IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 January 2023, accepted 9 March 2023, date of publication 10 March 2023, date of current version 16 March 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3256302*

APPLIED RESEARCH

# Proactive Auto-Scaling Approach of Production Applications Using an Ensemble Model

**MOHAMED SAMIR, KHALED T. WASSIF, AND SOHA H. MAKADY**
Faculty of Computers and Artificial Intelligence, Cairo University, Giza 12613, Egypt
Corresponding author: Soha H. Makady (s.makady@fci-cu.edu.eg)

**ABSTRACT** The resource usage behaviors of application workloads are currently the primary concern of cloud providers offering hosting services. These services should be able to adapt to workload changes by automatically provisioning and de-provisioning resources so that, at all times, the existing resources in a system match the current service demand. Such behavior can be achieved manually by hiring a DevOps team to manage the application's resources. Another option would be automating the resource provisioning processing using automated rules. Once such rules are met, the hosting environment will scale the resources accordingly. However, managing a DevOps team or creating flaky rules can lead to over-scaling application resources. This work proposes a new approach: a proactive auto-scaling framework built on an ensemble model. Such a model utilizes several machine learning techniques to scale application resources to match resource demand before the need arises. We evaluated our solution against three real production applications hosted on Cegedim Cloud Hosting Environment, an industrial environment serving several cloud applications from various domains, and against other machine learning models used in similar proactive auto-scaling experiments mentioned in past work. The experimentation results show that predicting application resources like CPU or RAM is feasible. Moreover, even in production environments, our ensemble model performs optimally in the CPU case and is near the optimal model when predicting RAM resources.

**INDEX TERMS** Auto-scaling, resource allocation, dynamic resource provisioning, resource management on clouds.

## I. INTRODUCTION

Cloud providers and virtualized data centers offer a standard feature called service elasticity. Service elasticity is the capability to automatically provide and de-provide resources to meet the present service demand as nearly as feasible at each moment to adjust the system to workload changes [1]. In addition, service elasticity reduces power consumption due to its avoidance of over-provisioning resources [2]. Service elasticity is a concept adopted by most service providers to satisfy peak demand periods and guarantee Quality of Service (QoS) to the users during the service lifetime. Therefore, service elasticity reduces the number of resources needed to implement the service.

Furthermore, it is considered a critical energy-saving mechanism for data centers and cloud providers [1]. Service elasticity mechanisms are classified into two categories: reactive and proactive. Reactive mechanisms work by monitoring the system resources and performance requirements and triggering a particular scaling act accordingly. The rule-based and threshold auto-scaling mechanisms are examples of reactive mechanisms used by cloud-based service providers. The rule-based systems work by providing a new node whenever the state of the system satisfies some rule [3]. While threshold auto-scaling systems work by withdrawing or adding a certain number of resources when a specific metric (e.g., the server load) exceeds or falls below a certain threshold. The main issue with reactive mechanisms is that the reaction time (the time from when a trigger condition is detected until the resources are available for use) may not be quick enough to prevent system overloading [4]. In addition, these mechanisms may lead to system instability because of the constant fluctuation of resource allocation [1].

The proactive (or predictive) technique uses statistical or mathematical models of observed workloads and system metrics to forecast the number of resources required

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

during the coming period [1]. Although most cloud platforms and providers utilize reactive models [5], there has been much research on predictive models based on time series analysis, queuing theory, reinforcement learning, or control theory [6]. Time series analysis is frequently utilized for systems with temporal patterns to build auto-scaling techniques [1]. Most of these suggestions rely on linear statistical time-series forecasting techniques, primarily Box and Jenkins auto-regressive models, to estimate service metrics based on prior observations [7], [8], [9]. However, those linear models may not adequately predict nonlinear input data, as these service measures can show nonlinear patterns. Several other studies used nonlinear regression models based on neural networks. The fundamental drawback of these approaches is the challenging task of creating an effective neural network topology, in addition to the slow training of the algorithms [1].

Due to the limitations of the previous methods and inspired by the current trends in AI, and machine learning-based orchestration of resources in the cloud environment [10], [11], this work proposes a proactive scaling framework based on an ensemble model. We implemented it on top of other machine learning models referenced in the related work. Our ensemble model works by operating different models in parallel and taking their most optimal predictions. Because the previous work only mentioned one model as its optimal solution for proactive scaling and did not try it with multiple applications or compare it to other models, that leaves two questions: first, is the model referenced in each previous work applicable to be used on all applications? Secondly, is there any better model in terms of accuracy?

Therefore, we compared our proposed ensemble model against models referenced in past work and other new models like Neural Hierarchical Interpolation for Time Series (N-Hits) [12], Temporal Fusion Transformer (TFT) [13], Facebook Prophet [14], LightGBM [15], and CatBoost [15]. Moreover, such new models were not used in the proactive autoscaling of resources in a cloud environment. Furthermore, This comparison was conducted on three industrial applications from different domains.

The results illustrate that our model exceeds all those models in most cases while ranking the second or third optimal model in a few other cases in the mean absolute error and root mean square error measurements. Furthermore, some models perform well on some data and applications in terms of accuracy, and in other applications, they have high prediction errors compared to our ensemble models, which have an acceptable error margin in all situations.

This paper is structured as follows: Section II shows the literature review, including the related work on dynamic resource provision and research mechanisms using machine learning forecasting techniques. Section III presents the proposed proactive scaling framework's technique. Section IV explains our evaluation setup and its results and main findings. Section V discusses the limitations. Section VI concludes the work and discusses future work.

## II. RELATED WORK

In this section, we discuss existing research trends in application scaling. The current automated scaling approaches for application resource provisioning can be mainly categorized into Auto-scaling and Proactive Scaling.

### A. REACTIVE SCALING TRENDS

Minxian Xu et al. introduced a multi-faceted scaling approach using reinforcement learning called CoScal to learn the scaling techniques efficiently [16]. This approach applies a hybrid scaling technique that combines vertical, horizontal, and brownout, making adaptive decisions via reinforcement Learning (RL). Their results demonstrate that CoScal reduces response time between 19% to 29% and decreases the connection time of services by 16% compared to state-of-the-art scaling techniques. However, they used simulated data from a demo microservice application called sock shop, and their data volume was small (only 500-minute data for evaluations). So there is no way to conclude that their approach will work against larger datasets against other approaches or even if it will ever work in a production application cloud setup.

Aslanpour et al. investigated how to improve current autoscaling methodologies by improving the tail of latencies for users' requests [17]. Their investigations discovered sources of tail latencies like 1) large requests, i.e., those data-intensive; 2) long-term scaling intervals; 3) instant analysis of scaling parameters; 4) conservative, i.e., tight, threshold tuning; 5) load-unaware surplus VM selection policies used for executing a scale-down decision; 6) cooldown feature and 7) VM start-up delay. However, the results were only concerned with scaling virtual machines and did not experiment with other application resources like CPU, RAM, or network bandwidth, so we do not know their impact on the tail of latencies for users' requests. Nevertheless, according to their results, the tail may show a different behavior by improving the average latency by auto-scaling mechanisms, which may require a tail-aware solution that they did not discuss how to develop.

### B. PROACTIVE SCALING TRENDS

Several proactive scaling strategies have been proposed. For example, M. Xu et al. proposed an efficient supervised learning-based Deep Neural Network (esDNN) approach for cloud workload prediction [18]. First, they utilized a sliding window to convert the multivariate data into a supervised learning time series that allows deep learning for processing. Then, a revised Gated Recurrent Unit (GRU) was applied to achieve accurate prediction. According to their results, esDNN can significantly reduce mean square errors, e.g., by 15%, rather than the GRU approach. However, their data volume is small (Alibaba's dataset was eight days' worth of data, and Google's dataset was 29 days' worth of data ). Secondly, they did not test their approach against other features and focused only on CPU despite their data having other

features, which makes their approach questionable regarding accurately predicting other application resources.

Lee et al. used several deep-learning-based predictive models to predict future demands and take action based on those predictions [3]. The utilized deep learning models were Long Short Term Memory (LSTM), a type of recurrent neural network (RNN), and Temporal Convolution Network (TCN). Furthermore, XGboost is implementing the Gradient Boosting Decision Tree algorithm oriented by the decision tree and boosting the ensemble method. However, such work had some limitations. For example, there was a disagreement between the business loss metric and the model loss metric, indicating contradictions in their model-building process. Also, the utilized data was limited, with no data spikes. Therefore, the proposed model's ability to predict a burst increase in the needed resources remains unknown.

Buchaca et al. used the AI4DL framework [19], [20] to characterize workload and discover resource consumption phases. First, the existing technology was advanced to an incremental phase discovery method that applies to more general ML workload types for training and inference. Next, a time-window MultiLayer Perceptron (MLP) was used to predict phases in containers with different types of workloads. Afterward, a predictive vertical auto-scaling policy resized the container dynamically according to phase predictions. However, MLP was only used for prediction and was not compared to other models. Additionally, the duration of keeping the containers running was ignored, so we do not know the volume of test data they used.

R. Moreno-Vozmediano et al. presented a novel predictive auto-scaling mechanism based on machine learning techniques for time series forecasting (moving average, linear regression, and SVM) and queuing theory [1]. The approach aimed to predict a distributed server's processing load accurately and estimate the appropriate number of resources that must be provisioned to optimize the service response time. Furthermore, fulfill the service-level agreement (SLA) contracted by the user using different machine learning models and comparing the result to reach the best model that gives the most optimal predictions. However, their data were simulated for a very short period (4 weeks). As a result, the data showed a linear trend which explains the good accuracy achieved by SVM and linear regression. However, such a linear trend is not a realistic state for production applications. For instance, production applications could be heavily loaded with user requests or crash. In such cases, SVM and Linear Regression could perform poorly.

EG Radhika et al. showed how Auto-Regressive Integrated Moving Average (ARIMA) and Recurrent Neural Network Long Short Term Memory (RNN-LSTM) techniques are used for predicting the future workload [2]. Furthermore, the RNN-LSTM deep learning approach had the lowest error rate when evaluating the performance metrics of the two techniques. Therefore, they concluded that it might be utilized to forecast future workloads for web applications in a private cloud. Afterward, Benifa et al. presented RLPAS (Reinforcement Learning based Proactive Auto-Scaler) algorithm [4]. Such an algorithm is based on the existing Reinforcement Learning (R.L.)-SARSA algorithm that learns the environment and allocates the needed resources in parallel. The performance of the RLPAS algorithm is validated using simulated workloads, and it outperforms existing auto-scaling approaches in terms of CPU utilization, response time, and throughput. Aslanpour et al. presented a proactive auto-scaling algorithm (PASA) with a heuristic predictor [5]. The predictor analyzes history with the help of the following techniques: (1) double exponential smoothing - DES, (2) weighted moving average - WMA, and (3) Fibonacci numbers. However, all such approaches [2], [4], [5] used simulated workloads or utilized open-source applications like RUBiS [21], RUBBoS [22], and Olio [23], which do not reflect actual user interaction compared to utilizing industrial applications.

## III. PROPOSED FRAMEWORK

To address the limitations of existing proactive auto-scaling algorithms within any cloud environment, we present a novel approach that simultaneously utilizes statistical models, machine learning models, and neural networks to provide a better prediction for the needed resources. The proposed approach is designed to be easily integrated into any cloud environment. Upon integration, the approach would access the resources' usage logs to forecast the needed resources in the future through an ensemble model. Such an ensemble model supersedes previous approaches by combining several machine learning models to provide an optimal prediction.

### A. FRAMEWORK OVERVIEW

The proposed approach has been constructed in a framework called ''Proactive Autoscaler Framework (PAF)''. Fig. 1 shows the usage flow of PAF by a deployment team. PAF is utilized within a cloud environment. Such cloud environment has deployed applications and an existing history for each deployed application. Such history should include application resource types like CPU and RAM, the consumption of each resource by the application each constant period (for example, every 13 minutes), and a long resource consumption history since Such long history is needed as training data.

Consider that it is needed to forecast resource X's future usage by a deployed application. The usage flow of PAF will proceed as follows:

1) PAF Calls the cloud environment hosting the application to fetch the application usage logs for resource X.
2) The cloud environment will respond with the inquired application's usage of Resource X.
3) PAF will run its prediction model on the retrieved usage data for Resource X and tries to forecast its future usage.
4) After PAF finishes the prediction, all results, including the forecasted values for Resource X usage and the recommended minimum and maximum values for that

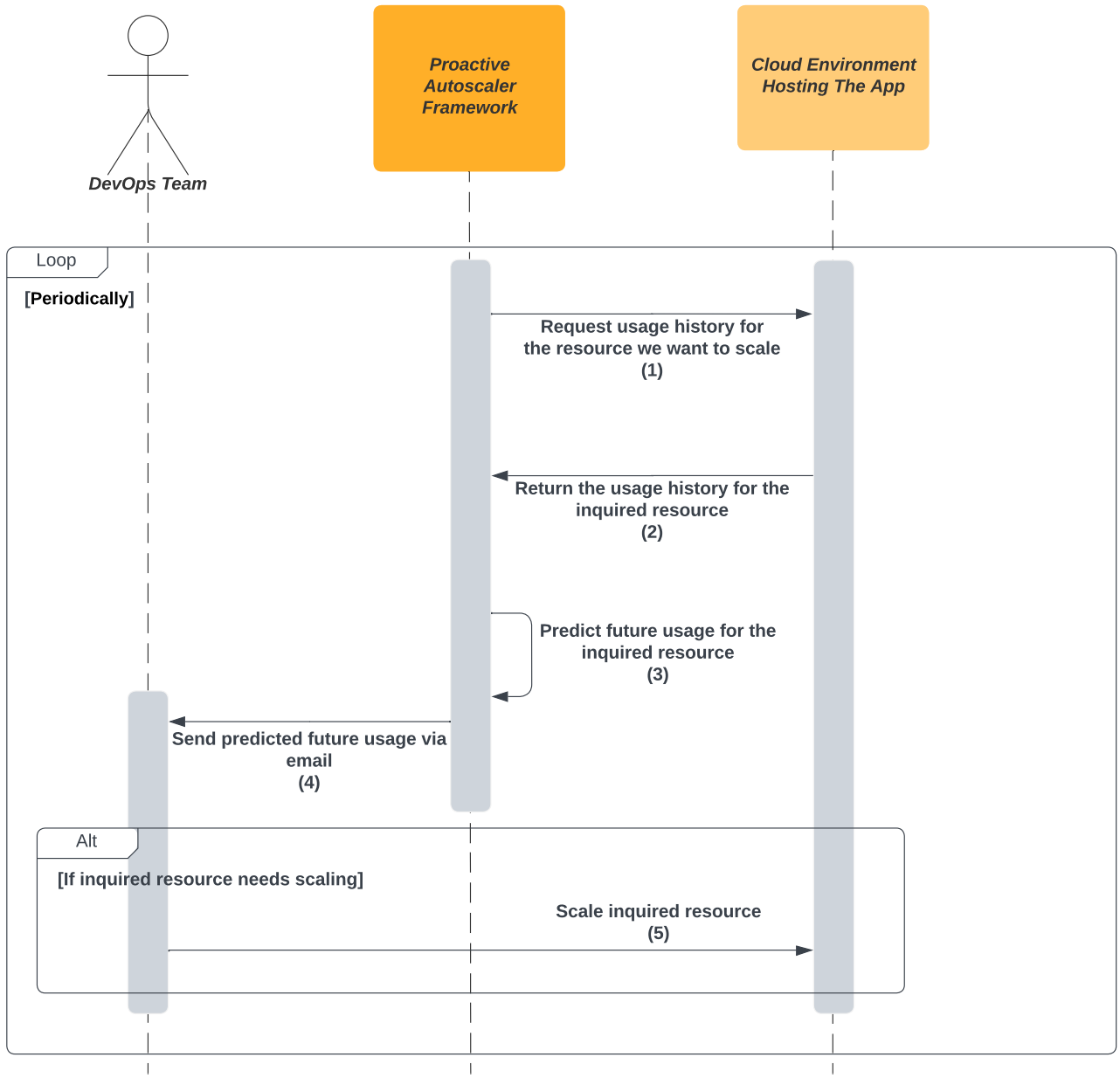



**FIGURE 1.** The usage flow of the proposed Proactive Autoscaler framework (PAF).

forecasted resource, are all documented and attached to an e-mail sent to the DevOps team.

### B. PREDICTION FRAMEWORK PARAMETERS

For our framework to run correctly, some configuration parameters must be provided by the DevOps team in the framework configuration file like:

1) The application ID. on the cloud hosting environment.
2) The resource we want to forecast.
3) The history duration that we will use as training data.
4) The duration of resource usage we want to predict.

### C. ENSEMBLE MODEL

An ensemble model is a machine learning technique that combines several base models to produce one optimal predictive model. To develop a generic non-domain specific model that fits almost all software applications (regardless of their business domain). We propose an ensemble model, which can forecast an application's needed resources (i.e., CPU, RAM) by utilizing seven models simultaneously. The proposed ensemble model is built on:

1) **Moving Average (MA)**: A statistical method for forecasting long-term trends. The technique represents

taking an average of a set of numbers in a given range while moving the range.

2) **Weekly Average (WA)**: A statistical method developed by us. The idea of this model came from observing that all applications show a similar workload pattern across weekdays. Such seasonality is used for forecasting. For example, to forecast next Sunday's data, we compute the average of all Sundays' data in the training data.
3) **Linear Regression (LR)**: A machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target-prediction value based on independent variables. It is mainly used for finding out the relationship between variables and forecasting.
4) **Polynomial Support Vector Machine (SVM Poly)**: a supervised machine learning algorithm that can be used for classification and regression problems. The SVM regression can approximate the value of the time series at time t, using the following function: $\hat{y_t} = b + \sum_{m=1}^{M} w_m \times K(x_i, x_m)$ where b is a constant (bias term); wm are the weight factors (W = $\{w1, w2, \ldots, w_M\}$ is the weight vector ); xt is the time series data window at time t, and K is the Polynomial kernel function. $K(x, x\prime) = (x.x\prime + 1)^P$, where $x.x\prime$ is the dot product of feature vectors x and $x\prime$, and p ∈ N is the exponent of the kernel chosen by the user.
5) **Radial Basis Support Vector Machine (SVM RBF)**: A supervised machine learning algorithm that can be used for classification and regression problems. The SVM regression can approximate the value of the time series at time t, using the following function: $\hat{y_t} = b + \sum_{m=1}^{M} w_m \times K(x_i, x_m)$ where b is a constant (bias term); wm are the weight factors (W = $\{w1, w2, \ldots, w_M\}$is the weight vector ); xt is the time series data window at time t, and K is the Radial kernel function. $K(x, x\prime) = e^{-\gamma||x-x\prime||^2}$, where $||x - x\prime||^2$ represent the Euclidean distance between feature vectors x and $x\prime$, and $\gamma \in R$ is a user-defined parameter.
6) **Facebook Prophet**: A machine learning algorithm for forecasting time series data based on an additive model where nonlinear trends fit with yearly, weekly, and daily seasonality, plus holiday effects, It works best with time series with strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend and typically handles outliers well [14].
7) **Long short-term memory (LSTM)**: A recurrent Neural Network used for time series forecasting [9].Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network (RNN) can process not only single data points (such as images), but also entire sequences of data (such as speech or video) [14].
8) **Neural Hierarchical Interpolation for Time Series (N-Hits)**: is an extension of the Neural Basis Expansion Analysis for Interpretable Time Series (N-BEATS) model that improves the accuracy of the predictions and reduces the computational cost. This is achieved by sampling the time series at different rates. That way, the model can learn short-term and long-term effects in the series. Then, generating the predictions will combine the forecasts made at different time scales, considering both long-term and short-term effects. This is called hierarchical interpolation [12].
9) **Temporal Fusion Transformer (TFT)**: is a Transformer-based model that leverages self-attention to capture the complex temporal dynamics of multiple time sequences [13].
10) **LightGBM**: is a gradient-boosting framework that uses tree-based learning algorithms. It uses two novel techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB), which fulfills the limitations of the histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks [15].
11) **XGBoost**: is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) [15].
12) **CatBoost**: CatBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi [15].

All the models mentioned above, except for N-Hits, TFT, Facebook Prophet, LightGBM, and CatBoost, were used in previous work. As previous work tested the models on open-source data with linear trends, we tried to address such limitations by (a) utilizing applications with nonlinear trends and (b) adding facebook prophet and LSTM since they can predict data with nonlinear trends.

Algorithm 1 shows the steps of the ensemble model. It first trains all the models used in its implementation ( Moving average, Weekly Average, Linear Regression, Support Vector Machine, Facebook Prophet, LSTM, N-Hits, TFT, LightGBM, XGBoost, and CatBoost ) with all the training data in the (trainArray) array. Afterward, such models are stored in the (models_array) variable.

Afterward, the ensemble model picks any random model of those to start predicting the first (forwardWindow) records in the first iteration (lines 1-3). Then in the second iteration, the ensemble model will look back at the last (backwardWindow) records and try to predict it with other models not selected in the last iteration (lines 5-21).

Suppose any of those models proves to have better accuracy than the already chosen model in the last iteration

**Algorithm 1** Ensemble Model Algorithm

**Input:** trainArray, testArray, backWindow, forwardWindow
**Output:** bestPredicitions
**Steps**

1: *currentPredictionModel* ← *models_array.getRandomModel*()
2: *currentTestRecords* ← *trainArray.get*(*trainArray.size* − *forwardWindow*, *trainArray.size*)
3: *bestPredicitions* ← *currentPredictionModel.predict*(*currentTestRecords*)
4: *idx* ← *forwardWindow*
5: **while** *idx* < *Test.length* **do**
6: *previousPredictedRecords* ← *bestPredicitions.get*(*idx* − *backWindow*, *idx*)
7: *actualResultsRecords* ← *testArray.get*(*idx* − *backWindow*, *idx*)
8: *bestMAE*, *bestRMSE* ← *getAccuarcyMeasures*(*previousPredictedRecords*, *actualResultsRecords*)
9: **for** each model in models_array **do**
10: *prediction* ← *model.predict*(*actualResultsRecords*)
11: *MAE*, *RMSE* ← *getAccuarcyMeasures*(*prediction*, *actualResultsRecords*)
12: **if** (*MAE* < *bestMAE*) or (*RMSE* < *bestRMSE*) **then**
13: *currentPredictionModel* ← *model*
14: *bestMAE* ← *MAE*
15: *bestRMSE* ← *RMSE*
16: **end if**
17: **end for**
18: *currentTestRecords* = *testArray.get*(*idx*, *idx* + *forwardWindow*)
19: *bestPredicitions* ← *bestPredicitions* + *currentPredictionModel.predict*(*currentTestRecords*)
20: *idx* ← *idx* + *forwardWindow*
21: **end while**

(line 12). In that case, the ensemble model will switch to that model and use it to predict the following (forwardWindow) records in this iteration (lines 13-15).

All iteration results are accumulated in an array called (bestPredicitions).

The (forwardWindow) and (backwardWindow) are configuration parameters that are set by the user of the proactive prediction framework

## IV. EVALUATION

### A. DATASET

We tested our Framework against three real production applications hosted on Cegedim Cloud Hosting Environment. Furthermore, we chose three applications from three different business domains since each application will differ in size and configuration. Such configuration was deliberately chosen to prove that no single machine learning model would properly provide correct predictions for all applications, unlike the Ensemble model, which combines several models to produce one optimal predictive model.

The input data chosen to train the forecasting model and evaluate our proposal were obtained from production applications by scrapping the virtual machines(VM) hosting each application for resource usage.

1) Application [A] is one of the most prominent applications used by thousands of users to communicate and schedule visits with their doctors. This application follows microservice architecture, and the entire application is scaled across twelve VM on the cloud environment
2) Application [B] is a pharmaceutical application used by thousands of users to order prescriptions and handle pharmacy management and sales. This application follows microservice architecture, and the entire application is scaled across eight VM on the cloud environment
3) Application [C] is also used for human resources administrations by many users. This application follows monolithic architecture, and the entire application is scaled across two VM on a cegedim cloud environment.

**TABLE 1. Used applications background.**

| Application | [A] | [B] | [C] |
|---|---|---|---|
| Users Count | 69,074 | 8,337 | 5,200 |
| App Size (KLOC) | 819 k | 98 k | 61 k |
| Collected Data Size(Megabytes) | 5.47M | 3.55M | 1.60M |
| Data Content | CPU & RAM usage | | |
| Data Period | Every 13 min for CPU and every 40 mins for RAM | | |
| Data Duration | August 2021 to April 2022 | | |

The resource allocation of each application was collected across nine months for our research purposes. A link to the data set can be found here.

Table 1 summarizes all applications used in our study, including details about user count, application size in kilo lines of code (KLOC), collected data size and content, the

period between data points, and the duration of the extracted data.

### B. FORECAST ACCURACY MEASURES

Different error measures can be used to evaluate the accuracy of the forecasting models. Some of the most common error measures that we used are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \quad [9]$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|} \quad [7]$$

We apply these accuracy measures to the forecast data within the forecasting horizon; assuming we know the real-time series values for this period, we obtain the real prediction error made by the forecasting model [1].

### C. PARAMETER SELECTION

While tuning the ensemble model's parameters, we faced a problem as the applications of interest differ in the development technologies they are built with, the business domain they serve, and, most importantly, their deployment configuration on the cloud. Given that a model to predict the optimal result for a dataset, it will need to be parameter tunned, but since all applications differ in its model's parameters. Hence, parameter tuning was made to reach common parameters that suit all applications under study.

- Forecasting model #1 (Moving average): After multiple regressions on all three applications, the optimal result for this model was yielded by the moving average (MA) of order two (i.e., an MA (p) model, with p = 2).
- Forecasting model #2 (Weekly average): No tunning parameter is needed since all it does is get the average of all previous points.
- Forecasting model #3 (Linear regression): After multiple regressions on all three applications, the optimal result for this model was yielded by an autoregressive (AR) model with a lag period of 24 h (i.e., an AR (p) model with p = 24)
- Forecasting model #4 (SVM with a polynomial kernel): After multiple regressions on all three applications, the optimal result for this model was yielded by SVM regression with a polynomial kernel of order p = 2.
- Forecasting models #5 (SVM with an RBF kernel): After multiple regressions on all three applications, the optimal result for this model was yielded by SVM regression with an RBF kernel With a $\gamma$ parameter of 1.0
- Forecasting models #6 (Facebook Prophet): After multiple regressions on all three applications, the optimal result was yielded by its default parameters (because of the many optimization functions that he runs in his implementation)
- Forecasting models #7 (LSTM): After multiple regressions on all three applications, 14 steps,200 epochs, and 0.1 dropout yielded the optimal result for this model, and we empirically tested with a higher step count. Still, they offered no better results than the order of 14 steps and took more time to execute.
- Forecasting models #8 (N-Hits): After multiple regressions on all three applications, 14 input chunk size, 1 output chunk size,56 batch size,200 epochs, and 0.1 dropout yielded the optimal result for this model.
- Forecasting models #9 (TFT): After multiple regressions on all three applications, 14 input chunk size, 1 output chunk size,4 lstm layers, 56 batch size,300 epochs, and 0.1 dropout yielded the optimal result for this model.
- Forecasting models #10 (LightGBM): After multiple regressions on all three applications, 1 lags, 1 output chunk size, and poisson likelihood yielded the optimal result for this model.
- Forecasting models #11 (XGBoost): After multiple regressions on all three applications, 1 lags, 1 output chunk size, and poisson likelihood yielded the optimal result for this model.
- Forecasting models #12 (CatBoost): After multiple regressions on all three applications, 1 lags, 1 output chunk size, and poisson likelihood yielded the optimal result for this model.
- Forecasting models #13 (Ensemble Model): After multiple regressions on all three applications, 14 steps backward and 1 step forward, yielded the optimal result for this model; we empirically tested with a higher step count, but they offered no better results than the order 14 steps, and 1 step forward and took more time to execute.

**TABLE 2. Each forecasting model with its optimal parameter(s).**

| Number | Forecasting Model | Parameters |
|---|---|---|
| 1# | Moving Average | MA(2) |
| 2# | Weekly Average | No parameters needed |
| 3# | Linear Regression | AR(24) |
| 4# | SVM Polynomial | $C = 1.9, e = 0.027, p = 1$ |
| 5# | SVM RBF Kernel | $C = 1.9, e = 0.027, \gamma = 1.0$ |
| 6# | Facebook Prophet | default parameters |
| 7# | LSTM | 14 steps,200 epochs, and 0.1 dropout |
| 8# | N-Hits | 14 input chunk size, 1 output chunk size,56 batch size,200 epochs, and 0.1 dropout |
| 9# | TFT | 14 input chunk size, 1 output chunk size,4 lstm layers, 56 batch size,300 epochs, and 0.1 dropout |
| 10# | LightGBM | 1 lags, 1 output chunk size, and poisson likelihood |
| 11# | XGBoost | 1 lags, 1 output chunk size, and poisson likelihood |
| 12# | CatBoost | 1 lags, 1 output chunk size, and poisson likelihood |
| 13# | Ensemble Model | 14 steps backward, 1 step forward |

Table 2 summarizes all models used in our study, including parameters used for each model during its operation under our ensemble model.

### D. RESULTS

The prediction models presented in this work forecast the CPU and RAM resources, using the parameters specified in

Section III-B. the reported results were produced by running each model with its perspective parameters on each month separately with a ratio of 75% training and 25% for testing.

When reporting the results, we will be reporting the results of Application [A] and Application [B] within the same figures, and Application [C] results will be reported in separate figures due to the variance in the used scale between those applications. Such variance is attributed to the difference in the data size between those applications. For instance, the application [A] data set is of size 5.7 megabytes due to its being a medical system, whereas the dataset of application [C] is of size 1.60 megabytes due to its being an HR system.

In those sections, we will explore the error variance results of the ensemble model and compare its accuracy to the accuracy of the models that the ensemble model is built on, in terms of MAE and RMSE, applying each of the different models individually to the same dataset.

### 1) CPU PREDICTION ERROR VARIANCE RESULTS

Fig. 2 shows the CPU MAE results for Applications [A] and [B], whereas Fig. 3 shows the CPU MAE results for Application [C]. For application [B], our ensemble model performed better than all the other models, while for applications [A] and [C], our model scored second place with a very small difference from the first place.

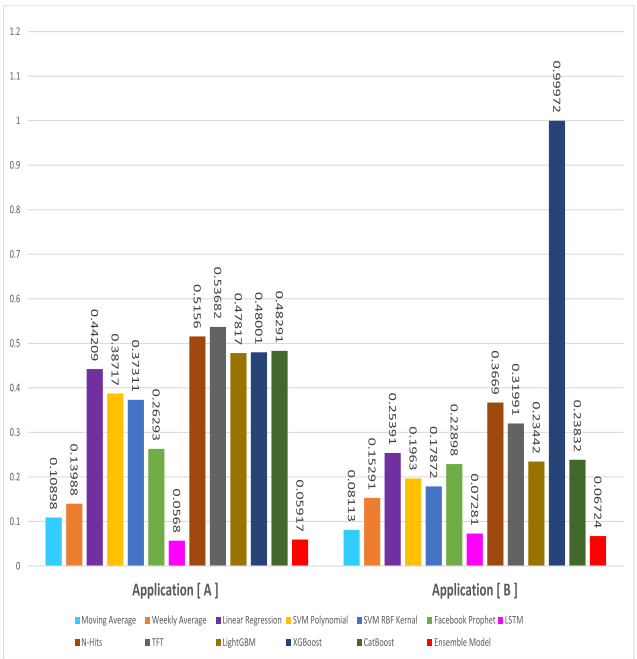



**FIGURE 2.** CPU MAE results for application [A] and application [B].

Fig. 4 shows the CPU RMSE results for Applications [A] and [B], whereas Fig. 5 shows the CPU RMSE results for Application [C]. For application [B], our ensemble model performed better than all the other models, while for applications [A] and [C], our model scored second place with a very small difference from the first place.

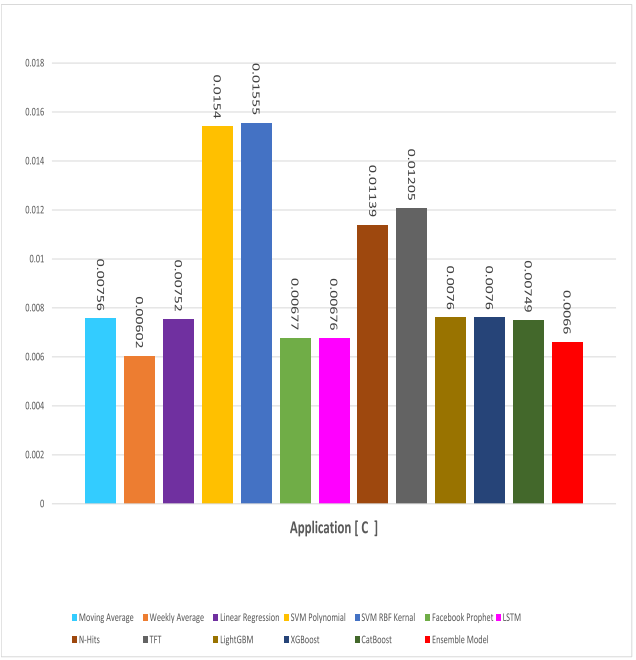



**FIGURE 3.** CPU MAE results for application [C].

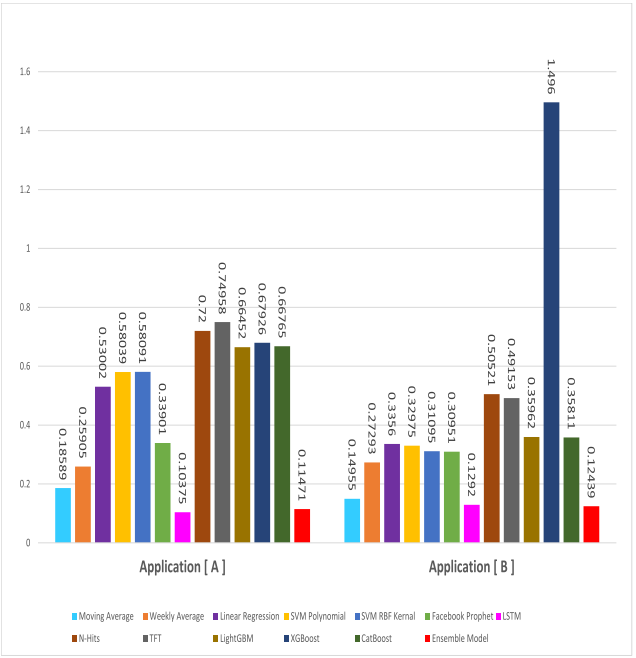



**FIGURE 4.** CPU RMSE results for application [A] and application [B].

### 2) RAM PREDICTION ERROR VARIANCE RESULTS

Fig. 6 shows the RAM MAE results for Applications [A] and [B], whereas Fig. 7 shows the RAM MAE results for Application [C]. Our ensemble model came second for application [A], third for Application [B], and first for application [C].

Fig. 8 shows the RAM RMSE results for Applications [A] and [B], whereas Fig. 9 shows the RAM RMSE results for Application [C]. Our ensemble model came second for

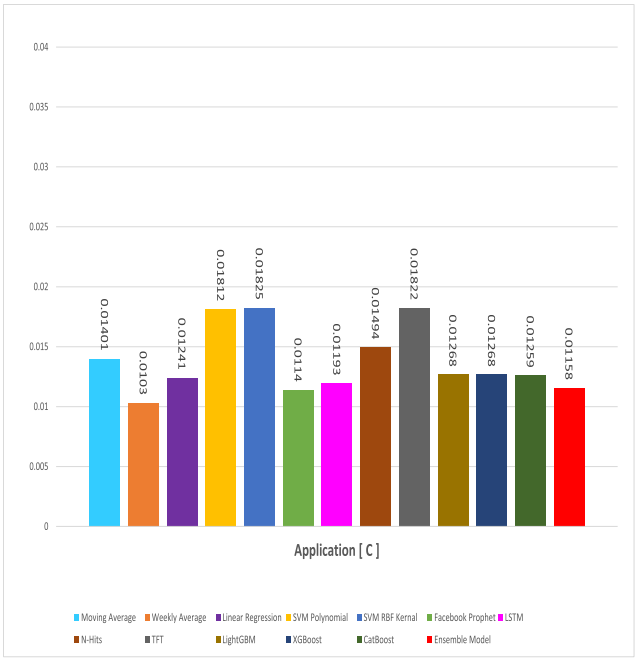



**FIGURE 5.** CPU RMSE results for application [C].

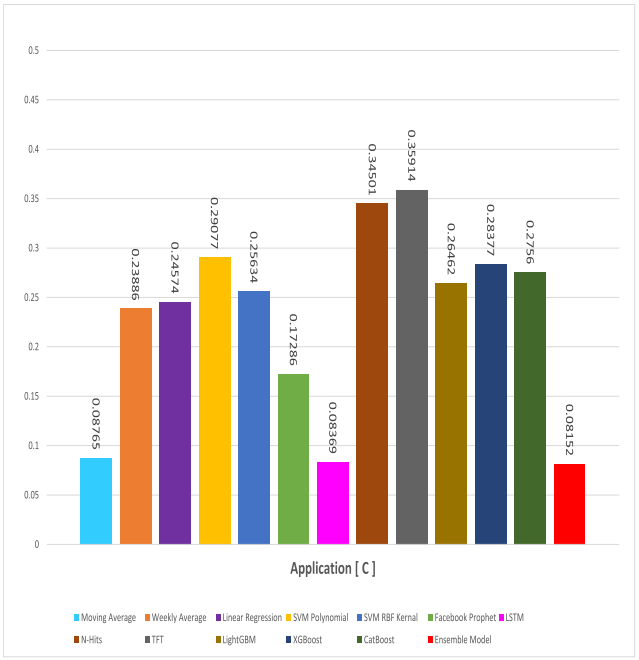



**FIGURE 7.** RAM MAE results for application [C].

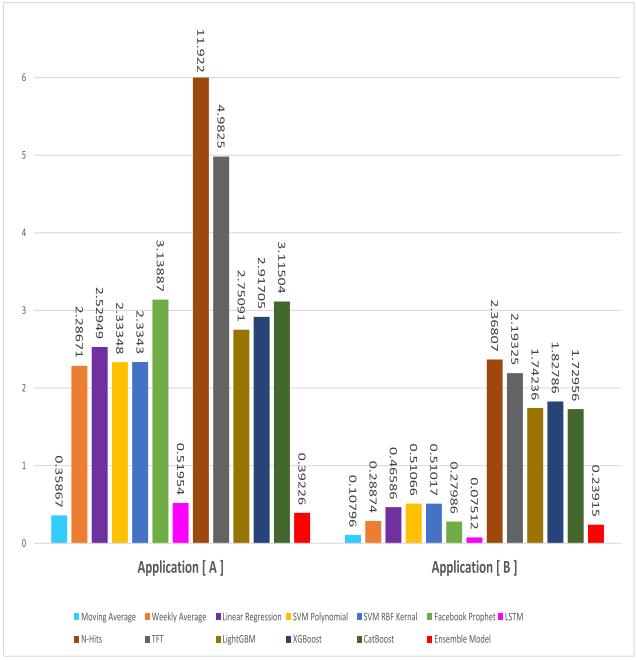



**FIGURE 6.** RAM MAE results for application [A] and application [B].

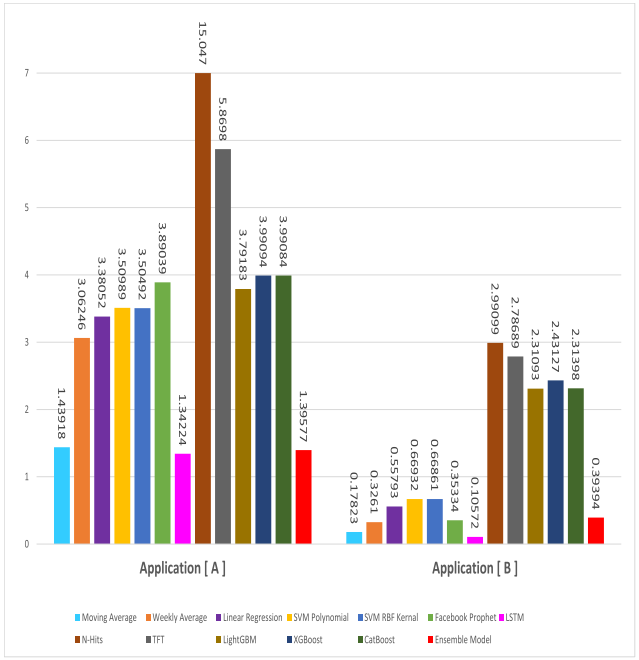



**FIGURE 8.** RAM RMSE results for application [A] and application [B].

application [A], fifth for Application [B], and second for application [C].

### 3) PREDICTION SAMPLE

Table 3 and Table 4 show an example of running the ensemble model on Application [A] CPU data and Application [C] RAM data. As can be seen in Table 3, the difference between the predicted usage for CPU for Application [A] and actual usage is minimal ( less than 0.2 core ) for the first bolded row in the table, which is quite acceptable for a production environment like Cegedim, which scales its cores by 2 cores minimum for each request.

The Same pattern holds for Table 4, as the difference between the predicted usage for RAM For Application [C] and actual usage is very small ( less than 33 Megabytes ) for the first bolded row in the table, which is quite acceptable

**TABLE 3.** CPU prediction results by PAF for application [A] on January 2022 across its 5 VMs from A [1], [2], [3], [4], [5] in CPU Cores.

| timestamp_date_format | A1 | A1_Predicted | A2 | A2_Predicted | A3 | A3_Predicted | A4 | A4_Predicted | A5 | A5_Predicted |
|---|---|---|---|---|---|---|---|---|---|---|
| **2022-01-24 06:08:00** | **0.1** | **0.118309125** | **0.15** | **0.114769496** | **0.1** | **0.070320383** | **0.11** | **0.09015774** | **0.08** | **0.078529775** |
| 2022-01-24 06:21:00 | 0.13 | 0.123457476 | 0.11 | 0.151372701 | 0.1 | 0.096666667 | 0.12 | 0.107234336 | 0.07 | 0.081656806 |
| 2022-01-24 06:34:00 | 0.17 | 0.145687237 | 0.16 | 0.123333333 | 0.13 | 0.089561328 | 0.1 | 0.115775086 | 0.08 | 0.073333333 |
| 2022-01-24 06:47:00 | 0.28 | 0.176977113 | 0.21 | 0.161230654 | 0.21 | 0.11 | 0.21 | 0.11 | 0.09 | 0.081656806 |
| 2022-01-24 07:00:00 | 0.5 | 0.263566107 | 0.36 | 0.210614637 | 0.35 | 0.196522817 | 0.28 | 0.200745165 | 0.08 | 0.08617036 |
| 2022-01-24 07:13:00 | 0.68 | 0.439102024 | 0.62 | 0.359635115 | 0.62 | 0.333909154 | 0.47 | 0.269142151 | 0.09 | 0.083333333 |
| 2022-01-24 07:26:00 | 0.87 | 0.60390985 | 0.72 | 0.620453596 | 0.75 | 0.602304876 | 0.59 | 0.456472933 | 0.08 | 0.08617036 |
| 2022-01-24 07:39:00 | 1.01 | 0.817211926 | 0.94 | 0.721406519 | 0.93 | 0.732887387 | 0.92 | 0.575888932 | 0.09 | 0.081656806 |
| 2022-01-24 07:52:00 | 1.47 | 0.972232997 | 1.09 | 0.944289923 | 1.2 | 0.914852083 | 1.16 | 0.907579303 | 0.08 | 0.086666667 |
| 2022-01-24 08:05:00 | 1.62 | 1.457270384 | 1.45 | 1.09659183 | 1.33 | 1.189401507 | 1.27 | 1.150822401 | 0.09 | 0.081656806 |
| 2022-01-24 08:18:00 | 1.96 | 1.609164476 | 1.64 | 1.46184659 | 1.67 | 1.319526911 | 1.5 | 1.262624621 | 0.09 | 0.086666667 |
| 2022-01-24 08:31:00 | 2.08 | 1.946366906 | 1.83 | 1.652772307 | 1.85 | 1.659651875 | 1.61 | 1.489256978 | 0.1 | 0.08617036 |
| 2022-01-24 08:44:00 | 2.09 | 2.063123941 | 1.88 | 1.842307091 | 1.9 | 1.839134693 | 1.73 | 1.593629122 | 0.1 | 0.093333333 |
| 2022-01-24 08:57:00 | 2.16 | 2.072802782 | 1.88 | 1.891975641 | 1.9 | 1.888872623 | 1.76 | 1.707047701 | 0.09 | 0.091391906 |
| 2022-01-24 09:10:00 | 2.35 | 2.140337467 | 1.79 | 1.891975641 | 2.11 | 1.888872623 | 1.7 | 1.735328555 | 0.1 | 0.08617036 |
| 2022-01-24 09:23:00 | 2.28 | 2.321787596 | 1.83 | 1.85 | 1.89 | 2.097054243 | 1.72 | 1.73 | 0.09 | 0.096666667 |

Table III shows the results of predicting the CPU for application [A]. As application [A] is replicated and has many instances, we show the prediction of each instance individually.

**TABLE 4.** RAM prediction results by PAF for application [C] on December 2021 across its 2 VMs from C [1], [2] in Giga RAM.

| timestamp_date_format | C1 | C1_Predicted | C2 | C2_Predicted |
|---|---|---|---|---|
| **2021-12-24 06:00:00** | **2.295879601** | **2.263819456** | **2.34410191** | **2.321556807** |
| 2021-12-24 06:40:00 | 2.293808218 | 2.105232239 | 2.346034525 | 2.098781347 |
| 2021-12-24 07:20:00 | 2.29450536 | 2.114748001 | 2.338183643 | 2.096838713 |
| 2021-12-24 08:00:00 | 2.327426914 | 2.153534412 | 2.384951357 | 2.171316624 |
| 2021-12-24 08:40:00 | 2.354888441 | 2.230947018 | 2.401509285 | 2.243987799 |
| 2021-12-24 09:20:00 | 2.35907102 | 2.235790253 | 2.411072729 | 2.255951166 |
| 2021-12-24 10:00:00 | 2.404421093 | 2.231138706 | 2.423387766 | 2.248400927 |
| 2021-12-24 10:40:00 | 2.468142267 | 2.32197579 | 2.520480398 | 2.579528016 |
| 2021-12-24 11:20:00 | 2.484941958 | 2.389958144 | 2.516898867 | 1.759927751 |
| 2021-12-24 12:00:00 | 2.486133575 | 2.452501773 | 2.515924219 | 1.771238962 |
| 2021-12-24 12:40:00 | 2.487456081 | 2.479739267 | 2.514561415 | 1.813451769 |
| 2021-12-24 13:20:00 | 2.509431597 | 2.486177205 | 2.515138146 | 1.812251729 |
| 2021-12-24 14:00:00 | 2.594855074 | 2.494340418 | 2.630615951 | 1.831996283 |
| 2021-12-24 14:40:00 | 2.646984104 | 2.530580917 | 2.68918395 | 1.834532659 |
| 2021-12-24 15:20:00 | 2.645896915 | 2.595915318 | 2.674255371 | 2.48573645 |
| 2021-12-24 16:00:00 | 2.64480615 | 2.594860077 | 2.678051712 | 2.595609029 |

Table IV shows the results of predicting the RAM for application [C]. As application [C] is replicated and has many instances, we show you the prediction of each instance individually.

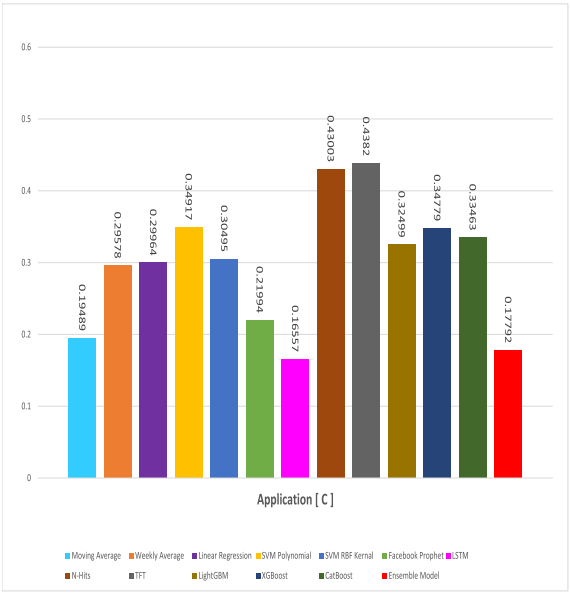



**FIGURE 9.** RAM RMSE results for application [C].

for a production environment like Cegedim, which scales its RAM by 2 Gigabyte minimum for each scaling request.

## V. LIMITATIONS

Our framework was intended to hook into a cloud environment, read the application of interest resources logs, and scale the resources automatically on the cloud environment without human intervention according to the ensemble model's prediction. However, unfortunately, the IT department rejected such an approach as they did not want to automate resource allocation, especially on costly production apps. So they advised that the predictions are sent to them by e-mail each week so they can choose to scale or not.

## VI. CONCLUSION AND FUTURE WORK

From the above results, we can conclude that:

- Forecasting resources for production applications as an idea is quite feasible, given that the error margin between the prediction and the actual result of resource utilization is very low compared to the smallest scaling unit for a production environment like the cegedim cloud hosting service.
- Using state-of-the-art machine learning models and neural networks only is not a good approach to solving the proactive auto-scaling problem, as the N-Hits and TFT may have good accuracy in predicting the resource demand for application [B] and application [C], but had

a lousy accuracy in predicting the resource demand for application [A] compared to older models like SVM And LSTM. That is why combining all models (old and new) under one ensemble model provided a proper resource prediction for all three applications.

- No optimal model can predict all cloud resources, as seen from the above results. For example, application [A] best model for Ram Prediction in terms of MAE is the Moving average, while for application [B], it was LSTM, but in all cases, our Ensemble model ranked second or third best accurate model in prediction and sometimes ranked the best accurate model as the case for application [B] CPU and application [C] RAM. Moreover, that is the beauty of ensemble models that always try to be near optimum.
- Since deep learning models like LSTM, Facebook prophet, N-Hits, and TFT take a long time to run in comparison to statistical models like Moving Average and gradient boosting Models like XGboost that take few seconds, And given that the margin of error between deep learning models and statistical and gradient boosting models is not that large. Owners of the prediction models may be persuaded to switch off deep learning models and make the ensemble model work with only statistical and gradient boosting models to have a quicker prediction and performance from the proactive framework.

Our work has many future research directions inspired by current AI trends [10], [11]:

- Proactive Load Balancer: Our framework has the potential to predict the needed number of containers or VMS needed by the applications. If we research how to integrate it into a load balancer, we can create a proactive load balancer that scales the application before the demand arises.
- Operation Plan Proactive Scaler framework: Our framework can be modified to forecast other application resources like network, physical memory, and even the number of containers or VMS needed by the application at any given time; accordingly, we can utilize our framework to scale multiple resources at any given time to achieve an operation plan made by the cloud operator, for instance, the cloud provider may require some application to provide high availability, low cost, or support energy saving mode. In such a case, our framework could have an operation plan proactive autoscaler may analyze and execute some scaling according to such a plan.
- Faster Proactive Auto Scaler framework: Integrating a Machine Learning-based optimization engine might increase the overall performance of the proactive auto scaler framework since it builds specific ML models for workload characterization and performance analysis based on analysis of monitoring data and system logs.
- Autonomous Proactive Auto Scaler: currently, our framework only provides scaling predictions to the DevOps team, and the DevOps team decides to scale. In the future, we need to investigate what will happen to the applications given that the framework makes the scaling decisions without the interventions of the DevOps team and if, in such scenarios, any problems arise to the applications that might cause violations of the application SLA agreement.

## AVAILABILITY OF DATA AND MATERIALS

The input data chosen to train and evaluate the ensemble model proposed in this work were obtained from real production applications hosted on cegedim cloud environment.

These input and test data sets are available as supplementary material to this manuscript and at link.

The link for the implemented proactive scaling framework in this manuscript is also available as on open source project at this link.

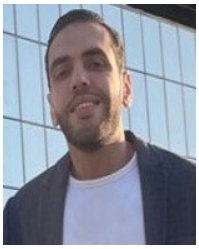
## ACKNOWLEDGMENT

Cegedim for providing the production data and support while development is greatly acknowledged.

## REFERENCES

[1] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic cloud services based on machine learning techniques," *J. Cloud Comput.*, vol. 8, no. 1, Dec. 2019.
[2] E. G. Radhika, G. S. Sadasivam, and J. F. Naomi, "An efficient predictive technique to autoscale the resources for web applications in private cloud," in *Proc. 4th Int. Conf. Adv. Electr., Electron., Inf., Commun. Bio-Inform. (AEEICB)*, Feb. 2018.
[3] C.-H. Lee, Z. He, Z. Li, X. Lu, J. Wang, and C. Wu, "A comparison of machine learning algorithms for automatic cloud resource scaling on a multi-tenant platform," *J. Phys., Conf. Ser.*, vol. 1828, no. 1, Feb. 2021, Art. no. 012039.
[4] J. V. B. Benifa and D. Dejey, "RLPAS: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Netw. Appl.*, vol. 24, no. 4, pp. 1348–1363, Aug. 2019.
[5] M. S. Aslanpour and S. E. Dashti, "Proactive auto-scaling algorithm (PASA) for cloud application," *Int. J. Grid High Perform. Comput.*, vol. 9, no. 3, pp. 1–16, Jul. 2017.
[6] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, Dec. 2014.
[7] G. Box, "Box and Jenkins: Time series analysis, forecasting and control," in *A Very British Affair*, 2013, pp. 161–215.
[8] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013.
[9] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *Proc. IEEE 4th Int. Conf. Cloud Comput.*, Jul. 2011.
[10] S. S. Gill et al., "AI for next generation computing: Emerging trends and future directions," *Internet Things*, vol. 19, Aug. 2022, Art. no. 100514.
[11] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10, pp. 1–35, Jan. 2022.
[12] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski, "N-HiTS: Neural hierarchical interpolation for time series forecasting," 2022, *arXiv:2201.12886*.
[13] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *Int. J. Forecasting*, vol. 37, no. 4, pp. 1748–1764, Oct. 2021.
[14] T. Toharudin, R. S. Pontoh, R. E. Caraka, S. Zahroh, Y. Lee, and R. C. Chen, "Employing long short-term memory and Facebook prophet model in air temperature forecasting," *Commun. Statist.-Simul. Comput.*, vol. 52, no. 2, pp. 1–24, 2021.

[15] Z. M. Omer and H. Shareef, "Comparison of decision tree based ensemble methods for prediction of photovoltaic maximum current," *Energy Convers. Manage., X*, vol. 16, Dec. 2022, Art. no. 100333.
[16] M. Xu, C. Song, S. Ilager, S. S. Gill, J. Zhao, K. Ye, and C. Xu, "CoScal: Multifaceted scaling of microservices with reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 4, pp. 3995–4009, Dec. 2022.
[17] M. S. Aslanpour, A. N. Toosi, R. Gaire, and M. A. Cheema, "Auto-scaling of web applications in clouds: A tail latency evaluation," in *Proc. IEEE/ACM 13th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2020.
[18] M. Xu, C. Song, H. Wu, S. S. Gill, K. Ye, and C. Xu, "EsDNN: Deep neural network based multivariate workload prediction in cloud computing environments," *ACM Trans. Internet Technol.*, vol. 22, no. 3, pp. 1–24, Aug. 2022.
[19] D. Buchaca, J. L. Berral, C. Wang, and A. Youssef, "Proactive container auto-scaling for cloud native machine learning services," in *Proc. IEEE 13th Int. Conf. Cloud Comput. (CLOUD)*, Oct. 2020.
[20] J. L. Berral, C. Wang, and A. Youssef. (Aug. 29, 2020). *AI4DL: Mining Behaviors of Deep Learning Workloads for Resource Management*. USENIX. Accessed: Sep. 5, 2022. [Online]. Available: https://www.usenix.org/conference/hotcloud20/presentation/berral
[21] OW2. *Rubis*. OW2 Projects—RUBiS. Accessed: Sep. 5, 2022. [Online]. Available: https://projects.ow2.org/view/rubis/
[22] OW2. *Rubbos*. OW2 Projects—RUBBoS. Accessed: Sep. 5, 2022. [Online]. Available: https://projects.ow2.org/view/rubbos/
[23] The Apache Software Foundation Incubator. *Olio*. Accessed: Sep. 5, 2022. [Online]. Available: https://incubator.apache.org/projects/olio.html



**MOHAMED SAMIR** received the B.Sc. degree in computer science from the Faculty of Computers and Information, Cairo University, in 2017, where he is currently pursuing the M.Sc. degree in software engineering with the Faculty of Computers and Information.

He is also a Senior Software Engineer with Cegedim, a health technology company-based in Boulogne-Billancourt, France, in 1969. His research interests include software evolution, software architecture, and software testing.

**KHALED T. WASSIF** received the B.Sc. degree (Hons.) in accounting from the Faculty of Commerce, Cairo University, in 1983, the Diploma (master's) degree in computer and information science from the Institute of Statistical Studies and Research, Cairo University, in 1986, and the master's and Ph.D. degrees in artificial intelligence from Cairo University, in 1991 and 1998, respectively.

He is currently a Professor with the Faculty of Computers and Artificial Intelligence, Cairo University. He has supervised or co-supervised 12 students on their Ph.D. dissertations and M.S. theses. He has published 30 research papers in international journals and conference proceedings. His research interests include machine learning, data mining, web mining, case-based reasoning, big data, and knowledge engineering. He is a Reviewer of the international *Egyptian Informatics Journal* and the *Egyptian Computer Science Journal*.

**SOHA H. MAKADY** received the B.Sc. and M.Sc. degrees (Hons.) in computer science from the Faculty of Computers and Information, Cairo University, Egypt, in 2002 and 2005, respectively, and the Ph.D. degree in software engineering from the University of Calgary, Canada, in 2015.

She is currently an Assistant Professor with the Faculty of Computers and Artificial Intelligence, Cairo University. She has supervised two M.Sc. students and one Ph.D. student. She is also supervising three Ph.D. students and three M.Sc. students. She has ten refereed research papers in international journals and conference proceedings. Her research interests include software evolution, software architecture, and software testing.