applied sciences

MDPI

Article

hLSTM-Aging: A Hybrid LSTM Model for Software Aging Forecast

Felipe Battisti [1], Arnaldo Silva [1], Luis Pereira [1], Tiago Carvalho [1], Jean Araujo [1,*], Eunmi Choi [2], Tuan Anh Nguyen [3,4,*] and Dugki Min [4,*]

[1] Departamento de Informática, Universidade Federal do Agreste de Pernambuco, Garanhuns 55292-270, Brazil; felipebattist@gmail.com (F.B.); arnaldovitorbarros@gmail.com (A.S.); luis-filipe.pereira@ufape.edu.br (L.P.); tiago.buarque@ufape.edu.br (T.C.)
[2] School of Software, College of Computer Science, Kookmin University, Seoul 02707, Korea; emchoi@kookmin.ac.kr
[3] Konkuk Aerospace Design-Airworthiness Research Institute (KADA), Konkuk University, Seoul 05029, Korea
[4] Department of Computer Science and Engineering, College of Engineering, Konkuk University, Seoul 05029, Korea
* Correspondence: jean.teixeira@ufape.edu.br (J.A.); anhnt2407@konkuk.ac.kr (T.A.N.); dkmin@konkuk.ac.kr (D.M.)

Abstract: Long-running software, such as cloud computing services, is now widely used in modern applications. As a result, the demand for high availability and performance has grown. However, these applications are more vulnerable to software aging issues and are more likely to fail due to the accumulation of mistakes in the system. One popular strategy for dealing with such aging-related problems is to plan prediction-based software rejuvenation activities based on previously obtained data from long-running software. Prediction algorithms enable the activation of a mitigation mechanism before the problem occurs. The long short-term memory (LSTM) neural network, the present state of the art in temporal series prediction, has demonstrated promising results when applied to software aging concerns. This study aims to anticipate software aging failures using a hybrid prediction model integrating long short-term memory models and statistical approaches. We emphasize the capabilities of each strategy in various long-running software scenarios and provide an untried hybrid model (hLSTM-aging) based on the union of Conv-LSTM networks and probabilistic methodologies, attempting to combine the strengths of both approaches for software aging forecasts. The hLSTM-aging prediction results revealed how hybrid models are a compelling solution for software-aging prediction. Experiments showed that hLSTM-aging increased MSE criteria by 8.54% to 50% and MAE criteria by 3.53% to 14.29% when compared to Conv-LSTM, boosting the model's initial performance.

Keywords: time series; software aging; resources prediction; mitigating techniques; LSTM

check for updates

Citation: Battisti, F.; Silva, A.; Pereira, L.; Carvalho, T.; Araujo, J.; Choi, E.; Nguyen, T.A.; Min, D. hLSTM-Aging: A Hybrid LSTM Model for Software Aging Forecast. *Appl. Sci.* **2022**, *12*, 6412. https://doi.org/10.3390/app12136412

Academic Editor: Peng-Yeng Yin

Received: 28 May 2022
Accepted: 20 June 2022
Published: 24 June 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

As the prevalence of software programs in our everyday lives grows, the demand for availability and high performance has emerged as a critical problem in software development. As a result, many cloud computing systems now function for extended periods without failing or going offline. This long operating duration, however, can lead to system failures and deterioration of the program's internal state, i.e., software aging [1].

Memory leakage, failure to relinquish file handles or connections, fragmentation-related difficulties, the recurrence of failures, data corruption, and other factors can all contribute to software aging, according to [2]. Isolated software faults do not usually result in system failure, but they can accumulate and cause error propagation across the software system. Such proliferation can cause the system to fail catastrophically.

As a result, there are several potential solutions for avoiding critical errors caused by the aging process of software. Alonso et al. [3] approach this problem as a prediction problem, monitoring certain selected applications and their usage of system resources over time.

Using such information, one may foresee where the system will begin to exhibit software aging issues and subsequently execute rejuvenation to avert them. Some existing techniques in the state of the art for predicting aging-related failures, such as in our situation, include employing time series [4], neural networks [5], and long short-term memory (LSTM) networks [6].

The earlier-mentioned problem of accumulating individual mistakes might be challenging to see empirically. However, such cumulative mistakes frequently leave unambiguous indications of system resource expenditure. A healthy program, for example, oscillates less in terms of speed and memory use.

By monitoring a system's resource usage over time, it is possible to determine when indicators of aging arise and when mitigation and rejuvenation actions should start execution. Furthermore, because this data collections spreads over time, it is only logical to interpret them as time series, that is, changeable data points tracked across time.

There are several algorithms commonly used for time series analysis that use statistical concepts, including the moving average (MA) [7] and its variants, the auto-regressive integrated moving average (ARIMA) [8] and the seasonal auto-regressive integrated moving average (SARIMA) [9]. Both models are suited to analyze time-series data or predict future points in the series. In addition, ARIMA models apply in some cases where the data show evidence of non-stationary behaviors. The model can use an initial differentiation step (corresponding to the "integrated" part of the model) one or more times to eliminate non-stationary characteristics. SARIMA works similarly to ARIMA but considers the seasonality parameter, thus making it more suitable for problems where the data have cyclic patterns. These algorithms, along with the auto regressive (AR) and the auto regressive moving average (ARMA) [10] that follow the same pattern, are explored in this research.

Recurrent neural networks, in addition to the statistical techniques outlined in the preceding paragraph, are cutting-edge for time-series analysis. Because of its capacity to discern patterns in sequences, this sort of network effectively forecasts new values in time series. LSTM models, on the other hand, are commonly utilized in this forecasting context at the cutting edge.

Although LSTM models have shown promising results in fitting patterns of software aging [6], as time-series challenges grow with technology, new architectures built from the original LSTM might be employed to fit more complicated data and tackle other problems. When selecting the appropriate technique for predicting software aging issues, the range of hybrid models for LSTM might be deceptive.

In this sense, the purpose of this work is to investigate how different LSTM models fit data acquired from software aging. We are looking for the ideal model for triggering a software rejuvenation mechanism before a chain propagation of faults leads to a critical failure of the program. In addition, we used the knowledge gained from the testing and studies to present a novel hybrid model based on probabilistic approaches and LSTM networks that may use the benefits of both strategies for time-series forecasting in software degradation situations.

The contributions of this work can be summarized as follows:

- A hybrid LSTM model (hLSTM-Aging) for software aging prediction integrating two highly accurate predictors: (i) the convolutional long short-term memory (Conv-LSTM) Network and (ii) the moving average (MA) model. The hybrid model may employ the MA model's capacity to provide strong but noisy predictions and then use the Conv-LSTM to filter the most significant features without sacrificing the accuracy of the real data pattern and make it useful to forecast a time series in the context of software aging.

- A software aging prediction framework for real-world aging prediction of software systems based on the proposed hLSTM-aging model.
- Extensive tests for predicting software aging utilizing a range of LSTM or MA architectures, along with the proposed hybrid model, for thorough analogies.

The remainder of this paper is organized as follows. Section 2 presents some related works. Section 3 presents the basic concepts about software aging and rejuvenation and time series. Section 4 describes the hybrid model. Section 5 presents the details about the experiments. In Section 6, we summarize the obtained results. Finally, Section 8 presents the conclusions and future works.

2. Related Works

A diverse range of machine learning models has been developed to forecast error propagation and detect potential software failures before they happen in the field of software aging. In this section, we present some research findings in this area, as well as those demonstrating ways to predict time series in other fields that might be applicable to computer system aging forecasting.

Zheng et al. [11] proposed a way of interpreting Just-in-Time software defect prediction using the random forest classifier as the base for a defect forecasting model as a solution to software aging problems. Compared to random prediction, their method increased accuracy by 36.8% on average. An investigation by Malhotra and Kamal [12] aimed to build machine learning classifiers that could predict defects in NASA datasets with imbalance. The authors' results suggest that machine learning classifiers by themselves cannot predict effectively. However, by applying oversampling methods, they can improve predictions. Using the ARIMA model to predict Eucalyptus Cloud resource consumption, Araujo et al. [13] showed the utility of probabilistic methods to the prediction of software aging. Using a machine learning model called Error Rate Reduction Pruning (ERP), Avresky et al. [14] constructed a framework for predicting the average failure rate of virtual machines in different cloud regions. Qiao et al. [15] used LSTM to predict software aging on Android systems in 2018. The authors evaluated performance using traditional metrics such as MAPE and MSE and found that LSTM had superior performance to other deep learning algorithms. Liu et al. [6] investigated the aging of cloud computing software using a hybrid ARIMA and LSTM method. The results obtained by their method were better than those of individual methods. Unfortunately, there is little information on why ARIMA and the Vanilla-LSTM model are effective for the hybrid algorithm. There are many different probabilistic methods and LSTM networks available in the current state of the art, and it would be helpful to have a better understanding of which ones are the most effective. We discuss related studies on other subjects related to this research and indicate a number of techniques suitable to be implemented for data forecasting of software aging problems in the following paragraphs.

Wang et al. [16] employed a hybrid method of the ARIMA algorithm and the BI-LSTM neural network to predict the early stabilization time of electrolytic capacitors. This work indicatesthat probabilistic models such as ARIMA can handle the linear parts of a time series, while the BI-LSTM is better for extracting the non-linear features.

As for the analysis of the fundamentals of software aging, Grottke et al. [1] demonstrated that the accumulation of internal errors in cloud software should lead to software aging problems and very likely an eventual software crash. Energy efficiency and carbon dioxide emission reduction can be interpreted as time series, and Xu et al. [17] proposed using auto-regressive models (AR) to analyze this type of data and were able to identify very important factors in reducing carbon dioxide emissions, such as energy efficiency. The moving average model (MA), employed by Ivanovski et al. [18] for accurate projection of the number of guests quarterly for one year in the future, allows sufficient preparation time for all parties engaged in the tourist sector. Other auto-regression models are also used for time-series analysis, such as the auto-regressive moving average (ARMA) used by Pappas et al. [19] to model the electricity demand loads in Greece—the model developed can be used to successfully predict the price of electricity—and the auto-regressive inte-

grated moving average (ARIMA) used by Kumar et al. [8] to predict sugarcane production in India. Studies indicated that between 2015 and 2017 sugarcane production would have an average growth rate of approximately 3% per year. Finally, the seasonal auto-regressive integrated moving-average (SARIMA) by Permanasari et al. [9] was used to predict the number of disease incidences in humans, more specifically to predict the occurrence of malaria in the United States.

Already in 2021, Yayam et al. [20] used a special deep neural network, the Stacked-LSTM, to estimate the state of health and the state of charge in li-ion batteries. The tests indicated this method is effectively applicable to quick chargers. In addition in 2021, Rahman et al. [21] proposed a language model for evaluating and finding logical errors in source codes using a bidirectional LSTM neural network, training the BiLSTM with a large number of source codes, outperforming other state-of-the-art models with recurrent neural networks (RNNs), and achieving an F-score of approximately 97%. Finally, He et al. [22] proposed the use of LSTM layers with additional convolutional layers to increase the performance of prediction models aiming to offer insights into gold price fluctuations. Several experiments were carried out on real data sets collected from the World Gold Council showing that this approach outperforms other classical methods of financial forecasting.

3. Background

This section presents an overview of software aging and rejuvenation and time series, which are essential for understanding this study.

3.1. Software Aging and Rejuvenation

The cumulative impact of faults propagated during a program's lifetime result in the development of software aging problems [1]. It should be mentioned that the aging-related effect on software is generally evident after a lengthy period of continuous operation [13].

Monitoring resource consumption over time is the first step in understanding the aging problem. Once the aging effects are identified, mitigating techniques may be implemented to lessen the aging consequences on the application [2]. These mitigation measures are referred to as software rejuvenation, a proactive fault management strategy aimed at cleaning up the system's internal state to prevent more severe crashes or failures in the future [23].

It is also critical to be accurate when deciding when to begin mitigation strategies, as this strategy can be quite costly in terms of processing time and resources [24]. When there is still a safety margin, picking the closest feasible point to a probable system failure is crucial to guarantee that the mitigation procedures are performed before any major software defects arise.

The software restart [25] is an example of a rejuvenation approach. As a result, the old program is stopped, and a new comparable process is established to replace it. Typically, this type of replacement may erase the aging effects acquired throughout the previous process's run-time. The fundamental issue is that when the system must be restarted, there is a downtime overhead produced by the restart operation, and the program becomes inaccessible during the rejuvenation mechanisms' execution.

3.2. Time Series

A time series is a representation of collected data across time. Time series are commonly used for data description, interpretation, process management, and prediction [26]. Time-series prediction is a procedure that aims to examine information from data from previous periods of time to forecast future values for the same data [27]. The sequence of events in a time series can have a significant impact on the complexity of prediction models. Recurrent neural networks (RNN) are commonly employed in time-series prediction because they maintain the results of prior calculations. The main disadvantage of utilizing a conventional RNN is that it takes a long time to train a model.

An alternative proposal is the long short-term memory networks (LSTM) [28], a type of recurrent neural network capable of learning order dependence in sequence prediction problems. As a consequence, it is also largely used to fit time-series data.

An LSTM has a chain structure that includes four neural networks and several memory units known as cells. First, significant information is added to the neuron via the input gate. The forget gate then removes information that is no longer helpful in the present neuron state. The bias is applied to the current and prior inputs after they have been multiplied by the weight matrices. The result is fed into a binary activation function (similar to sigmoid). If the output state is zero, the information is deleted. If the output state is one, the information is saved for later use. Finally, the output gate is in charge of retrieving useful data from the neuron and sending it to the next neuron.

There are also probabilistic methods in statistics, such as auto-regressive (AR), moving average (MA) [7], auto-regressive moving average (ARMA) [10], auto-regressive integrated moving average (ARIMA) [8], and seasonal auto-regressive integrated moving average (SARIMA) [9], that can be used for time-series forecasting. These techniques are all based on probabilistic notions.

The auto-regressive (AR) model is a multiple regression model in statistics that anticipates the variable of interest in a time series using a linear combination of previous variable values. It is auto-regressive because it regresses the variable against itself. An auto-regressive model, often known as order p, may be expressed as

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \tag{1}$$

where $\varphi_1, \ldots, \varphi_p \varphi_1, \ldots, \varphi_p$ are parameters, c is a constant, and the random variable $\varepsilon_t \varepsilon_t$ is white noise [29].

The moving average model [30] states a linear co-dependence of the time series current value with the past error terms. It is assumed that such error terms are independent and regularly distributed. MA(q) denotes the moving average model of q order. The actual value is expressed by the model as a linear combination of the time-series mean, the current error term, and the prior error term. Mathematically, the expression is,

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{2}$$

where μ stands for the time-series mean, ε represents the present error, and θ is the past error terms.

The auto-regressive–moving-average (ARMA) model is based on the assumption that the time series is stationary and varies evenly around a specific period. Because it depicts a time series by running it through a recursive filter and then a non-recursive linear filter, it operates as a hybrid of the MA and AR models. The ARMA order consists of two integers (p,q) that serve as the orders of the AR and MA components, respectively. The ARMA model is noted as:

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i x_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}. \tag{3}$$

The auto-regressive integrated moving average (ARIMA) model is a generalization of the ARMA model. The "integrated" component of the term refers to the reversal of differences. An ARIMA(p, q, d) utilizes the AR and MA model with a different variable d which is used to remove the trend and convert a non-stationary time series to a stationary one. The seasonal auto-regressive integrated moving average (SARIMA) is an ARIMA extension that takes into consideration a time series' seasonal information.

4. hLSTM-Aging: A Hybrid LSTM Model

According to Liu et al. [6], combining probabilistic approaches and LSTM networks improves prediction power for time series in the context of software aging. The Conv-LSTM network and the moving average (MA) model are used in our suggested technique, which combines two highly accurate predictors.

These two models were chosen after conducting exploratory studies on all of the LSTM architectures and probabilistic models. In our studies, both MA and Conv-LSTM performed better overall.

Intuitively, this combination makes a lot of sense. The MA can provide accurate but noisy forecasts. We may select the most significant properties using Conv-LSTM convolutions without sacrificing integrity to the real data pattern, making it meaningful to forecast a time series in the context of software aging. This combination is designed as follows:

Given a time series, represented by T, and with n points in time:

$$T = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & ... & t_n \end{bmatrix} \tag{4}$$

To generate a new P time-series representation, we employ the moving average model. To do so, we train the MA model on the T series, and after training, the MA model predicts a value for each T-value. The MA model's output is the new representation of what we term P.

$$P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & ... & p_n \end{bmatrix} \tag{5}$$

The P time series must then be restructured for the prediction procedure. The size of the time-series subsets, which will serve as X inputs in the prediction job, is first specified. Then, using the information from sub-sequence X, a second number is picked to specify how far forward in the time series the suggested approach should anticipate. As an example, if the value of points ahead is one and the size of the sub-sequence is four, we will have an $X_{initial}$ input and $Y_{initial}$ target in the following formats.

$$X_{initial} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 & p_4 \end{bmatrix} \tag{6}$$

$$Y_{initial} = P_5 \tag{7}$$

In this data format with defined sets X and Y, simple LSTM models could already give the final prediction of the data. However, since we chose to use Conv-LSTM, all the points X_i of X must be in an $m \times m$ format that resembles an image.

For this example, we picked four as the sub-sequence size number for the X points on purpose because these sub-sequences must be transformed into $m \times m$ matrices, where the size of the sub-sequence must equal the square of m. So, in this case, $m = 2$. Furthermore, as a general rule:

$$SS = m^2 \tag{8}$$

where SS stands for sub-sequence size.

Figure 1 shows how a point with size four, predicting one point ahead in the time series, is structured for the Conv-LSTM model.

The Conv-LSTM model has a straightforward structure (Figure 2). To make a prediction, we first employ a layer using the Conv-LSTM and 64 filters, followed by a dense layer in the output. After defining the Conv-LSTM, we utilize the model to fit the time series P. Then, using the Conv-LSTM, we anticipate the original time series, completing the flow of the suggested technique. The complete workflow is illustrated in Figure 3.

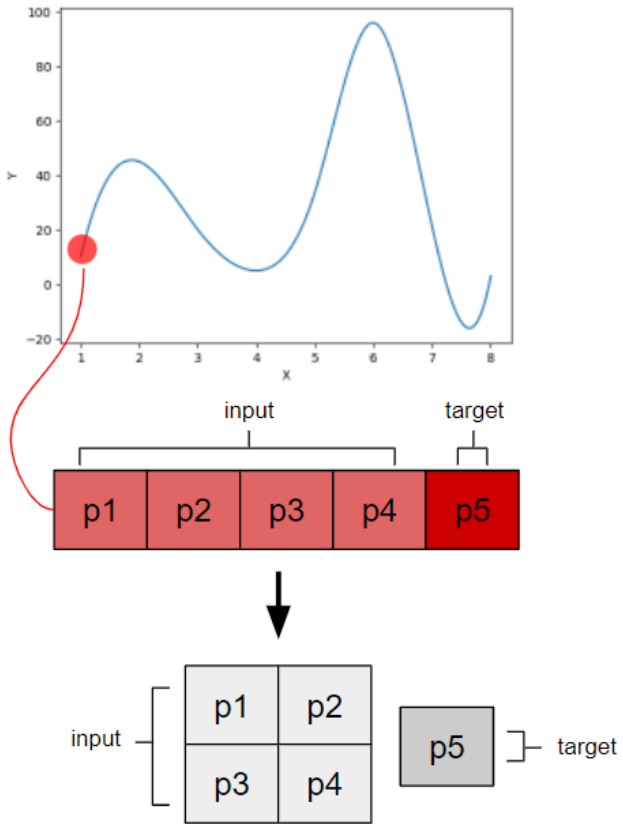


Figure 1. Data structure for Conv-LSTM model.

Algorithm 1 below displays the proposed method structure. Given a time series T, an MA model is trained and then fed with the same time series T, generating a time series of errors P. As stated before, P and T need a restructure operation before the ConvLSTM workflow. Therefore, by training ConvLSTM with error time series P and predicting with the original series T, we have F as the final time series predicted by the hybrid method.

Algorithm 1 The hLSTM algorithm

Input: T
Output: F
$MA.fit(T)$
$P \leftarrow MA.predict(T)$
$P \leftarrow reshape(P)$
$T \leftarrow reshape(T)$
$ConvLSTM.fit(P)$
$F \leftarrow ConvLSTM.predict(T)$

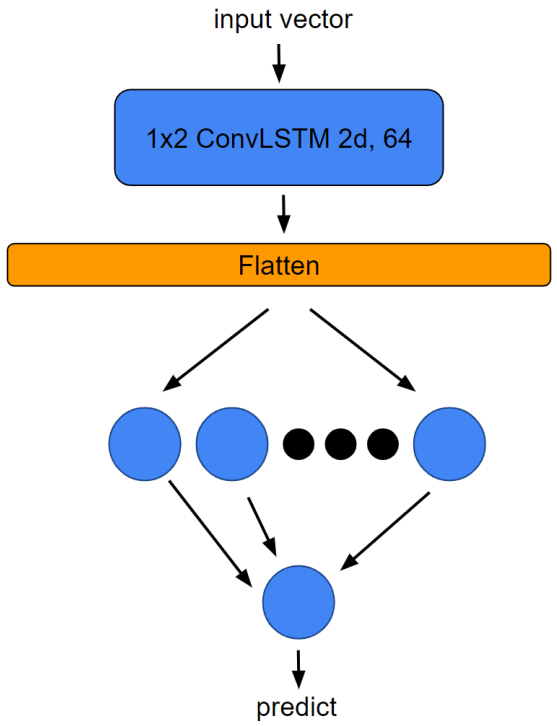


Figure 2. Conv-LSTM architecture.

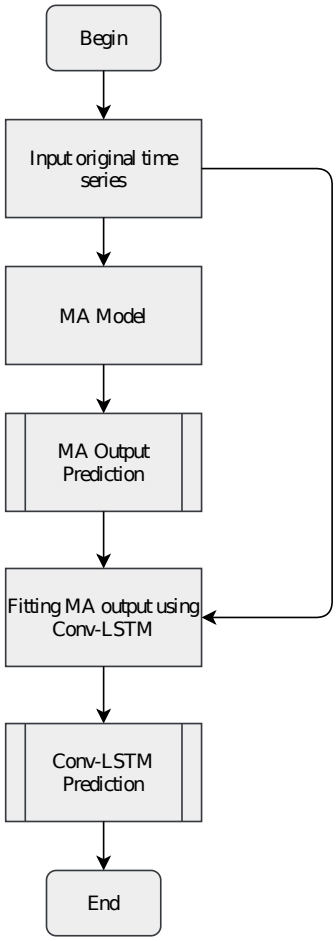


Figure 3. hLSTM-Aging prediction process.

5. Experiments and Analyses

5.1. Database

Oliveira et al. [31] generated the program aging monitoring data sets. Our trials used four measures, listed in order of prediction difficulty: (i) CPU utilization, (ii) disk usage, (iii) virtual memory resident set size, and (iv) memory used. These four indicators were selected because they are well-known metrics that are frequently tracked in the context of software aging. Furthermore, their behavior and complexity varies. They may then be used to test the trained models in various circumstances.

5.2. Evaluation Metrics

We measured the accuracy of the time-series prediction using three error metrics: MAPE, MSE, and MAE [32].

5.2.1. MAPE

The mean absolute percentage error (MAPE) expresses the accuracy as a ratio that is equivalent to the following expression:

$$\text{MAPE} = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{y_t - \hat{y}_t}{y_t} \right|$$

where y_t is the actual value from the time series and $\hat{y}_t$ is the value predicted by the model in evaluation. Therefore, the MAPE measures the average prediction error ratio concerning the series true value. It is important to mention that this metric is very sensitive to error and can fail due to a division by zero.

5.2.2. MSE

The mean squared error (MSE) measures the average difference between the predicted and the real values. The following formula defines the MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

where y_i and $\hat{y}_i$ are the real and predicted value, respectively.

5.2.3. MAE

The mean absolute error (MAE) measures the error between paired observations expressing the same phenomenon. In prediction cases, the y_i can be the predicted value and $\hat{y}_i$ the actual value observed. The following formula can define it:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$

5.3. Experimental Design

The LSTM architectures chosen for the experiments were: (i) Vanilla-LSTM, (ii) Stacked-LSTM, (iii) bidirectional LSTM, (iv) CNN LSTM and (v) ConvLSTM. As cited in the related works section, these LSTM architectures showed satisfactory results in the available research. They are standard in the state of the art, and since the experiment aimed to compare the hLSTM with a variety of non-hybrid methods, these architectures fit the purpose. The hyper-parameters were the same in all experiments conducted: the learning rate was 0.001. The train and test sets kept a design so that every four sequential numbers in the time series acted as a point, and the fifth number was the output for the point. In the case of the CNN LSTM and the Conv-LSTM, the input vector needs a reshape, turning the four numbers point into a 2d volume of 2×2 dimension. The LSTM and variant models used 100 epochs for training, and the loss function used was the MSE.

The software aging time series chosen for evaluation were CPU utilization, disk usage, virtual memory resident set size, and the memory used in two different scenarios, with only one virtual machine in the system while the second case had five. These time series were divided using 75% percent of the data for training and the rest for testing. As a comparison method, a table containing the MSE, MAPE, and MAE metrics was generated for each time series evaluated.

In addition, the generated graphs had the same test and training division for all experiments, and the data display form was also the same. First, the original data was plotted in blue, then the prediction values for the train set in red were added to the graph, and finally, the values in green refer to the predicted data for the test set. The *Y*-axis represents the actual values according to the tested data, and the *X*-axis represents the passage of time in minutes.

6. Results Analysis

The *Y*-axis represents the watched software measures in the following plots, and the *X*-axis represents the time in the aging scale relative to the collected data.

6.1. Metrics

For the CPU utilization, Table 1 shows a clear difference between the accuracy of the LSTM networks and the probabilistic methods according to the metrics used. While the LSTM-based methods were superior to MSE results, the probabilistic methods showed slightly better MAPE values. Among the LSTM models, the Conv-LSTM had better results but was not very far from the other LSTM models. The SARIMA model had the best results among the statistical methods compared to the rest of the methods. The proposed combination (moving average + Conv-LSTM) had better results when compared to the individual methods in almost all of the evaluation metrics.

Table 1. Results of evaluation metrics for CPU utilization data.

Architecture	MSE-Train/Test	MAPE-Train/Test	MAE-Train/Test
Vanilla-LSTM	$5 \times 10^{-6}/3 \times 10^{-5}$	$1.01 \times 10^{-1}/1.95 \times 10^{-1}$	$1.6 \times 10^{-3}/6.1 \times 10^{-3}$
Stacked-LSTM	$1 \times 10^{-5}/2 \times 10^{-4}$	$1.36 \times 10^{-1}/5.06 \times 10^{-1}$	$2 \times 10^{-3}/1.5 \times 10^{-2}$
Bi-LSTM	$6 \times 10^{-6}/2 \times 10^{-6}$	$1.09 \times 10^{-1}/1.8 \times 10^{-2}$	$1 \times 10^{-3}/5 \times 10^{-4}$
CNN-LSTM	$6 \times 10^{-6}/5 \times 10^{-5}$	$1.11 \times 10^{-1}/2.24 \times 10^{-1}$	$1 \times 10^{-3}/7 \times 10^{-3}$
Conv-LSTM	$\mathbf{6 \times 10^{-6}/3 \times 10^{-5}}$	$\mathbf{1.04 \times 10^{-1}/1.43 \times 10^{-1}}$	$\mathbf{1 \times 10^{-3}/4 \times 10^{-3}}$
AR	$1.548 \times 10^{0}/5 \times 10^{-4}$	$2 \times 10^{-4}/5 \times 10^{-3}$	$3 \times 10^{-4}/1.84 \times 10^{-2}$
MA	$1.140 \times 10^{0}/1.841 \times 10^{0}$	$3.03 \times 10^{-1}/4.31 \times 10^{-1}$	$3.25 \times 10^{-1}/1.350 \times 10^{0}$
ARMA	$1.845 \times 10^{0}/7 \times 10^{-3}$	div by zero$/7.8 \times 10^{-2}$	$7.643 \times 10^{0}/7.4 \times 10^{-2}$
ARIMA	$1.649 \times 10^{0}/1.1 \times 10^{-2}$	$2 \times 10^{-4}/2.9 \times 10^{-2}$	$3 \times 10^{-4}/9.5 \times 10^{-2}$
SARIMA	$\mathbf{1.739 \times 10^{0}/6.8 \times 10^{-2}}$	$\mathbf{1 \times 10^{-4}/7.0 \times 10^{-2}}$	$\mathbf{1 \times 10^{-4}/2.25 \times 10^{-1}}$
hLSTM	$\mathbf{5 \times 10^{-6}/2 \times 10^{-5}}$	$\mathbf{8.6 \times 10^{-2}/1.37 \times 10^{-1}}$	$\mathbf{2 \times 10^{-5}/4 \times 10^{-3}}$

Concerning the disk usage (Table 2), the results show similar behavior to the previous time series evaluated (CPU utilization). In addition, the moving average technique presented better results than the rest of the probabilistic methods and better MAPE results concerning LSTM-based methods. Once again, the LSTM models had similar results, but the BI-LSTM had better performance when compared to the others. Here, the proposed combination (moving average + Conv-LSTM) method had better performance in training than the individual methods and had a close result in the test data.

Table 2. Results of evaluation metrics for disk usage.

Architecture	MSE-Train/Test	MAPE-Train/Test	MAE-Train/Test
Vanilla-LSTM	$8 \times 10^{-6}/2 \times 10^{-5}$	$6.6361 \times 10^{1}/4.4 \times 10^{-2}$	$5 \times 10^{-4}/3 \times 10^{-4}$
Stacked-LSTM	$1 \times 10^{-5}/2 \times 10^{-5}$	$6.6272 \times 10^{1}/1.03 \times 10^{-1}$	$8 \times 10^{-4}/8 \times 10^{-4}$
Bi-LSTM	$\mathbf{9 \times 10^{-6}/2 \times 10^{-5}}$	$\mathbf{2.1420 \times 10^{1}/2.8 \times 10^{-2}}$	$\mathbf{6 \times 10^{-4}/1 \times 10^{-4}}$
CNN-LSTM	$1 \times 10^{-5}/5 \times 10^{-5}$	$7.8694 \times 10^{1}/7.4 \times 10^{-2}$	$1 \times 10^{-3}/5 \times 10^{-4}$
Conv-LSTM	$9 \times 10^{-6}/2 \times 10^{-5}$	$5.8370 \times 10^{1}/3.4 \times 10^{-2}$	$6 \times 10^{-4}/2 \times 10^{-4}$
AR	$7.019 \times 10^{0}/1.8 \times 10^{-2}$	$5 \times 10^{-4}/1.52 \times 10^{-1}$	$1 \times 10^{-4}/1.21 \times 10^{-1}$
MA	$\mathbf{1.3 \times 10^{-2}/1.02 \times 10^{-1}}$	$\mathbf{6.43 \times 10^{-1}/3.47 \times 10^{-1}}$	$\mathbf{9.6 \times 10^{-2}/2.92 \times 10^{-1}}$
ARMA	$7.021 \times 10^{0}/1.9 \times 10^{-2}$	$3 \times 10^{-3}/1.455 \times 10^{-1}$	$1 \times 10^{-4}/1.20 \times 10^{-1}$
ARIMA	$7.020 \times 10^{0}/3.0 \times 10^{-2}$	$4 \times 10^{-4}/1.84 \times 10^{-1}$	$1 \times 10^{-4}/1.24 \times 10^{-1}$
SARIMA	$7.021 \times 10^{0}/1.7 \times 10^{-2}$	$3 \times 10^{-4}/1.55 \times 10^{-1}$	$1 \times 10^{-4}/1.20 \times 10^{-1}$
hLSTM	$\mathbf{8 \times 10^{-6}/6 \times 10^{-5}}$	$\mathbf{1.23 \times 10^{-1}/6 \times 10^{-5}}$	$\mathbf{5 \times 10^{-4}/1 \times 10^{-3}}$

For the VmRSS time series (Table 3), the test results followed the same pattern we observed for the disk usage measure. The Vanilla-LSTM showed better results in the test experiments concerning the other LSTM-based models. For the probabilistic methods, the moving average technique had a better performance. The hybrid combination (moving average + Conv-LSTM) achieved better results in all the evaluation metrics.

Table 3. Results of evaluation metrics for VmRSS.

Architecture	MSE-Train/Test	MAPE-Train/Test	MAE-Train/Test
Vanilla-LSTM	$1 \times 10^{-4}/1.4 \times 10^{-2}$	$4.99 \times 10^{-1}/2.4194 \times 10^{5}$	$4 \times 10^{-3}/7.3 \times 10^{-2}$
Stacked-LSTM	$1 \times 10^{-4}/5.7 \times 10^{-2}$	$5.55 \times 10^{-1}/4.6351 \times 10^{5}$	$4 \times 10^{-3}/1.44 \times 10^{-1}$
Bi-LSTM	$\mathbf{9 \times 10^{-5}/3.73 \times 10^{-1}}$	$\mathbf{4.78 \times 10^{-1}/1.2014 \times 10^{6}}$	$\mathbf{4 \times 10^{-3}/3.65 \times 10^{-1}}$
CNN-LSTM	$1 \times 10^{-4}/1.3 \times 10^{-2}$	$6.59 \times 10^{-1}/2.3476 \times 10^{5}$	$5 \times 10^{-3}/1.3 \times 10^{-2}$
Conv-LSTM	$1 \times 10^{-4}/7 \times 10^{-3}$	$5.25 \times 10^{-1}/1.6974 \times 10^{5}$	$4 \times 10^{-3}/5.2 \times 10^{-2}$
AR	$8.937 \times 10^{0}/2.30 \times 10^{-1}$	3×10^{-3}/div by zero	$3 \times 10^{-3}/2.98 \times 10^{-1}$
MA	$\mathbf{3 \times 10^{-4}/2.30 \times 10^{-1}}$	**1.6×10^{-2}/div by zero**	$\mathbf{1.3 \times 10^{-2}/2.98 \times 10^{-1}}$
ARMA	$9.125 \times 10^{0}/2.29 \times 10^{-1}$	3×10^{-3}/div by zero	$2 \times 10^{-3}/2.96 \times 10^{-1}$
ARIMA	$9.397 \times 10^{0}/3.68 \times 10^{-1}$	2×10^{-3}/div by zero	$2 \times 10^{-3}/4.32 \times 10^{-1}$
SARIMA	$9.401 \times 10^{0}/2.72 \times 10^{-1}$	2×10^{-3}/div by zero	$2 \times 10^{-3}/3.52 \times 10^{-1}$
hLSTM	$\mathbf{6 \times 10^{-5}/1.5 \times 10^{-2}}$	$\mathbf{4.34 \times 10^{-1}/1.5 \times 10^{-2}}$	$\mathbf{3 \times 10^{-3}/7.7 \times 10^{-2}}$

In the one VM memory used table (Table 4), all the LSTM methods showed similar results, but the Conv-LSTM still demonstrated an advantage in the testing experiments, while for the statistical models, once again the moving average had a better performance when compared with the other methods. The results were better for the proposed combination (moving average + Conv-LSTM) than those from the training set's methods. The hybrid combination had slightly worse results in both MSE and MAE metrics but surpassed the individual methods in the MAPE metric in the test set.

Finally, for the memory used with five virtual machines (Table 5), the comparison of metrics demonstrated that Vanilla-LSTM achieved better test results in MSE and MAPE and a regular performance for MAE concerning the other LSTM methods. The ARIMA model achieved the best performance in all evaluation metrics compared to the other probabilistic methods. The proposed combination did not show better results when comparing the metrics.

Even though the LSTM models' results showed very similar patterns, both hybrid models BI-LSTM and Conv-LSTM stood above the others in a more general analysis. The moving average method had the best performance in the MSE metric in the probabilistic methods field. The LSTM methods were better at predicting the test set values when compared to the probabilistic methods. However, even with good results, the LSTM models still had trouble adjusting noisy data. As the hybrid model between ARIMA and

Vanilla-LSTM in [6] had good results, the combination of the MA and the Conv-LSTM model also demonstrated a way of improving the forecasting of the software aging problems with hybrid models.

Table 4. Results of evaluation metrics for memory used with one virtual machine.

Architecture	MSE-Train/Test	MAPE-Train/Test	MAE-Train/Test
Vanilla-LSTM	$4 \times 10^{-4}/7 \times 10^{-4}$	$1.944 \times 10^{0}/2.888 \times 10^{0}$	$6 \times 10^{-3}/2.2 \times 10^{-2}$
Stacked-LSTM	$4 \times 10^{-4}/6 \times 10^{-4}$	$2.098 \times 10^{0}/2.605 \times 10^{0}$	$7 \times 10^{-3}/1.9 \times 10^{-2}$
Bi-LSTM	$4 \times 10^{-4}/3 \times 10^{-4}$	$1.938 \times 10^{0}/1.351 \times 10^{0}$	$6 \times 10^{-3}/1.0 \times 10^{-2}$
CNN-LSTM	$4 \times 10^{-4}/4 \times 10^{-4}$	$2.066 \times 10^{0}/2.012 \times 10^{0}$	$7 \times 10^{-3}/1.5 \times 10^{-2}$
Conv-LSTM	$\mathbf{4 \times 10^{-4}/3 \times 10^{-4}}$	$\mathbf{1.998 \times 10^{0}/1.037 \times 10^{0}}$	$\mathbf{7 \times 10^{-3}/8 \times 10^{-3}}$
AR	$4 \times 10^{-4}/3.8 \times 10^{-2}$	$1.9 \times 10^{-2}/2.30 \times 10^{-1}$	$6 \times 10^{-3}/1.49 \times 10^{-1}$
MA	$\mathbf{8 \times 10^{-3}/1.09 \times 10^{-1}}$	$\mathbf{3.08 \times 10^{-1}/4.27 \times 10^{-1}}$	$\mathbf{7.9 \times 10^{-2}/3.18 \times 10^{-1}}$
ARMA	$4 \times 10^{-4}/1.1 \times 10^{-2}$	$1.7 \times 10^{-2}/1.19 \times 10^{-1}$	$6 \times 10^{-3}/7.5 \times 10^{-2}$
ARIMA	$4 \times 10^{-4}/6.0 \times 10^{-2}$	$2.0 \times 10^{-2}/3.04 \times 10^{-1}$	$7 \times 10^{-3}/2.00 \times 10^{-1}$
SARIMA	$4 \times 10^{-4}/1.29 \times 10^{-2}$	$1.77 \times 10^{-2}/1.2 \times 10^{-2}$	$6 \times 10^{-3}/7.8 \times 10^{-2}$
hLSTM	$\mathbf{2 \times 10^{-4}/1.3 \times 10^{-3}}$	$\mathbf{1.429 \times 10^{0}/1 \times 10^{-3}}$	$\mathbf{6 \times 10^{-3}/1.8 \times 10^{-2}}$

Table 5. Results of evaluation metrics for memory used with five virtual machines.

Architecture	MSE-Train/Test	MAPE-Train/Test	MAE-Train/Test
Vanilla-LSTM	$\mathbf{2.22 \times 10^{-5}/1.41 \times 10^{-5}}$	$\mathbf{5.85 \times 10^{-1}/5.92 \times 10^{-1}}$	$\mathbf{1.9 \times 10^{-3}/1.9 \times 10^{-3}}$
Stacked-LSTM	$2.15 \times 10^{7}/1.44 \times 10^{7}$	$9.20 \times 10^{-1}/6.06 \times 10^{-1}$	$2.81 \times 10^{3}/2.51 \times 10^{3}$
Bi-LSTM	$1.47 \times 10^{-4}/1.08 \times 10^{-4}$	$1.29 \times 10^{0}/5.58 \times 10^{-1}$	$7.0 \times 10^{-3}/5.1 \times 10^{-3}$
CNN-LSTM	$1.53 \times 10^{-4}/5.09 \times 10^{-4}$	$1.33 \times 10^{0}/1.99 \times 10^{0}$	$7.2 \times 10^{-3}/1.82 \times 10^{-2}$
Conv-LSTM	$1.99 \times 10^{-4}/1.56 \times 10^{-4}$	$1.58 \times 10^{0}/8.68 \times 10^{-1}$	$8.5 \times 10^{-3}/7.9 \times 10^{-3}$
AR	$1.46 \times 10^{-4}/1.08 \times 10^{-3}$	$3.72 \times 10^{-3}/4.22 \times 10^{-1}$	$1.08 \times 10^{-3}/1.32 \times 10^{-1}$
MA	$8.92 \times 10^{-3}/9.80 \times 10^{-2}$	$1.63 \times 10^{-1}/3.48 \times 10^{-1}$	$7.75 \times 10^{-2}/3.11 \times 10^{-1}$
ARMA	$1.57 \times 10^{-4}/4.92 \times 10^{-3}$	$1.18 \times 10^{-2}/6.57 \times 10^{-2}$	$6.46 \times 10^{-3}/5.99 \times 10^{-2}$
ARIMA	$\mathbf{1.61 \times 10^{-4}/8.87 \times 10^{-4}}$	$\mathbf{1.24 \times 10^{-2}/2.83 \times 10^{-2}}$	$\mathbf{6.73 \times 10^{-3}/2.56 \times 10^{-2}}$
SARIMA	$1.62 \times 10^{7}/4.55 \times 10^{8}$	$6.75 \times 10^{-3}/4.35 \times 10^{-2}$	$2.09 \times 10^{3}/1.82 \times 10^{4}$
hLSTM	$\mathbf{1.82 \times 10^{-4}/3.43 \times 10^{-4}}$	$\mathbf{1.41 \times 10^{0}/1.57 \times 10^{0}}$	$\mathbf{8.2 \times 10^{-3}/1.39 \times 10^{-2}}$

6.2. Visualization

In this section, the time-series models previously discussed are visually compared. Three result plots are displayed for each software aging time series tested for comparison: one for the LSTM-derived models, one for the probabilistic set of methods, and the other for a proposed combination. Those plots make it possible to understand how the models overlap the original time series and show that the LSTM models have a better generalization performance.

Analyzing the CPU utilization in Figure 4, all techniques can achieve a good prediction in the training set. However, when comparing the results for the test set, the Conv-LSTM and the hybrid method had a much better performance fitting the data.

For the disk usage plots in Figure 5, even though the MA in the training test was able to fit the pattern of the data, the bidirectional LSTM and the proposed combination both surpassed the probabilistic method.

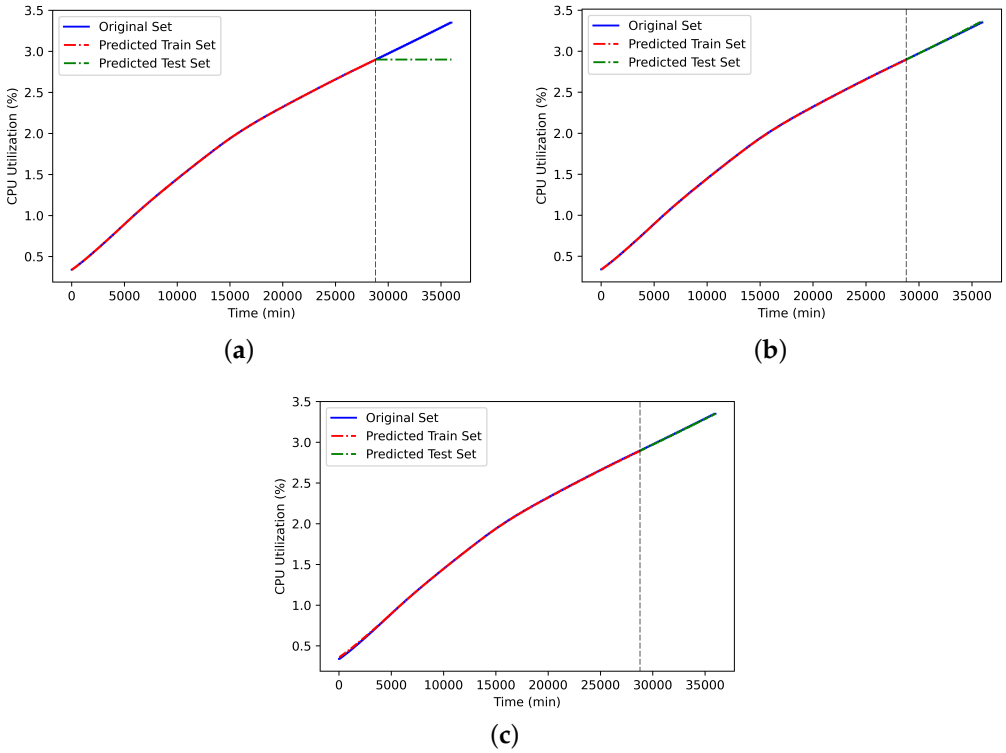


Figure 4. Graphics for data type "CPU utilization": (**a**) SARIMA; (**b**) Conv-LSTM; (**c**) proposed (moving average + Conv-LSTM).

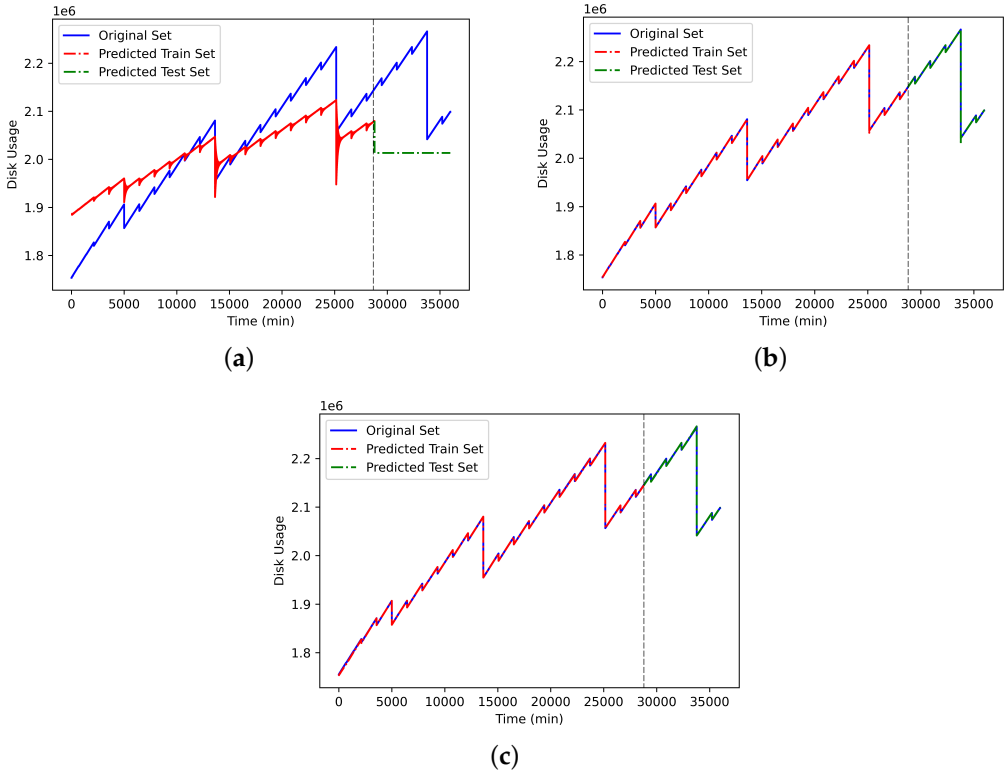


Figure 5. Graphics for data type "disk usage": (**a**) moving average; (**b**) bidirectional LSTM; (**c**) proposed (moving average + Conv-LSTM).

In the VmRSS plots in Figure 6, the MA had some trouble adapting to the most extreme points in the data and had a worse result in the test set, while the Vanilla-LSTM

and the proposed method were able to get a better generalization in both the training and testing sets.

For the memory used Figure 7 plots, it becomes clear that the Conv-LSTM method was able to fit the pattern of the data better, but the MA was able to achieve results closer to the spike points in the data. The combination of both the Conv-LSTM model and the moving average model achieved better results, being able to hit spike points and fit the curve pattern of the data.

Observing Figure 8, the idea initiated in the analysis of Figure 7 is reinforced. The LSTM method can adapt to time-series patterns but does not generalize well enough to hit spike points. On the other hand, the proposed combination achieves a satisfactory result in training and testing.

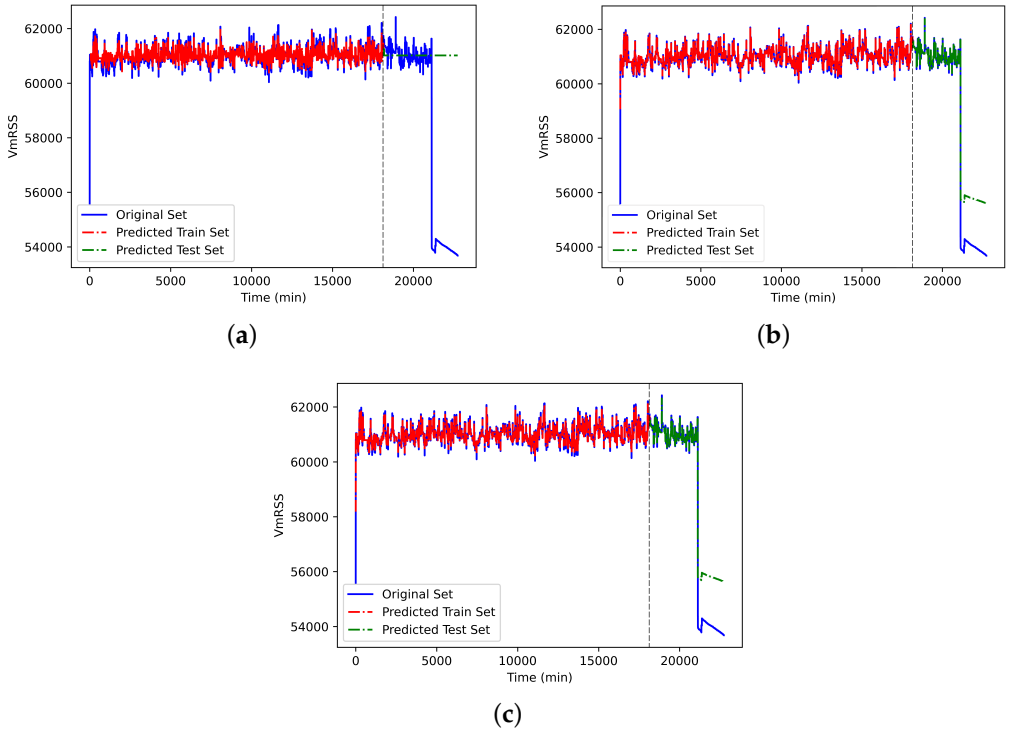


Figure 6. Graphics for data type "VmRSS": (**a**) moving average; (**b**) Vanilla-LSTM; (**c**) proposed (moving average + Conv-LSTM).

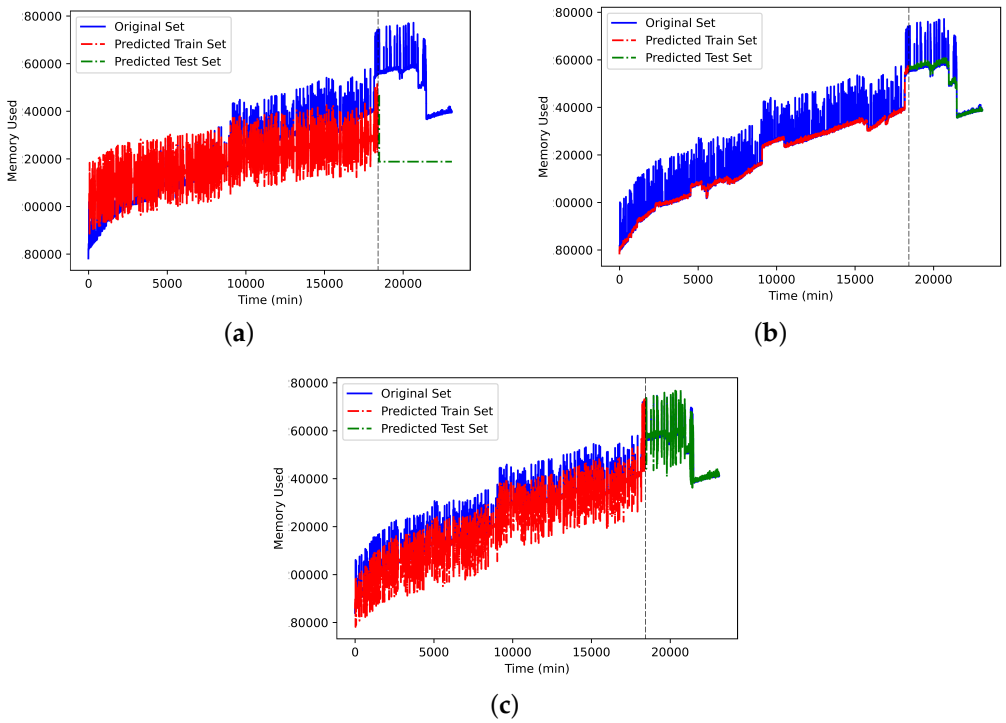


Figure 7. Graphics for data type "memory used with one VM": (**a**) moving average; (**b**) Conv-LSTM; (**c**) proposed (moving average + Conv-LSTM).

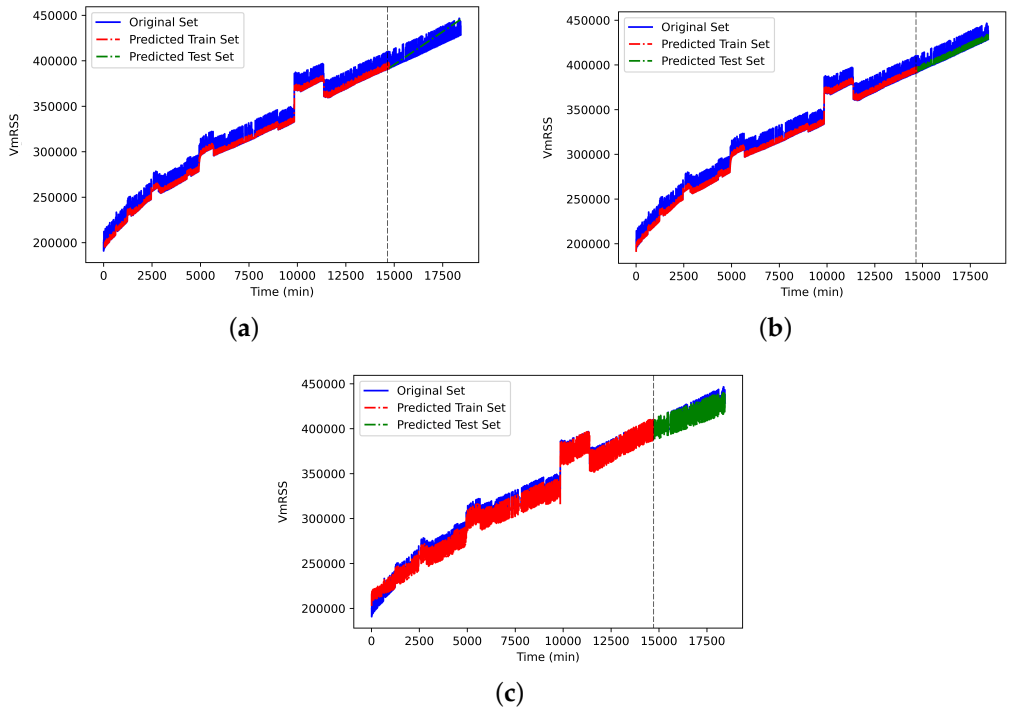


Figure 8. Graphics for data type "memory used with five VM": (**a**) ARIMA; (**b**) Vanilla-LSTM; (**c**) proposed (moving average + Conv-LSTM).

7. Discussion

The findings of the studies show that the probabilistic approaches during the learning phase focus mostly on the peak locations, whereas the LSTM methods focus on the growth and decay elements of the curves. Both qualities are critical in the context of failure

prevention; a peak point might indicate a memory overflow, and the time-series growth and decay pattern makes it easier to predict how resource consumption will be after a given period. Nonetheless, the combination of both techniques appears to be the most appropriate way. The moving average learns to adapt to noise, while the Conv-LSTM learns to filter out the most important parts for prediction using convolutions, and residual connections learn time-series behavioral patterns. In real-world applications, being closer to these two features is more essential than having the lowest error estimate rates. Using two separate methodologies in a hybrid manner, on the other hand, is not unique in software aging prediction. Hybrid models have been explored in previous studies but without any explanation as to why each of the selected forecasting methods is better suited to software aging forecasting. A significant contribution of this work is the fact that it allowed for the selection of the two best model options for the hLSTM hybrid method after testing both probabilistic methods and LSTM architectures. Using this framework, it is also suggested that the combination of several different forecasting techniques can yield a new signature combination of forecasting techniques specific to an individual problem.

8. Conclusions

In this study, we examined the state of the art in time-series forecasting for issues related to software aging. Our study categorized the most commonly used approaches into two major groups: (i) LSTM models; and (ii) probabilistic approaches, and evaluated them all on identical data that had differing distributions and complexity. On the basis of the time-series tables and figure plots, the LSTM-related models fit better to the data patterns, while, on the other hand, the probabilistic models were able to better capture the spike points within the data gathered from the software aging problems, resulting in higher variance of the noise data. Consequently, by combining the features of both tactics, a hybrid method is simple and outperforms the original methods by combining the moving average and the Conv-LSTM network. In our study, we found that hybrid models were the most effective for predicting software aging. In the future, we plan to introduce hybrid techniques that use computational intelligence tactics such as genetic algorithms and neural networks.

Author Contributions: Conceptualization, methodology, and supervision, J.A.; project administration, formal analysis and investigation, T.A.N.; resources, funding acquisition and investigation, E.C. and D.M.; validation, A.S., L.P. and T.C.; writing—original draft, F.B., A.S., L.P. and T.C.; writing—review and editing, J.A., T.A.N., E.C. and D.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1A2C2094943 and 2020R1A6A1A03046811).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Grottke, M.; Matias, R., Jr.; Trivedi, K. The fundamentals of software aging. In Proceedings of the IEEE International Conference on Software Reliability Engineering Workshops, Seattle, WA, USA, 11–14 November 2008; pp. 1–6. [CrossRef]
2. Cotroneo, D.; Natella, R.; Pietrantuono, R.; Russo, S. A Survey of Software Aging and Rejuvenation Studies. *J. Emerg. Technol. Comput. Syst.* **2014**, *10*, 1–34. [CrossRef]
3. Alonso, J.; Belanche, L.; Avresky, D.R. Predicting software anomalies using machine learning techniques. In Proceedings of the IEEE 10th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 25–27 August 2011; pp. 163–170. [CrossRef]
4. Araujo, J.; Matos, R.; Alves, V.; Maciel, P.; Souza, F.V.d.; Matias, R., Jr.; Trivedi, K.S. Software Aging in the Eucalyptus Cloud Computing Infrastructure: Characterization and Rejuvenation. *J. Emerg. Technol. Comput. Syst.* **2014**, *10*, 1–22. [CrossRef]

5. Sudhakar, C.; Shah, I.; Ramesh, T. Software rejuvenation in cloud systems using neural networks. In Proceedings of the International Conference on Parallel, Distributed and Grid Computing, Solan, India, 11–13 December 2014; pp. 230–233. [CrossRef]
6. Liu, J.; Tan, X.; Wang, Y. CSSAP: Software aging prediction for cloud services based on ARIMA-LSTM hybrid model. In Proceedings of the IEEE International Conference on Web Services (ICWS), Milan, Italy, 8–13 July 2019; pp. 283–290. [CrossRef]
7. Connor, J.; Martin, R.; Atlas, L. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Netw.* **1994**, *5*, 240–254. [CrossRef] [PubMed]
8. Kumar, M.; Anand, M. An Application Of Time Series Arima Forecasting Model For Predicting Sugarcane Production In India. *Stud. Bus. Econ.* **2014**, *9*, 81–94.
9. Permanasari, A.; Hidayah, I.; Bustoni, I.A. SARIMA (Seasonal ARIMA) implementation on time series to forecast the number of Malaria incidence. In Proceedings of the International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia, 7–8 October 2013; pp. 203–207. [CrossRef]
10. Hannan, E.J. *Multiple Time Series*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
11. Zheng, W.; Shen, T.; Chen, X.; Deng, P. Interpretability application of the Just-in-Time software defect prediction model. *J. Syst. Softw.* **2022**, *188*, 111245. [CrossRef]
12. Malhotra, R.; Kamal, S. An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data. *Neurocomputing* **2019**, *343*, 120–140. [CrossRef]
13. Araujo, J.; Matos, R.; Maciel, P.; Vieira, F.; Matias, R.; Trivedi, K.S. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. In Proceedings of the IEEE Third International Workshop on Software Aging and Rejuvenation, Hiroshima, Japan, 29 November–2 December 2011; pp. 38–43. [CrossRef]
14. Avresky, D.R.; Pellegrini, A.; Di Sanzo, P. Machine learning-based management of cloud applications in hybrid clouds: A Hadoop case study. In Proceedings of the IEEE 16th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 30 October–1 November 2017; pp. 1–5. [CrossRef]
15. Qiao, Y.; Zheng, Z.; Fang, Y. An empirical study on software aging indicators prediction in android mobile. In Proceedings of the IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Memphis, TN, USA, 15–18 October 2018; pp. 271–277. [CrossRef]
16. Wang, Z.; Qu, J.; Fang, X.; Li, H.; Zhong, T.; Ren, H. Prediction of early stabilization time of electrolytic capacitor based on ARIMA-Bi_LSTM hybrid model. *Neurocomputing* **2020**, *403*, 63–79. [CrossRef]
17. Xu, B.; Lin, B. Assessing CO_2 emissions in China's iron and steel industry: A dynamic vector autoregression model. *Appl. Energy* **2016**, *161*, 375–386. [CrossRef]
18. Ivanovski, Z.; Milenkovski, A.; Narasanov, Z. Time Series Forecasting Using a Moving Average Model for Extrapolation of Number of Tourist. *UTMS J. Econ.* **2018**, *9*, 121–132.
19. Pappas, S.; Ekonomou, L.; Karamousantas, D.C.; Chatzarakis, G.; Katsikas, S.; Liatsis, P. Electricity Demand Loads Modeling using AutoRegressive Moving Average (ARMA) models. *Energy* **2008**, *33*, 1353–1360. [CrossRef]
20. Yayan, U.; Arslan, A.; Yucel, H. A Novel Method for SoH Prediction of Batteries Based on Stacked LSTM with Quick Charge Data. *Appl. Artif. Intell.* **2021**, *35*, 1–19. [CrossRef]
21. Rahman, M.; Watanobe, Y.; Nakamura, K. A Bidirectional LSTM Language Model for Code Evaluation and Repair. *Symmetry* **2021**, *13*, 247. [CrossRef]
22. He, Z.; Zhou, J.; Dai, H.N.; Wang, H. Gold Price Forecast Based on LSTM-CNN Model. In Proceedings of the IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Fukuoka, Japan, 5–8 August 2019; pp. 1046–1053. [CrossRef]
23. Huang, Y.; Kintala, C.; Kolettis, N.; Fulton, N. Software rejuvenation: Analysis, module and applications. In Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing, Pasadena, CA, USA, 27–30 June 1995; pp. 381–390. [CrossRef]
24. Laird, L. *Software Measurement and Estimation: A Practical Approach*; John Wiley & Sons: Hoboken, NJ, USA, 2006. [CrossRef]
25. Matias, R.; Filho, P.J.F. An experimental study on software aging and rejuvenation in web servers. In Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, IL, USA, 17–21 September 2006, Volume 1, pp. 189–196. [CrossRef]
26. Chen, C.W.S.; Chiu, L.M. Ordinal Time Series Forecasting of the Air Quality Index. *Entropy* **2021**, *23*, 1167. [CrossRef] [PubMed]
27. Brockwell, P.J.; Davis, R.A. Stationary time series. In *Time Series: Theory and Methods*; Springer: New York, NY, USA, 1991; pp. 1–41.
28. Ma, X.; Tao, Z.; Wang, Y.; Yu, H.; Wang, Y. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transp. Res. Part C Emerg. Technol.* **2015**, *54*, 187–197. [CrossRef]
29. Shumway, R.H.; Stoffer, D.S. *Time Series Analysis and Its Applications (Springer Texts in Statistics)*; Springer: Berlin/Heidelberg, Germany, 2005.
30. Hyndman, R.; Athanasopoulos, G. *Forecasting: Principles and Practice*, 2nd ed.; OTexts: Melbourne, VIC, Australia, 2018.

31. Oliveira, F.; Araujo, J.; Matos, R.; Lins, L.; Rodrigues, A.; Maciel, P. Experimental Evaluation of Software Aging Effects in a Container-Based Virtualization Platform. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 11–14 October 2020; pp. 414–419. [CrossRef]
32. Schwarz, G. Estimating the Dimension of a Model. *Ann. Stat.* **1978**, *6*, 461–464. [CrossRef]