, namely: fully-connected recurrent neural networks (FC-RNN), long-short memory networks (LSTM), and gated recurrent unit networks (GRU). The evaluation is performed on 25 real-world runtime software metrics that are categorized into 3 classes (*i.e., crash rate*, *hang time*, and *waiting time*) and gathered from 8 different software applications, resulting in a total of 14,575 individual data points. These metrics were recorded over a period of one and a half years on the large-scale IT infrastructure of a company[1], which comprises thousands digital devices. In this study, we distinguish between *short-* and *long-term* software metrics. This distinction refers to the distance of the forecasting target with respect to the current week of the software metric under analysis.

We show that, overall, RNN models are more effective than SARIMA and naive baselines, with FC-RNN providing the best forecasting accuracy. Nonetheless, our evaluation underscores the lack of a "silver bullet" method that outperforms all others across the considered metrics. For instance, we found that when dealing with specific classes of metrics, such as application waiting times, SARIMA offers the most accurate prediction. Furthermore, we found that benefits of using RNN models are valuable until the forecasting horizon does not exceed approximately one week.

The main contribution of this work are:

- A first empirical assessment of TSF methods when applied to runtime software metrics.
- An investigation of how different TSF methods behave when applied to different classes of metrics.
- A sensitivity analysis of TSF on runtime software metrics while varying forecasting horizons.

Paper Structure. The remainder of this paper is organized as follows. Section 2 provides background on TSF within and outside the software domain. Section 3 outlines the experimental design used for our empirical study. In Section 4, we detail our research questions along with the corresponding findings. Section 5 discusses potential threats to validity, and Section 6 concludes this paper.

2 BACKGROUND

Over the years, a vast variety of phenomena have been captured and modeled by leveraging the concept of time series, which can be defined as an ordered sequence of data points gathered at regular time intervals. Time series forecasting (TSF) approaches aim at predicting future evolution of the variable of interest by learning from its historical data while looking for patterns, trends, and seasonality in the data. Due to the easy porting of the time series concept, the effectiveness of TSF methods has been widely investigated in diverse areas, such as medicine, environment, system engineering, finance and more [3, 8, 17, 22, 44, 47, 50, 63]. Forecasting techniques can be differentiated between one-step-ahead and multistep-ahead ones. In the first case, the goal is to predict just the next value of the time series, whereas multiple values are predicted simultaneously in the second case. Generally, the number of forecast values that are generated at a time is denoted as the *forecasting horizon*.

Earliest techniques relied on straightforward heuristics as they evolved over time in more complex statistical models, such as Exponential Smoothing [13] and Autoregressive Integrated Moving Average (ARIMA) models [14]. The wide popularity of machine learning techniques, also driven by the increasing availability of data, has induced a strong interest in artificial neural networks (ANNs) for this purpose [27]. Some studies also explored the combination of statistical models and deep learning models, for instance Zhang et al. leveraged statistical models to fit the linear part of the time series and ANN to model the residual part [65].

Usually, a time series is fed into a neural network by creating consecutive shifted input windows to predict the datapoint(s) that follows the input sequence, thus modeling the task as a supervised learning problem. As dealing with sequential data is a very frequent task in machine learning, several neural network architectures have been expressly designed for sequence prediction, such as recurrent neural networks (RNN). Examples of most popular RNN are Long Short-Term Memory (LSTM) [26, 28, 47] and Gated Recurrent Unit (GRU) [18] models. Despite the extensive employment of TSF techniques, the *No Free Launch Theorem* [62] denies the possibility to build and select a single method that outperforms all the others across all time series, and for this reason a comprehensive evaluation of several models is often needed for the forecasting effectiveness.

Motivated by the necessity of studying the behavior of software applications over time [24, 56], and driven by the vast amount of data produced by modern software applications, researchers have started to apply TSF methods in software contexts, with the goal of anticipating potential issues that may arise from software evolution [3, 10]. For instance, Jia et al. [31] applied a hybrid prediction framework, namely DGRU, to predict software aging and determine the optimal rejuvenation time. Krishna et al. [35] proposed a model to predict the number of bug reports and enhancement requests that are going to be generated in the next month by exploiting the history of issue reports. Amin et al. [3] investigated the employment of ARIMA models in order to predict software reliability as an alternative solution to fight the Software Reliability Growth Models (SRGMs) limitations. Other researchers have explored the use of TSF for enabling proactive autonomous decisions in self-aware systems. For instance, Bauer et al. [10] conducted a preliminary investigation into how TSF methods can be integrated within self-aware systems. Additional research has investigated the exploitability of TSF in predicting Quality of Service (QoS) attributes [30, 52, 60]. For example, Syu et al. [52] conducted an empirical study on TSF methods applied to web services quality attributes.

Albeit considerable work has been carried out on the application of TSF methods in specific software contexts, their application on runtime software metrics and the derived benefits/drawbacks in a broader monitoring environment should be more thoroughly investigated. To fill this gap, we conducted an empirical evaluation

[1]Due to privacy concerns, the company choose not to reveal itself.

of TSF methods on several runtime software metrics (by varying nature) gathered from a large IT infrastructure.

3 EXPERIMENT DESIGN

In this section, we describe our experiment design, including the dataset, the TSF methods (and baselines) we studied, and the specific experimental procedures we used to gather metric predictions.

3.1 Dataset

Our dataset consists of 25 distinct runtime software metrics collected from the IT infrastructure of a large company, which includes more than 30k digital devices with heterogeneous hardware and software configurations. For each of the 25 metrics, the dataset provides a time series of daily observations collected over a period of one and a half years, thus resulting in a total of 583 measurements per metric. Each metric (*i.e.,* times series) concerns to a specific runtime aspect of a particular software application. For example, one metric might represent the *crash rate* of a specific web browser, while another could indicate the *hang time* of a particular email client. The dataset encompasses 8 distinct software applications from a diverse array of types, including web browsers, communication platforms, and word processors. Each application was monitored on 3 different runtime aspects, which we refer to as *metric classes*. In the following, we describe the semantics of each metric class.

- *Crash rate* denotes the average number of observed crashes of an application per hour of use. It is calculated by dividing the total number of observed application crashes within the IT infrastructure by the cumulative hours of application usage across all devices.
- *Hang time* represents the percentage of application usage time during which the application remains in a "hang" state, *i.e.,* when the application is unresponsive and not performing any active processing tasks. This runtime aspect can be critical for diagnosing potential issues within software applications.
- *Waiting time* is defined as the time users spend in waiting for an application to respond while it is actively running. This metric encompasses periods when the application is unresponsive, known as the "hang" state, as well as application and network loading times. It reports the percentage of the total application usage time that is spent in a "waiting" state across all devices within the IT infrastructure.

As a result, the dataset contains 24 time series (*i.e.,* 3 for each of the 8 software applications), plus an additional one that reports the *crash rate* related to operating system failures, such as *blue screen of death* [36] or *kernel panics* [49].

Due to privacy concerns, we cannot make publicly available the dataset used in our study.

3.2 Models and Baselines selection

For our empirical study, we selected four TSF methods, including one autoregressive moving-average model and three recurrent neural network models. We chose these types of models because they have been successfully applied to time series analysis in previous research [3, 22, 26, 31]. Specifically, our selection comprises the following TSF methods: (i) Seasonal Autoregressive Integrated Moving Average (SARIMA) [59], (ii) Fully Connected Recurrent Neural Network (FC-RNN) [40, 64], (iii) *Long Short-Term Memory* (LSTM) [28] and (iv) *Gated Recurrent Unit* (GRU) [18].

SARIMA is an extension of the autoregressive integrated moving-average (ARIMA) model [14] that addresses the seasonal fluctuations often observed in time series data. It is based on the integration of additional seasonal terms, which allow the model to capture both short- and long-term dependencies, as well as repetitive patterns that occur at fixed intervals. This versatility makes it suitable for forecasting time series where the data exhibit periodicity.

FC-RNN is a variant of neural networks where the outputs are fed back into the network as inputs, thus forming directed cycles. This creates a recurrent connection pattern that allows the network to maintain a state that can theoretically hold information about previous inputs indefinitely. The term "fully connected" denotes that each neuron in a given layer is connected to every neuron in both the preceding and subsequent layers.

LSTM is a specialized form of RNN designed to address the challenge of learning long-term dependencies. Unlike standard RNNs, LSTMs include a series of gated cell states that regulate the flow of information. These gates control the persistence and updating of information within the cell state. This architecture allows LSTMs to effectively retain important information over extended sequences while discarding irrelevant data, thus making them particularly suitable for TSF.

GRU is a particular form of RNN, introduced to solve the vanishing gradient problem inherent to traditional RNNs. GRUs simplify the LSTM approach by combining the forget and input gates into a single "update gate" [18], while also merging the cell state and hidden state. This results in a more streamlined model that requires fewer parameters without sacrificing performance.

To aid the interpretability of results, we compare the forecasting accuracy of TSF models with the ones of two naïve baseline models, namely *seasonal naïve* (sNaïve) and *seasonal monthly mean* (sMM).

sNaïve operates on the assumption of temporal recurrence, by repeating the last observed values for future forecasts. In other words, this method projects the most recent seasonal data forward to the next equivalent season, thus assuming cyclical repetition. For instance, when predicting values for the upcoming week, the method simply replicates the observations from the previous week. This method is commonly used as baseline for comparative analysis in TSF studies [9, 22, 27, 34, 43, 52].

sMM leverages recent seasonal weekly trends to forecast future values. Specifically, it computes predictions by calculating the mean of data points that correspond to the same weekday in the preceding month. For example, to forecast the value for an upcoming Monday, sMM would average the values from all Mondays in the last month. Domain experts from the company that provided the dataset have recommended this baseline approach, as it reflects the established analytical practices of their software monitoring team.

3.3 Experimental procedure

To account for the distinct characteristics of the TSF methods investigated in our study, we adopt two distinct procedures tailored specifically for the RNN and SARIMA models, respectively.

SARIMA evaluation. For SARIMA, we independently create one model instance for each of the 25 time series (*i.e.,* runtime software metrics). The fitting process in SARIMA models consists of estimating the $(p, d, q)(P, D, Q)_s$ parameters, where: p and P represent the order of the autoregressive (AR) terms for the non-seasonal and seasonal parts respectively; d and D denotes the degree of differencing needed to render the series stationary on both non-seasonal and seasonal levels; q and Q indicate the order of the moving average (MA) terms for the non-seasonal and seasonal components; and s denotes the seasonality period of the time series data. To estimate the d and D parameter, we considered the number of times we applied differencing to obtain a stationary time series according to the ADFuller [21] test. The best values of (p, q, P, Q) have been searched by fitting the model with all the values ranging from 0 to 3 for each parameter, and by selecting the parameters configuration that gave the lowest *Akaike Information Criterion* (AIC) [2, 51] value after the model fitting. We set the s parameter to 7, which corresponds to a one week seasonality. The fitting process is performed using the initial 467 consecutive measurements of the time series (*i.e.,* 80% of the data points), with the remaining 20% of the measurements reserved for assessing the forecasting accuracy of the model. For the evaluation, starting from the initial input window of 467 consecutive measurements, we progressively increase the input window by one time unit and use the model to forecast the subsequent 14 measurements. This methodology enables us to simulate a realistic scenario of progressive forecasting, in which the model is continuously tested as new data becomes available in the time series. As a result, we obtain for each time series a set of 89 *forecast segments* F, each consisting of 14 predicted measurements.

RNN evaluation. To train the RNN models, instead, we employ a *sliding window segmentation* technique [29], which divides a longer time series of measurements into smaller, overlapping segments of fixed size. Specifically, we use a sliding window of 28 consecutive measurements (*i.e.,* 4 weeks) to generate multiple overlapping segments. The initial 14 measurements serves as the input segment (or *look-back* period), whereas the remaining measurements form the *forecast segment*. Following the best practice for deep-learning models [42, 63], we scaled the data using the *z-score* normalization technique, thus ensuring that the data falls within a comparable range. Specifically, we standardize each of the 28 measurements within the window segment by using the process outlined in Equation (1):

$$z_i = \frac{x_i - \mu_{input}}{\sigma_{input}} \tag{1}$$

where z_i is the standardized value of the i^{th} data point of the window segment, x_i is the original data point value, μ_{input} is the mean of the input segment measurements, and σ_{input} is their standard deviation. We use the mean and standard deviation of the input segment, instead of the entire window of 28 measurements, to ensure that our evaluation process reflects realistic forecasting scenarios, where future measurements (*i.e.,* the forecast segments) are not yet known.

To evaluate a RNN model, we independently train one model instance for each of the 25 time series, by using the initial 72% consecutive window segments (*i.e.,* 394 segments) for training, the subsequent 8% for validation (*i.e.,* 19 segments), and the remaining 20% for testing (*i.e.,* 89 segments). For the RNN models under consideration (*i.e.,* FC-RNN, LSTM, GRU), the neural network architecture is designed as follows. The first layer of the architecture consists of 14 units of the specific RNN type (either FC-RNN, LSTM, or GRU). The architecture includes a second layer composed of another 14 units of the same type, which is encapsulated within a bidirectional layer. Finally, the processed information is channeled through a fully connected layer with 14 units, by employing the hyperbolic tangent (tanh) activation function to produce the final output. We use the Adam optimizer [33] with an initial learning rate of 0.001, using the mean absolute error (MAE) as the loss function. Additionally, we implement a reduce-on-plateau strategy, which decreases the learning rate when no improvement is observed in the validation loss for 20 epochs. The training is conducted for a maximum of 500 epochs, with an early stopping mechanism that terminates the process if no improvement in the validation loss is observed for 50 consecutive epochs. The best model is selected basing on the lowest validation loss observed throughout the training epochs. As a result of the evaluation process, similarly to SARIMA, we obtain for each RNN model and time series a set of forecast segments F, each consisting of 14 measurements.

Notation. For notational convenience, we use F^i to denote the forecast segment that aims to predict the time series data points beginning from the i^{th} position. For instance, F^{500} represents the forecast segment predicting data from the 500^{th} to the 513^{th} position of the time series. Additionally, we use F_j to denote the j^{th} element of the forecast segment, with $1 \leq j \leq 14$. Figure 1 graphically illustrates the example presented above (F^{500}). The black circles represent the data used as input by the models and the red ones are the predicted measurements.

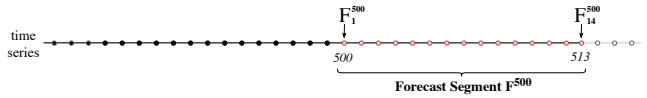


Figure 1: Forecast segment within the time series.

Ultimately, for each TSF method and time series, we obtain 89 forecast segments F^i, with $482 \leq i \leq 570$, where 482 denotes the starting position of the first forecast segment appearing in the time series, and 570 denotes the starting position of the last forecast segment of the time series.

Implementation details. The experiment has been implemented in *Python*. The `SARIMAX` class of *statsmodels*[2] library has been used for implementing SARIMA models. RNNs have been implemented using `SimpleRNN` (for FC-RNN models), `LSTM` and `GRU` classes of *Tensorflow*[3].

[2]https://www.statsmodels.org/stable/index.html
[3]https://www.tensorflow.org/

4 RESEARCH QUESTIONS AND FINDINGS

In this section, we describe in detail the research questions and experimental results of our empirical study. For sake of clarity, we address each research question in a separate subsection. Each subsection is structured as follows: we first outline the objective of the research question, then we describe the methodology used to gather the answers, and finally we discuss the results.

4.1 RQ$_1$: How effective are TSF methods when applied to predict short-term runtime software metrics?

Objective. With the first research question, we want to study the effectiveness of TSF methods in predicting short-term runtime software metrics. Specifically, our focus is on evaluating the accuracy of these methods in "one-step-ahead" forecasting, which entails predicting runtime software metrics for the immediate next day.

Methodology. To address this research question, we employ a commonly used metric for measuring forecasting accuracy, namely SMAPE (Symmetric Mean Absolute Percentage Error).

The values of SMAPE range from 0% to 200%, where 0% indicates perfect forecasting accuracy, while 200% represents the worst accuracy. Given that the aim of this research question is to evaluate the forecasting accuracy of TSF methods in predicting next-day metrics, we calculate the SMAPE by focusing exclusively on the first element of each forecast segment F^i, *i.e.,* the next-day prediction.

Formally, given a time series Y and a particular TSF method, we calculate the corresponding SMAPE as follows:

$$\text{SMAPE} = \frac{100\%}{n-k+1} \sum_{t=k}^{n} \frac{\left|F_1^t - Y_t\right|}{\frac{1}{2}\left(\left|F_1^t\right| + \left|Y_t\right|\right)} \tag{2}$$

where k and n denote the positions of the first and last elements of the time series used for evaluation (specifically, k=482 and n=513 in our case). F_1^t denotes the first element of the forecast segment F^t (*i.e.,* the next-day prediction of the TSF method for the t^{th} element of the time series), and Y_t denotes the actual time series measurement at position t.

As a result of this process, for each TSF method, we obtain 25 SMAPE results, *i.e.,* one per time series.

In order to assess (and compare) the forecasting accuracy of different TSF methods, we plot the SMAPE distribution of each TSF method using box plots, and report the associated descriptive statistics (*e.g.,* mean, median). Additionally, we conduct an analysis to determine whether each TSF method outperforms the naive baselines. Indeed, the practical applicability of a TSF method may be questioned if it does not improve upon these baselines. To accomplish this, we employ the Wilcoxon signed-rank test [61] for each pair of <*TSF method, baseline*> to compare their respective SMAPE values. We set the significance level at 0.05, meaning that differences with p-values below this threshold are considered statistically significant. In addition to the Wilcoxon signed-rank test, we utilize the common language effect size [39], in the version proposed by Vargha and Delaney ($\hat{A}_{12}$) [57], to assess the magnitude of the differences observed. Given two related paired samples X and Y, the common language effect size is the proportion of pairs where X is higher than Y.

$$\hat{A}_{12} = P(X > Y) + .5 \times P(X = Y) \tag{3}$$

The $\hat{A}_{12}$ value, which ranges from 0 to 1, is interpreted using the thresholds provided by Vargha and Delaney [57]. A $\hat{A}_{12}$ value of 0.5 suggests that there is no significant difference in forecasting accuracy between the TSF method and the baseline. A value of $\hat{A}_{12}$ larger than 0.5 indicates that the TSF method is likely to yield more accurate forecasts than the baseline, *i.e.,* lower SMAPE. Specifically, the magnitude of the difference is considered as *small* (S^+), *medium* (M^+) and *large* (L^+) if the $\hat{A}_{12}$ value is greater than or equal to 0.56, 0.64 and 0.71, respectively. Conversely, a value of $\hat{A}_{12}$ lower than 0.5 indicates that the TSF method yields worse forecasting accuracy than the baseline. In this case, the effect size is considered as *small* (S^-), *medium* (M^-) and *large* (L^-) if the $\hat{A}_{12}$ value is lower than or equal to 0.44, 0.34 and 0.29, respectively. We consider comparisons that report $\hat{A}_{12}$ larger than 0.44 and lower than 0.56 as negligible, and therefore not meaningfully different.

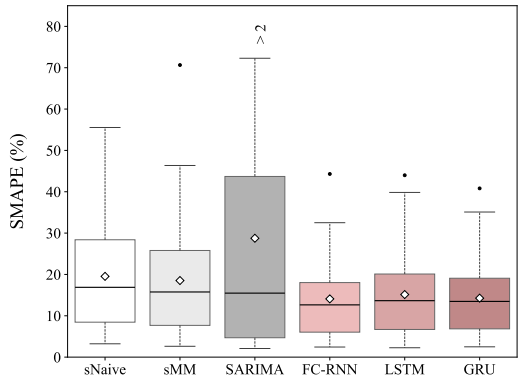


Figure 2: RQ$_1$. SMAPE distribution for each TSF Method. Boxplots are highlighted in red for RNN models, grey for SARIMA and white/silver for the baselines.

	SMAPE Statistics				
	Mean (μ)	Median ($\tilde{x}$)	Std Dev (σ)	Min	Max
sNaïve	19.53	16.88	14.23	3.23	55.54
sMM	18.52	15.76	15.77	2.62	70.65
SARIMA	28.75	15.48	33.02	2.09	116.95
FC-RNN	14.09	12.64	10.31	2.44	44.30
LSTM	15.14	13.65	10.94	2.27	43.99
GRU	14.26	13.48	9.91	2.49	40.82

Table 1: RQ$_1$. SMAPE descriptive statistics for TSF methods and baselines.

Results. We reported the results concerning the SMAPE distribution of TSF methods in Figure 2 and Table 1. A first observation is that RNN-based methods demonstrate promising forecasting accuracies, as evidenced by their average SMAPE values: 14.09% for FC-RNN, 15.14% for LSTM, and 14.26% for GRU. Among them,

	sNaïve	sMM
SARIMA	0.50 (-), p=0.596	0.48 (-), p=0.164
FC-RNN	**0.63 (S^+), p=<0.001**	**0.57 (S^+), p=0.001**
LSTM	**0.59 (S^+), p=<0.001**	0.55 (-), p=0.027
GRU	**0.60 (S^+), p=<0.001**	**0.56 (S^+), p=0.006**

Table 2: RQ_1. Results of the comparison between TSF methods and baselines. Each cell is formatted as "$\hat{A}_{12}$, *p-value*", where $\hat{A}_{12}$ represents the Vargha-Delaney effect size, and the p-value is the result of the Wilcoxon signed-rank test. The interpretation of the $\hat{A}_{12}$ value is also provided in brackets. Comparisons where TSF methods outperform baselines with statistical significance (*p-value*< 0.05) and a non-negligible effect size are highlighted in bold.

FC-RNN appears to be the most effective, by providing the lowest median and mean SMAPE values. These results are comparable to, or even better than, those reported in recent TSF research [9, 10, 37]. From Figure 2, it can be observed that RNN-based methods (shown in red in Figure 2) demonstrate better forecasting accuracy than baselines, as indicated by their lower SMAPE values. In support of this, we also notice in Table 1 that RNN models show considerably lower mean and median SMAPE values than those provided by baselines. For example, if we compare the worst-performing RNN model in terms of mean and median SMAPE (*i.e.,* LSTM) with the best-performing baseline (namely, sMM) we still observe an improvement in mean (18.52% versus 15.14%) and median (15.76% versus 13.62%) values. Another interesting result is that RNN-based methods exhibit higher stability in forecasting accuracy when compared to other approaches, *i.e.,* the prediction error tends to vary less from one time series to another. Indeed, as shown in Table 1, RNN-based methods exhibit lower standard deviations in SMAPE. Specifically, among RNN-based methods we observe a maximum standard deviation of 10.94% (LSTM), which is notably lower than the 14.23% and 15.77% standard deviations observed for sNa"ive and sMM baselines, respectively. This observation is remarked by Figure 2, which displays a narrower inter-quartile range (IQR) for RNN-based methods that is also more shifted towards the bottom, thus indicating lower errors. This suggests that RNN-based methods provides more accurate and stable prediction than baselines. Further confirmation of this finding comes from the results of the Wilcoxon signed-rank test presented in Table 2. Both FC-RNN and GRU demonstrate statistically significant improvements over the baselines ($p < 0.05$), with a non-negligible effect size ($\hat{A}_{12} \geq 0.56$). Additionally, LSTM outperforms sNaive with a small effect size. These results indicate considerable benefits in employing RNN for TSF of short-term runtime software metrics.

An analysis of Figure 2 reveals that SARIMA results in the widest SMAPE IQR, thus indicating significant variation in its forecasting accuracy. This observation is further supported by the data in Table 1, which shows SARIMA having the highest standard deviation, at 33.02%, among all evaluated methods. Moreover, SARIMA provides the worst average forecasting accuracy, with an average SMAPE of 28.75%. While these results might suggest that SARIMA is poorly suited for predicting short-term runtime software metrics, a closer examination of its SMAPE distribution reveals some interesting insights. For example, as shown in Table 1, SARIMA achieves the lowest minimum SMAPE value (2.09%) and its median SMAPE (15.48%) is lower than those of the naive baselines. Additionally, Figure 2 shows that SARIMA provides the lowest first quartile in SMAPE. These observations indicate that SARIMA performs quite well on a specific subset of time series, potentially even outperforming other approaches on these instances.

Summary. RNN-based methods are more effective in predicting short-term runtime software metrics, with FC-RNN being the most effective one (average SMAPE of 12.64%). FC-RNN and GRU outperform both baselines with statistical significance ($p < 0.05$) with non-negligible effect sizes ($\hat{A}_{12} \geq 0.56$). SARIMA is less stable in forecasting accuracy, by exhibiting SMAPE values that significantly vary from one time series to another (standard deviation of 33.02%). To answer RQ_1, our analysis demonstrates that TSF methods, particularly RNN-based methods, are quite effective in accurately predicting short-term runtime software metrics. These findings highlight the potential benefits of applying TSF methods into real-world software monitoring contexts.

4.2 RQ_2: Do TSF methods exhibit diverse forecasting accuracy over different classes of runtime software metrics?

Objective. The second research question aims to assess the forecasting accuracy of TSF models for each class of runtime software metric (*i.e., crashes rate/hang times/waiting times*). We want to study whether TSF methods perform consistently across diverse metric classes or their accuracy varies significantly depending on the specific class of metric being forecasted.

Methodology. To address this research question, we reuse the SMAPE values previously calculated for RQ_1. However, rather than examining these values in aggregate, we group them per metric class. We investigate the forecasting accuracy of TSF methods across each metric class by analyzing their corresponding SMAPE distributions through box plots, and by examining the associated descriptive statistics. For each metric class, we also compare the forecasting accuracy of TSF methods with naive baselines using Wilcoxon signed-rank test. This is done for each *<TSF method, baseline>* pair to determine if there is a statistically significant difference in their SMAPE values. Additionally, we employ the Vargha-Delaney $\hat{A}_{12}$ to assess the effect size.

Results. The SMAPE distribution computed over all the metric classes is depicted in Figure 3, and the associated descriptive statistics are reported in Table 3. We examined the results in two ways: (i) by evaluating the general forecasting accuracy of TSF methods on individual software metric classes, and (ii) by analyzing how each TSF method behaves in relation to each software metric class.

With the first analysis, we want to study how TSF methods collectively perform when applied to different classes of metrics. In Figure 3, we observe that SMAPE box plots for same metric classes exhibit similar behavior across various TSF methods. For instance, when observing the SMAPE distribution related to *waiting time*, we notice significantly lower errors compared to those reported for other metric classes. This can be observed in both Figure 3 and Table 3: the IQR, mean, and median of SMAPE for *waiting*

		Baselines		TSF Methods			
		sNaive	sMM	SARIMA	FC-RNN	LSTM	GRU
Metric Class	SMAPE Stat						
Crash Rate	Mean (μ)	24.48	24.05	38.76	17.40	19.07	17.95
	Median ($\tilde{x}$)	18.55	15.76	18.24	13.56	14.70	14.97
	Std Dev (σ)	16.40	20.88	37.06	9.72	11.21	9.82
	Max	55.54	70.65	116.95	32.51	39.83	35.10
	Min	6.12	5.07	4.55	5.24	5.03	5.33
Hang Times	Mean (μ)	21.62	18.33	39.32	16.24	16.52	15.86
	Median ($\tilde{x}$)	19.02	16.61	34.91	13.44	14.01	13.40
	Std Dev (σ)	14.31	13.18	36.52	12.70	12.33	11.48
	Max	50.65	46.35	116.26	44.30	43.99	40.82
	Min	5.65	3.98	3.64	3.97	4.09	4.07
Waiting Times	Mean (μ)	11.87	12.50	6.91	8.20	9.34	8.53
	Median ($\tilde{x}$)	8.72	8.78	4.19	5.90	6.34	6.86
	Std Dev (σ)	8.85	9.98	5.77	6.01	7.38	6.03
	Max	28.39	29.99	17.28	18.04	21.77	19.08
	Min	3.23	2.62	2.09	2.44	2.27	2.49

Table 3: RQ$_2$. SMAPE descriptive statistics for TSF methods and baselines grouped by software metric class.

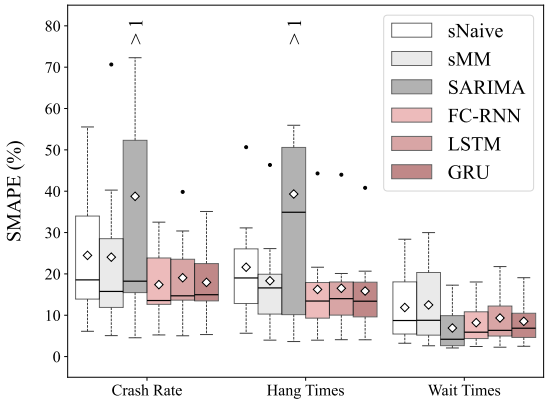


Figure 3: RQ$_2$. SMAPE distribution of TSF methods grouped by metric class.

time are consistently lower than those reported for other metric classes, regardless of the TSF method employed. This suggests that time series pertaining to the same metric classes may share common characteristics that influence the forecasting accuracy of TSF methods, thus leading to consistently better or worse outcomes. Indeed, time series related to *waiting time* appear significantly more predictable than those related to *crash rate* or *hang time*. Both *hang time* and *crash rate* show larger and more dispersed (*i.e.,* higher standard deviation σ) SMAPE values, with *crash rate* looking as the hardest to predict. A possible motivation for this result could be that *waiting time* metrics exhibit a seasonal pattern that recurs over time, while *crash rate* (or *hang time*) metrics are influenced by more unpredictable factors, such as unexpected software bugs or hardware malfunctions.

In our second analysis, we investigate the forecasting accuracy of various TSF methods for each class of runtime software metric. In Figure 3 we observe that RNN-based methods, *i.e.,* FC-RNN, LSTM, and GRU, perform generally well across the different metric classes. RNN-based methods show lower (or comparable) SMAPE distributions than those reported by naive baselines. This is confirmed by the results of the Wilcoxon signed-rank test, reported in Table 4. For instance, FC-RNN outperforms sNaive in all the metric classes with a statistically significant difference ($p < 0.05$) and a medium effect size ($\hat{A}_{12} \geq 0.64$). LSTM also outperforms sNaive with a small effect size ($\hat{A}_{12} \geq 0.56$) on *crash rate* and *hang time*. GRU, similarly, outperforms sNaive with small and medium effect sizes on *crash rate* and *hang time*, respectively. Compared to the sMM baseline, both LSTM and GRU report statistically significant lower SMAPE values on *hang time* and *waiting time*, with either small or medium effect sizes. Overall, FC-RNN demonstrates the best forecasting accuracy across the different metric classes, by providing statistically significant improvement over both naive baselines in all metric classes, except on *crash rate* when compared to sMM.

Another noteworthy result concerns the diverse forecasting accuracy provided by SARIMA over different metric classes. By examining Figure 3, we observe that SARIMA provides substantially different SMAPE values over different metric classes. For instance, while it reports relatively high errors for *crash rate* and *hang time* (*e.g.,* average of 38.79% and 39.32%, respectively), considerably low SMAPE values are reported for *waiting time* (*e.g.,* average of 6.91%). This diversity is remarked in the comparison with naive baselines. According to Table 4, SARIMA outperforms both sNaive and sMM on *waiting time* with statistically significant difference, and large and medium effect sizes, respectively. However, for other metric classes, SARIMA does not provide any improvement over the baselines. Even more, it provides worse forecasting accuracy than sMM on *crash rate*. Nonetheless, by analyzing both Figure 3 and Table 3, we can notice that SARIMA provides the best forecasting accuracy on *waiting time* among different TSF methods, with the lowest mean and median values (respectively, 6.91% and 4.19%). This may suggest that such method performs particularly well when dealing with more predictable time series that show recurring patterns, while it may not fit well on more irregular runtime software metrics, such as *crash rate* or *hang time*.

Summary. When collectively analyzed, TSF methods exhibit diverse forecasting accuracy depending on the class of runtime software metric. For instance, TSF methods show higher effectiveness when dealing with *waiting time* metrics, while they are less effective

TSF Method	Baseline	Software Metric Class		
		Crash Rate	Hang Times	Waiting Times
SARIMA	sNaive	0.42 (-), p=0.129	0.36 (-), p=0.312	**0.73 (L^+), p=0.008**
	sMM	**0.38 (S^-), p=0.008**	0.34 (-), p=0.148	**0.70 (M^+), p=0.008**
FC-RNN	sNaive	**0.65 (M^+), p=0.004**	**0.64 (M^+), p=0.008**	**0.69 (M^+), p=0.008**
	sMM	0.52 (-), p=0.496	**0.59 (S^+), p=0.008**	**0.64 (M^+), p=0.016**
LSTM	sNaive	**0.60 (S^+), p=0.012**	**0.62 (S^+), p=0.008**	0.61 (-), p=0.109
	sMM	0.53 (-), p=0.57	0.56 (-), p=0.055	0.59 (-), p=0.109
GRU	sNaive	**0.59 (S^+), p=0.02**	**0.64 (M^+), p=0.008**	0.61 (-), p=0.055
	sMM	0.52 (-), p=0.652	**0.58 (S^+), p=0.023**	**0.62 (S^+), p=0.023**

Table 4: RQ_2. Results of the comparison between TSF methods and baselines over each class of metric. Each cell is formatted as "$\hat{A}_{12}$, *p-value*", where $\hat{A}_{12}$ represents the Vargha-Delaney effect size, and the p-value is the result of the Wilcoxon signed-rank test. The interpretation of the $\hat{A}_{12}$ value is also provided in brackets. Comparisons where TSF methods outperform baselines with statistical significance ($p\text{-}value < 0.05$) and a non-negligible effect size are highlighted in bold.

on classes of metrics more closely related to software malfunctions, such as *hang times* and *crash rate*. RNN-based methods demonstrate the most consistent effectiveness across different metric classes, with FC-RNN being the most effective one. However, when dealing with classes of more predictable metrics, such as *waiting time*, SARIMA has been shown to be the most effective.

4.3 RQ_3: To what extent does forecasting accuracy degrade when applied to predict longer-term runtime software metrics?

Objective. The goal of this research question is to assess how forecasting accuracy decreases when predicting longer-term runtime software metrics. The TSF methods that we consider are able to generate multi-step ahead forecasts. This means that, at any given time, a TSF method can be queried to predict (for instance) the runtime software metrics of the next 14 days. It is expected that near-term predictions (*e.g.,* for the forthcoming days) will generally be more accurate than those for longer terms (*e.g.,* the following week). Through this research question, we aim to evaluate the extent to which the accuracy decreases as the forecasting horizon is progressively extended.

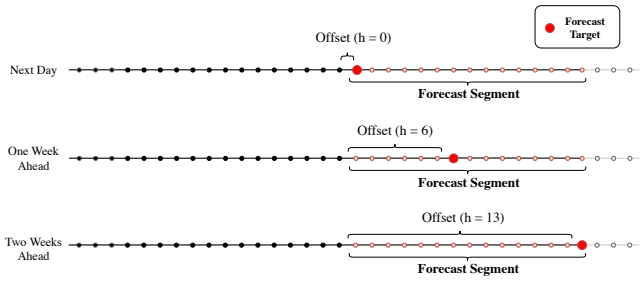


Figure 4: RQ_3. Three representative offset scenarios: next day ($h = 0$), one week ahead ($h = 6$), and two weeks ahead ($h = 13$).

Methodology. To achieve this research goal, we introduce the concept of *offset* (h), which we define as the number of days between the last time step used as input and the *forecast target*. This essentially simulates scenarios where, at a given moment, the goal is to predict the metric for a specific future day. For example, an offset $h = 0$ indicates a scenario where the forecast target is the next day, while an offset $h = 6$ corresponds to a scenario where the goal is to predict runtime software metrics for the same day in the following week. Figure 4 graphically illustrates the concept of offset and forecast target. The first scenario represents the case where the goal is to predict the metrics of next day ($h = 0$), the second scenario targets predictions for the same day in the following week ($h = 6$), and the third scenario uses the same day of two weeks ahead as forecast target ($h = 13$).

For each specific offset h, we calculate the SMAPE of a given TSF method applied to a particular time series Y as follows:

$$\text{SMAPE} = \frac{100\%}{n-k+1} \sum_{t=k}^{n} \frac{\left|F^t_{1+h} - Y_{t+h}\right|}{\frac{1}{2}\left(\left|F^t_{1+h}\right| + |Y_{t+h}|\right)} \tag{4}$$

where F^t_{1+h} represents the $(1+h)^{th}$ element of the forecast segment F^t, corresponding to the prediction for the $1+h$ steps ahead day (*i.e.,* the forecast target). Y_{t+h} indicates the actual measurement observed in the time series on that particular day. For example, with a $h = 6$ offset, SMAPE is calculated by considering exclusively the errors at the 7^{th} elements across all forecast segments of the time series. This corresponds to assessing the forecasting accuracy of a TSF method in predicting the runtime software metric for the same day in the following week.

As results of this process, for each TSF method, we compute 350 SMAPE values, corresponding to each combination of time series and offset h, with $0 \leq h \leq 13$.

We employ box plots and line plots to illustrate the relationship between SMAPE and the offset h. The analysis of these plots aids in gaining a better understanding of how forecasting accuracy degrades as the forecasting horizon is extended.

Results. Figure 5 displays the SMAPE distribution of each TSF method for three representative offset scenarios: next day (h=0), one week ahead (h=6), and two weeks ahead (h=13). Figure 6 shows the trend of the mean SMAPE of each TSF method as the offset h increases. Figure 7 reports the same information for each metric class, separately.

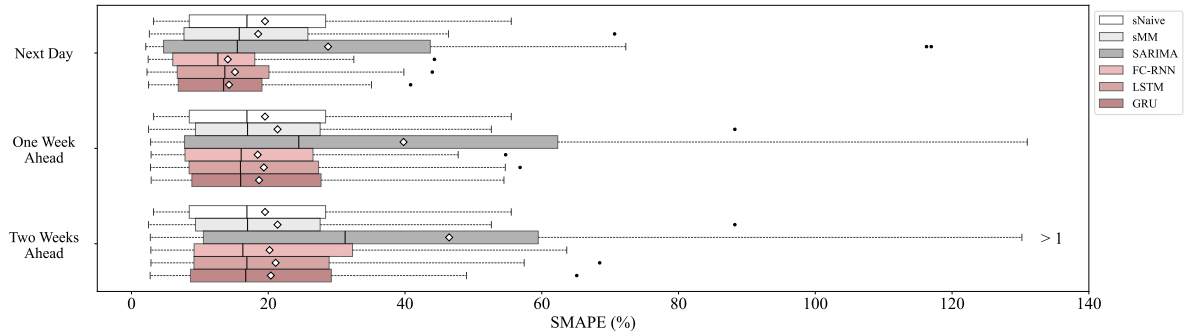
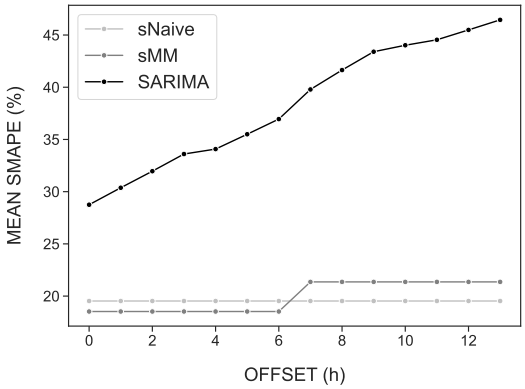
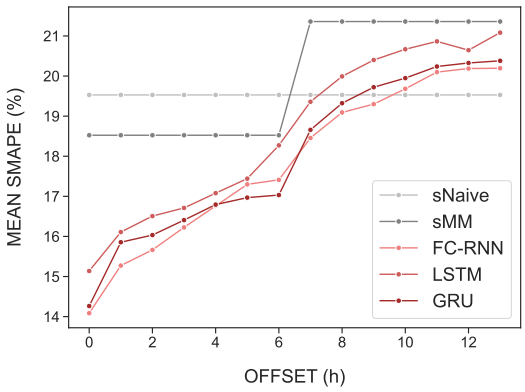


Figure 5: RQ$_3$. SMAPE distribution of TSF methods under three representative offset scenarios: next day ($h = 0$), one week ahead ($h = 6$), and two weeks ahead ($h = 13$).



(a) SARIMA vs baselines.



(b) RNNs vs baselines.

Figure 6: RQ$_3$. Relationship between mean SMAPE and offset (h).

Figure 6 clearly highlights an increasing trend in the mean SMAPE as the offset increases. This increasing trend is also observable in individual classes of metrics, as shown in Figure 7. By looking at Figure 5, we can also notice that the SMAPE tends to become more variable as the offset increases, thus indicating less stability in forecasting accuracy on long-term predictions. This is particularly visible in RNN-based methods, which show an IQR that consistently increases with each offset scenario.

Another interesting outcome of our analysis concerns the comparison of the forecasting accuracy of TSF methods with the baselines. Specifically, upon examining Figure 6, we notice a critical offset beyond which RNN-based methods begin to achieve comparable or even worse forecasting accuracy than the ones of baselines. This specific turning point is observed at approximately a week ahead ($h \approx 7$). A similar behavior is also observable on each individual class of runtime software metric, as shown in Figure 7. Nonetheless, each class of metric exhibits slightly different patterns. For instance, in Figure 7, we notice that mean SMAPE of RNNs begin to exceed those of baselines at offset $h = 9$ on *crashes rate*, instead RNNs start to exceed baselines error at offset $h = 6$ for *waiting times*, . This finding is also evident in Figure 5, where a clear degradation is observed in the SMAPE distribution, moving from the *next day* to the *one week ahead* scenario. Overall, these results suggest that the benefits of employing RNNs for predicting runtime software metrics vanish when the prediction target exceeds the current week. Our results highlight the importance of accounting for the changing dynamics of runtime software metrics, thus outlining the necessity of incorporating recent temporal patterns. In a practical software monitoring context, this emphasizes the need to regularly update predictions as new data becomes available.

Another interesting observation is that, albeit RNN models exhibit similar behavior, in some cases they start to exceed the baselines at different offsets. For example, by looking at Fig. 7, we noticed that LSTM start to exceed the sMM on *hang time* as early as offset is equal to $h = 3$, while for FC-RNN this happens at offset $h = 6$. Furthermore, on *hang times*, GRU outperforms the baselines across the entire offset range, although from offset $h = 6$ its SMAPE becomes very similar to that of sMM. This reveals that the choice of the specific RNN model can have non-trivial impact on the forecasting accuracy of longer-term runtime software metrics.

We then examined the accuracy of TSF models in detail for each class of software metrics. In line with the findings discussed in RQ$_2$, Figure 7 reveals that predictions for *waiting times* metrics consistently displayed lower errors than other metric classes, even in longer-term forecasts, with a maximum mean SMAPE of 16%. It is interesting to note that SARIMA, which is the best-performing

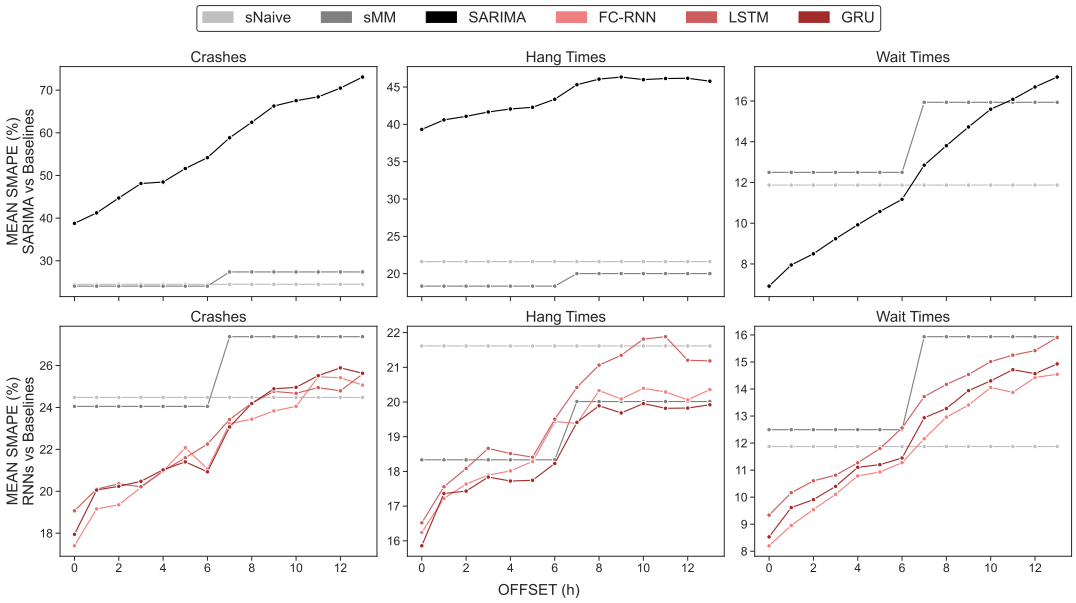


Figure 7: RQ$_3$. Relationship between mean SMAPE and offset (h) grouped by metric class. The first row displays results for SARIMA, while the second row shows the results of RNN-based methods.

method for *waiting time* according to RQ$_2$, shows here a similar behavior to that of RNN-based methods. In fact, we observe that SARIMA only begins to underperform the baselines when the offset exceeds one week ($h = 6$).

Summary. By answering RQ$_3$, our findings reveal that short-term forecasts are consistently more accurate than longer-term ones. Namely, the SMAPE values demonstrate a rising trend as the offset increases. Our results suggest that the advantages of using TSF methods vanish if the offset exceeds approximately one week. These findings offer practical insights into the suitability of TSF methods for predicting long-term runtime software metrics.

5 THREATS TO VALIDITY

Construct validity. A potential threat to construct validity in our study pertains to the selection of TSF methods. The choice of different TSF methods might yield different results. To mitigate this threat, we selected well-established TSF methods, by including a traditional autoregressive moving average model and three types of recurrent neural networks. Another threat concerns the choice of a metric for assessing the effectiveness of the TSF methods. The forecasting accuracy of the studied methods was evaluated using SMAPE, which is a widely accepted scale-independent error measure. However, it is important to note that other error metrics could partially alter the study outcomes.

Internal validity. The implementation of RNN and SARIMA models in our study was carried out by using *tensorflow* and *statsmodels*, respectively. The choice of these specific libraries might introduce biases that could potentially influence the study results. Nonetheless, these are well-established libraries in the field of data analysis. The choice of hyper-parameters can significantly affect the forecasting accuracy of TSF methods. Different hyper-parameter configurations might lead to different outcomes. To mitigate this, we tried to maintain consistent hyper-parameter settings across different TSF methods wherever feasible. For example, all RNN-based methods were configured with an identical number of layers and units, while utilizing the same optimizer. Furthermore, we applied uniform early stopping strategies and epochs.

External validity. The results of our empirical study may not generalize to other different runtime software metrics. However, our analysis was based on a dataset comprising 25 different runtime software metrics collected from 8 distinct software applications, which span three metric classes (*i.e.,* crash rate, hang times, and waiting times). To the best of our knowledge, there are no other TSF studies that have utilized a dataset involving such diversity in runtime software metrics.

6 CONCLUSION

In this empirical study, we investigated the effectiveness of popular TSF methods for predicting runtime software metrics. Our results demonstrate that: (i) TSF methods are indeed effective for short-term predictions, with RNN-based methods emerging as the most effective ones; (ii) no single method offers the best forecasting accuracy across all runtime software metrics; (iii) the benefits of applying TSF methods diminish when the forecasting horizon extends beyond the current week. We encourage future research to expand upon the depth and breadth of our study scope. For instance, future studies could delve into hyper-parameter tuning and explore further TSF methods, including multivariate ones that can simultaneously consider multiple runtime metrics. Additionally, given the rapid expansion of transfer learning in various domains [4, 41, 66], the investigation of the application of pre-trained TSF models for predicting runtime software metrics is a promising avenue for future research.

ACKNOWLEDGEMENTS

This work is partially supported by Italian Government (Ministero dell'Università e della Ricerca, PRIN 2022 PNRR): "RECHARGE: monitoRing, tEsting, and CHaracterization of performAnce Regressions" (cod.P2022SELA7), and by "ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing", funded by European Union – NextGenerationEU.

REFERENCES

[1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 1737–1749. https://doi.org/10.1109/ICSE48619.2023.00149

[2] H. Akaike. 1974. A new look at the statistical model identification. *IEEE Trans. Automat. Control* 19, 6 (1974), 716–723. https://doi.org/10.1109/TAC.1974.1100705

[3] Ayman Amin, Lars Grunske, and Alan Colman. 2013. An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software* 86, 7 (2013), 1923–1932. https://doi.org/10.1016/j.jss.2013.03.045

[4] Ognjen Antonijevi, Slobodan Jeli, Branislav Bajat, and Milan Kilibarda. 2023. Transfer learning approach based on satellite image time series for the crop classification problem. *Journal of Big Data* 10, 1 (2023). https://doi.org/10.1186/s40537-023-00735-2

[5] Dan Ardelean, Amer Diwan, and Chandra Erdman. 2018. Performance Analysis of Cloud Applications. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 405–417. https://www.usenix.org/conference/nsdi18/presentation/ardelean

[6] Chetan Bansal, Sundararajan Renganathan, Ashima Asudani, Olivier Midy, and Mathru Janakiraman. 2020. DeCaf: Diagnosing and Triaging Performance Issues in Large-Scale Cloud Services. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 201–210. https://doi.org/10.1145/3377813.3381353

[7] Wei Bao, Jun Yue, and Yulei Rao. 2017. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE* 12, 7 (07 2017), 1–24. https://doi.org/10.1371/journal.pone.0180944

[8] Oren Barkan, Jonathan Benchimol, Itamar Caspi, Eliya Cohen, Allon Hammer, and Noam Koenigstein. 2023. Forecasting CPI inflation components with Hierarchical Recurrent Neural Networks. *International Journal of Forecasting* 39, 3 (2023), 1145–1162. https://doi.org/10.1016/j.ijforecast.2022.04.009

[9] André Bauer, Marwin Züfle, Simon Eismann, Johannes Grohmann, Nikolas Herbst, and Samuel Kounev. 2021. Libra: A Benchmark for Time Series Forecasting Methods. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Virtual Event, France) *(ICPE '21)*. Association for Computing Machinery, New York, NY, USA, 189–200. https://doi.org/10.1145/3427921.3450241

[10] André Bauer, Marwin Züfle, Nikolas Herbst, Albin Zehe, Andreas Hotho, and Samuel Kounev. 2020. Time Series Forecasting for Self-Aware Systems. *Proc. IEEE* 108, 7 (2020), 1068–1093. https://doi.org/10.1109/JPROC.2020.2983857

[11] Peter Bauer, Alan Thorpe, and Gilbert Brunet. 2015. The quiet revolution of numerical weather prediction. *Nature* 525, 7567 (2015), 47–55. https://doi.org/10.1038/nature14956

[12] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site reliability engineering: How Google runs production systems.* " O'Reilly Media, Inc.".

[13] Baki Billah, Maxwell L. King, Ralph D. Snyder, and Anne B. Koehler. 2006. Exponential smoothing model selection for forecasting. *International Journal of Forecasting* 22, 2 (2006), 239–247. https://doi.org/10.1016/j.ijforecast.2005.08.002

[14] George.E.P. Box and Gwilym M. Jenkins. 1976. *Time Series Analysis: Forecasting and Control.* Holden-Day.

[15] Brian Brazil. 2018. *Prometheus: Up & Running: Infrastructure and Application Performance Monitoring.* " O'Reilly Media, Inc.".

[16] Junming Cao, Bihuan Chen, Chao Sun, Longjie Hu, Shuaihong Wu, and Xin Peng. 2022. Understanding Performance Problems in Deep Learning Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 357–369. https://doi.org/10.1145/3540250.3549123

[17] Pousali Chakraborty, Marius Corici, and Thomas Magedanz. 2021. System Failure Prediction within Software 5G Core Networks using Time Series Forecasting. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. 1–7. https://doi.org/10.1109/ICCWorkshops50388.2021.9473530

[18] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[19] Vittorio Cortellessa and Luca Traini. 2020. Detecting Latency Degradation Patterns in Service-Based Systems. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 161–172. https://doi.org/10.1145/3358960.3379126

[20] David Daly, William Brown, Henrik Ingo, Jim O'Leary, and David Bradford. 2020. The Use of Change Point Detection to Identify Software Performance Regressions in a Continuous Integration System. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 67–75. https://doi.org/10.1145/3358960.3375791

[21] D. Dickey and Wayne Fuller. 1979. Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *JASA. Journal of the American Statistical Association* 74 (06 1979). https://doi.org/10.2307/2286348

[22] Dimitrios Effrosynidis, Evangelos Spiliotis, Georgios Sylaios, and Avi Arampatzis. 2023. Time series and regression methods for univariate environmental forecasting: An empirical evaluation. *Science of The Total Environment* 875 (2023), 162580. https://doi.org/10.1016/j.scitotenv.2023.162580

[23] Liz Fong-Jones and Charity Majors. 2022. *Observability Engineering.* O'Reilly Media, Incorporated.

[24] E. Fuentetaja and D. Bagert. 2002. Software Evolution from a Time-Series Perspective. In *Proceedings of the International Conference on Software Maintenance (ICSM'02) (ICSM '02)*. IEEE Computer Society, USA, 226.

[25] Shi Han, Yingnong Dang, Song Ge, Dongmei Zhang, and Tao Xie. 2012. Performance Debugging in the Large via Mining Millions of Stack Traces. In *Proceedings of the 34th International Conference on Software Engineering* (Zurich, Switzerland) *(ICSE '12)*. IEEE Press, 145–155.

[26] Amirreza Heidari and Dolaana Khovalyg. 2020. Short-term energy use prediction of solar-assisted water heating system: Application case of combined attention-based LSTM and time-series decomposition. *Solar Energy* 207 (09 2020), 626–639. https://doi.org/10.1016/j.solener.2020.07.008

[27] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2021. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting* 37, 1 (2021), 388–427. https://doi.org/10.1016/j.ijforecast.2020.06.008

[28] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[29] Robin John Hyndman and George Athanasopoulos. 2018. *Forecasting: Principles and Practice* (2nd ed.). OTexts, Australia.

[30] R. J. Hyndman, E. Wang, and N. Laptev. 2015. Large-Scale Unusual Time Series Detection. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE Computer Society, Los Alamitos, CA, USA, 1616–1619. https://doi.org/10.1109/ICDMW.2015.104

[31] Kai Jia, Xiao Yu, Chen Zhang, Wenhua Hu, Dongdong Zhao, and Jianwen Xiang. 2023. Software Aging Prediction for Cloud Services Using a Gate Recurrent Unit Neural Network Model Based on Time Series Decomposition. *IEEE Transactions on Emerging Topics in Computing* 11, 3 (2023), 580–593. https://doi.org/10.1109/TETC.2023.3258503

[32] Mike Julian. 2017. *Practical monitoring: effective strategies for the real world.* " O'Reilly Media, Inc.".

[33] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[34] Nikolaos Kourentzes. 2013. Intermittent demand forecasts with neural networks. *International Journal of Production Economics* 143 (05 2013), 198–206. https://doi.org/10.1016/j.ijpe.2013.01.009

[35] Rahul Krishna, Amritanshu Agrawal, Akond Rahman, Alexander Sobran, and Tim Menzies. 2018. What is the Connection between Issues, Bugs, and Enhancements? Lessons Learned from 800+ Software Projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice* (Gothenburg, Sweden) *(ICSE-SEIP '18)*. Association for Computing Machinery, New York, NY, USA, 306–315. https://doi.org/10.1145/3183519.3183548

[36] Microsoft Learn. 2023. Blue screen data. https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/blue-screen-data 01.

[37] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2020. The M4 Competition: 100, 000 time series and 61 forecasting methods. *International Journal of Forecasting* 36, 1 (Jan. 2020), 54–74. https://doi.org/10.1016/j.ijforecast.2019.04.014

[38] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming Performance Variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 409–425. https://www.usenix.org/conference/osdi18/presentation/maricq

[39] Kenneth O McGraw and Seok P Wong. 1992. A common language effect size statistic. *Psychological bulletin* 111, 2 (1992), 361.

[40] Engin Cemal Mengüç and Nurettin Acır. 2018. Kurtosis-Based CRTRL Algorithms for Fully Connected Recurrent Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 29, 12 (2018), 6123–6131. https://doi.org/10.1109/TNNLS.2018.2826442

[41] Erik Otović, Marko Njirjak, Dario Jozinović, Goran Mausa, Alberto Michelini, and Ivan Štajduhar. 2021. Intra-domain and cross-domain transfer learning for time series data – How transferable are the features? *Knowledge-Based Systems* 239 (12 2021), 107976. https://doi.org/10.1016/j.knosys.2021.107976

[42] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. 2020. Deep Adaptive Input Normalization for Time Series Forecasting. *IEEE Transactions on Neural Networks and Learning Systems* 31, 9 (2020), 3760–3765. https://doi.org/10.1109/TNNLS.2019.2944933

[43] M. Peixeiro. 2022. *Time Series Forecasting in Python.* Manning.

[44] Francesco Piccialli, Fabio Giampaolo, Edoardo Prezioso, David Camacho, and Giovanni Acampora. 2021. Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion. *Information Fusion* 74 (2021), 1–16. https://doi.org/10.1016/j.inffus.2021.03.004

[45] Casey Rosenthal and Nora Jones. 2020. *Chaos engineering: system resiliency in practice.* O'Reilly Media.

[46] Julia Rubin and Martin Rinard. 2016. The Challenges of Staying Together While Moving Fast: An Exploratory Study. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas) *(ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 982–993. https://doi.org/10.1145/2884781.2884871

[47] Alaa Sagheer and Mostafa Kotb. 2019. Time series forecasting of petroleum production using deep LSTM recurrent networks. *Neurocomputing* 323 (2019), 203–213. https://doi.org/10.1016/j.neucom.2018.09.082

[48] Jeffrey Shaman and Alicia Karspeck. 2012. Forecasting seasonal outbreaks of influenza. *Proceedings of the National Academy of Sciences* 109, 50 (2012), 20425–20430. https://doi.org/10.1073/pnas.1208772109 arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1208772109

[49] D.P. Siewiorek and R.S. Swarz. 1998. *Reliable Computer Systems: Design and Evaluation.* A K Peters. https://books.google.it/books?id=EOrVzQEACAAJ

[50] James H. Stock and Mark W. Watson. 2002. Forecasting Using Principal Components from a Large Number of Predictors. *J. Amer. Statist. Assoc.* 97, 460 (2002), 1167–1179. http://www.jstor.org/stable/3085839

[51] Suhartono Suhartono. 2011. Time Series Forecasting by using Seasonal Autoregressive Integrated Moving Average: Subset, Multiplicative or Additive Model. *Journal of Mathematics and Statistics* 7 (01 2011), 20–27. https://doi.org/10.3844/jmssp.2011.20.27

[52] Yang Syu, Jong-Yih Kuo, and Yong-Yi Fanjiang. 2017. Time series forecasting for dynamic quality of web services: An empirical study. *Journal of Systems and Software* 134 (2017), 279–303. https://doi.org/10.1016/j.jss.2017.09.011

[53] Luca Traini and Vittorio Cortellessa. 2023. DeLag: Using Multi-Objective Optimization to Enhance the Detection of Latency Degradation Patterns in Service-Based Systems. *IEEE Transactions on Software Engineering* 49, 6 (2023), 3554–3580. https://doi.org/10.1109/TSE.2023.3266041

[54] Luca Traini, Vittorio Cortellessa, Daniele Di Pompeo, and Michele Tucci. 2022. Towards effective assessment of steady state performance in Java software: are we there yet? *Empirical Software Engineering* 28, 1 (2022), 13. https://doi.org/10.1007/s10664-022-10247-x

[55] Luca Traini, Daniele Di Pompeo, Michele Tucci, Bin Lin, Simone Scalabrino, Gabriele Bavota, Michele Lanza, Rocco Oliveto, and Vittorio Cortellessa. 2021. How Software Refactoring Impacts Execution Time. *ACM Trans. Softw. Eng. Methodol.* 31, 2, Article 25 (dec 2021), 23 pages. https://doi.org/10.1145/3485136

[56] W.M. Turski. 2002. The reference model for smooth growth of software systems revisited. *IEEE Transactions on Software Engineering* 28, 8 (2002), 814–815. https://doi.org/10.1109/TSE.2002.1027802

[57] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the "CL" Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. http://www.jstor.org/stable/1165329

[58] Kaushik Veeraraghavan, Justin Meza, David Chou, Wonho Kim, Sonia Margulis, Scott Michelson, Rajesh Nishtala, Daniel Obenshain, Dmitri Perelman, and Yee Jiun Song. 2016. Kraken: Leveraging Live Traffic Tests to Identify and Resolve Resource Utilization Bottlenecks in Large Scale Web Services. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 635–651. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/veeraraghavan

[59] William Wei. 2006. *Time Series Analysis: Univariate and Multivariate Methods, 2nd edition, 2006.*

[60] Gary White, Andrei Palade, and Siobhán Clarke. 2018. Forecasting QoS Attributes Using LSTM Networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2018.8489052

[61] Frank Wilcoxon. 1992. *Individual Comparisons by Ranking Methods.* Springer New York, New York, NY, 196–202. https://doi.org/10.1007/978-1-4612-4380-9_16

[62] D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82. https://doi.org/10.1109/4235.585893

[63] Hongju Yan and Hongbing Ouyang. 2018. Financial Time Series Prediction Based on Deep Learning. *Wireless Personal Communications* 102, 2 (01 Sep 2018), 683–700. https://doi.org/10.1007/s11277-017-5086-2

[64] Subhash Chand Yogi, Vibhu Kumar Tripathi, and Laxmidhar Behera. 2021. Adaptive Integral Sliding Mode Control Using Fully Connected Recurrent Neural Network for Position and Attitude Control of Quadrotor. *IEEE Transactions on Neural Networks and Learning Systems* 32, 12 (2021), 5595–5609. https://doi.org/10.1109/TNNLS.2021.3071020

[65] G. Peter Zhang. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* 50 (2003), 159–175.

[66] Xiaofeng Zhou, Naiju Zhai, Shuai Li, and Haibo Shi. 2023. Time Series Prediction Method of Industrial Process With Limited Data Based on Transfer Learning. *IEEE Transactions on Industrial Informatics* 19, 5 (2023), 6872–6882. https://doi.org/10.1109/TII.2022.3191980