IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 April 2024, accepted 21 May 2024, date of publication 29 May 2024, date of current version 10 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3406670

RESEARCH ARTICLE

Stateless System Performance Prediction and Health Assessment in Cloud Environments: Introducing cSysGuard, an Ensemble Modeling Approach

NUTT CHAIRATANA[1] **AND RATHACHAI CHAWUTHAI**[2]
[1]Department of Robotics and AI Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
[2]Department of Computer Engineering, School of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand

Corresponding author: Rathachai Chawuthai (rathachai.ch@kmitl.ac.th)

ABSTRACT Stateless cloud computing presents remarkable scalability and cost-effectiveness by offering dynamically adjustable resources tailored to fluctuating demands, eliminating the constraints of stateful architectures. However, the challenges presented by dynamic workload are substantial in the context of system health monitoring, frequently leading to service interruptions owing to insufficient resources. It underscores the need for the development of more efficient monitoring systems. Our study introduces cSysGuard, a novel framework designed to enhance monitoring capabilities within cloud environments. The methodology employs an ensemble regression model with a stacking strategy to forecast dynamic performance metrics. The algorithm also leverages a classification model to assess the system's health based on forecasted metrics, effectively identifying potential failures in the future. Under the configuration utilized, our evaluations demonstrated increased predictive performance with cSysGuard in forecasting various metrics compared to traditional models. The results showed an improvement of up to a remarkable 2.28-fold increase, varying significantly based on the specific metric under consideration. In addition, the effectiveness of health assessment was achieved through Decision Trees with hyperparameter tuning, resulting in a macro-averaged F1 score of 89.79%. This research contributes to both the theoretical and practical aspects of server monitoring, presenting a solution that assesses system performance metrics and health to tackle dynamic challenges in cloud infrastructure.

INDEX TERMS Cloud computing, stacking ensemble models, machine learning, non-functional testing, predictive maintenance, resource allocation, proactive system metrics, stateless application.

I. INTRODUCTION

Cloud computing, renowned for its scalability [1] and flexibility [2], has revolutionized the management of system resources. Within its broad spectrum, stateless cloud computing plays a crucial role. This paradigm facilitates the flexible allocation of resources without retaining session information [3], significantly aiding in instance scaling. In such cases, it significantly enhances system reliability and adaptability to fluctuating demands while maintaining optimal system utilization. As easily integrated with various architectures, stateless computing has become the preferred framework for application hosting, directly catering to the dynamic demands of the digital environment.

Nonetheless, the unpredictable nature of cloud computing [4] often complicates effective resource allocation, carrying the risk of unexpected system failures that violate service-level agreements or SLAs [5]. Even though cloud providers offer auto-scaling mechanisms [6] for dynamic resource adjustment, they often rely on predefined rules or simple threshold-based techniques [7], [8]. While certain providers also provide predictive methods [9] for auto-scaling, the

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

systems are typically pattern-based [10]. These strategies often struggle with scaling within a reasonable timeframe during irregular workload patterns, such as traffic spikes, leading to inefficient resource management. A more robust framework is required to handle these challenges in such cases.

Furthermore, existing monitoring frameworks primarily focus on real-time monitoring data but rarely provide continuous analysis of system health based on current captured data. This limitation arises because the interplay between resource metrics and their sufficiency assessment—determining whether they are adequate or inadequate—is intricate. It is often characterized by nonlinear dependencies from various factors, such as application characteristics and system configuration. Therefore, relying solely on individual metrics to assess resource allocation adequacy effectively can be challenging.

To address these challenges, we require a sophisticated tool that accurately forecasts key metrics and assesses system health using these indicators in dynamic cloud environments. In response, we have developed cSysGuard, denoted as ''Cloud System Guard,'' a comprehensive ensemble predictor framework designed to handle unpredictable nature effectively. It builds upon our previous research [11], which primarily focused on real-time system-health analysis. With this advancement, cSysGuard comprises two main components: a system performance metric forecasting module and a health assessment feature. It predicts performance metrics and then promptly uses these forecasts to assess potential system failures. Unlike traditional real-time monitoring tools [51], [52], cSysGuard emphasizes machine learning-based prediction with customizable internal architecture. This flexibility allows users to tailor the configuration to their needs, maximizing predictive accuracy and robustness in rapidly changing cloud environments.

This manuscript elucidates our research on stateless applications, introducing cSysGuard, an ensemble machine learning framework devised for precise system metrics forecasting and health status diagnosis. The narrative begins with a thorough literature review (Section II), establishing a foundational understanding and drawing parallels with similar endeavors in the field. We then transition to a comprehensive overview of cSysGuard's methodology (Section III), detailing the high-level workings, component functionalities, and data preprocessing strategies. The discussion elaborates on system metrics forecasting (Section IV), highlighting the regression analysis framework, input-output formats, and the model's evaluation and tuning processes. Subsequently, we explore the system health assessment phase (Section V), focusing on classification analysis, predictor structure, and performance optimization techniques. The paper culminates in the Results and Discussion sections (Section VI), presenting a critical evaluation of cSysGuard's predictive capabilities, followed by a conclusion (Section VII) that encapsulates the essential findings and suggests avenues for future research.

II. LITERATURE REVIEW

This section provides a comprehensive synthesis of cloud computing with machine-learning backgrounds, insights from prior studies, and the techniques utilized in our research.

A. ADAPTATION: CLOUD WITH MACHINE LEARNING

Although cloud applications introduce innovative resource management strategies, they also present challenges in monitoring resources. Its variable nature complicates the effective tracking of fluctuating system metrics. Consequently, resource utilization can result in inefficiency and diminished system performance. It underscores the need for more flexible monitoring methods to precisely assess and adapt to the ever-changing requirements of cloud environments.

Therefore, there is growing interest in leveraging machine learning techniques to address these challenges [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28]. Machine learning-driven predictive analytics, widely used in many fields [29], can analyze complex data patterns to identity potential anomalies. This proactive approach enhances system reliability and resource allocation, ensuring scalable and responsive services. Integrating machine learning with cloud computing represents a significant advancement in improving system resource management.

B. RELATED WORK

Notably, many methodologies frequently favor singular predictive models [12], [13], [14], [15], [16], [17], [18], using universal models irrespective of specific system nuances, a practice often known as the ''one-size-fits-all'' approach. However, as emphasized by Kim et al. [22], relying on a single predictor model does not adequately address the cloud workload dynamics and short-term volatility. Singh et al. [19] proposed an alternative solution using a Support Vector Machine to categorize workloads into broad categories like ''very low,'' ''low,'' ''medium,'' and ''high.'' Although the method offers a simplified overview of the status, the reliance on categorical variables is insufficient for delivering precise values in applications that require detailed nuances. Gao et al. [20] and Caron et al. [21] proposed forecasting methods based on historical pattern data. Due to their heavy reliance on historical trends, these methods are limited in rapidly evolving environments, particularly when encountering novel patterns that emerge without precedent. In contrast, cSysGuard employs an ensemble machine-learning approach that leverages the unique strengths of various models to achieve robust predictive capabilities. This strategy effectively adapts to the rapidly evolving cloud environments.

Several studies have proposed ensemble model approaches that incorporate numerous predictors. Kim et al. [22] achieved balanced predictions by employing multiple models using a weighted averaging strategy. This method maximizes the strength of each model, calibrated by their predictive

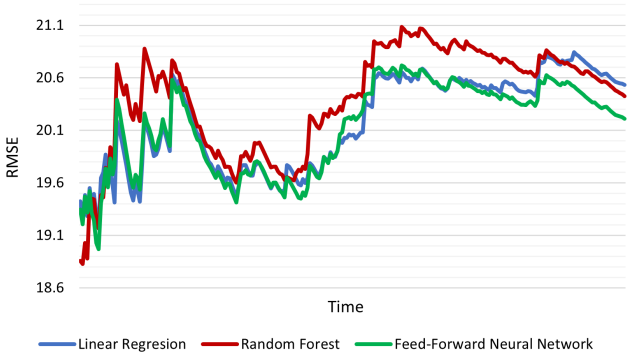

FIGURE 1. RMSE-based performance of meta models. This figure illustrates the fluctuating performances of different meta-models for CPU requests (unit: percent) over 20 minutes in a dynamic environment.

performance. Nonetheless, as Section VI-C indicates, relying solely on weighted averaging yields a lower accuracy than cSysGuard, where high predictive accuracy is essential for anticipating potential failures. Mehmood et al. [23] proposed a stacking ensemble model using Decision Trees as a meta-model to finalize the predictions from the base models. Our findings indicate that a single meta-model may not always provide robust performance, especially in unpredictable environments. Fig. 1 illustrates the varying performances of meta-models for Central Processing Unit (CPU) requests in a dynamic cloud environment over time. The data indicates that the best-performing meta-model changes periodically, presenting a challenge to select the best meta-model for the specific usage. Rather than building upon a single meta-model or weight averaging, cSysGuard employs a multi-layered stacking ensemble approach with a weighted averaging strategy, effectively amalgamating the contributions of all models in the output generation.

On the other hand, deep learning techniques have been extensively utilized in several studies [24], [25], [26], [27], [28]. Although deep learning techniques are potent instruments for predictive analysis, they inevitably require substantial computational resources, even when applied to simpler metric patterns. Meanwhile, cSysGuard offers a balanced strategy for computational resource utilization and predictive accuracy by judiciously selecting well-suited models with specific data characteristics. Even the best accuracy is considered without regard to computational complexity; cSysGuard still performs better than standalone deep learning models, owing to its incorporation of a diverse array of predictive models.

Our prior research focused on system health assessments. Although the inputs remain consistent with our current work, the approach in this study diverges by employing forecasted values instead of actual historical data. This methodology integrates our regression analytics to predict these metrics and feeds the results into our classification health assessment process. This innovative approach enhances our capability to identify potential future system failures by leveraging insights to inform proactive measures for resource management.

C. TECHNIQUE

This subsection provides a comprehensive overview of the essential techniques and a foundation for understanding the advancements in our study area.

1) LOW-PASS BUTTERWORTH FILTER

The low-pass Butterworth filter, a key concept in signal processing [30], attenuates high-frequency noise while maintaining signal integrity. Characterized by its maximally flat magnitude response and ripple-free passband, the filter ensures a smooth frequency response up to the cut-off frequency, making it ideal for applications requiring undistorted outputs. Its transfer function, defined by the filter order, influences the steepness of the roll-off at the cut-off point [31]. Higher-order filters offer a faster passband-to-stopband transition but increase the complexity and risk of phase distortion. In the data analysis, the filter actively smooths out noise and data variations, bolstering the reliability of the analytical results.

2) SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE

The synthetic minority over-sampling technique (SMOTE) addresses imbalanced data by generating synthetic samples for the minority class, thereby improving the data distribution for machine learning applications. It creates new instances through interpolation between minority samples and their neighbors [32], enhancing diversity and reducing overfitting risks. However, its effectiveness depends on careful parameter tuning, including the number of neighbors and the amount of synthetic data generated. The versatility of SMOTE makes it applicable across various domains, from fraud detection to medical diagnoses, where it significantly contributes to more equitable and accurate predictive modeling.

3) ENSEMBLE MODEL: STACKING

The stacking approach in ensemble models features a multi-layered structure, each recognized as a Level-N layer, for enhanced prediction. Initially, the base models process the data independently, offering diverse analytical perspectives. Higher-level models then collectively analyze their outputs to enhance predictive accuracy. Each subsequent level integrates and improves upon the previous ones [33], reducing biases and variances, thus enhancing the overall predictive accuracy. Fig. 2 visually represents the multi-layer stacking approach, presenting the flow from Level-0 to Level-N layers.

4) REGRESSION ANALYTICS

The Autoregressive Integrated Moving Average (ARIMA) model is essential for time-series analysis, effectively capturing temporality with autoregressive features and moving averages for non-seasonal forecasting [34]. The Seasonal Autoregressive Integrated Moving Average (SARIMA) builds on ARIMA by incorporating seasonality, making it adept at predicting seasonal fluctuations in time series data.

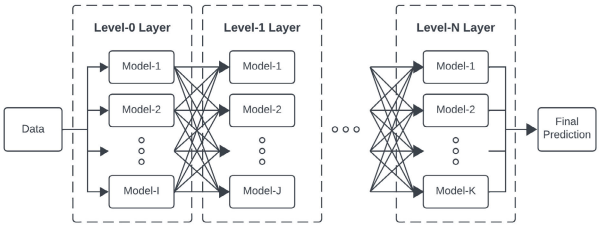

FIGURE 2. Multi-layer stacking in ensemble modeling. This figure depicts the layered architecture of our ensemble model, from initial Level-0 models to advanced aggregation layers, highlighting the sequential prediction and refinement process.

Exponential Smoothing (ETS) is a time-honored technique for forecasting that assigns exponentially decreasing weights over time. It captures trends and seasonalities in time-series data [35], and its simplicity makes it particularly effective for quick and short-term forecasting.

Linear Regression is a fundamental model in statistical analysis [36] and is ideal for predicting a continuous dependent variable using one or more independent variables. It assumes a linear relationship between the variables. A simple LR involves a single independent variable, whereas multiple LRs extend to multiple variables. This model is particularly effective in scenarios where a linear correlation exists, with coefficients indicating the influence of each variable.

Random Forest creates multiple decision trees to improve the prediction accuracy and robustness. It leverages bagging and feature randomness in tree construction [37], reducing overfitting and enhancing diversity. This method effectively handles complex datasets, offering a reliable performance in various applications.

Feedforward Neural Networks feature a straightforward input, hidden, and output layer structure [38]. Using sequential neuron connections, they excel in complex pattern recognition and prediction. Highly versatile and efficient, they are ideal for handling large datasets. Their widespread use in applications such as speech recognition in predictive analytics stems from their proficient nonlinear modeling of intricate relationships between inputs and outputs.

Convolutional Neural Networks (CNN), renowned for their efficacy in image processing, also excel in time-series and regression tasks owing to their ability to prioritize patterns through convolutional layers [39]. It enables practical applications in audio processing and natural language tasks, extending their use beyond visual data analysis.

Temporal Convolutional Networks (TCN) combine the strengths of CNN with the temporal sensitivity suited for sequence modeling [40]. They effectively handle long input sequences in complex time series forecasting tasks.

Recurrent Neural Networks (RNN) are fundamental models for sequential data analysis. It has the unique ability to retain information across sequences [41], making it essential for text processing, speech recognition, and time-series analysis. Nevertheless, RNN encounters challenges with long-term dependencies, leading to the development of advanced versions, such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM). The GRU incorporates a simplified gating mechanism to address the vanishing gradient problem [42], balancing computational efficiency with the ability to learn long-term dependencies. This advantage makes the GRU suitable for complex tasks such as language modeling and nuanced time-series analysis. Alternatively, LSTM selectively retains or forgets information [43], making it practical for extended sequences. Its capability to handle long-term dependencies makes LSTM ideal for challenging tasks, such as predictive text generation, advanced time series forecasting, and language translation.

5) CLASSIFICATION ANALYTICS

Decision Trees are known to simplify complex decision-making. They created an intuitive, tree-like structure by segmenting datasets into branches based on feature values [44]. This method divides data into smaller, more manageable subsets, effectively handling nonlinear relationships among variables and facilitating hierarchical decision processes.

A Support Vector Machine (SVM) is used in classification and regression tasks. It offers robust performance in high-dimensional spaces owing to its margin maximization principle, which forms a hyperplane for data classification [45].

K-Nearest Neighbors (KNN) is an instance-based algorithm that classifies data by analyzing the majority class among its "K" nearest neighbors, relying on a similarity principle [46]. The algorithm's accuracy is sensitive to the chosen number of neighbors and the specific distance metric applied, thereby impacting its adaptability to various data.

Logistic Regression, essential for binary classification in machine learning [47], uses a logistic function to convert predictor variables into probabilities, thereby accommodating nonlinear variable relationships. In various fields, like medical diagnosis and spam detection, employing maximum likelihood estimation for coefficient calculation is pivotal.

Neural Networks, inspired by the structure of biological neurons, are composed of complex interconnected layers of artificial neurons or nodes [48]. It begins with an input layer, includes one or more hidden layers, and ends with an output layer. Each node simulates a biological neuron, calculates a weighted sum of inputs, and applies a nonlinear transformation commonly termed the activation function.

6) EVALUATION METHOD

The Root Mean Square Error (RMSE) is a critical metric for assessing regression tasks, quantifying the square root of the mean of the squared discrepancies between the predicted and actual values. By squaring the errors before averaging, the RMSE heavily penalizes the larger discrepancies, making it particularly sensitive to significant prediction errors. This characteristic enhances its utility in offering a precise measure of model accuracy. The formula is detailed in (1)

$$RMSE = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(y_k - \hat{y_k})^2} \tag{1}$$

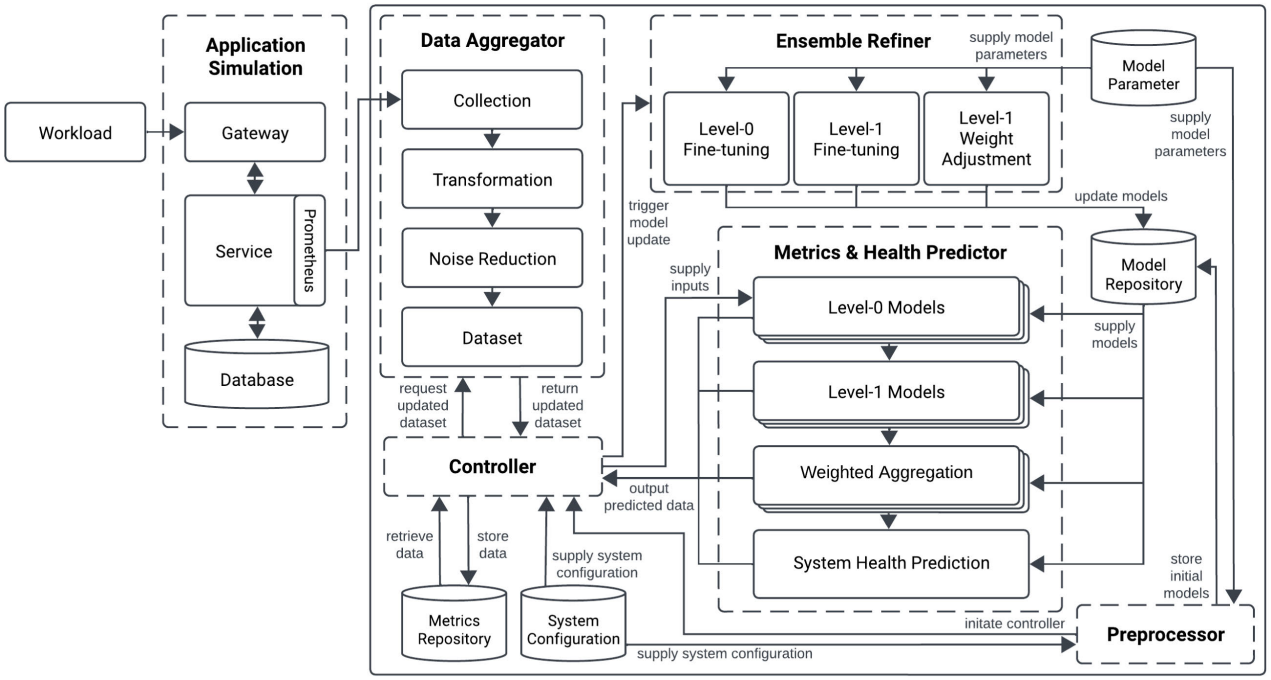

FIGURE 3. cSysGuard architecture. This figure demonstrates the process of system preprocessing, data collection, performance metrics forecasting, health assessment, and continuous model refinement.

where y_k represents the actual value, $\hat{y_k}$ denotes the predicted value, and n signifies the number of observations. A lower RMSE indicates better accuracy compared to a higher one.

The F1 score is a key metric for evaluating classification tasks. It is computed by harmonically averaging precision, which is the ratio of true positives to all positive predictions, indicating the accuracy of positive classifications, and recall, the ratio of true positives to all actual positives, reflecting the ability to identify all relevant instances. These calculations are elaborated in (2), (3), and (4) for class n.

$$\text{Precision}_n = \frac{TP_n}{TP_n + FP_n} \tag{2}$$

$$\text{Recall}_n = \frac{TP_n}{TP_n + FN_n} \tag{3}$$

$$\text{F1-Score}_n = 2 \times \frac{\text{Precision}_n \times \text{Recall}_n}{\text{Precision}_n + \text{Recall}_n} \tag{4}$$

In addition to these equations, TP represents true positives, FP signifies false positives, and FN indicates false negatives. An F1 score of 1 signifies perfect precision and recall, indicating that all predictions are accurate and complete. Conversely, a score of 0 represents the lowest accuracy, where the model fails to identify any true positive values.

III. METHODOLOGY OVERVIEW

This section provides a detailed overview of the methodology used in our study, focusing on the design and data handling of cSysGuard.

A. CSYSGUARD ARCHITECTURE

Fig. 3 comprehensively depicts the architecture of cSysGuard, comprising six core components: application simulation, data aggregator, controller, metrics and health predictor, ensemble refiner, and preprocessor. Each is integral to the prediction process. The following sections briefly describe the system workflow and each element, elucidating their functions and collaborative interactions.

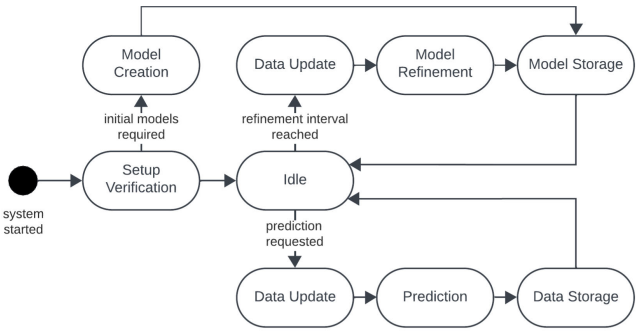

FIGURE 4. cSysGuard execution workflow. The figure illustrates cSysGuard's operational sequence, which mainly encompasses the stages of preprocessing, data prediction, and model refinement.

1) EXECUTION WORKFLOW

Fig. 4 reveals the execution workflow of cSysGuard. Initially, cSysGuard determines whether the configuration necessitates the creation of models. In such cases, it initiates and stores these models for future analysis; otherwise, it bypasses the preprocessing step and proceeds directly to the primary process using the pre-existing models stored in the repository. In the main phase, the system remains idle until a task emerges. Upon reaching the prediction intervals, the algorithm collects data from the target application to access the most recent inputs and then activates the

predictors to generate and store the forecast. In addition to preserving model performance, the refinement process is activated at predefined intervals to utilize the latest dataset to align models with current data trends. Post-refinement, updated models are stored. This cycle of prediction and refinement recurs over time, ensuring cSysGuard's processes are perpetually optimized and up-to-date.

2) APPLICATION SIMULATION

This component emulates an application cloud environment, providing a means to acquire the dataset. The study leveraged Digital Ocean's cloud service [49], deploying three Linux virtual machines [50], each with 1vCPU, 1GB of memory, and a 10GB disk. The API gateway is responsible for routing requests to the stateless application service, which utilizes a database for data storage. Our target service incorporates a monitoring tool called Prometheus [51] to collect and store metrics over time methodically. In addition, the simulator employs a custom script to simulate dynamic workloads, manipulating the intensity of various metrics in unique combinations for each request. For instance, one scenario may involve high CPU and memory usage with low usage in other metrics, whereas another could stress high bandwidth usage while keeping the rest low. This approach enables the model to evaluate the system's performance under fluctuating conditions, focusing on randomness and swift changes. These assumptions mirror real-world cloud environments, enhancing the generalizability of cSysGuard by ensuring it can adapt to unpredictable and rapidly changing workloads.

3) DATA AGGREGATOR

This component is responsible for updating and structuring the dataset for modeling and analysis. Initially, the process aggregates the system data from the application service and then transforms the metrics into a suitable format for predictive modeling. To enhance the accuracy of the forecasts, it also performs a noise reduction process to minimize data inconsistencies. Finally, the component forwards the processed dataset to the controller for further processing.

4) CONTROLLER

This component is crucial for orchestrating cSysGuard's logic flow. It begins by fetching the essential setting parameters, such as the model refinement and prediction schedules, from the configuration repository. The controller initiates the models to generate system metrics and health forecasts. Afterward, it archives the predictions in the metrics repository. Beyond these tasks, it manages the model maintenance by periodically signaling the refiner to tune models as per the schedule. Additionally, it consistently acquires the latest data from the aggregator for system predictions and refinements. This adept coordination by the controller is vital to cSysGuard's seamless operation and overall dependability and efficiency.

5) METRICS AND HEALTH PREDICTOR

This component mainly oversees the prediction process, including forecasting system performance metrics and health assessments. Once activated by the controller, the prediction process is initiated with the most recent dataset as the input. Represented as an ensemble regression model, the first three layers—Level 0, Level 1, and Weight Aggregation—mainly focus on forecasting performance metrics, as detailed in Section IV. Besides, we incorporate several sets of predictors within the forecasting layers, with each set specializing in specific metrics. Upon completion, it consolidates and forwards the results to the final layer for system health assessment, requiring inputs from all metric sets for a thorough evaluation, as detailed in Section V. Moreover, each layer sends its predictions to the controller for storage, serving as valuable data for subsequent model evaluation and refinement. This component ensures precise and timely predictions through a comprehensive and systematic ensemble architecture.

6) ENSEMBLE REFINER

This component maintains the regression model's proficiency in identifying the trends and data characteristics. Upon activation by the controller, the refiner fine-tunes the Level-0 and Level-1 models by utilizing the latest dataset. Similarly, it adjusts the Level-1 weights based on the range of historical predictions, dynamically aligning the models' strengths with their predictive efficacy. Once completed, the system promptly updates the repository with the newly refined model and weights. Through this iterative refinement, the component enhances the adaptability and precision of the models, ensuring they remain effective in a dynamic environment.

7) PREPROCESSOR

This component is essential for setting up the models cSysGuard requires to initiate the primary process. The preprocessor commences by retrieving the list of models from the system configuration. Then, it actively compiles the models with predefined parameters and training datasets. Upon completion, the calibrated models are stored in the repository. However, the preprocessing step is optional if the repository already contains the models.

B. DATA PREPROCESSING

In the data collection, we implemented a custom script to access Prometheus' local storage to receive resource utilization and performance indicators. The dataset [57] comprises approximately 8,000 data points, each accumulated every 5 seconds. The metrics include CPU and memory requests (expressed as percentages), network inbound and outbound rates (measured in gigabytes per second, GB/s, or megabytes per second, MB/s), transactions per second (TPS), and response time (in seconds, s, or milliseconds, ms), as illustrated in Fig. 5. Moreover, we collected the system's

TABLE 1. Transformed system performance metrics dataset with server health in 40 seconds.

CPU Request[1]	Memory Request[1]	Inbound Bandwidth (MB/s)	Outbound Bandwidth (MB/s)	TPS	Response Time (ms)	System Health (binary)
0.12	0.505	3160	3430	3.6	2910	1
0.124	0.504	3590	3120	2.2	1150	0
0.186	0.542	5920	5290	2.2	2560	1
0.126	0.505	3530	3900	4	4060	1
0.184	0.526	4890	4310	2	2320	0
0.150	0.534	4610	4090	3	1880	0
0.174	0.533	6250	5490	2.4	1370	0
0.164	0.535	3490	3150	2.8	1950	0

[1]Values are normalized to a range from 0 to 1.

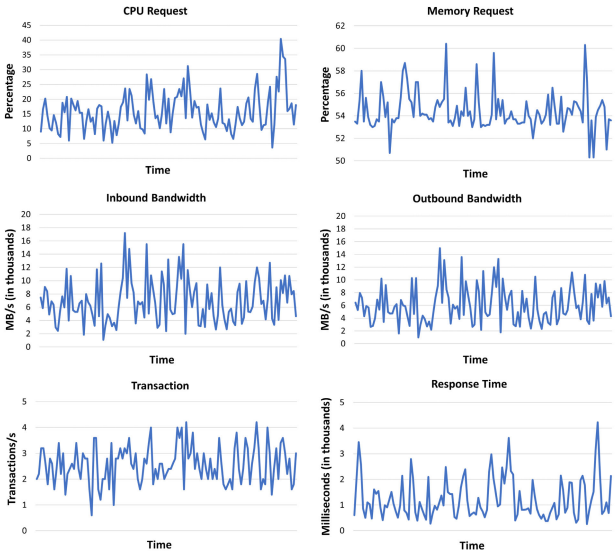

FIGURE 5. System performance metrics across different parameters. This figure displays a series of graphs that showcase CPU and memory requests, inbound and outbound bandwidth, transaction rates, and average response times, all measured over a 10-minute period.

real-time health status, which was determined by scrutinizing HTTP response codes [53]. When the script detects predefined error response codes, it classifies the system health as "unhealthy"; in all other cases, it designates the status as "healthy." Although stateless applications rarely leverage disk storage, which is not part of the training input, we still monitor it to ensure its proper stateless behavior.

Subsequently, the system performs a series of transformations to format the data appropriately for subsequent modeling and analysis. Table 1 shows the modification results, outlining how various metrics were adjusted to meet the analytical requirements. Specifically, we normalized CPU and memory requests to a scale from 0 to 1, transformed inbound and outbound bandwidth in megabytes per second (MB/s), and quantified response time in milliseconds (ms). We retained the TPS feature in its original form as collected. The mechanism also transformed system status from the original textual designations into a numerical format, encoding "healthy" as 0 and "unhealthy" as 1, facilitating binary analysis.

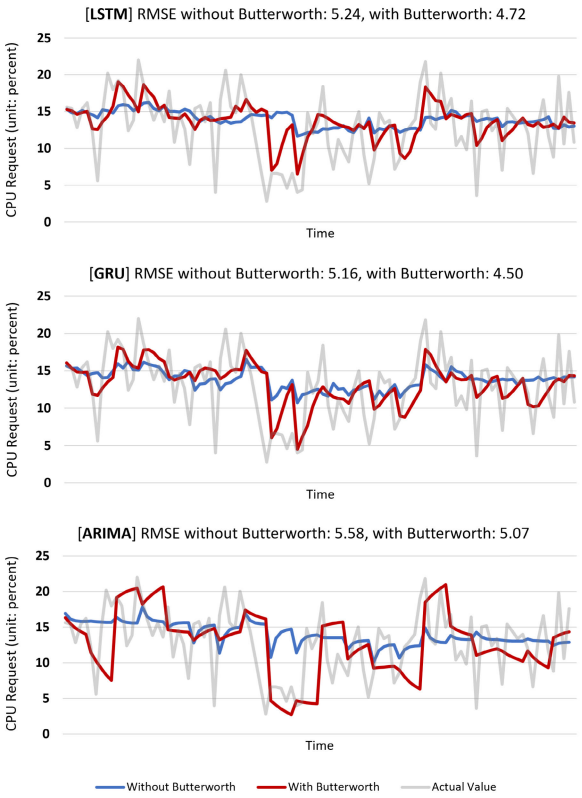

FIGURE 6. Comparative predictions and RMSE with/without low-pass butterworth filter. This figure displays the 10-minute predictions for LSTM, GRU, and ARIMA models in CPU requests, illustrating the comparison between scenarios with filtered and unfiltered data.

Eventually, the system performs a noise reduction process utilizing a low-pass Butterworth filter to minimize data inconsistencies. To demonstrate an enhanced forecasting accuracy, Fig. 6 showcases the impact of noise reduction by contrasting the predictions and RMSE of LSTM, GRU, and ARIMA models using both noise-filtered and raw data inputs with actual values. It highlights the enhancement achieved by applying a filter to reduce noise. The process then yields and stores a well-refined dataset for further analysis.

IV. SYSTEM METRICS FORECASTING

This section describes the detailed process of forecasting performance metrics. This comprehensive examination

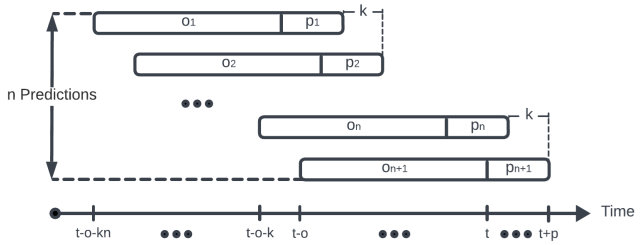

FIGURE 7. Regression prediction sequence. This figure displays the prediction cycle, integrating *o* observations and *p* forecast steps within a set interval *k* over *n* predictive cycles.

covers the intricacies of Level-0 and Level-1 learners, Weight Aggregation, and the approaches used for comparing models and tuning hyperparameters in regression models.

A. LEVEL-0 LEARNER

Level-0 learners comprise base models designed for time-series prediction that utilize datasets aggregated directly from the system for their training. At each predefined interval, these models use the most recent dataset to predict forthcoming values, considering two key elements: the predetermined count of past data points (observation steps) and the quantity of future data points (prediction steps) to be predicted. To illustrate, Fig. 7 visualizes the sequence of time series predictions, showcasing observation steps "o" and prediction steps "p" for each interval "k" across "n" predictions. cSysGuard allows the adjustment of these parameters to meet user-specific requirements. In this study, we configured the settings with 30 observation steps and five forecast steps, aligning the interval with the forecasting steps.

Our setup has operated various models, including the ARIMA, SARIMA, ETS, CNN, TCN, RNN, GRU, and LSTM. Each brings a unique forecasting approach, enabling us to capture various aspects and patterns within the data. In addition to deep learning, we incorporated two hidden layers into the architecture: the first equipped with 64 neurons and the second with 32 neurons, both employing ReLU activation. We used the default settings provided by the Statsmodels [54] and TensorFlow [55] libraries. Despite this, we observed that the library's default parameter values for deep learning led to suboptimal predictions. Therefore, parameter adjustments are required. Table 2 outlines the model-fitting configuration, ensuring the retention of initial learning proficiency of deep learning models. Upon prediction, the models forward the results to the subsequent layer, Level 1.

B. LEVEL-1 LEARNER

Level-1 learners serve as meta-models designed for refining the regression outputs and constitute the second layer in our predictive framework. Their main objective is to improve the forecast precision by leveraging the outcomes from Level-0 models as inputs to produce refined outputs. This integration enables Level-1 learners to fine-tune the initial forecasts, enhancing the overall predictive performance of cSysGuard.

TABLE 2. Parameters influencing performance in deep learning model training.

Adjusted Parameter	Values	Default
Epochs	50	1
Batch Sizes	32	none
Default Parameter	**Values**	
Shuffle	true	
Validation Split	0.0	
Validation Steps	none	
Validation Batch Size	none	
Validation Frequency	1	
Initial Epoch	0	
Steps Per Epoch	none	
Callbacks	none	
Class Weight	none	
Sample Weight	none	

This layer comprises three distinct models: Linear Regression, Random Forest, and Feedforward Neural Networks. Linear Regression provides simplicity and interoperability to effectively capture linear relationships within the data. Random Forest contributes to reducing overfitting and improving generalization. Feedforward Neural Networks excel at capturing complex, nonlinear patterns and interactions in the data. This combination leverages the strengths of each model, ensuring that the stacking approach can effectively generalize across different metrics and system behaviors to enhance the predictive accuracy and robustness of cSysGuard.

For Linear Regression and Random Forest, we employed the standard parameters provided by the Scikit-learn library [56]. For Feedforward Neural Networks, we replicated the deep learning structure and model-fitting parameters of Level-0 models using the TensorFlow library.

To optimally balance cSysGuard's predictive performance and computational complexity, we adopted a selective approach to determine which Level-0 models would be fed into specific Level-1 models for predictions. The selection depends on the impact each model has on the predictive performance for particular metrics, evaluated by the Pearson correlation coefficient, r [58], which is defined as follows:

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{5}$$

where x_i represents the predicted metric values from Level-0 models, $\bar{x}$ signifies the average predicted metric across Level-0 models, y_i indicates the actual metric values, and $\bar{y}$ reflects the mean of the actual metrics. Upon score calculation, we sorted the models in descending order based on their correlations to gauge their impact. For instance, Fig. 9 illustrates the Pearson correlations of each base model for predicting CPU requests. We then incrementally incorporated Level-0 models into each Level-1 learner, starting with those having the highest correlation scores, and assessed

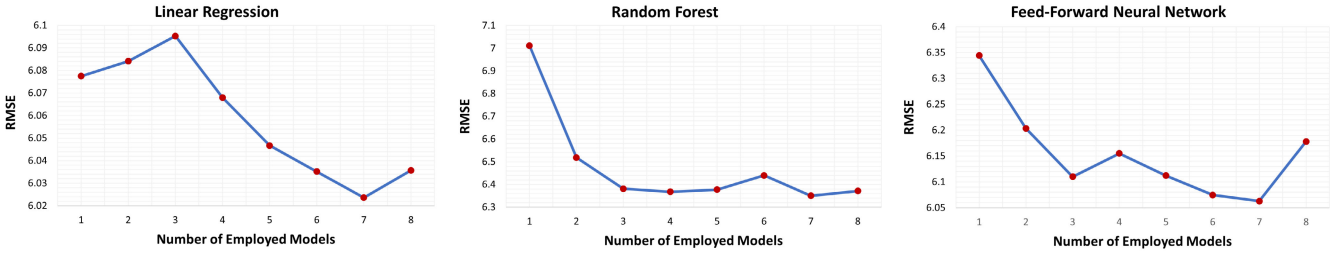

FIGURE 8. Incremental performance improvement in level-1 models. This figure illustrates RMSE progression for Level-1 models predicting CPU requests (unit: percent), highlighting the incremental performance enhancements achieved with each addition of a Level-0 model.

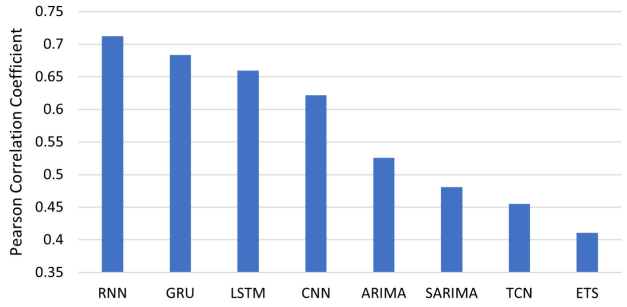

FIGURE 9. Correlation analysis of level-0 models in CPU request prediction. This figure ranks the base models based on the Pearson correlation scores.

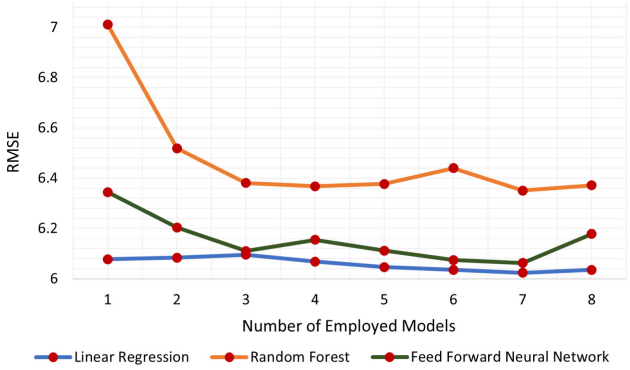

FIGURE 10. Consolidated performance analysis of level-1 models. This figure synthesizes the RMSE achievements of Level-1 models in forecasting CPU requests (unit: percent), encapsulating the collective efficacy derived from integrating multiple Level-0 models.

the predictive performance of each Level-1 model using the RMSE. This iterative process continued until all base models were utilized and assessed. For example, Fig. 8 demonstrates the improvement in the performance of Level-1 models for predicting CPU requests by adding each model. Alternatively, Fig. 10 offers a unified visualization of the enhancements across the models, presented in a single comprehensive graph. Our analysis aimed to pinpoint an "elbow point" [59], where incorporating additional models led to minimal gains in the predictive accuracy for the specific metric. This selective approach to model integration significantly conserves computational resources while ensuring high predictive precision, demonstrating a successful trade-off between performance and computational efficiency in intricate forecasting tasks.

C. WEIGHTED AGGREGATION LAYER

The weighted aggregation layer is the final regression stage for predicting forthcoming metrics. To effectively leverage the contributions of each Level-1 model, the system receives predictive model outputs and refines the final prediction using a weight-averaging method [60] by assigning higher weights to more accurate models. In addition to the weight calculation, the layer computes the Performance Vector (PV) of Level-1 learners based on past t refinement iterations. The PV is represented as a $3 \times t$ matrix:

$$PV = \begin{bmatrix} PE_{LR,1} & PE_{LR,2} & \cdots & PE_{LR,t} \\ PE_{RF,1} & PE_{RF,2} & \cdots & PE_{RF,t} \\ PE_{FF,1} & PE_{FF,2} & \cdots & PE_{FF,t} \end{bmatrix} \tag{6}$$

where it consolidates the prediction errors, measured by RMSE, for Linear Regression (LR), Random Forest (RF), and Feedforward Neural Networks (FF) across t refinement iterations. The refinement process enables ongoing updates to the model performance, ensuring sustained accuracy in a rapidly changing cloud environment.

Nevertheless, incorporating the RMSE directly as weights presents a significant challenge, as it may not adequately reflect model correlations. A model with a slightly lower RMSE may not enhance the ensemble diversity if similar to another model. To mitigate this issue, we developed a normalized weighting system derived from the PV. The calculation scales the errors into normalized weights, identified as $NormWeight_{i,j}$, converting them to a range between 0 and 1 for comparability across different models. The weight normalization formula is outlined in (7):

$$NormWeight_{i,j} = \frac{\exp(-\alpha \times PE_{i,j})}{\sum_{k \in \{LR,RF,FF\}} \exp(-\alpha \times PE_{k,j})} \tag{7}$$

where the weights are calculated by inversely relating them to their RMSE, expressed as $\exp(-\alpha \times PE_{k,j})$, ensuring that models with lower errors are more reliable than those with higher errors. In addition, α acts as a sensitivity controller, adjusting the weight assignment based on error magnitude. α facilitates a subtle balance between emphasizing and moderating differences in model errors and acts as a controllable safeguard to prevent integer overflow [61] during exponential system computations. To maximize the weight difference, our

study aimed to maximize each weight value while ensuring that outcomes stayed within an acceptable range as controlled by α. This strategy allows for a more pronounced distinction in model weights, which is particularly beneficial when the models' performances are nearly identical.

To derive the final prediction, we integrated the outputs from our Level-1 learners through a weighted averaging approach. The final prediction for a given time point m, represented by P_m, is determined as (8):

$$P_m = \sum_{k \in \{LR, RF, FF\}} NormWeight_{k,j} \times P_{k,m} \tag{8}$$

where each model's prediction, denoted as $P_{k,m}$, is multiplied by its respective normalized weight, $NormWeight_{k,m}$. This process guarantees that each model's contribution to the final prediction is proportional to its performance. By harmonizing the models' strengths, this approach yields a more precise and reliable ensemble forecast.

D. MODEL COMPARISON

Our approach evaluated cSysGuard's efficacy in forecasting system metrics by comparing its performance with traditional models used in the Level-0 layer. We utilized RMSE to compare and benchmark the performance across different metrics. This method clearly explains the precision and dependability of cSysGuard, emphasizing its capability to manage dynamic environments effectively.

E. HYPERPARAMETER TUNING

After comparing the models, we concentrated on developing a strategy to enhance cSysGuard further. Therefore, we selected one model from each Level-0 and Level-1 layer and investigated their impacts by adjusting the hyperparameters through Bayesian optimization [62]. Subsequently, we evaluated the improvements using the RMSE as our benchmark.

Given the system configuration constraints outlined in Section VI-E, we limited our hyperparameter tuning to particular models and metrics to mainly focus on a methodology optimizing cSysGuard. With an emphasis on enhancing CPU request predictions, the study focused on refining the RNN and Random Forest models, designated as Level-0 and Level-1, respectively. These models were selected not only due to their numerous tunable parameters but also to demonstrate that our hyperparameter tuning methodology is effective across deep learning and traditional machine learning techniques in a stacking architecture.

The hyperparameters covered various factors within specified ranges, as detailed in Table 3, directly affecting model performance. For the RNN, these parameters include data structuring elements (such as observation steps), model architecture (covering learning rate, the number of neurons, and activation functions in both the first and second layers), and model training configurations (incorporating epochs and batch size). For the Random Forest, considered parameters entail the number of trees (estimators), tree depth (max depth), minimum samples for a split (min sample split), minimum samples per leaf (min sample leaf), maximum number of features considered for splitting a node (max features), maximum number of leaf nodes (max-leaf nodes), minimum impurity decrease for a split (min impurity decrease), whether bootstrap samples are used (bootstrap), and the function to measure the quality of a split (criterion).

TABLE 3. Range of hyperparameters for recurrent neural networks (RNN) and random forest (RF) tuning in CPU request analysis.

Range of Hyperparameters for RNN (Level-0)	
Parameter	**Range of Values**
Observation Steps	5 to 100
Learning Rate	0.001 to 0.1
Number of Neurons in Layer 1	20 to 80
Number of Neurons in Layer 2	20 to 80
Activation Functions in Layer 1	relu, tanh, sigmoid
Activation Functions in Layer 2	relu, tanh, sigmoid
Epochs	10 to 150
Batch Size	16 to 64
Range of Hyperparameters for RF (Level-1)	
Parameter	**Range of Values**
Number of Estimators	10 to 1000
Max Depth	3 to 30
Min Sample Split	2 to 20
Min Sample Leaf	1 to 20
Max Features	auto, sqrt, log2
Max Leaf Nodes	10 to 1000
Min Impurity Decrease	0 to 0.1
Bootstrap	true, false
Criterion	squared error, absolute error, poisson, friedman mse

V. SYSTEM HEALTH ASSESSMENT

This section describes the system health assessment approach, focusing on layer concepts, model comparisons, and hyperparameter tuning in the classification models.

A. HEALTH PREDICTION LAYER

The health predictor represents the last layer in our predictive framework. It has a classification model designed to determine the system's health. Based on inputs from the forecasting layer that precedes it, this model categorizes the system's condition as either ''healthy'' or ''unhealthy.''

Moreover, we utilized the sliding window technique to incorporate data trends into the prediction process to enhance predictive accuracy. To forecast the target point, the model concentrates on the latest data within a specified range, defined by the window size. As illustrated in Fig. 11, cSysGuard adopts this technique for system health assessment, concentrating on the ten latest forecast points from the regression layer to ascertain the current system status. This method effectively captures the temporal patterns of

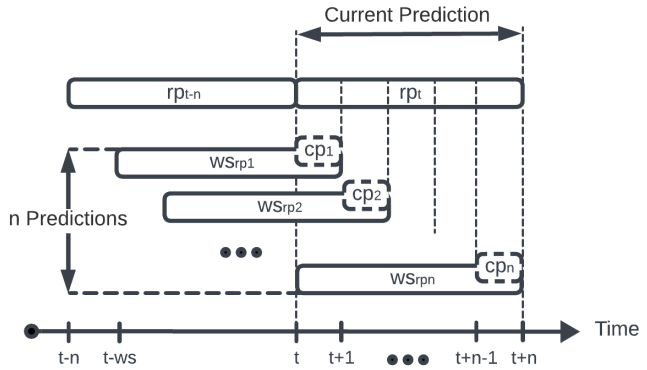

FIGURE 11. Sliding window technique in classification prediction. This figure demonstrates the sliding window technique for system health assessment, featuring the window size *ws* of regression predictions *rp* over *n* cycles, resulting in classification predictions *cp*.

data, leading to more accurate forecasts of future system conditions.

In addition to the imbalanced nature of our dataset [63], characterized by a higher prevalence of "healthy" statuses, there is a risk of models favoring the majority class and underperforming on the crucial but rarer "unhealthy" class. To address this, we implemented SMOTE to enhance the representation of the minority class, thereby enhancing the model equity by creating synthetic instances of this class to improve its detection.

B. MODEL COMPARISON

An essential aspect of our methodology is the comparative analysis of various machine learning models to assess their performance efficacy. For this analysis, we selected five prominent models: Decision Trees, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, and Neural Networks. These models were initially performed with the default parameters provided by the Scikit-learn and TensorFlow libraries [55], [56]. In addition to Neural Networks, our configuration featured a two-layered structure: the first layer had six nodes using the ReLU activation function, and the second comprised a single node with a Sigmoid activation function.

To enhance the model's generalizability, the evaluation process employed 10-fold cross-validation, conducting several training rounds and validation on different data splits to derive the average scores. We utilized precision, recall, and F1 score with macro averaging [64], ensuring equal weighting of each category in the assessment throughout the cross-validation process. The model with the highest macro-averaged F1 score is the most optimal, as it demonstrates a balanced trade-off between precision and recall, with the specifics detailed in (9), (10), and (11).

$$\text{Precision}_{\text{macro}} = \frac{\text{Precision}_{\text{healthy}} + \text{Precision}_{\text{unhealthy}}}{2} \quad (9)$$

$$\text{Recall}_{\text{macro}} = \frac{\text{Recall}_{\text{healthy}} + \text{Recall}_{\text{unhealthy}}}{2} \quad (10)$$

TABLE 4. Hyperparameter ranges in decision tree tuning.

Parameter	Range of Values
Max Depth	3 to 500
Min Samples Split	2 to 100
Min Samples Leaf	1 to 100
Criterion	gini, entropy
Max Features	none, auto, sqrt, log2
Max Leaf Nodes	10 to 1000
Min Impurity Decrease	0 to 5

$$\text{F1-Score}_{\text{macro}} = 2 \times \frac{\text{Precision}_{\text{macro}} \times \text{Recall}_{\text{macro}}}{\text{Precision}_{\text{macro}} + \text{Recall}_{\text{macro}}} \quad (11)$$

C. HYPERPARAMETER TUNING

Subsequently, we carefully refined the model's hyperparameters through Bayesian optimization to enhance its predictive capabilities with our dataset. We methodically tested a range of parameter values to identify the combination that resulted in the highest macro-averaged F1 score. This process allowed us to optimize the model's accuracy and tailor the model specifically to the nuances of the dataset.

Table 4 presents the hyperparameters of the Decision Trees, identified as the top-performing model in Section VI-B. These influential hyperparameters encompass max depth, min sample split, min sample leaf, criterion, max features, max-leaf nodes, and min impurity decrease.

VI. RESULT AND DISCUSSION

This section presents our empirical findings for machine-learning models in system metrics and health forecasting. It also provides an insightful discussion of the findings.

A. SYSTEM METRICS FORECASTING EVALUATION

This subsection details the experimental results of the system metrics forecasting, including two main categories: model comparison and hyperparameter tuning.

1) MODEL COMPARISON

Table 5 provides a detailed performance evaluation of cSysGuard compared to the selected Level-0 models across various metrics. Deep learning models, such as GRU, LSTM, and RNN, show competitive predictive performance due to their ability to capture complex temporal dependencies and nonlinear patterns in the data. In contrast, traditional models like ARIMA and SARIMA generally lag in most metrics because they rely on more straightforward, linear assumptions and are less effective at handling the variability and complexity of dynamic cloud environments. Overall, cSysGuard shows superior proficiency in most areas. Notably, in response time, it achieves a significant 2.28-fold increase in performance compared to ETS. Notably, this comparison also highlights cSysGuard's capability to surpass advanced deep learning models, such as GRU and LSTM, typically known for their strong performance in predictive

TABLE 5. RMSE-based performance comparison of cSysGuard with selected level-0 models across specific metrics.

	cSysGuard	GRU	LSTM	RNN	CNN	ARIMA	SARIMA	TCN	ETS
CPU Request (percent)	**4.793**	4.917	5.008	4.796	5.254	6.816	7.724	5.825	
Memory Request (percent)	**1.494**	1.518	1.553	1.487	1.587	2.284			
Inbound Bandwidth (MB/s)	**2778**	2795	2827	2787	2897	3658		3070	
Outbound Bandwidth (MB/s)	**2368**	2388	2416	2368	2471	3122		2632	
TPS	**0.635**	0.641	0.646	0.637	0.674	0.834	0.831	0.707	
Response Time (ms)	**640.2**	649.1	665.7	642.3	742.0	1049.2		847.5	1460.0

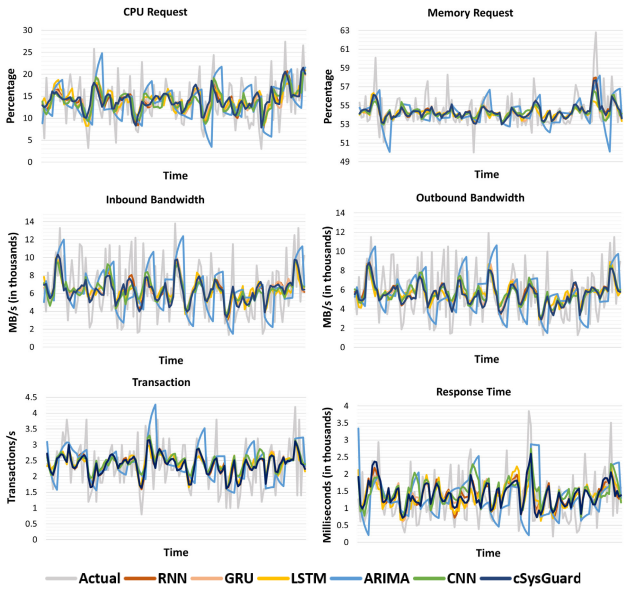

FIGURE 12. System performance metrics prediction. This figure compares cSysGuard's prediction results with those of commonly used models (GRU, RNN, LSTM, ARIMA, CNN) against actual results for CPU and memory requests, bandwidth, transaction rates, and response times.

tasks. These results demonstrate that cSysGuard effectively surpasses traditional models in certain areas, reinforcing its robustness and adaptability in diverse system-metric forecasting scenarios.

Additionally, Fig. 12 visualizes the predicted values compared to the actual values for various metrics. Each graph compares the performance of cSysGuard with commonly used models such as GRU, RNN, LSTM, ARIMA, and CNN. The visualizations clarify cSysGuard's predictive capabilities, highlighting its effectiveness in forecasting diverse system metrics. Given their alignment with performance benchmarks and the vast array of adjustable parameters, we opted for RNN and Random Forest as our Level-0 and Level-1 models in our optimization experiment.

2) HYPERPARAMETER TUNING

Table 6 lists the optimal RNN and Random Forest parameters for CPU usage prediction. Simultaneously, Table 7 presents the improved performance of each model and cSysGuard. Interestingly, the tuning process not only augmented the individual performances of the RNN and Random Forest but also enhanced the predictive capabilities of the subsequent layers in the architecture. This finding suggests a synergistic effect, where refining one system component positively influences overall efficacy, thereby underscoring the interconnected nature of the models within the system.

TABLE 6. Optimal parameters for recurrent neural networks (RNN) and random forest (RF) tuning in CPU request analysis.

Optimal RNN (Level-0) Parameters	
Parameter	**Value**
Observation Steps	64
Learning Rate	0.001
Number of Neurons in Layer 1	49
Number of Neurons in Layer 2	55
Activation Functions in Layer 1	relu
Activation Functions in Layer 2	relu
Epochs	100
Batch Size	16
Optimal RF (Level-1) Parameters	
Parameter	**Value**
Number of Estimators	545
Max Depth	27
Min Sample Split	5
Min Sample Leaf	10
Max Features	auto
Max Leaf Nodes	13
Min Impurity Decrease	0.00035269496460264014
Bootstrap	true
Criterion	friedman mse

TABLE 7. RMSE-based performance of recurrent neural networks (RNN), random forest (RF), and cSysGuard in CPU request (unit: percent).

	RNN	RF	cSysGuard
Base Performance	4.796	4.856	4.793
Level-0: Tuned RNN	**4.695**	**4.736**	**4.691**
Level-1: Tuned RF	4.695	**4.706**	**4.684**

B. SYSTEM HEALTH ASSESSMENT EVALUATION

This subsection specifically delves into the experimental results of the system health assessment, dividing them into model comparison and hyperparameter tuning.

TABLE 8. Performance of base models and tuned decision trees in system health assessment.

	Macro-avg Precision	Macro-avg Recall	Macro-avg F1 Score
Base Performance			
Decision Trees	0.8856	0.8802	0.8812
Support Vector Machine	0.8255	0.8231	0.8223
K-Nearest Neighbors	0.8372	0.7819	0.7736
Neural Networks	0.8515	0.8159	0.8060
Logistic Regression	0.8380	0.8380	0.8368
Optimal Performance			
Tuned Decision Trees	0.9012	0.8968	0.8979

TABLE 9. Optimal parameters for tuning decision trees.

Parameter	Values
Max Depth	129
Min Samples Split	2
Min Samples Leaf	1
Criterion	gini
Max Features	none
Max Leaf Nodes	637
Min Impurity Decrease	0

1) MODEL COMPARISON

Table 8 presents a comparative analysis of various base models' performance, focusing on the macro-averaged precision, recall, and F1 score. The analysis reveals that the Decision Trees outperform the other base models by achieving the highest F1 score. This performance highlights its efficiency in accurately identifying positive instances while minimizing false positives, as evidenced by its high recall and precision. Such a balanced effectiveness profile underscores the suitability of Decision Trees for reliable system health status prediction, establishing it as a robust choice for such assessments. Based on the evaluation, we focused on Decision Trees for in-depth analysis during hyperparameter tuning.

2) HYPERPARAMETER TUNING

With Decision Trees identified as the top-performing base model, Table 8 offers a comparative analysis showcasing the impact of hyperparameter tuning on model performance. In addition, Table 9 presents the optimal parameter settings of our model. The results indicate that the optimized Decision Trees outperform the original model, showing enhancements of approximately 1.56% in precision, 1.66% in recall, and 1.67% in the F1 score. It demonstrates the improved performance of its original version and other base models.

Following the hyperparameter tuning process, we embarked on a detailed analysis of the model's classification accuracy. We constructed an averaged confusion matrix, denoted as Fig. 13. It synthesized the outcomes from each iteration of cross-validation conducted with our sample

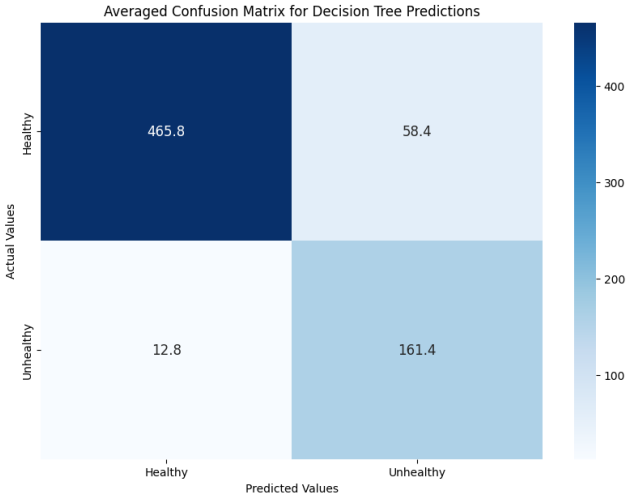

FIGURE 13. Confusion matrix visualization for system health assessment. This figure illustrates the averaged confusion matrix, capturing the predictive accuracy of server status as healthy or unhealthy.

data, offering a transparent and quantifiable measure of cSysGuard's predictive accuracy. The matrix delineates two principal classes indicative of server status: "healthy" and "unhealthy." In this matrix, columns represent predicted labels, and rows correspond to actual labels. This graphical representation is crucial for evaluating the precision and reliability of our model in distinguishing the server's operational states.

C. COMPARATIVE ANALYSIS OF DIFFERENT ENSEMBLE METHODOLOGY

Our study introduces an innovative ensemble stacking model to enhance predictive accuracy. In detail, the regression framework is a three-tiered architecture, comprising base models in the first layer, meta-models in the second, and employing a weighted averaging approach in the third. Then, we conducted a comparative analysis incorporating existing research studies alongside a novel experiment we devised, all utilizing similar ensemble modeling approaches.

Specifically, we have selected two research studies. Both employ two-tiered architectural frameworks. The first literature, by Mehmood et al. [23], proposed an ensemble model with a single meta-model (Decision Trees) in the second layer, which leverages predictions from the base models to determine the outcome. The second publication, by Kim et al. [22], introduced an ensemble model framework named CloudInsight. It incorporates a weighted average strategy in its second layer to aggregate and summarize predictions from the base models. In addition, we conducted an experiment that involves simple averaging in the third layer of cSysGuard. The study aimed to determine if a more streamlined model structure or a more straightforward aggregation method could achieve superior results without adding extra complexity. Therefore, we constructed these frameworks and applied them to our dataset to determine

TABLE 10. RMSE performance comparison: Single Meta-Model, CloudInsight, and cSysGuard (simple averaging: cSG-SA & weighted averaging: cSG-WA) across metrics.

	1 Meta-Model (Mehmood *et al.*)	Cloud Insight (Kim *et al.*)	cSG-SA	cSG-WA
CPU Request (percent)	6.6708	4.9131	4.6914	4.6845
Memory Request (percent)	2.1594	1.5442	1.4943	1.4941
Inbound Bandwidth (MB/s)	4052.5	2887.4	2778.2	2777.6
Outbound Bandwidth (MB/s)	3536.6	2462.9	2369.0	2368.7
TPS	0.9117	0.6377	0.6346	0.6348
Response Time (ms)	978.21	780.79	640.41	640.20

their effectiveness and compare them to the cSysGuard architecture.

Table 10 presents the comparative effectiveness of each methodology. Employing CloudInsight or a single meta-model does not match the performance efficacy of cSysGuard's configuration (cSG-WA). Alternatively, while using a simple averaging strategy in cSysGuard (cSG-SA) aligns closely with the current version, the critical advantage of weighted averaging is its flexibility; it enables users to tailor weight calculations based on specific needs, such as Level-1 model error sensitivity or collected error range. This flexibility enhances overall performance to favor better-performing models by fine-tuning their weights, emphasizing the importance of cSysGuard's nuanced three-tier layering strategy.

D. cSysGuard PERFORMANCE OVERHEAD

Subsequently, our analysis shifted towards assessing the impact on system performance, specifically examining the computational overhead. We experimented with cSysGuard on a host machine with a 2.9 GHz Intel Core i7-7820HK Quad-Core processor and 16GB DDR4 RAM. Table 11 showcases a comprehensive comparison, highlighting differences in key performance indicators such as average prediction and refinement times across three machine learning models (RNN, GRU, and LSTM) chosen for their substantial operational times and the cSysGuard (cSG) framework. Based on the result, cSysGuard exhibits marginally higher processing times than the selected high-performing models. However, it is essential to note that cSysGuard's processing duration is contingent upon the complexity of the underlying model it incorporates. Employing a more streamlined model within the system can reduce overall processing times for cSysGuard. Conversely, based on our findings, enhancing the host's resources can also reduce overall processing time. cSysGuard is fundamentally influenced by the characteristics

TABLE 11. Comparative analysis of prediction and refinement times (unit: seconds) for Selected ML models and cSysGuard (cSG).

	RNN	GRU	LSTM	cSG
Prediction Process	0.382	0.362	1.061	2.366
Refinement Process (Parallel Training)	74.29	132.40	141.84	152.55

of the employed models and the capabilities of its operating machine.

E. SYSTEM SPECIFICATION BIAS

As Section III-A2 outlined, our methodology deployed the system with specific configurations, including CPU type, memory capacity, and the operating system used. It is essential to acknowledge that differences in these specifications can influence the overall modeling process and its predictive outcomes. Such variations, in turn, can potentially affect the predictive accuracy and generalizability of cSysGuard.

Therefore, the study recommends that users consider adjusting cSysGuard's setup to suit their server specifications. Users can select different models and quantities, optimize parameters, fine-tune data configuration and prediction interval, or adjust Level-1 model error sensitivity. cSysGuard offers flexibility to align with users' operational goals and requirements, maximizing the advantages of ensemble learning.

VII. CONCLUSION AND FUTURE WORK

This paper introduces cSysGuard, a novel and resilient monitoring framework designed to enhance the prediction of system performance metrics and health assessments in dynamic cloud environments. To forecast performance metrics effectively, cSysGuard implements an advanced ensemble model using a stacking approach that integrates various predictors, such as the Level-0 and Level-1 models. Subsequently, the framework dynamically yields an accurate final metric prediction through a weighted averaging strategy. Furthermore, cSysGuard examines these forecasts to determine system failover through a fine-tuned classification model.

In our study, cSysGuard consistently outperformed the other standard models, encompassing traditional statistical and advanced deep learning approaches. It showcased significant proficiency, from nearly equivalent to an impressive increase of up to 2.28 times compared to conventional models. In addition to the system health assessment, it leveraged Decision Trees refined through hyperparameter tuning, achieving a macro-averaged F1 score of 89.79%. These outcomes highlight cSysGuard's effectiveness in delivering precise performance metrics and health analysis, showcasing it as a comprehensive system monitoring tool.

In future work, we intend to expand the capabilities of cSysGuard beyond its focus on detecting system-health failures based on resource sufficiency. Future iterations will

consider integrating other system failures, such as network-related issues, security breaches, and application-level errors. By addressing these additional dimensions, cSysGuard will become a more versatile framework adaptable to a broader range of scenarios and capable of providing comprehensive monitoring across various aspects of system health.

REFERENCES

[1] VMware by Broadcom. *What is Cloud Scalability?* Accessed: Jan. 14, 2024. [Online]. Available: https://www.vmware.com/topics/glossary/content/cloud-scalability.html

[2] Synopsys. *How Cloud Computing Flexibility Enables Design Changes*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.synopsys.com/cloud/insights/cloud-computing-flexibility.html

[3] Red Hat. (Dec. 21, 2023). *Stateful vs Stateless*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless

[4] Q. Zhang and R. Boutaba, "Dynamic workload management in heterogeneous cloud computing environments," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, Krakow, Poland, May 2014, pp. 1–7, doi: 10.1109/NOMS.2014.6838288.

[5] L. Rosencrance, S. Louissaint, and K. Brush. (2024). *Service-Level Agreement (SLA)*. TechTarget. Accessed: Jan. 25, 2024. [Online]. Available: https://www.techtarget.com/searchitchannel/definition/service-level-agreement

[6] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, and M. D. Assuncao, "Evaluating auto-scaling strategies for cloud computing environments," in *Proc. IEEE 22nd Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Paris, France, Sep. 2014, pp. 187–196, doi: 10.1109/MASCOTS.2014.32.

[7] Amazon Web Services. (Jul. 4, 2023). *EC2 Auto Scaling*. [Online]. Available: https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scale-based-on-demand.html

[8] Google Cloud. *Autoscaling Groups of Instances*. Accessed: Jun. 14, 2023. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler

[9] P. Wenda. (Jul. 2, 2021). *AutoScaling: Introducing Compute Engine Predictive Autoscaling*. Google Cloud. Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/blog/products/compute/introducing-compute-engine-predictive-autoscaling

[10] Google Cloud. *Scaling Based on Predictions*. Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/compute/docs/autoscaler/predictive-autoscaling

[11] N. Chairatana and R. Chawuthai, "Cloud stateless server failover prediction using machine learning on proactive system metrics," in *Proc. 18th Int. Joint Symp. Artif. Intell. Natural Lang. Process. (iSAI-NLP)*, Bangkok, Thailand, Nov. 2023, pp. 1–6, doi: 10.1109/isai-nlp60301.2023.10354585.

[12] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen, "Cost-effective cloud HPC resource provisioning by building semi-elastic virtual clusters," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Denver, CO, USA, Nov. 2013, pp. 1–12.

[13] A. A. Bankole and S. A. Ajila, "Predicting cloud resource provisioning using machine learning techniques," in *Proc. 26th IEEE Can. Conf. Electr. Comput. Eng. (CCECE)*, Regina, SK, Canada, May 2013, pp. 1–4, doi: 10.1109/CCECE.2013.6567848.

[14] N. Suksriupatham and A. Hoonlor, "Workload prediction with regression for over and under provisioning problems in multi-agent dynamic resource provisioning framework," in *Proc. 17th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Bangkok, Thailand, Nov. 2020, pp. 128–133, doi: 10.1109/JCSSE49651.2020.9268289.

[15] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Oct. 2015, doi: 10.1109/TCC.2014.2350475.

[16] K. Dmytro, T. Sergii, and P. Andiy, "ARIMA forecast models for scheduling usage of resources in IT-infrastructure," in *Proc. 12th Int. Sci. Tech. Conf. Comput. Sci. Inf. Technol. (CSIT)*, vol. 1, Lviv, Ukraine, Sep. 2017, pp. 356–360, doi: 10.1109/STC-CSIT.2017.8098804.

[17] P. Sekwatlakwatla, M. Mphahlele, and T. Zuva, "Traffic flow prediction in cloud computing," in *Proc. Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, Durban, South Africa, Nov. 2016, pp. 123–128, doi: 10.1109/ICACCE.2016.8073735.

[18] D. Lee, D. Lee, M. Choi, and J. Lee, "Prediction of network throughput using ARIMA," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Fukuoka, Japan, Feb. 2020, pp. 1–5, doi: 10.1109/ICAIIC48513.2020.9065083.

[19] S. T. Singh, M. Tiwari, and A. S. Dhar, "Machine learning based workload prediction for auto-scaling cloud applications," in *Proc. OPJU Int. Technol. Conf. Emerg. Technol. Sustain. Develop. (OTCON)*, Chhattisgarh, India, Feb. 2023, pp. 1–6, doi: 10.1109/OTCON56053.2023.10114033.

[20] J. Gao, H. Wang, and H. Shen, "Machine learning based workload prediction in cloud computing," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Honolulu, HI, USA, Aug. 2020, pp. 1–9, doi: 10.1109/ICCCN49398.2020.9209730.

[21] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2010, pp. 456–463, doi: 10.1109/CLOUDCOM.2010.65.

[22] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Forecasting cloud application workloads with CloudInsight for predictive resource management," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1848–1863, Jul. 2022, doi: 10.1109/TCC.2020.2998017.

[23] T. Mehmood, S. Latif, and S. Malik, "Prediction of cloud computing resource utilization," in *Proc. 15th Int. Conf. Smart Cities, Improving Quality Life Using ICT IoT (HONET-ICT)*, Islamabad, Pakistan, Oct. 2018, pp. 38–42, doi: 10.1109/HONET.2018.8551339.

[24] S. Li, J. Bi, H. Yuan, M. Zhou, and J. Zhang, "Improved LSTM-based prediction method for highly variable workload and resources in clouds," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Toronto, ON, Canada, Oct. 2020, pp. 1206–1211, doi: 10.1109/SMC42975.2020.9283029.

[25] S. Bhagavathiperumal and M. Goyal, "Workload analysis of cloud resources using time series and machine learning prediction," in *Proc. IEEE Asia–Pacific Conf. Comput. Sci. Data Eng. (CSDE)*, Melbourne, VIC, Australia, Dec. 2019, pp. 1–8, doi: 10.1109/CSDE48274.2019.9162385.

[26] P. N, S. Chhetri, S. R. Kini, G. G. Mishra, and S. Kumar, "Resource provisioning in cloud using ARIMA and LSTM technique," in *Proc. IEEE 2nd Mysore Sub Sect. Int. Conf. (MysuruCon)*, Mysuru, India, Oct. 2022, pp. 1–7, doi: 10.1109/MysuruCon55714.2022.9972689.

[27] W. Chen, C. Lu, K. Ye, Y. Wang, and C.-Z. Xu, "RPTCN: Resource prediction for high-dynamic workloads in clouds based on deep learning," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Portland, OR, USA, Sep. 2021, pp. 59–69, doi: 10.1109/Cluster48925.2021.00038.

[28] M. Hassan, H. Chen, and Y. Liu, "DEARS: A deep learning based elastic and automatic resource scheduling framework for cloud applications," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun., Big Data Cloud Comput., Social Comput. Netw., Sustain. Comput. Commun. (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Melbourne, VIC, Australia, Dec. 2018, pp. 541–548, doi: 10.1109/BDCLOUD.2018.00086.

[29] A. Radwan, M. Amarneh, H. Alawneh, H. I. Ashqar, A. AlSobeh, and A. A. A. R. Magableh, "Predictive analytics in mental health leveraging LLM embeddings and machine learning models for social media analysis," *Int. J. Web Services Res.*, vol. 21, no. 1, pp. 1–22, Feb. 2024, doi: 10.4018/ijwsr.338222.

[30] J. H. Challis, "Signal processing," in *Experimental Methods in Biomechanics*. Cham, Switzerland: Springer, 2021, doi: 10.1007/978-3-030-52256-8_4.

[31] L. D. Paarmann, "Butterworth filters," in *Design and Analysis of Analog Filters* (The International Series in Engineering and Computer Science), vol. 617. Boston, MA, USA: Springer, 2003, doi: 10.1007/0-306-48012-3_3.

[32] S. Satpathy. (Nov. 17, 2023). *SMOTE for Imbalanced Classification With Python*. Analytics Vidhya. Accessed: Jan. 28, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques

[33] R. Dey and R. Mathur, "Ensemble learning method using stacking with base learner: A comparison," in *Proc. Int. Conf. Data Anal. Insights*, vol. 727. Singapore: Springer, 2023, pp. 159–169, doi: 10.1007/978-981-99-3878-0_14.

[34] A. C. Harvey, "ARIMA models," in *The New Palgrave* (Time Series and Statistics), J. Eatwell, M. Milgate, and P. Newman, Eds. London, U.K.: Palgrave Macmillan, 1990, doi: 10.1007/978-1-349-20865-4_2.

[35] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting With Exponential Smoothing: The State Space Approach*. Cham, Switzerland: Springer, 2008, doi: 10.1007/978-3-540-71918-2.

[36] G. James, D. Witten, T. Hastie, and R. Tibshirani, "Linear regression," in *An Introduction to Statistical Learning* (Springer Texts in Statistics). New York, NY, USA: Springer, 2021, doi: 10.1007/978-1-0716-1418-1_3.

[37] Y. Liu, Y. Wang, and J. Zhang, "New machine learning algorithm: Random forest," in *Information Computing and Applications* (Lecture Notes in Computer Science), vol. 7473, B. Liu, M. Ma, and J. Chang, Eds. Berlin, Germany: Springer, 2012, doi: 10.1007/978-3-642-34062-8.

[38] S. Skansi, "Feedforward neural networks," in *Introduction to Deep Learning* (Undergraduate Topics in Computer Science). Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-73004-2_4.

[39] S. Tripathy and R. Singh, "Convolutional neural network: An overview and application in image classification," in *Proc. 3rd Int. Conf. Sustain. Comput.*, vol. 1404. Singapore: Springer, 2022, doi: 10.1007/978-981-16-4538-9_15.

[40] C. Lea, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks: A unified approach to action segmentation," in *Proc. Comput. Vis. ECCV Workshops*, in Lecture Notes in Computer Science, vol. 9915. Cham, Switzerland: Springer, 2016, pp. 47–54.

[41] S. A. Marhon, C. J. F. Cameron, and S. C. Kremer, "Recurrent neural networks," in *Handbook on Neural Information Processing* (Intelligent Systems Reference Library), vol. 49, M. Bianchini, M. Maggini, and L. Jain, Eds. Berlin, Heidelberg: Springer, 2013, doi: 10.1007/978-3-642-36657-4_2.

[42] S. Kostadinov. (Dec. 16, 2017). *Understanding GRU Networks*. Towards Data Sci. Accessed: Jan. 17, 2024. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[43] K. Smagulova and A. P. James, "Overview of long short-term memory neural networks," in *Deep Learning Classifiers With Memristive Networks* (Modeling and Optimization in Science and Technologies), vol. 14, A. James, Ed. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-14524-8_11.

[44] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/bf00116251.

[45] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, Jul. 1995, doi: 10.1007/BF00994018.

[46] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.

[47] J. S. Cramer, "The origins and development of the logit model," in *Logit Models From Economics and Other Fields*. Cambridge, U.K.: Cambridge Univ. Press, 2003, pp. 149–157, doi: 10.1017/CBO9780511615412.010.

[48] B. Mer, J. Reinhardt, and M. T. Strickland, "Neural networks: An introduction," in *Physics of Neural Networks*, 2nd ed. Berlin, Germany: Springer, 1995, doi: 10.1007/978-3-642-57760-4.

[49] *Digital Ocean*. Accessed: Jan. 14, 2024. [Online]. Available: https://www.digitalocean.com

[50] Google Cloud. *What is a Virtual Machine?* Accessed: Jan. 14, 2024. [Online]. Available: https://cloud.google.com/learn/what-is-a-virtual-machine

[51] *Prometheus*. Accessed: Jun. 15, 2023. [Online]. Available: https://prometheus.io

[52] *Zabbix*. Accessed: May 26, 2023. [Online]. Available: https://www.zabbix.com

[53] *MDN Web Docs: HTTP Response Status Codes*. Accessed: Jun. 15, 2023. [Online]. Available: https://developer.mozilla.org/en-U.S./docs/Web/HTTP/Status

[54] *Statsmodels*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.statsmodels.org

[55] *TensorFlow*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.tensorflow.org

[56] *Scikit-learn*. Accessed: Jan. 15, 2023. [Online]. Available: https://www.scikit-learn.org

[57] N. Chairatana and R. Chawuthai, Jan. 31, 2024, "Cloud stateless system performance metrics and status," IEEE Dataport, doi: 10.21227/8wf2-2y40.

[58] S. Turney. (Jun. 22, 2023). *Pearson Correlation Coefficient*. Scribbr. Accessed: Jan. 15, 2024. [Online]. Available: https://www.scribbr.com/statistics/pearson-correlation-coefficient

[59] B. Saji. (2021). *Elbow Method for Finding the Optimal Number of Clusters in K-Means*. Analytics Vidhya. Accessed: Jan. 15, 2024. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/01/in-depth-intuition-of-k-means-clustering-algorithm-in-machine-learning

[60] D. Schumacher. *Understanding Weighted Average*. SERP AI. Accessed: Jan. 15, 2024. [Online]. Available: https://serp.ai/weighted-average

[61] ScienceDirect. *Integer Overflow*. Accessed: Jan. 15, 2024. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/integer-overflow

[62] W. Wang. (2020). *Bayesian Optimization Concept Explained in Layman Terms*. Towards Data Sci. Accessed: Jan. 15, 2024. [Online]. Available: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f

[63] J. Brownlee. (2020). *A Gentle Introduction to Imbalanced Classification*. Mach. Learn. Mastery. Accessed: Jun. 15, 2023. [Online]. Available: https://machinelearningmastery.com/what-is-imbalanced-classification

[64] A. Tariq. (2023). *What is the Difference Between Micro and Macro Averaging?* Educative. Accessed: Jun. 15, 2023. [Online]. Available: https://www.educative.io/answers/what-is-the-difference-between-micro-and-macro-averaging

NUTT CHAIRATANA received the B.Eng. degree in computer innovation engineering from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 2022, where he is currently pursuing the M.Eng. degree in robotics and computational intelligence systems with the Department of Robotics and AI Engineering.

From 2022 to 2023, he was a Software Engineer with RentSpree. He is a Machine Learning Engineer with Kasikorn Business-Technology Group, Thailand.

RATHACHAI CHAWUTHAI received the B.Eng. degree in computer engineering from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 2006; the M.Eng. degree in information management from Asian Institute of Technology, Pathum Thani, Thailand, in 2012; and the Ph.D. degree in informatics from SOKENDAI and National Institute of Informatics, Kanagawa, Japan, in 2016.

From 2006 to 2010, he was a Software Engineer with Thomson Reuters. From 2012 to 2013, he was a Senior Software Engineer with Punsarn Asia. From 2013 to 2016, he was a Research Assistant with the National Institute of Informatics. He is currently an Assistant Professor with the King Mongkut's Institute of Technology Ladkrabang.