IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Publication date xxxx 00, 0000, final version XXXX 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2024.0429000

Optimizing LSTM Forecasting Models for AIOps via Manta Ray Foraging Algorithm

WESLLEY SANTANA ROSALEM[1], (Membro, IEEE), FABRÍCIO STEINLE AMOROSO[1], RENATO DIAS DE SOUZA[1], FELIPE MAHLOW[1], KELTON AUGUSTO PONTARA[1], (Senior Member, IEEE)

[1]São Paulo State University (UNESP), School of Sciences, 17033-360 Bauru, São Paulo, Brazil

Corresponding author: Weslley Santana Rosalem (e-mail: w.rosalem@unesp.br).

ABSTRACT This study introduces a novel approach for hyperparameter optimization of Long Short-Term Memory (LSTM) networks using the Manta Ray Foraging Optimization (MRFO) algorithm, applied to resource usage forecasting in Artificial Intelligence for IT Operations (AIOps). The method was evaluated on two real-world datasets: the publicly available Google Cluster Traces 2019 and a Prometheus-based memory usage dataset collected from a Linux server by the authors. MRFO outperformed Grid Search, Random Search, Particle Swarm Optimization (PSO), and a fixed baseline configuration. On the Google Cluster Traces dataset, MRFO achieved a 6.8% reduction in Mean Absolute Error (MAE) and 13.6% in Mean Absolute Percentage Error (MAPE); on the Prometheus dataset, it reduced MAE by 6.1% and Root Mean Squared Error (RMSE) by 9.0% compared to a proposed LSTM baseline. These improvements enable proactive identification of potential downtimes and support resource allocation strategies that reduce service disruptions and improve cost predictability in cloud infrastructures. The proposed pipeline is implemented in TensorFlow and is fully reproducible and extensible to multivariate forecasting. Future directions include hybrid meta-heuristics, anomaly detection, and root cause correlation to advance autonomous data center management.

INDEX TERMS AIOps, Hyperparameter Optimization, LSTM, MRFO, Resource Forecasting, Time Series Forecasting

I. INTRODUCTION

Accurate prediction of computational resources, such as memory usage, CPU, and network, is essential for the efficient management of data centers in modern cloud computing environments [1]. Artificial Intelligence for IT Operations (AIOps) integrates advanced machine learning techniques and automation to forecast fluctuations in resource demand, enabling proactive allocation, cost optimization, and greater system reliability [2]. Inadequate resource management in IT environments can lead to significant increases in operational costs [3], due to factors such as overprovisioning, underutilization, and lack of visibility into actual infrastructure usage [4]. Additionally, IT service downtime can result in substantial financial losses, directly affecting business continuity and customer satisfaction [5]. In this context, approaches that combine predictive analytics with operations automation, such as those offered by AIOps solutions, have been explored to mitigate these risks and optimize operational efficiency [6].

Time series data from data centers exhibit complex dynamics, including seasonality (e.g., daily or weekly usage patterns), abrupt peaks (e.g., during traffic spikes), and nonlinear trends (e.g., due to application scalability) [7]. These characteristics challenge traditional statistical models, such as autoregressive integrated moving average (ARIMA), which struggle with nonlinear dependencies [8]. Modern approaches tackle the problem using advanced machine learning techniques [9]. Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), are particularly well-suited to capture long-term temporal dependencies [10]. LSTMs mitigate the vanishing gradient problem present in standard RNNs, allowing them to learn patterns across extensive time horizons, which is crucial for predicting resource demands based on days- or weeks-long usage histories. However, LSTM performance is highly sensitive to the choice of hyperparameters, such as the num-

ber of units, learning rate, batch size, dropout rate, and regularization strength [11]. Manual tuning or grid-based methods like Grid Search become computationally infeasible for high-dimensional hyperparameter spaces, often requiring thousands of training iterations [12].

To address this challenge, meta-heuristic algorithms have gained attention for their ability to efficiently explore complex and non-convex spaces [13]. The Manta Ray Foraging Optimization (MRFO) algorithm [14], inspired by the foraging behaviors of manta rays, offers a robust balance between global exploration and local exploitation. MRFO employs three distinct strategies—chain foraging, cyclone foraging, and somersault foraging—to navigate the hyperparameter space, dynamically adapting to avoid local optima [14]. Compared to other meta-heuristics, such as Particle Swarm Optimization (PSO), Genetic Algorithms (GA), Butterfly Optimization Algorithm (BOA), and Grey Wolf Optimizer (GWO), MRFO presents several advantages [15]. In contrast, MRFO's diverse foraging strategies enable a more effective balance between exploration and exploitation, with empirical studies showing faster convergence rates than PSO in similar problems [16].

This work presents four main contributions to the field of AIOps and machine learning model optimization. First, it proposes the application of the MRFO algorithm for the automated tuning of LSTM hyperparameters in resource forecasting tasks for IT environments, addressing a significant gap in the literature on optimization methods for automated IT operations. Second, it presents the creation of a novel dataset for system resource monitoring, collected directly from a real server infrastructure using Prometheus[1], thus contributing new, real-world data for the research community. Third, it develops a reproducible training and evaluation pipeline based on TensorFlow, validated on real datasets—specifically the Google Cluster Traces 2019 [17], [18] and the aforementioned Prometheus-based dataset[2]—ensuring the practical applicability of the solution in operational scenarios. Finally, it conducts a detailed analysis of seasonal patterns in the data, such as daily seasonality with frequency $f = 1/48$, and investigates the sensitivity of model hyperparameters, offering valuable insights for the efficient deployment of LSTMs in AIOps contexts.

The article is organized as follows: Section II presents a review of related work, covering the use of LSTMs for time series forecasting, the application of meta-heuristics for hyperparameter optimization, challenges and frameworks in AIOps, and the main gaps identified in the literature. Section III establishes the theoretical foundations, covering the concepts of AIOps and time series, the operation of LSTM networks, the Manta Ray Foraging Optimization (MRFO) algorithm, and the evaluation metrics used in this study. Section IV describes the adopted methodology, detailing the datasets used, problem formulation, MRFO optimization setup, baseline methods, and experimental environment. Section V presents the obtained results, including a critical analysis of method performance, sensitivity to variables, observed seasonal patterns, and computational cost. Finally, Section VI concludes the article, summarizing the main contributions and discussing future directions, such as the development of hybrid meta-heuristics, the extension to multivariate forecasting, and the implementation of scalable AIOps approaches.

[1]https://prometheus.io/

[2]https://github.com/weslleyrosalem/UNESP_Paper-MRFO/tree/main/data

II. RELATED WORK

A. LSTM AND TIME SERIES FORECASTING

LSTM networks have been widely used in time series forecasting within the AIOps context due to their ability to capture long-term dependencies—an essential feature for modeling complex temporal patterns. This architecture was originally developed to overcome the limitations of traditional recurrent neural networks (RNNs), such as the vanishing gradient problem, allowing relevant information to be retained and reused over extended temporal sequences [10]. Experimental evidence indicates that LSTM models outperform classical statistical approaches, such as ARIMA, especially in scenarios characterized by nonlinearity and high data complexity [19]. This advantage stems from the LSTM's ability to effectively represent subtle temporal dynamics. In addition, systematic reviews of deep learning architectures confirm the robustness of LSTM models, particularly in contexts involving noisy and unstable data [20]. Advanced LSTM architectures, such as encoder-decoder models with attention mechanisms, have proven effective for workload forecasting in heterogeneous cloud environments. By assigning dynamic weights to historical data, these networks enhance prediction accuracy in mixed workload scenarios, such as continuous services and batch tasks. Complementary techniques, such as splitting long sequences into subwindows (scroll prediction), further help reduce accumulated errors in long-term forecasting, reinforcing the feasibility of adaptive LSTM models for resource management in dynamic environments [21]. More recently, architectures like xLSTM-Time have even outperformed transformer-based models in long-term forecasting tasks, consolidating the central role of LSTMs in modern time series solutions [22].

B. METAHEURISTICS FOR HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization is one of the main challenges in training deep learning models, such as LSTM networks, whose performance strongly depends on the appropriate selection of these parameters. In high-dimensional search spaces, studies indicate that the Random Search method is more efficient than Grid Search, achieving comparable performance with a significantly smaller number of evaluations—up to 90% fewer in conventional neural networks and approximately 35% fewer in deep architectures [12]. Strategies based on metaheuristic algorithms have been explored as effective alternatives. One example is the proposal of a

satellite network traffic forecasting model using enhanced GRU networks with attention mechanisms, in which the Particle Swarm Optimization (PSO) algorithm was applied for hyperparameter optimization. The approach outperformed traditional models, such as conventional GRU, SVM, and FARIMA, with reductions in prediction errors of 26.9%, 37.2%, and 57.8%, respectively [23]. In another proposal, an LSTM model was developed and optimized with an improved genetic algorithm (IGA) for short-term traffic flow prediction, resulting in significant accuracy gains compared to conventional LSTM, although with a considerable increase in computational complexity due to the evolutionary tuning process [24]. A comparative analysis among hyperparameter optimization techniques for neural networks showed that the genetic algorithm achieved performance similar to Random Search but with greater flexibility in complex search spaces. However, all evaluated methods (Grid Search, Random Search, and Genetic Algorithm) required high execution times, with Grid Search being the slowest due to its exhaustive nature [25]. In the field of bio-inspired approaches, the use of the Grey Wolf Optimizer (GWO) demonstrated good results in engineering problems, but it shows limitations in exploring high-dimensional search spaces, such as those involved in hyperparameter tuning [26]. More recent alternatives, such as the Whale Optimization Algorithm (WOA), have shown faster convergence and require fewer function evaluations compared to PSO in several standardized benchmarks, as evidenced by performance tests and convergence curves presented in the study [27], while the Ant Colony Optimization (ACO) offers another promising strategy based on collective behavior [28]. These approaches reinforce the importance of advanced optimization techniques in improving the performance of predictive models applied to the AIOps context.

C. CHALLENGES AND AIOPS FRAMEWORKS

AIOps faces critical challenges in IT operations, such as resource forecasting, anomaly detection, and incident management. These capabilities are essential for addressing the increasing complexity and volume of data in modern IT environments [29]. The need for proactive forecasting to reduce downtime is widely recognized, given that service interruptions can result in significant costs for companies.

In multi-cloud environments, AIOps encounters additional challenges, such as data heterogeneity and latency constraints. Federated learning has been explored as a solution in AIOps, enabling collaborative model training across distributed data centers while preserving privacy. The study demonstrated that federated learning can achieve up to 95% of the accuracy of centralized models while reducing inter-cloud data transfer by 87% [30].

III. THEORETICAL FOUNDATIONS

A. AIOPS AND TIME SERIES FORECASTING

AIOps (Artificial Intelligence for IT Operations) refers to the application of artificial intelligence techniques, such as machine learning and predictive models, to automate and enhance IT operations management. In data centers and cloud environments, where large volumes of operational data are continuously generated, these techniques enable proactive monitoring, anomaly detection, and real-time decision-making [31]. Historically, the analysis of temporal data in computing systems has been conducted using classical statistical time series methods. One widely used approach is additive decomposition, in which a time series x_t is represented as the sum of three distinct components [32]:

$$x_t = T_t + S_t + R_t, \tag{1}$$

where T_t represents the long-term trend, S_t the seasonal component associated with periodic patterns, and R_t the residual term that captures random variations and noise. This decomposition allows for an interpretable analysis of the series structure, often used in preprocessing and diagnostic steps. Moreover, the presence of seasonal patterns can be investigated using the periodogram, which estimates the power spectral density of the time series [33]. Relevant peaks at specific frequencies indicate dominant periodicities — for instance, a peak at $f = 1/48$ (for series with a 30-minute interval) signals daily seasonality.

Although these tools are useful for explicitly characterizing time series, they are not directly used in this work. Instead, we adopt an approach based on recurrent neural networks of the LSTM type, which are capable of automatically capturing implicit trends and seasonal patterns in the data, without the need for formal decomposition. Thus, the concepts presented here aim to contextualize the problem within the classical time series framework, reinforcing the motivations for adopting more flexible and adaptive methods such as LSTMs.

B. LSTM NETWORKS

LSTM networks represent a major advancement for AIOps applications, where the continuous analysis of temporal data streams is essential to maintaining stability and efficiency in complex IT systems. Figure 1 provides a detailed view of the internal structure of an LSTM cell, illustrating how sequential information is processed in a controlled and adaptive manner. At the center of the LSTM cell is the cell state, denoted as c_t, which acts as a long-term memory line capable of carrying information across many time steps [34], [35]. This design enables the model to retain critical historical patterns, such as seasonal workload fluctuations or recurring system behaviors, even as new data is introduced. The figure shows this as a horizontal path that runs through the cell, intersecting with gates that determine how information is updated or forgotten.

The forget gate, represented in the figure by the leftmost sigmoid unit (σ), receives the previous hidden state h_{t-1} and the current input x_t, and decides which parts of the previous cell state c_{t-1} should be discarded [34], [35]. This mechanism is especially relevant in AIOps contexts, as it allows the model to ignore spurious or outdated signals,

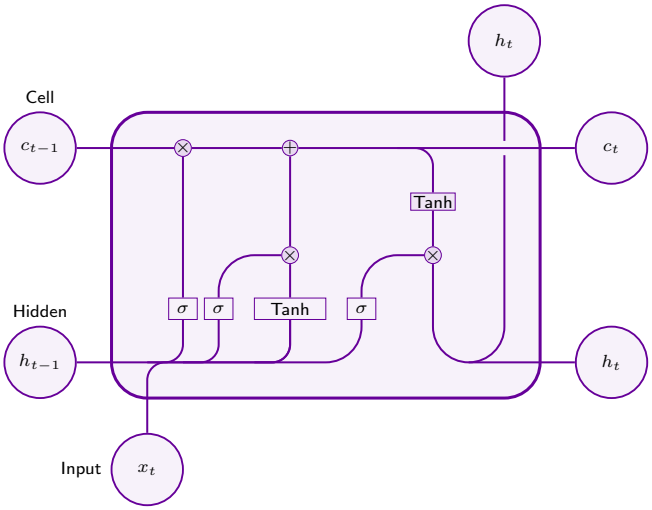

FIGURE 1. Internal architecture of an LSTM cell. It illustrates how the previous cell state (c_{t-1}) and hidden state (h_{t-1}), together with the current input (x_t), interact through a series of gates (forget, input, and output) to produce the updated cell state (c_t) and hidden state (h_t).

such as isolated errors or transient load spikes, that are not useful for future predictions. Next, the input gate combines a sigmoid activation with a tanh transformation to assess which new information should be added to the cell state. The current input and previous hidden state are again processed to determine both the values to update and their magnitude. This process allows the LSTM to learn from recent incidents, refining its internal representation of what constitutes normal or anomalous behavior in the system.

After these operations, the cell state is updated by combining the retained content from the past with the selected new information. The output gate, located at the right side of the cell in the figure, applies a final sigmoid function to the same input and hidden state pair, and filters the now-updated cell state through a tanh function to generate the new hidden state h_t. This hidden state is the external output of the cell and also serves as a memory for the next time step [34], [35]. In the figure, x_t denotes the input at time t, which could correspond to metrics, logs, or alert data in an AIOps pipeline. The elements labeled h_{t-1} and c_{t-1} represent the hidden and cell states from the previous time step. After passing through the internal mechanisms of the LSTM, the model yields c_t and h_t, shown in the diagram as respectively [34], [35].

By coordinating these internal components, the LSTM cell maintains a balance between memory retention and adaptability. This capability is especially powerful in environments like AIOps, where the model must not only react to new data but also identify latent dependencies across time. For instance, it may detect that a gradual increase in database errors earlier in the day correlates with latency spikes observed hours later, uncovering relationships that traditional models would miss. Beyond anomaly detection, this ability enables proactive forecasting. An LSTM can recognize subtle signs that resemble previously observed failure patterns, allowing it to anticipate issues before they escalate. When connected to orchestration tools, such models can even trigger automated responses, such as resource reallocation or service isolation, in response to predictive signals.

C. MANTA RAY FORAGING OPTIMIZATION (MRFO)

The Manta Ray Foraging Optimization (MRFO) algorithm is inspired by three natural foraging strategies of manta rays: chain foraging, cyclone foraging, and somersault foraging [14], [34]. In the chain foraging mechanism, manta rays align head-to-tail and move cooperatively toward food sources. Each individual updates its position based on the current global best and the agent in front of it. Cyclone foraging simulates the spiral swimming pattern of manta rays around plankton-rich areas, allowing the algorithm to explore local regions more intensively. Somersault foraging introduces random large jumps around the best-known solution, enhancing global search capabilities and avoiding premature convergence. A mathematical formulation is given by [14], and is presented below.

The mathematical model of chain foraging is given by:

$$x_i^d(t+1) = \begin{cases} x_i^d(t) + r \cdot (x_{\text{best}}^d(t) - x_i^d(t)) \\ \quad +\alpha \cdot (x_{\text{best}}^d(t) - x_i^d(t)), & i = 1 \\ x_i^d(t) + r \cdot (x_{i-1}^d(t) - x_i^d(t)) \\ \quad +\alpha \cdot (x_{\text{best}}^d(t) - x_i^d(t)), & i = 2, \ldots, N \end{cases} \tag{2}$$

in this formulation, $x_{\text{best}}^d(t)$ denotes the best-known solution in the d-th dimension at iteration t, and $x_{i-1}^d(t)$ is the position of the previous individual in the population. The term r is a random number uniformly drawn from the interval $[0, 1]$, and α is a learning coefficient that controls the degree of attraction toward the best solution. For the first individual ($i = 1$), the movement depends solely on the global best, while subsequent individuals ($i = 2, \ldots, N$) are additionally influenced by their immediate predecessor, modeling cooperative motion in a chain formation.

The cyclone foraging phase is modeled as:

$$x_i^d(t+1) = \begin{cases} x_{\text{rand}}^d + r \cdot (x_{\text{rand}}^d - x_i^d(t)) \\ \quad +\beta \cdot (x_{\text{rand}}^d - x_i^d(t)), & i = 1 \\ x_{\text{rand}}^d + r \cdot (x_{i-1}^d(t) - x_i^d(t)) \\ \quad +\beta \cdot (x_{\text{rand}}^d - x_i^d(t)), & i = 2, \ldots, N \end{cases} \tag{3}$$

Here, x_{rand}^d is a randomly generated point in the search space for dimension d, and β is a scalar coefficient that controls the strength of the spiral movement around this random target. The term r again introduces stochasticity. For the first agent, the update is based on the difference from the random point alone, while for the others, the influence of the previous individual is also included. This mechanism emulates the spiral convergence of manta rays toward prey and facilitates local exploitation around new regions.

Somersault foraging is represented by:

$$x_i^d(t+1) = x_i^d(t) + S \cdot (r_2 \cdot x_{\text{best}}^d - r_3 \cdot x_i^d(t)), \quad (4)$$

where S is the somersault factor, typically set to 2, and r_2, r_3 are two independent random numbers uniformly drawn from $[0, 1]$. This equation enables sudden, long-range jumps in the direction of the global best solution, which helps diversify the search and increases the probability of escaping local optima. The magnitude of the jump is adaptively scaled by both the distance to the best position and random variability.

Together, these three strategies allow MRFO to balance exploration and exploitation in the search space, leading to effective global optimization.

D. EVALUATION METRICS

This study evaluates forecasting performance using four metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Symmetric Mean Absolute Percentage Error (SMAPE). RMSE penalizes large errors due to its quadratic nature and is sensitive to outliers: $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}$. MAE provides a linear error measure, offering robustness: $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$. MAPE normalizes errors by the actual values but becomes unstable as $y_i \to 0$: $\text{MAPE} = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$. SMAPE addresses this by symmetrizing the denominator, making it more reliable for AIOps datasets with near-zero values: $\text{SMAPE} = \frac{200\%}{N}\sum_{i=1}^{N}\frac{|\hat{y}_i - y_i|}{|\hat{y}_i| + |y_i|}$.

IV. METHODOLOGY

A. DATASETS

Two real-world datasets were used in this study. The first dataset, referred to as Google Cluster Traces 2019, consists of 30 days of memory usage data captured from a Google Borg cell. The data is aggregated in 30-minute intervals and was obtained from [17], [18]. To ensure comparability and numerical stability, the data was normalized to the range [0,1] using a logarithmic transformation. Autocorrelation analysis revealed strong daily seasonality, characterized by a lag of 48, corresponding to a frequency of $f = 1/48$.

The second dataset was created by the authors and is based on real memory usage metrics collected from a production Linux server over a 7-day period. These metrics, extracted via Prometheus, were resampled at 30-minute intervals. This dataset serves to validate our observations in a real-world operational context. For interpretability, memory usage values were converted into megabytes (MB), resulting in a dataset with an average of 802.25 MB and a standard deviation of 68.33 MB. Like the first dataset, daily seasonality was evident and statistically significant, with an autocorrelation lag of 48 and a correlation coefficient of $r = 0.65$.

To address scale differences between the datasets, the following preprocessing transformation was applied to each time point:

$$x'_t = \frac{\log(1 + x_t) - \min(\log(1 + X))}{\max(\log(1 + X)) - \min(\log(1 + X))}. \quad (5)$$

This transformation ensures that the data are mapped to a uniform scale while preserving the relative magnitudes of the original values. Outliers were identified using the standard 3-sigma rule, defined as data points where $|x_t - \bar{x}| > 3\sigma$, and were subsequently imputed using cubic spline interpolation to maintain temporal continuity. Additionally, for the Prometheus dataset, which contained up to 20% missing values, interpolation was used to fill the gaps, ensuring that the temporal structure and statistical properties of the series were preserved for downstream analysis.

B. PROBLEM FORMULATION

The prediction task consists of forecasting the next time step resource usage value (x_{t+1}) given a sequence of past observations $\mathbf{x}_t = [x_{t-n_{\text{steps}}+1}, \ldots, x_t]$. The model is trained to minimize the Mean Absolute Error (MAE) between the predicted and actual values. To assess the influence of temporal context on predictive performance, different time window lengths were tested, specifically $n_{\text{steps}} \in \{24, 48, 72, 96\}$, corresponding to time spans of 12, 24, 36, and 48 hours respectively (considering a 30-minute sampling interval).

The Long Short-Term Memory (LSTM) network architecture adopted in this study consists of two stacked LSTM layers, each configured with a number of units chosen from the set $\{64, 128, 192, 256\}$. To mitigate overfitting and improve generalization, both L2 regularization and dropout were employed. The L2 penalty coefficient was selected from the values $\{10^{-6}, 10^{-5}, 10^{-4}\}$, while dropout rates of $\{0.1, 0.25, 0.4\}$ were evaluated. The input to the model consists of sequences with a fixed length defined by the chosen value of n_{steps}, and the output is a single dense layer responsible for predicting the target value at the next time step.

C. MRFO OPTIMIZATION

Hyperparameter tuning was performed using the Manta Ray Foraging Optimization (MRFO) algorithm. The search space encompassed six key hyperparameters of the LSTM model. These include the number of units in the LSTM layers, ranging from 64 to 256 (treated as integers), and the learning rate, sampled from a log-uniform distribution within the interval $[10^{-4}, 10^{-2}]$. The batch size was treated as a discrete parameter, with candidate values {16, 32, 64}. The total number of training epochs was varied between 10 and 100, while dropout rates were sampled from a continuous interval between 0.1 and 0.4. Finally, L2 regularization coefficients were drawn from a log-uniform distribution spanning $[10^{-6}, 10^{-4}]$. This range of values was selected to provide sufficient flexibility in controlling overfitting, training stability, and convergence speed.

The MRFO algorithm was configured with a population size of $N = 10$ agents and a maximum number of $T_{\max} = 20$ iterations per optimization run. The search dynamics of MRFO are governed by a combination of three behavioral strategies inspired by manta ray foraging patterns. Specifically, 40% of the agents followed the chain foraging

strategy, 40% adopted the cyclone foraging behavior, and the remaining 20% utilized somersault foraging to perform global exploration. The algorithmic parameters associated with these strategies were set as $\alpha = 1.5$, $\beta = 1.0$, and an initial somersault coefficient $\gamma_0 = 2$, as suggested in prior empirical studies. These values provide a balanced trade-off between convergence efficiency and solution diversity in high-dimensional hyperparameter spaces.

D. BASELINES

To evaluate the effectiveness of the MRFO-based hyperparameter optimization, we compared its performance against several baseline strategies commonly used in model selection. The first baseline consists of a fixed configuration defined by the following parameters: $n_{\text{steps}} = 48$, number of LSTM units set to 100, dropout rate $dr = 0.3$, and L2 regularization coefficient $l2 = 10^{-4}$. This configuration serves as a reference point for assessing improvements introduced by adaptive tuning methods.

In addition to the fixed baseline, we implemented an exhaustive Grid Search over a predefined parameter space to identify the best-performing combination through brute-force evaluation. As another standard alternative, Random Search was performed by sampling 120 configurations uniformly from the search space, offering a computationally cheaper yet competitive strategy compared to exhaustive search.

We also included Particle Swarm Optimization (PSO) as a metaheuristic baseline. The PSO algorithm was configured with a swarm of 30 particles and executed over 10 iterations, allowing the population to explore the solution space through velocity and position updates guided by individual and collective experience. These comparative strategies provide a diverse spectrum of approaches—ranging from deterministic to stochastic, and from exhaustive to nature-inspired—for benchmarking the MRFO's optimization performance.

E. EXPERIMENTAL SETUP

All experiments were conducted using Google Colab Pro[3], equipped with a high-performance NVIDIA $A100$ GPU featuring 40 GB of dedicated VRAM and 80 GB of system RAM. This environment provided sufficient computational power for training deep neural networks and performing optimization tasks efficiently. The software stack was built using Python 3.11 and included the following key libraries: TensorFlow 3.10 [36] for implementing and training LSTM models, NumPy 2.0.2 [37] for numerical operations, Pandas 2.2.2 [38] for data manipulation, and Scikit-learn 1.5.2 [39] for utility functions such as preprocessing and evaluation metrics.

To maintain temporal coherence in the time series, the datasets were split into three partitions following chronological order: 70% of the data was used for training, 10% for validation, and the remaining 20% for testing. This approach avoids data leakage and ensures realistic performance evaluation. Each hyperparameter optimization method—MRFO, PSO, Grid Search, and Random Search—was executed once, relying on internal validation to guide the search process. In contrast, the fixed baseline configuration was evaluated across 20 independent runs using random seeds ranging from 42 to 61. This repeated execution was applied only to the Google Cluster Trace dataset to account for stochasticity in model initialization and training, providing a robust estimate of baseline variability.

[3]https://colab.research.google.com/

V. RESULTS AND DISCUSSION

A. PERFORMANCE EVALUATION

The performance of the different hyperparameter optimization methods was assessed on both datasets using four common regression metrics: RMSE, MAE, MAPE, and SMAPE. Lower values indicate better predictive performance for all metrics. Tables 1 and 2 present the detailed results for the Google Cluster Trace and Prometheus datasets, respectively.

TABLE 1. Performance comparison on the Google Cluster Trace dataset using RMSE, MAE, MAPE, and SMAPE.

Method	RMSE	MAE	MAPE (%)	SMAPE (%)
Baseline	0.0743	0.0585	11.20	10.89
Grid Search	0.0769	0.0591	10.95	10.35
Random Search	0.0789	0.0608	11.00	10.62
PSO	0.0821	0.0634	11.67	11.07
MRFO	**0.0696**	**0.0545**	**9.68**	**9.49**

As shown in Table 1, the MRFO algorithm achieved the best results across all four metrics when applied to the Google Cluster Trace dataset. Specifically, MRFO reduced the MAE by approximately 6.8% and the MAPE by 13.6% compared to the fixed baseline. It also outperformed the widely-used Grid Search and Random Search techniques, which achieved similar but inferior results. PSO, while competitive, presented the worst performance among the optimization-based methods, suggesting potential limitations in its exploration-exploitation balance for this task. Notably, the improvements observed in SMAPE and MAPE indicate not only lower absolute errors but also enhanced proportional accuracy, which is particularly relevant for dynamic time series with variable scales.

TABLE 2. Performance comparison on the Prometheus dataset using RMSE, MAE (in MB), MAPE, and SMAPE.

Method	RMSE (MB)	MAE (MB)	MAPE (%)	SMAPE (%)
Baseline	68.57	57.01	6.94	7.11
Grid	62.79	54.45	6.79	6.78
Random	62.89	54.34	6.73	6.77
PSO	62.59	54.17	6.76	6.74
MRFO	**62.44**	**53.54**	**6.60**	**6.67**

In Table 2, the performance trends observed on the Google dataset are corroborated. MRFO again achieved the lowest error scores across all metrics, reducing the MAE by 6.1% and RMSE by 9.0% relative to the baseline. Although the

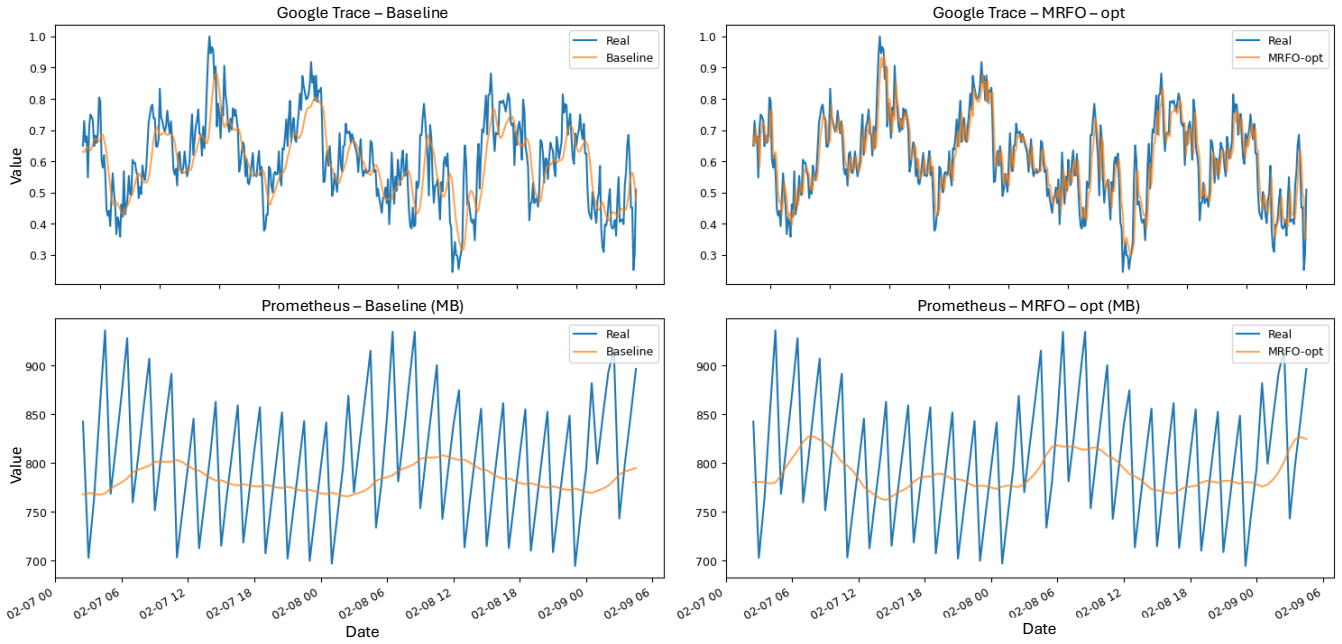

FIGURE 2. Comparison between real values and predictions using Baseline and MRFO-opt models on two datasets: Google Trace (top) and Prometheus (bottom). The left column presents results for the Baseline model, while the right column shows the performance of the MRFO-opt optimized model.

performance margins between methods are somewhat narrower in this real-world dataset, MRFO consistently provided the most accurate forecasts. Random Search and Grid Search performed similarly, with minimal variation in error magnitudes. PSO slightly improved over these traditional methods, but not enough to surpass MRFO. The relatively smaller differences in metrics across methods for Prometheus suggest a lower sensitivity to hyperparameter tuning, possibly due to the dataset's shorter time horizon and smoother variance.

Figure 2 visually reinforces the quantitative results reported in Tables 1 and 2, highlighting the improvements achieved by the MRFO-optimized model in relation to the Baseline. In the top row, which corresponds to the Google Cluster Trace dataset, both models generally follow the real series trend, but the MRFO-opt model demonstrates superior alignment with local variations. The predicted signal is smoother yet remains more faithful to the original peaks and troughs, which supports the improvements observed in RMSE, MAE, and particularly in the proportional metrics MAPE and SMAPE. These visual observations validate the numerical gains shown in Table 1, where MRFO achieved the lowest error rates across all metrics.

The bottom row presents results from the Prometheus dataset, where the predicted values correspond to memory usage in megabytes. Here, the difference between the predicted and actual values are bigger. In the case of the Prometheus dataset, neither model was able to precisely reproduce the strong periodic oscillations present in the real data. Nevertheless, the metrics presented in Table 2 indicate that the MRFO-opt model achieved better results compared to the Baseline, with lower RMSE and MAE values, as well as slight improvements in MAPE and SMAPE, reflecting a closer approximation to the actual values.

Taken together, the visual and quantitative evidence demonstrates that the MRFO optimization approach not only reduces overall prediction error but also enhances the model's capacity to track complex temporal structures in real-world data. This dual strength—in absolute precision and relative proportionality—illustrates the practical advantage of using MRFO for hyperparameter tuning in LSTM-based time series forecasting, particularly in operational contexts where both accuracy and stability are critical. Taken together, these results highlight MRFO's robustness and superior optimization capability across distinct temporal datasets. Its adaptive foraging strategies—balancing local exploitation and global exploration—enabled it to identify highly effective hyperparameter configurations that improved both absolute and relative error performance. These findings validate the applicability of MRFO for hyperparameter optimization in LSTM-based time series forecasting tasks.

B. SENSITIVITY ANALYSIS

To evaluate the effect of temporal context length on forecasting performance, a sensitivity analysis was conducted over different time window sizes using the Grid Search method. Specifically, $n_{\text{steps}} \in \{24, 48, 72, 96\}$ was tested, correspond-

ing to time spans of 12 to 48 hours. Longer windows allow the model to capture long-term dependencies in resource usage, which can be beneficial in workloads exhibiting structured periodicity. However, they also increase the number of trainable parameters, leading to a higher risk of overfitting, especially in the presence of noisy or irregular behavior. Conversely, shorter windows reduce model complexity and overfitting risk but may omit relevant seasonal patterns.

The MRFO algorithm demonstrated its capacity to balance this trade-off automatically. By leveraging its dynamic foraging strategies, MRFO consistently selected optimal window sizes that aligned with the underlying periodic structure in the data. This adaptivity enabled the algorithm to capture temporal dependencies effectively while maintaining generalization, outperforming both heuristic (fixed) and exhaustive (Grid Search) strategies. Notably, MRFO frequently converged to values of n_{steps} that reflect the dominant seasonal cycle observed in the data, further supporting its interpretability.

C. SEASONAL PATTERNS

Autocorrelation analysis was applied to both datasets to identify temporal regularities. In both the Google Cluster Trace and Prometheus datasets, a pronounced peak at a lag of 48 time steps (equivalent to 24 hours at 30-minute intervals) was observed, confirming the presence of strong daily seasonality. This seasonal behavior was characterized by recurring workload cycles that reflect operational routines such as job scheduling or user access patterns.

These findings directly informed the design of the time window search space and the evaluation of window lengths in the hyperparameter tuning process. In particular, the inclusion of $n_{\text{steps}} = 48$ ensured that the models could align their receptive field with the primary periodic component of the signal. MRFO was particularly effective in identifying such seasonally-aligned configurations, which contributed significantly to its improved forecasting accuracy across both datasets.

D. COMPUTATIONAL COST

Table 3 summarizes the computational cost, measured as wall-clock training time in minutes, for each hyperparameter tuning strategy on the two datasets.

TABLE 3. Computational cost (in minutes) for each optimization method on both datasets. Lower values are better.

Method	Google Trace (min)	Prometheus (min)
Baseline	**8.0**	**3.3**
Grid Search	22.1	14.3
Random Search	17.4	12.1
PSO	70.2	51.6
MRFO	14.1	14.3

As shown in Table 3, the baseline configuration, which uses fixed hyperparameters and no optimization, required the least computational effort. With training times of 8.0 minutes for the Google dataset and 3.3 minutes for the Prometheus dataset, it serves as a computational lower bound and provides a useful benchmark for evaluating the overhead introduced by tuning strategies. Although the baseline is efficient, this speed comes at the cost of suboptimal performance, as previously shown in Tables 1 and 2. In contrast, while PSO yielded relatively competitive predictive performance (specially in Table 2), it incurred the highest computational cost due to its iterative particle updates and slower convergence. In contrast, Grid Search and Random Search were more efficient but less accurate, highlighting the well-known trade-off between search thoroughness and computational overhead. MRFO demonstrated a favorable balance between these extremes: it achieved the best accuracy with a runtime comparable to Random Search and significantly lower than PSO.

These results underscore MRFO's practical suitability for AIOps deployments, where both accuracy and efficiency are critical. Its runtime profile and performance gains suggest strong potential for parallelization, making it a promising candidate for integration into distributed learning frameworks.

VI. CONCLUSION

This study presents the application of Manta Ray Foraging Optimization (MRFO) for hyperparameter tuning of LSTM models in the context of resource forecasting for AIOps. The proposed approach achieved substantial improvements, reducing MAE by 6.8% on the Google Cluster Trace and 6.1% on the Prometheus dataset, along with a 9.0% reduction in RMSE for Prometheus. These gains enable more proactive resource management, reducing system downtime and improving cost predictability. The reproducible pipeline, built on TensorFlow, ensures practical applicability in production environments. Despite its effectiveness, the approach has limitations. The study focused solely on univariate memory usage prediction, which restricts generalizability to multivariate contexts. Additionally, the MRFO optimization process, while efficient, still incurs a computational cost of approximately 14 minutes in the testes datasets, which may limit its use in real-time scenarios.

Future improvements could include the use of hybrid metaheuristic methods, such as combining MRFO with PSO or genetic algorithms, to accelerate convergence without sacrificing predictive accuracy. The model could also be extended to multivariate forecasting by incorporating CPU, network, and disk usage, with canonical correlation analysis aiding in the identification of dependencies among metrics. Validation on other datasets, may help assess generalization. Transfer learning could further support adaptation to heterogeneous cloud environments. To enhance adaptability in dynamic scenarios, sliding window mechanisms from 1 to 24 hours could be applied to manage concept drift and balance latency versus accuracy. The scalability of MRFO might be improved by parallelizing its evaluation process across Kubernetes-based clusters, potentially reducing execution time without compromising solution quality. Additionally, exploring fed-

erated AIOps could enable distributed, privacy-preserving forecasting with performance gains over centralized models. Finally, deployment on edge devices could be made feasible by compressing the MRFO-LSTM model to maintain low-latency forecasting while keeping MAE under 60MB.

This work provides a contribution toward scalable and intelligent resource management in cloud computing, with practical experimentation in the context of AIOps. By investigating the use of advanced optimization techniques such as MRFO, it offers insights that may assist in improving operational efficiency. The findings suggest potential for developing more proactive, cost-aware, and adaptable IT systems capable of handling dynamic workloads.

REFERENCES

[1] S. Ouhame, Y. Hadi, and A. Ullah, "An efficient forecasting approach for resource utilization in cloud data center using cnn-lstm model," Neural Computing and Applications, vol. 33, no. 16, pp. 10043–10055, 2021.

[2] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 4–5, IEEE, 2019.

[3] D. C. Pinha and R. S. Ahluwalia, "Flexible resource management and its effect on project cost and duration," Journal of Industrial Engineering International, vol. 15, pp. 119–133, 2019.

[4] S. Singh and I. Chana, "Resource provisioning and scheduling in clouds: Qos perspective," The Journal of Supercomputing, vol. 72, pp. 926–960, 2016.

[5] S. S. Wang and U. Franke, "Enterprise it service downtime cost and risk transfer in a supply chain," Operations Management Research, vol. 13, no. 1, pp. 94–108, 2020.

[6] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. H. Hoi, "Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges," 2023.

[7] S. Mazumdar and A. S. Kumar, "Forecasting data center resource usage: An experimental comparison with time-series methods," in International Conference on Soft Computing and Pattern Recognition, pp. 151–165, Springer, 2016.

[8] M. Corduas, "Nonlinearity tests in time series analysis," Journal of the Italian Statistical Society, vol. 3, pp. 291–313, 1994.

[9] S. Kashyap, A. Singh, and S. S. Gill, "Machine learning-centric prediction and decision based resource management in cloud computing environments," Cluster Computing, vol. 28, no. 2, p. 130, 2025.

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

[11] B. El Haddaoui, R. Chiheb, R. Faizi, and A. El Afia, "On the sensitivity of lstms to hyperparameters and word embeddings in the context of sentiment analysis," in International Conference On Big Data and Internet of Things, pp. 529–542, Springer, 2021.

[12] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," The journal of machine learning research, vol. 13, no. 1, pp. 281–305, 2012.

[13] B. Benaissa, M. Kobayashi, M. Al Ali, T. Khatir, and M. E. A. E. Elmeliani, "Metaheuristic optimization algorithms: An overview," HCMCOU Journal of Science–Advances in Computational Structures, pp. 33–61, 2024.

[14] W. Zhao, Z. Zhang, and L. Wang, "Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications," Engineering Applications of Artificial Intelligence, vol. 87, p. 103300, 2020.

[15] D. Zhu, S. Wang, C. Zhou, and S. Yan, "Manta ray foraging optimization based on mechanics game and progressive learning for multiple optimization problems," Applied Soft Computing, vol. 145, p. 110561, 2023.

[16] F. A. Alturki, H. MH Farh, A. A. Al-Shamma'a, and K. AlSharabi, "Techno-economic optimization of small-scale hybrid energy systems using manta ray foraging optimizer," Electronics, vol. 9, no. 12, p. 2045, 2020.

[17] J. Wilkes et al., "Google cluster-usage traces v3," Google Inc., Mountain View, CA, USA, Technical Report, 2020.

[18] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format+ schema," Google Inc., White Paper, vol. 1, pp. 1–14, 2011.

[19] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "A comparison of arima and lstm in forecasting time series," in 2018 17th IEEE international conference on machine learning and applications (ICMLA), pp. 1394–1401, Ieee, 2018.

[20] P. Lara-Benítez, M. Carranza-García, and J. C. Riquelme, "An experimental review on deep learning architectures for time series forecasting," International journal of neural systems, vol. 31, no. 03, p. 2130001, 2021.

[21] Y. Zhu, W. Zhang, Y. Chen, and H. Gao, "A novel approach to workload prediction using attention-based lstm encoder-decoder network in cloud environment," EURASIP Journal on Wireless Communications and Networking, vol. 2019, pp. 1–18, 2019.

[22] M. Alharthi and A. Mahmood, "xlstmtime: Long-term time series forecasting with xlstm," AI, vol. 5, no. 3, pp. 1482–1495, 2024.

[23] Z. Liu, W. Li, J. Feng, and J. Zhang, "Research on satellite network traffic prediction based on improved gru neural network," Sensors, vol. 22, no. 22, p. 8678, 2022.

[24] J. Zhang, S. Qu, Z. Zhang, and S. Cheng, "Improved genetic algorithm optimized lstm model and its application in short-term traffic flow prediction," PeerJ Computer Science, vol. 8, p. e1048, 2022.

[25] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: a big comparison for nas," arXiv preprint arXiv:1912.06059, 2019.

[26] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," Advances in engineering software, vol. 69, pp. 46–61, 2014.

[27] S. Mirjalili and A. Lewis, "The whale optimization algorithm," Advances in engineering software, vol. 95, pp. 51–67, 2016.

[28] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," IEEE computational intelligence magazine, vol. 1, no. 4, pp. 28–39, 2007.

[29] Q. Cheng, D. Sahoo, A. Saha, W. Yang, C. Liu, G. Woo, M. Singh, S. Saverese, and S. C. Hoi, "Ai for it operations (aiops) on cloud platforms: Reviews, opportunities and challenges," arXiv preprint arXiv:2304.04661, 2023.

[30] A. Sreerangapuri, "Federated learning: Revolutionizing multi-cloud ai while preserving privacy," International Journal of Computer Applications Information Technology, vol. 7, pp. 587–602, 07 2024.

[31] Z. Zhong, Q. Fan, J. Zhang, M. Ma, S. Zhang, Y. Sun, Q. Lin, Y. Zhang, and D. Pei, "A survey of time series anomaly detection methods in the aiops domain," 2023.

[32] Pennsylvania State University, "Lesson 5: Decomposition of time series." https://online.stat.psu.edu/stat510/lesson/5/5.1, 2023. Accessed: 2025-04-30.

[33] S. B. Chudo and G. Terdik, "Modeling and forecasting time-series data with multiple seasonal periods using periodograms," Econometrics, vol. 13, no. 2, p. 14, 2025.

[34] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, Deep learning, vol. 1. MIT press Cambridge, 2016.

[35] M. Nielsen, Neural Networks and Deep Learning. Determination Press, 2015.

[36] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[37] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," Computing in Science amp; Engineering, vol. 13, p. 22–30, Mar. 2011.

[38] W. McKinney and P. Team, "Pandas-powerful python data analysis toolkit," Pandas—Powerful Python Data Analysis Toolkit, vol. 1625, 2015.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

WESLLEY SANTANA ROSALEM received the B.S. degree in Computer Science from Universidade Anhembi Morumbi in 2014. He received an MBA in Enterprise Architecture from FIAP in 2017, and a postgraduate degree in Marketing from the University of São Paulo (USP) in 2019. He also earned an Executive MBA in Business Administration from Fundação Getulio Vargas (FGV) in 2021. He is currently pursuing the M.Sc. degree in Computer Science at São Paulo State University (UNESP), with a focus on AIOps and deep learning methods for time series forecasting and anomaly detection in Kubernetes environments. His research interests include artificial intelligence, AIOps, deep learning, time series analysis, and intelligent resource management in cloud computing. He currently works as a Principal AI Solutions Architect for the LATAM region at Red Hat. He is a member of the IEEE.

FABRÍCIO STEINLE AMOROSO received the B.S. degree in Computer Science from São Paulo State University (UNESP), Brazil, in 2022. He is currently pursuing the M.Sc. degree in Computer Science at UNESP and conducts research at the Laboratory of Applied Security and Networks (LARS), focusing on cybersecurity applications of deep learning. His current research interests include intrusion detection systems, deepfake detection, and intelligent security mechanisms for computer networks.

RENATO DIAS DE SOUZA He is currently a Ph.D. student in Computer Science at São Paulo State University (UNESP). He received his M.Sc. in Science and Technology of Materials in 2025 and his B.Sc. in Materials Physics, also from UNESP, in 2019. His research focuses on the application of Machine Learning, Generative Artificial Intelligence, and Quantum Computing.

FELIPE MAHLOW received his Ph.D. in Computer Science from São Paulo State University (UNESP) in 2025, where he focused on classical and quantum machine learning and their applications in Quantum Information Science. He also earned a teaching degree in Physics and a bachelor's degree in Materials Physics from UNESP in 2020 and 2023, respectively. His research encompasses the use of Machine Learning, Generative Artificial Intelligence, and Quantum Computing.

KELTON AUGUSTO PONTARA DE COSTA (Senior Member, IEEE) received the B.Sc. degree in systems analysis from the "Sagrado Coração" University (USC), the M.Sc. degree in computer science from the "Euripides Foundation" of Marilia University (UNIVEM), and the Ph.D. degree from São Paulo University (USP). He was a Postdoctoral Researcher of computer networks with Campinas State University (UNICAMP) and a Postdoctoral Researcher of anomaly detection in computer networks with São Paulo State University (UNESP), Bauru. He is currently a Professor of computer science and information systems courses and a Professor of the master's and doctoral programs in computer science with UNESP. His current research interests include computer science, with emphasis on systems architecture computing and distributed systems, management in computer networks, computer security, anomaly detection systems and signatures in computer networks, and data flow analysis in computer networks.