Cluster Computing (2025)28:176
https://doi.org/10.1007/s10586-024-04887-5

Adaptive horizontal scaling in kubernetes clusters with ANN-based load forecasting

Lucileide M. D. da Silva[1,2,3] · Pedro V. A. Alves[1,2] · Sérgio N. Silva[1,2] · Marcelo A. C. Fernandes[1,2,4]

Received: 23 July 2024 / Revised: 19 October 2024 / Accepted: 4 November 2024

Abstract
In modern cloud environments, efficient management of computational resources is a critical challenge due to the growing demand for scalable and high-performance applications. Horizontal scaling in Kubernetes (K8s) clusters is essential for dynamically adjusting resources to match workload demands. However, their reactive nature often limits traditional autoscaling methods like K8s Horizontal Pod Autoscaler (HPA), leading to inefficiencies under variable loads. To overcome these limitations, more advanced and adaptive scaling approaches are needed. Thus, this study introduces an adaptive approach to horizontal scaling in K8s clusters using Artificial Neural Networks (ANNs) for load forecasting, referred to as ANN-HS. The proposed method aims to enhance the efficiency of resource consumption and optimize replica allocation compared to the standard HPA. By leveraging pre-trained regression models, ANN-HS dynamically adjusts resources to meet varying demands, ensuring adherence to latency requirements and improving overall system performance. Experimental results demonstrate that ANN-HS outperforms traditional HPA methods, offering a scalable and flexible solution for managing microservices in cloud environments. This approach provides a robust framework for optimizing horizontal scaling in Kubernetes, contributing to the advancement of intelligent resource management in cluster computing. Experimental results show that ANN-HS significantly improves resource utilization compared to Kubernetes' HPA. Specifically, ANN-HS reduces CPU consumption by approximately 50% while maintaining Service Level Agreement (SLA) compliance with an average violation rate of less than 10%. Additionally, ANN-HS reduces the number of replicas needed by 66.67%, optimizing resource allocation under varying load conditions.

Keywords Horizontal scaling · Kubernetes · Machine learning · Artificial neural networks · HPA

1 Introduction

Cluster computing has become a cornerstone of modern computing, enabling the efficient processing and management of large-scale applications across distributed systems. Horizontal scaling, a crucial aspect of cluster management, involves dynamically adjusting the number of running instances to match workload demands. Kubernetes (K8s), a widely adopted container orchestration platform, offers built-in solutions like the Horizontal Pod Autoscaler (HPA) to facilitate this process. However, the increasing complexity and variability of workloads necessitate more advanced and adaptive scaling techniques to maintain performance and resource efficiency in Kubernetes environments [1–5].

Horizontal scaling in K8s clusters is a critical challenge for ensuring the efficiency and performance of applications in cloud environments [1, 2, 6]. With the increasing complexity and demand of HTTP traffic, it is essential to adopt innovative approaches to dynamically adjust the number of application replicas running in the cluster [7–9]. In this context, the use of Machine Learning (ML) techniques has

✉ Marcelo A. C. Fernandes
mfernandes@dca.ufrn.br

1 InovAI Lab, nPITI/IMD, Federal University of Rio Grande do Norte (UFRN), Natal 59078-900, RN, Brazil

2 Leading Advanced Technologies Center of Excellence (LANCE), nPITI/IMD, UFRN, Natal 59078-900, RN, Brazil

3 Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN), Santa Cruz 59200-000, RN, Brazil

4 Department of Computer and Automation Engineering, UFRN, Natal 59078-900, RN, Brazil

Published online: 21 January 2025

emerged as a promising solution for predicting the load associated with HTTP traffic, enabling horizontal scaling based on real-time information.

Traditional scaling approaches in K8s, such as the HPA [10], offer basic resource adjustment functionalities [11]. However, these methods are often limited by their reactivity and dependence on static metrics like CPU usage. Recent literature shows that methods based on advanced forecasts and machine learning can overcome these limitations, providing proactive and adaptive scaling that better responds to demand variability and maintains required service levels [12, 13].

This work aims to develop an adaptive approach for horizontal scaling in Kubernetes clusters using Artificial Neural Networks (ANN) for load prediction, called ANN-HS. The proposed method aims to improve resource consumption efficiency and optimize replica allocation compared to the traditional HPA. Using pre-trained regression models, ANN-HS dynamically adjusts resources to meet variable demands, ensuring latency requirements and enhancing overall system performance in cloud computing environments.

To evaluate the effectiveness of the proposed approach, experiments were conducted on a Kubernetes cluster implemented on two distinct virtual machines. Using jMeter to generate HTTP traffic demands and Prometheus to collect cluster metrics, the ANN-based load prediction system was compared to Kubernetes' HPA. The results of these tests provided valuable insights into the performance and efficiency of the machine learning approach compared to traditional horizontal scaling strategies, demonstrating the potential of this technique to optimize resource management in Kubernetes environments.

The highlights of this study can be summarised as follows:

- Introduction of ANN-HS: A novel approach leveraging ANNs for dynamic and automatic horizontal scaling in K8s clusters.
- Enhanced Performance: Demonstrates superior efficiency in resource consumption and optimized replica allocation compared to the traditional Kubernetes HPA.
- Experimental Validation: Comprehensive evaluation in a real-world Kubernetes environment, utilizing jMeter for traffic generation and Prometheus for metric collection, substantiating the effectiveness of the proposed method.
- Resource Optimization and SLA Maintenance: Significant improvements in resource utilization while ensuring adherence to SLAs, showcasing the potential of ANN-HS to maintain high service quality under varying loads.

2 Related works

Recent studies have explored various approaches to enhance horizontal scaling in K8s, utilizing advanced machine learning techniques. The study by [14] introduces an intelligent autonomous autoscaling system for microservices in cloud applications, featuring a generic auto-scaling algorithm. Using reinforcement learning agents to identify autoscaling threshold values, the system was evaluated with real data and demonstrated significant improvements in microservice response times, up to 20% better than the standard Kubernetes autoscaling. This approach provides a customized autoscaling solution with minimal user involvement.

Similarly, [6] propose and evaluate autoscaling mechanisms for serverless computing platforms, utilizing reinforcement learning techniques to optimize resource provisioning in response to variable workloads. Their developed solutions can learn and autonomously adjust scaling policies, considering metrics such as latency, throughput, Service Level Agreements (SLA) violations, resource usage, and associated costs.

The study by [15] proposes an auto-scaling engine for Kubernetes that leverages various AI-based forecasting methods. The engine integrates Auto-Regressive (AR), Hierarchical Temporal Memory (HTM), and Long Short-Term Memory (LSTM) models to adjust resource allocation based on real-time demand predictions proactively. The paper demonstrates that AI-driven auto-scaling can effectively balance the trade-off between resource over-provisioning and SLA violations, offering a more efficient approach to managing cloud-based web applications.

Another relevant work by [16] addresses the challenge of resource management in Kubernetes, particularly the timely scaling of resources during workload fluctuations. They propose an elastic scaling method in K8s based on time series forecasting using Holt-Winter and GRU (Gated Recurrent Unit) models. This method predicts resource needs to dynamically adjust K8s clusters, improving resource utilization and maintaining service quality.

The research by [17] and [18] explore the use of bidirectional LSTM networks for deep learning-based autoscaling in K8s. They aim to improve the accuracy of scaling decisions by predicting future resource demands, using deep learning techniques to dynamically adjust the number of service replicas. The Bi-LSTM model can predict future workloads by analyzing historical time series data, enabling more precise and faster resource provisioning.

The HANSEL system, as proposed by [18], enhances the horizontal scaling of microservices on the Kubernetes platform by leveraging a Bi-LSTM load prediction

algorithm with an attention mechanism. This system addresses the limitations of Kubernetes' built-in HPA by incorporating both active and reactive scaling methods. The combination of these methods allows HANSEL to accurately predict and respond to dynamic microservice loads, thereby optimizing resource utilization and ensuring compliance with SLAs. The experimental results demonstrate that HANSEL can improve resource utilization by approximately 20%, making it a significant advancement over traditional scaling methods.

The double tower deep learning architecture, as proposed by [19] in their study on intelligent horizontal autoscaling in edge computing, utilizes two parallel neural network structures to process distinct sets of input data independently before merging their outputs for final decision-making. This design incorporates a tower dedicated to local metrics, employing LSTMs to handle time-series data, and another tower focused on global metrics, utilizing Feedforward Neural Networks (FNNs) to process aggregated infrastructure data. By combining the outputs of these towers, the architecture enhances the accuracy and robustness of predictive models, offering significant improvements in resource utilization and performance optimization in dynamic edge computing environments.

The research by [20] presents an innovative approach to automating the configuration of Kubernetes clusters. By leveraging digital twins and machine learning, specifically using a Random Forest Regressor, the system autonomously adjusts the HPA settings in response to dynamic workload changes. This method reduces resource utilization while maintaining low request latency.

In the context of managing resource scalability for Kubernetes edge clusters, [21] proposes an advanced autoscaling mechanism that leverages various machine learning techniques to predict incoming request rates and adjust resources accordingly. Their approach, unlike the traditional HPA in Kubernetes, which relies on reactive decision-making based on current metrics, incorporates proactive scaling strategies. By utilizing methods such as AR, Hierarchical Temporal Memory (HTM), and LSTM, the proposed system dynamically selects the most suitable forecasting model to minimize request losses and optimize resource allocation. This research demonstrates a significant improvement in handling the fluctuating demands of edge applications, offering a solution for maintaining service quality and efficiency in cloud environments.

These studies collectively highlight the potential of Artificial Intelligence in enhancing the auto-scaling capabilities of K8s. This work proposes a solution based on Artificial Neural Networks (ANN) for load prediction in HTTP applications in a K8s environment, referred to here as *Artificial Neural Network Horizontal Scaling* (ANN-HS). The ANN is trained with a dataset containing CPU load per replica, received packets per minute (rpm) per replica, and the number of replicas under different traffic scenarios. Based on these data, the ANN can make load predictions, which are then converted into a new number of replicas, allowing for dynamic and automatic horizontal scaling of the cluster.

Table 1 presents a summary of the main approaches in the state of the art related to horizontal scaling in Kubernetes clusters, highlighting the techniques used, key metrics, limitations, and the contributions of each work. As shown, approaches based on reinforcement learning and more complex neural networks, such as bidirectional LSTM and double-tower architectures, offer significant resource utilization and response time improvements. However, many of these solutions need help with implementation complexity and adjustments in highly dynamic load pattern scenarios. In contrast, the present work stands out for its simplicity and efficiency, utilizing ANN to predict the load and dynamically adjust resources in real-time, achieving significant reductions in CPU consumption and the number of replicas while maintaining required SLA levels. This comparison highlights the advantage of the proposed approach in terms of ease of implementation and operational efficiency, especially when compared to K8s standard HPA.

3 Materials and methods

Figure 1 describes the architecture of the ANN-HS, where an ANN is used to predict the load on a web application, enabling horizontal scaling control in a K8s cluster. At each n-th time instant, the ANN-HS receives as input the variables $x^{\text{cpu}}(n)$, $x^{\text{rp}}(n)$, and $m^{\text{p}}(n)$, representing the CPU load in millicores, the rate of rpm, both normalized by the number of application replicas, and the number of replicas of a given application, respectively. The variables $x^{\text{cpu}}(n)$ and $x^{\text{rp}}(n)$ are generated by a module referred to here as the *Normalization Module* (NM), which captures the application's metrics through a *Monitoring Service* (MS) associated with the K8s cluster. The MS captures the CPU load metrics of all application replicas, $m^{\text{cpu}}(n)$, the rate of packets received by all application replicas in rpm, $m^{\text{rp}}(n)$, and the number of application replicas, $m^{\text{p}}(n)$. At each n-th time instant, the ANN-HS produces the output in the variable $y(n)$, representing a predicted load value on the web application. The predicted load value at the n-th instant, $y(n)$, is used to calculate the next number of replicas to be achieved by the K8s, $m^{\text{p}}(n+1)$. The calculation of the next number of replicas is performed by a module called the *Load-to-Replica Converter Module* (LRCM), which can use anything from a simple linear

Table 1 Summary of the state of the art in horizontal scaling in Kubernetes clusters

References	Approach	Technique used	Key metrics	Limitations	Comparison with this work
(2021) [14]	Intelligent autoscaling for microservices	Reinforcement Learning	20% improvement in response time	Requires significant time to learn and adjust to workload changes	Fast load prediction using ANN, providing continuous and dynamic adjustments in real time
(2020) [15]	Adaptive AI-based autoscaling	AR, HTM, LSTM	Optimization between resource provisioning and SLA violations	Complexity in handling highly dynamic load patterns	Simplicity in using ANN for accurate prediction of variable loads
(2021) [17]	LSTM-based autoscaling for Kubernetes	Bidirectional LSTM	Improved accuracy in scaling decisions	Complex implementation and parameter tuning for different scenarios	Simpler solution with MLP, maintaining high accuracy and real-time response
(2021) [18]	HANSEL: Adaptive horizontal scaling of microservices using Bi-LSTM	Bi-LSTM with attention mechanism	20% better resource utilization	Requires combining active and reactive methods to achieve accuracy	Continuous load prediction without needing complex combinations of methods
(2022) [19]	Intelligent autoscaling in edge computing	Double-tower neural network architecture	Improved accuracy and robustness in edge computing scenarios	Requires separate processing of local and global metrics before decision-making	Simpler implementation, yet equally efficient in cloud environments
This work	Adaptive horizontal scaling in Kubernetes clusters	Artificial Neural Network (ANN)	50% reduction in CPU usage, 66.67% reduction in replicas, SLA violation $< 10\%$	Potential initial training overhead of the model	

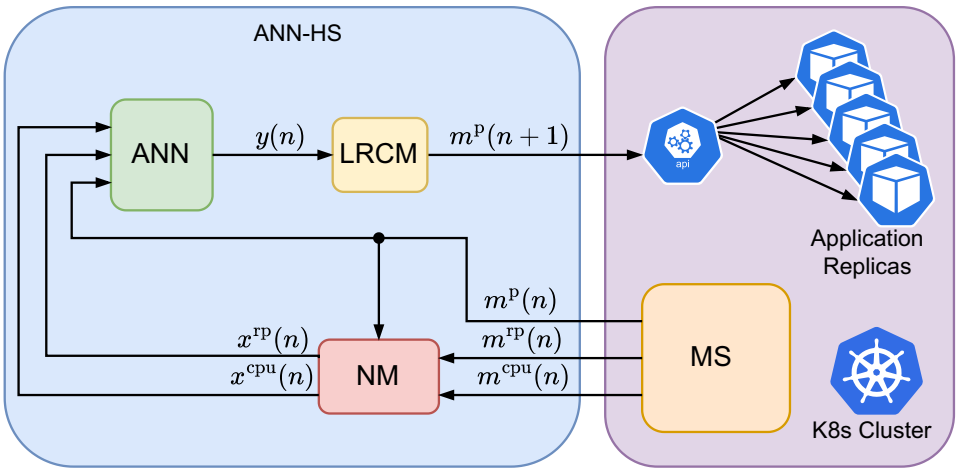

Fig. 1 ANN Horizontal Scaling (ANN-HS) Architecture

conversion to a more complex metric to indicate the next number of replicas based on the load predicted by the ANN. The number of replicas is adjusted directly through the K8s API.

The variables $x^{\mathrm{cpu}}(n)$ and $x^{\mathrm{rp}}(n)$ can be expressed as

$$x^{\mathrm{cpu}}(n) = \frac{m^{\mathrm{cpu}}(n)}{m^{\mathrm{p}}(n)} \tag{1}$$

and

$$x^{\mathrm{rp}}(n) = \frac{m^{\mathrm{rp}}(n)}{m^{\mathrm{p}}(n)}. \tag{2}$$

The variable $m^{\mathrm{p}}(n+1)$ can be expressed as

$$m^{\mathrm{p}}(n+1) = f(y(n)) \tag{3}$$

where $f(\cdot)$ is a function that maps the predicted load to the number of replicas requested by K8s. The variable $m^{\mathrm{p}}(n+1)$ is an integer value between 1 and $m^{\mathrm{p}}\mathrm{max}$, where $m^{\mathrm{p}}\mathrm{max}$ represents the maximum number of replicas. Based on

Fig. 1, Eqs. 1 and 2 are implemented by the NM, and Eq. 3 is implemented by the LRCM.

The ANN plays a crucial role in load prediction in K8s. By receiving the CPU load metrics and the rate of rpm, normalized by the number of replicas, the ANN is trained to estimate the application's current load. Based on the load predicted by the ANN, the system can make intelligent decisions about adjusting the number of replicas, promoting on-demand horizontal scaling. The ANN allows the system to be more adaptive and responsive to traffic fluctuations, optimizing resource consumption and improving the application's performance in the K8s environment.

The research methodologies included two distinct approaches to explore and validate the proposed ANN-based horizontal scaling in K8s environments. The first approach focused on training the ANN to predict the load of a web application running in a K8s cluster. The second approach involved evaluating and testing the proposed method in a running K8s environment. For both, the experimental environment consisted of a MicroK8s [22] implementation in a virtual machine (VM) setup. This setup included a VM named VM Cluster (VMC) with 4 CPUs, 12 GB of memory, and Ubuntu 20.04 server, and another VM named VM Load Generator (VMLG) with 3 CPUs, 12 GB of memory, and Ubuntu 22.04 server. The VMLG was used to generate HTTP traffic demands on the application running in K8s. The jMeter [23] tool was employed to generate HTTP traffic demands, while Prometheus [24] was utilized to capture relevant cluster metrics. The web application was implemented in Java using Dropwizard and executed a Fibonacci sequence of order 15 in response to each request made by jMeter to the K8s web application. The host machine was equipped with 128 GB of RAM, and 16 CPUs and ran Windows 10.

3.1 ANN training

3.1.1 Dataset creation

Figure 2 details the scheme used for generating the training dataset for the ANN [25]. A total of $K = 50$ experiments were conducted, varying the load (number of users) in a range from 1 to 99, with intervals of 2. Thus, each k-th load experiment had N_k users, where $N_k \in 1, 3, \ldots, 97, 99$ for $k = 1, \ldots, K$. Each k-th load experiment with N_k users was executed $V = 20$ times, varying the number of replicas, R_i, of the web application in K8s from 1 to 20, in steps of 1, i.e., $R_i \in 1, 2, \ldots, 19, 20$ for $i = 1, \ldots, V$. In total, $K \times V = 1000$ experiments were conducted, where each j-th experiment, $\mathbf{e}_j$ can be expressed as

$$\mathbf{e}_j = \mathbf{e}_{k,i} = \{N_k, R_i\} \text{ para } j = 1, \ldots, K \times V, \tag{4}$$

where

$$k = \left\lfloor \frac{j-1}{V} \right\rfloor + 1 \tag{5}$$

and

$$i = j - V \times (k-1). \tag{6}$$

Both variables $\mathbf{e}_j$ and $\mathbf{e}_{k,i}$ represent the experiment for generating the dataset, with the k-th value of the number of users and the i-th value of the number of replicas. Each experiment was controlled by a module referred to here as the *Experiment Controller Module* (ECM). As illustrated in Fig. 2, $u_{k,i}$ represents the virtual user created by jMeter associated with the experiment $\mathbf{e}_{k,i}$.

The value V was selected for dataset creation based on preliminary experimental analysis to simulate a moderate load scenario in a K8s environment, representing a significant but not extreme traffic demand. This choice aimed to capture relevant variations in CPU usage, $x^{\text{cpu}}(n)$, number of replicas, $m^{\text{p}}(n)$, and packet reception rate, $x^{\text{rp}}(n)$, allowing the model to learn the system's dynamics under typical conditions. The moderate load ensured a complex profile to test the system's dynamic adjustment capabilities without overwhelming resources. Additionally, multiple variations of V were used to cover different load levels and ensure robust model training.

Each j-th experiment, $\mathbf{e}_j$, (see Eqs. 4, 5, and 6) used the load profile illustrated in Fig. 3. This profile was constructed using the *Open Model Thread Group*, which defines the number of virtual users created over time by jMeter to generate traffic demand on the web application running in K8s [23].

Figure 3 illustrates the virtual user creation profile used in jMeter to simulate HTTP request demand in the Kubernetes environment. This profile was designed to capture the typical traffic behavior seen in production web applications, which often experience fluctuating loads, including traffic spikes followed by a gradual decrease in access. The pattern consists of three key phases: an initial ramp-up, where the number of virtual users gradually increases, followed by a constant load period, and finally, a ramp-down, where the number of users decreases. The ramp-up simulates the natural increase in traffic, such as when a website or web service begins to receive more users during peak times, while the ramp-down reflects the decline in traffic after these high-usage periods.

The load pattern (profile), as depicted in Fig. 3, is critical in evaluating the effectiveness of the proposed horizontal scaling system, ANN-HS. This system is tasked with dynamically adjusting the number of replicas in response to

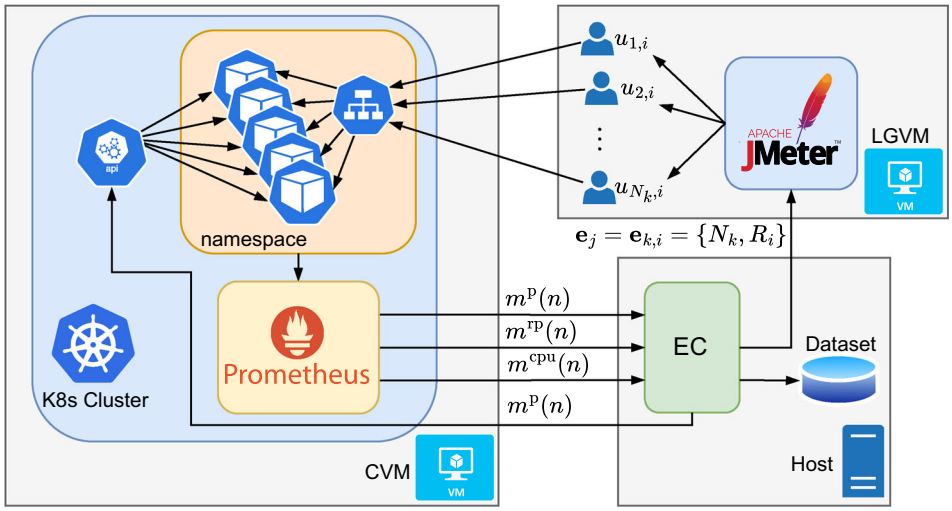

Fig. 2 Scheme used for dataset creation for load prediction

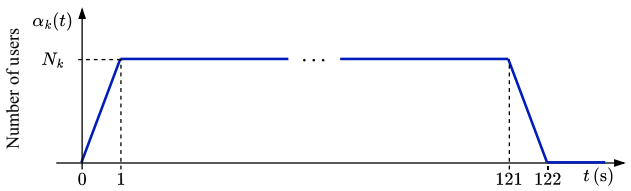

Fig. 3 Profile of virtual user creation by jMeter using the *Open Model Thread Group* for dataset generation. Each j-th experiment, $\mathbf{e}_j$, (see Eq. 4) used this profile

fluctuations in HTTP request demand. The load pattern is particularly crucial during the ramp-up phase, where the system must swiftly identify the traffic surge and provision additional replicas to prevent overload and ensure response times remain within the SLA parameters. Similarly, during the constant load phase, the system must strike a delicate balance between resource usage and the number of necessary replicas. In the ramp-down phase, ANN-HS must intelligently reduce the number of replicas to prevent resource overuse without compromising performance. Therefore, the load pattern shown in Fig. 3 is pivotal in validating ANN-HS's ability to anticipate and adapt to changing HTTP traffic conditions in real-time.

With a duration of 122 seconds, the user creation load profile is characterized by a ramp-up in the first 1 second until reaching a quantity of N_k users. Between 1 and 121 seconds, the requests remain constant at N_k, and after 121 seconds, the number of users ramps down (following the same ramp-up rate) until 122 seconds. The number of virtual users at a given time, $\alpha_k(t)$, can be expressed as

$$\alpha_k(t) = \begin{cases} N_k t & \text{if } 0 < t \leq 1, \\ N_k & \text{if } 1 < t \leq 121, \\ -N_k t + 122 N_k & \text{if } 121 < t \leq 122, \\ 0 & \text{if } t > 122, \end{cases} \tag{7}$$

where N_k is the maximum number of virtual users that the load profile can reach for the k-th load value associated with the j-th experiment, $\mathbf{e}_j$ (see Eqs. 4, 5, and 6).

The experiments generated 1000 datasets, with data captured every 2 s over a duration of 122 s. Thus, each j-th experiment $\mathbf{e}_j$ produced a dataset, $\mathbf{C}_j$, which can be expressed as:

$$\mathbf{C}_j = \mathbf{C}_{k,i} = [\mathbf{c}_{k,i,1}, \ldots, \mathbf{c}_{k,i,v}, \ldots, \mathbf{c}_{k,i,M_{k,i}}] \tag{8}$$

where $M_{k,i}$ is the number of samples captured in each j-th experiment (see Eqs. 4, 5, and 6) and $\mathbf{c}_{k,i,v}$ is the v-th sample captured during the j-th experiment, which is expressed as:

$$\mathbf{c}_{k,i,v} = [x^{\mathrm{cpu}}_{k,i,v}, x^{\mathrm{rp}}_{k,i,v}, m^{\mathrm{p}}_{k,i,v}] \tag{9}$$

where $x^{\mathrm{cpu}}_{k,i,v}$, $x^{\mathrm{rp}}_{k,i,v}$ and $m^{\mathrm{p}}_{k,i,v}$ represent the CPU load per replica (see Eq. 1) used by the application in millicores, the rate of packets received per replica (see Eq. 2) of the application in rpm, and the number of replicas, respectively. At the end of the experiments, a dataset of approximately 61,000 rows was obtained.

3.1.2 Preprocessing

The preprocessing stage is crucial in preparing the data before training the ANN. In this phase, two main operations were performed: the removal of outliers and the calculation of average CPU consumption and packets received per experiment. Initially, a dataset with approximately 61, 000 entries was created. Outliers with values above the 95% percentile were removed, thus eliminating extremely atypical and unrepresentative data. This approach ensured the integrity of the data used in training the ANN.

After removing the outliers, the next step was calculating each experiment's average CPU consumption per replica in milli-cores and the rate of packets received per replica in rpm. By calculating the average of these resources for each experiment, we obtained a central value that smooths out temporary fluctuations and reduces potential noise in the data. This methodology provides a more stable and reliable perspective on the average load patterns of the web application across different traffic scenarios. As a result, the ANN is trained with more consistent and relevant data, improving its ability to accurately learn the relationship between the inputs and the predicted load.

Consequently, a new dataset was generated for training the ANN, resulting in a dataset of 1,000 lines, where each line corresponds to an experiment [25]. Each line of this new dataset can be expressed as

$$\mathbf{g}_j = \mathbf{g}_{k,i} = [\bar{x}^{\text{cpu}}_{k,i}, \bar{x}^{\text{rp}}_{k,i}, m^{\text{p}}_{k,i}] \tag{10}$$

where

$$\bar{x}^{\text{cpu}}_{k,i} = \frac{1}{M_{i,k}} \sum_{v=1}^{M_{i,k}} x^{\text{cpu}}_{k,i,v} \tag{11}$$

and

$$\bar{x}^{\text{rp}}_{k,i} = \frac{1}{M_{i,k}} \sum_{v=1}^{M_{i,k}} x^{\text{rp}}_{k,i,v}. \tag{12}$$

Table 2 presents the descriptive statistics of the dataset used in this study. It shows measures such as the mean, median, standard deviation, minimum and maximum values, as well as the first and third quartiles (Q1 and Q3) for each variable analyzed, including CPU usage, number of replicas (pods), and packet rate.

3.1.3 Training

In this study, using a Multi-Layer Perceptron (MLP) type ANN was crucial for load prediction and horizontal scaling in K8s environments. Configured as a regression model, the ANN was designed to predict a continuous load value based on three input variables: CPU load per replica in milli-cores, packet reception rate per replica in rpm, and the number of application replicas running in the K8s cluster.

Table 2 Descriptive statistics of the dataset

Variable	Mean	Median	Std. Dev	Min	Max	Q1 - Q3
$x^{\text{cpu}}(n)$	28.01	18.36	28.09	1.38	148.94	10.52 - 35.36
$x^{\text{rp}}(n)$	53.69	28.30	82.11	0.28	621.83	14.21 - 54.40
$m^{\text{p}}(n)$	10.50	10.50	5.77	1.00	20.00	5.50 - 15.50

The ANN's architecture consisted of three hidden layers with 200, 200, and 100 neurons, respectively, and an output layer. The hidden layers played a crucial role in enabling the network to learn the complex and non-linear patterns associated with the application load, allowing for more accurate modeling of the relationship between the input variables and the predicted load. The ReLU activation function was used to introduce non-linearity, which helps the network to capture these patterns better. The training process was conducted with a maximum iteration limit of 1000 to ensure convergence, and the regularization strength was set to zero to prioritize fitting the training data without applying any penalty to the model's complexity.

The choice of hyperparameters, including the number of neurons, activation function, iteration limit, and regularization strength, was fine-tuned after extensive testing with different values. This iterative process aimed to identify the optimal configuration that provided the best balance between training accuracy and generalization to new data, ensuring that the model performed effectively across varying load scenarios.

During training, iterative adjustments were made to the ANN to minimize the error between the generated predictions and the actual observed values. The regression algorithm adjusted the connections' weights between neurons, enhancing the network's ability to effectively and accurately correlate the inputs with the predicted load.

The total training time for the model was 69.85 s, performed on a computer equipped with an Intel(R) Core(TM) i7-10700 CPU at 2.90 GHz, 128 GB of DDR4 RAM at 2666 MHz, a 512 GB SSD, and an NVIDIA GeForce GTX 1650 graphics card with 4 GB. After training, the final model size was 495,760 bytes, and it achieved a throughput of 107,630 observations per second (obs/sec), demonstrating high efficiency in prediction speed.

3.2 Methodology associated with the ANN testing

Figure 4 details the scheme used to test the ANN-HS proposal. A total of $H = 20$ experiments were conducted, varying the load (number of users) from 5 to 100 in increments of 5. Thus, each k-th load experiment had L_k users, where $L_k \in \{5, 10, \ldots, 95, 100\}$ for $k = 1, \ldots, H$. Each k-th test experiment can be represented by the variable $\mathbf{t}_k$.

Each k-th test experiment, $\mathbf{t}_k$, utilized the load profile illustrated in Fig. 5. This profile was constructed using the *Open Model Thread Group*, which determines the number of virtual users created over time by jMeter to generate

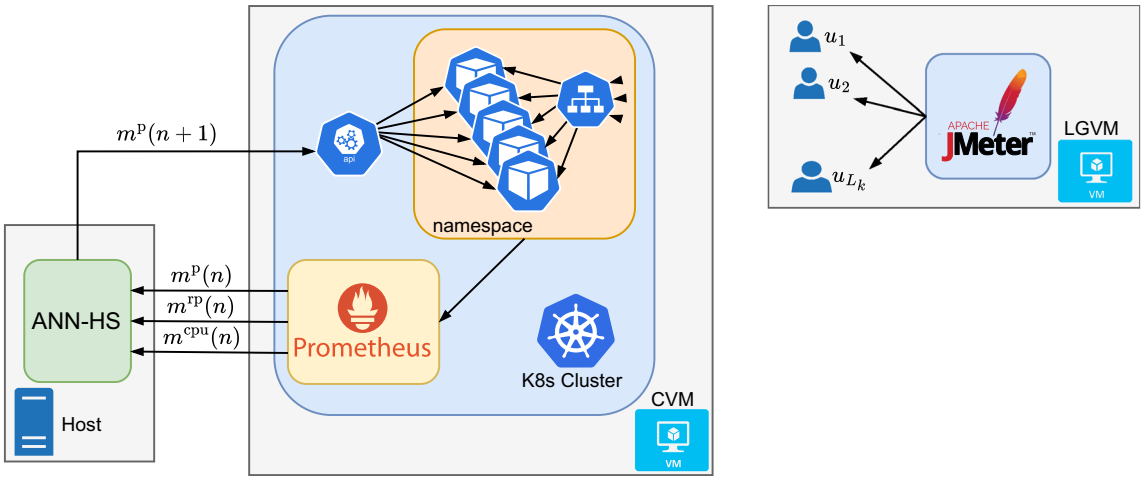

Fig. 4 Scheme used for the k-th test experiment of the ANN-HS

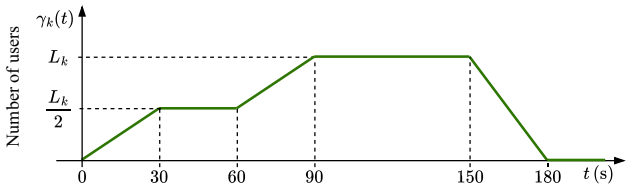

Fig. 5 Profile for creating virtual users via jMeter using *Open Model Thread Group* for the ANN-HS tests. Each k-th test experiment, $\mathbf{t}_k$, utilized this profile

traffic demand on the web application running on Kubernetes [23].

With a duration of 180 seconds, the load profile for the k-th test experiment, $\mathbf{t}_k$, is characterized by a ramp-up in the first 30 seconds until reaching a count of $\frac{L_k}{2}$ users. In the next 30 seconds, the load remains constant at $\frac{L_k}{2}$ users. Upon reaching 60 seconds, the load begins a second ramp-up over 30 seconds until it reaches L_k users. After 90 seconds, the load remains constant at L_k users for the next 60 seconds. At 150 seconds, the load begins a ramp-down to zero over the next 30 seconds. The number of virtual users at any given time, $\gamma_k(t)$, can be expressed as

$$\gamma_k(t) = \begin{cases} \frac{L_k t}{60} & \text{if } 0 < t \leq 30, \\ \frac{L_k}{2} & \text{if } 30 < t \leq 60, \\ \frac{L_k t}{60} - \frac{L_k}{2} & \text{if } 60 < t \leq 90, \\ L_k & \text{if } 90 < t \leq 150, \\ \frac{-L_k t}{30} + 6L_k & \text{if } 150 < t \leq 180, \\ 0 & \text{if } t > 180, \end{cases} \tag{13}$$

where L_k is the maximum number of virtual users the load profile can reach for the k-th load value associated with the k-th experiment.

The ANN-HS was implemented and executed both on the experiment's host and on the previously mentioned virtual machines. Using the Matlab development environment, version 596681, the system utilized the Java Fabric8 Kubernetes-Client library [26] to interact with the Kubernetes API and perform dynamic scaling of the web application replicas. This approach allowed the ANN-HS to operate as an external component to the cluster, making scaling decisions based on collected metrics and sending scaling commands to the K8s cluster. The necessary metrics for scaling were collected using Prometheus [24], which served as the monitoring service, as illustrated in Fig. 1. For this study, a maximum value of $m^{\mathrm{p}}_{\max} = 20$ was set, and a linear function was applied to convert the predicted load into the number of replicas. Equation 14 presents the formula for calculating the replicas.

$$m^{\mathrm{p}}(n+1) = \left\lceil y(n) \times \frac{20}{100} \right\rceil. \tag{14}$$

In the testing methodology, the native K8s Horizontal Pod Autoscaler (HPA) was used as a benchmark for comparison with the proposed ANN-HS. HPA is a built-in K8s feature that automatically adjusts the number of pod replicas based on utilization metrics, such as CPU usage or custom metrics [1, 2]. The testing approach was similar to the ANN-HS, where the load was varied by the number of users, as depicted in the load profile shown in Fig. 5 and following Eq. 13. The metric used for the HPA was the CPU load per replica, with an average utilization target of 10%. The replicas would be scaled up or down by the HPA to maintain the average CPU resource utilization at about 10% of the total available capacity.

The HPA was deployed with the parameters: *averageUtilization: 10*, *minReplicas: 1*, *maxReplicas: 20*, *scaleDown: stabilizationWindowSeconds: 0*, and *scaleUp:*

stabilizationWindowSeconds: 0. The comparison between the ANN-HS and the HPA allowed for a more comprehensive and detailed analysis of the performance and effectiveness of each horizontal scaling approach. The results were important in identifying the advantages and limitations of each solution and provided valuable insights for enhancing the scalability and adaptability of applications in cloud environments.

4 Results

4.1 ANN training

Figure 6 shows the relationship between the predicted load value for all experiments, y, and the actual load value observed in the dataset, N. The Root Mean Squared Error (RMSE) was calculated as 0.9431. The Mean Squared Error (MSE) was calculated as 0.8895. The coefficient of determination (R^2) was calculated as 0.9989, which is a statistical metric indicating the proportion of variance in the dependent variable that is predictable from the independent variables. A value close to 1 indicates that the model explains the data variability very well. Finally, the Mean Absolute Error (MAE) was calculated as 0.6073. These metrics were obtained exclusively from the validation dataset, demonstrating that the trained regression model performed excellently in predicting load on data that were not used during training.

The use of validation data ensures that the reported performance reflects the model's ability to generalize to new, unseen data rather than merely fitting the training data. The low error values and high R^2 score indicate that the model can accurately predict the load under various conditions, confirming no evidence of overfitting. This supports the conclusion that ANN-HS is effective for load prediction and can be a reliable approach for horizontal scaling in K8s environments.

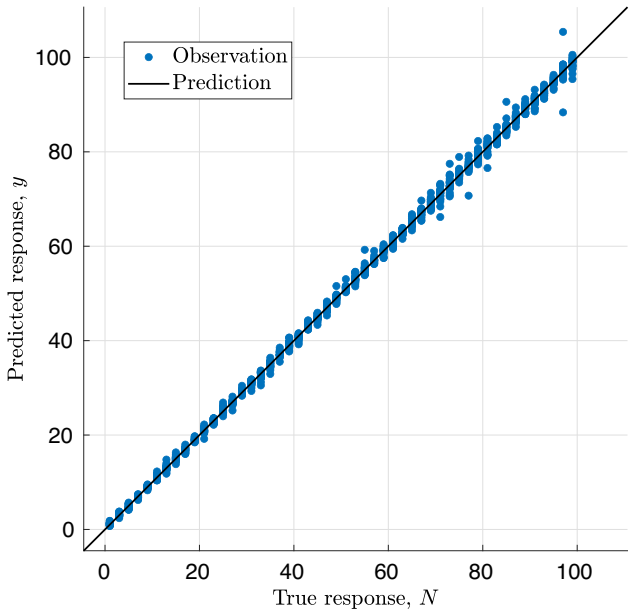

Fig. 6 Relationship between the predicted load value, y, and the actual load value observed in the dataset, N

4.2 ANN-HS testing

Figures 7, 8, and 9 display the curves for CPU consumption in milli-core, $m_k^{\text{cpu}}(t)$, the packet rate $x_k^{\text{rp}}(t)$ per replica in rpm, and the number of replicas $m_k^{\text{p}}(t)$ over time for the experiments $k = 46$ ($L_{46} = 50$).

Meanwhile, Figs. 10, 11, and 12 illustrate the distribution of values associated with the average metrics for CPU consumption, packets received in rpm/replica, and number of replicas over the 180 seconds associated with each k-th test experiment for all L_k values. For each k-th experiment, the values $\bar{m}_k^{\text{cpu}}$, $\bar{x}_k^{\text{rp}}$, and $\bar{m}_k^{\text{p}}$ were generated and can be characterized as

$$\bar{m}_k^{\text{cpu}} = \frac{1}{180}\sum_{t=1}^{180} m_k^{\text{cpu}}(t), \tag{15}$$

$$\bar{x}_k^{\text{rp}} = \frac{1}{180}\sum_{t=1}^{180} x_k^{\text{rp}}(t), \tag{16}$$

and

$$\bar{m}_k^{\text{p}} = \frac{1}{180}\sum_{t=1}^{180} m_k^{\text{p}}(t). \tag{17}$$

Finally, Fig. 13 presents the distribution of the violation rate for a Service Level Agreement (SLA) of latency, δ, for all test experiments. The violation rate for each k-th test experiment, $\mathbf{t}_k$, can be expressed as

$$\delta_k = \frac{1}{M}\sum_{j=1}^{M_k} \beta_{k,j} \tag{18}$$

where M_k is the number of requests made by jMeter during the k-th test and $\beta_{k,j}$ can be expressed as

$$\beta_{k,j} = \begin{cases} 0 & \text{if } r_{k,j} < \text{SLA}_{\max} \\ 1 & \text{if } r_{k,j} \geq \text{SLA}_{\max} \end{cases} \tag{19}$$

where $r_{k,j}$ is the latency of the j-th request of the k-th test and $\text{SLA}_{\max}$ is the maximum allowable violation rate [27]. Figure 13 presents the results for an $\text{SLA}_{\max} = 250\,\text{ms}$.

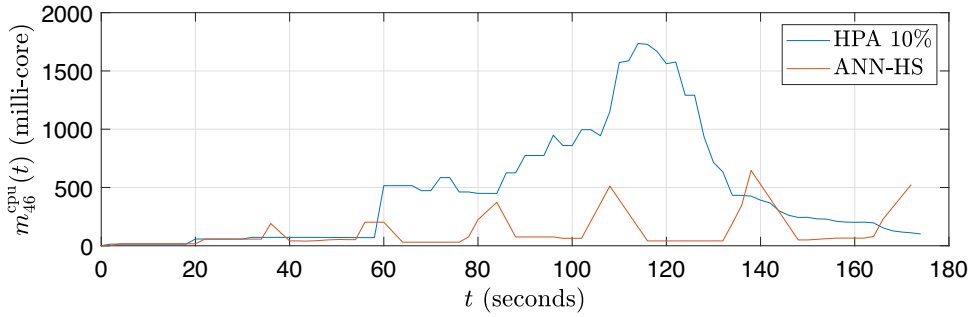

Fig. 7 CPU consumption curves in milli-core for $k = 46$ ($L_{46} = 50$)

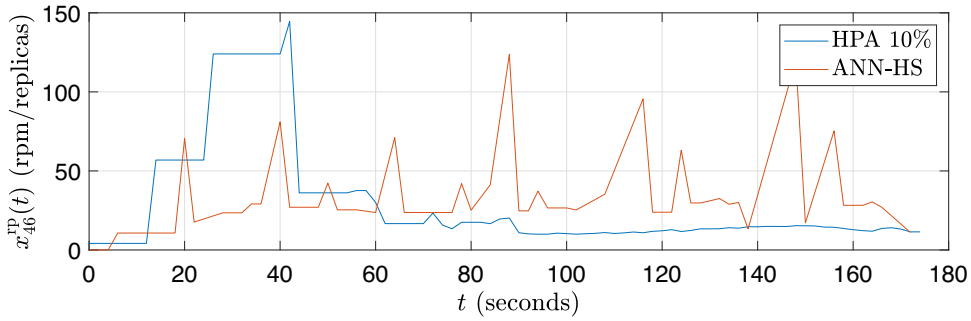

Fig. 8 Curves of packet reception rate in rpm/replica for $k = 46$ ($L_{46} = 50$)

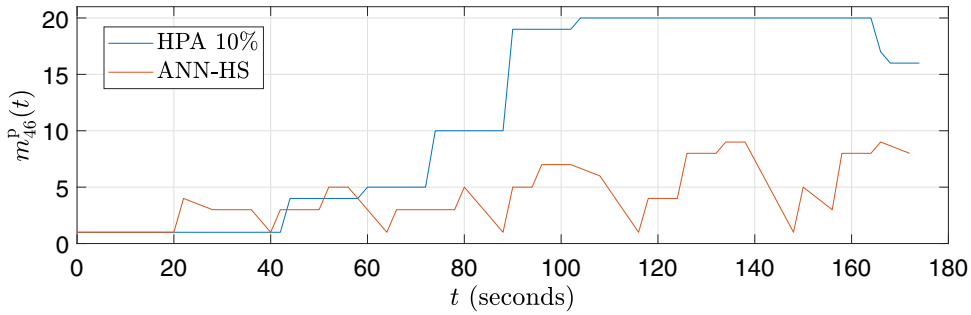

Fig. 9 Curves of the number of replicas for $k = 46$ ($L_{46} = 50$)

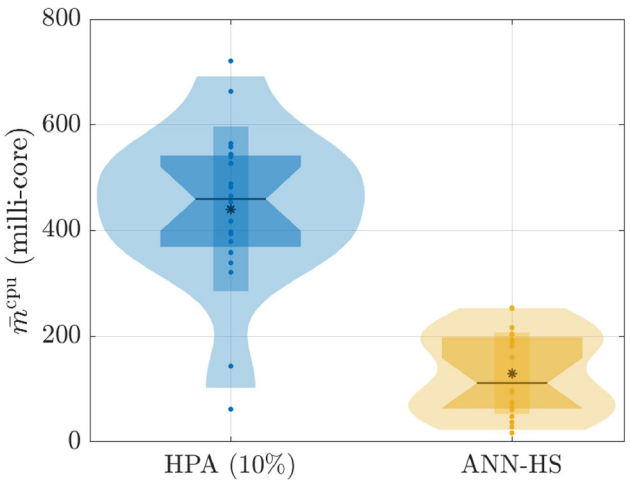

Fig. 10 Distribution of the values associated with the average CPU consumption over 180 seconds for all tested L_k values

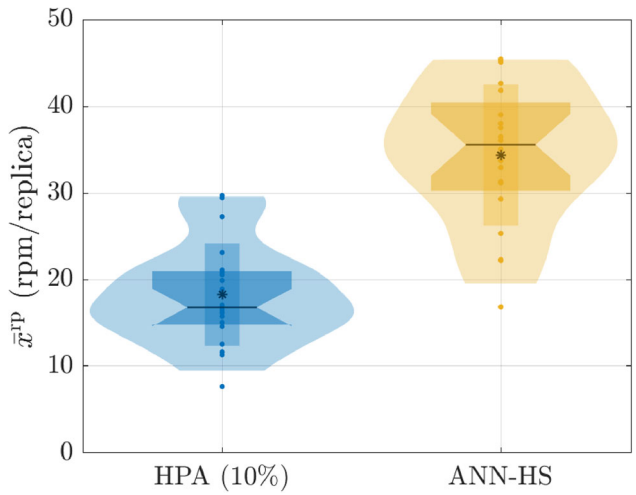

Fig. 11 Distribution of the values associated with the average packet reception rate in rpm/replicas over 180 seconds for all tested L_k values

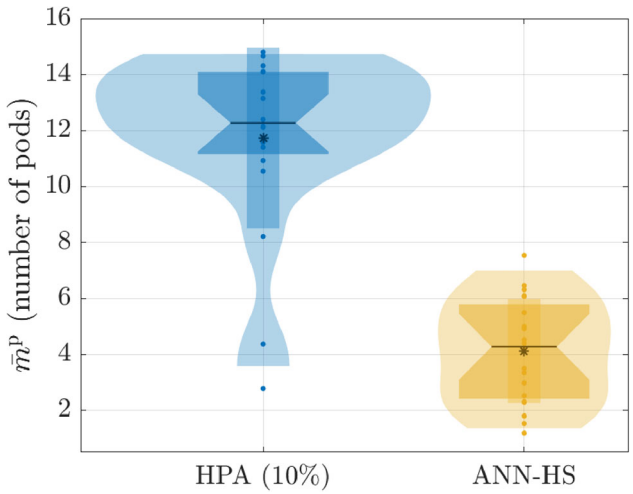

Fig. 12 Distribution of the values associated with the average number of replicas over 180 seconds for all tested L_k values

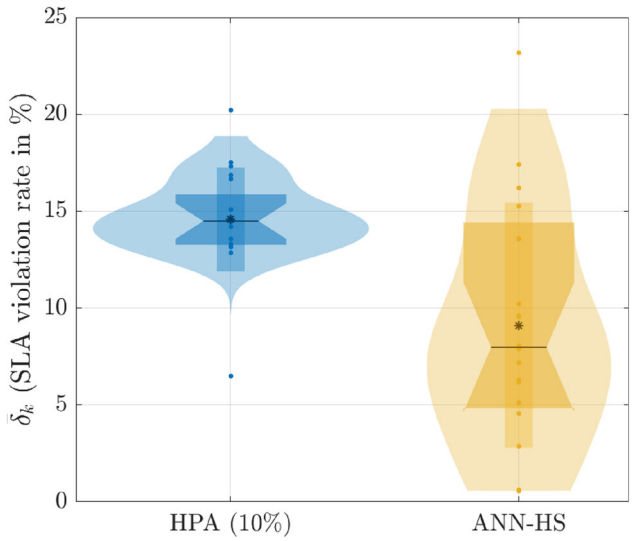

Fig. 13 Distribution of the average associated with the violation rate for a latency SLA for $SLA_{max} = 250\,ms$

5 Discussion

The results highlight the advantages of ANN-HS over HPA, particularly in terms of customization and reduction in the number of required replicas. During experiments conducted under various stress scenarios (changing L_k values) using jMeter, it was observed that ANN-HS maintained lower CPU consumption than HPA. This performance is demonstrated in Fig. 7, which shows the results from a specific test of 180 seconds ($L_k = 50$). This finding is further supported by Fig. 10, which encompasses all the tests conducted.

By deploying fewer replicas than HPA in all experiments, ANN-HS demonstrated the ability to reduce CPU usage rates. This more efficient optimization of workload distribution among these replicas leads to lower CPU consumption per unit, signifying more effective resource utilization and reduced system overhead. This feature is particularly beneficial in stress scenarios, enabling a better balance between resource demand and system capacity. This effect is visible in Fig. 9 for a specific test lasting 180 seconds ($N_k = 50$), and is detailed in Fig. 12 for all conducted tests.

As illustrated in Figs. 8 and 11, ANN-HS behavior results in a higher packet rate per replica in rpm compared to HPA. This difference suggests that HPA tends to underutilize replicas, as it allocates more than ANN-HS to meet the same stress demand (various L_k values).

Significant differences between HPA and ANN-HS are observed concerning the violation rate of the latency SLA of 250 ms, as shown in Fig. 13. While HPA records violations concentrated around 10% to 20%, ANN-HS displays a broader distribution, ranging from 0% to 20% violations. The average violation rate for HPA is around 15%, whereas for ANN-HS it is less than 10%. This difference suggests that ANN-HS more effectively handles variations in load and demand, adapting the number of replicas more appropriately to maintain latency levels within the limits set by the SLA.

One of the key advantages of ANN-HS is its highly accurate and efficient ability to adjust the number of replicas. By analyzing factors such as workload, packet reception rate, and the number of pods or replicas, ANN-HS employs a previously trained regression model to determine the ideal number of replicas needed, thus avoiding excessive resource allocation. This translates to more optimized use of resources available in the K8s environment. The customized flexibility and refined adjustments provided by ANN-HS have the potential to enhance user experience and ensure the quality of service delivered.

Although techniques such as LSTM, GRU, and autoencoders are commonly used for time series forecasting and modeling load patterns, a different approach was adopted in this work. Instead of employing a traditional forecasting method focused on predicting the following load samples based on historical data, the approach concentrated on identifying a multi-dimensional signature of the load profile. Three variables were selected to characterize the application's behavior: CPU, number of replicas (pods), and packet reception rate. These variables were chosen because they capture the current state of the application and its impact on computational resources, providing a view of how the system responds to varying traffic levels.

The approach aimed to optimize real-time dynamic resource adjustment without modeling long-term temporal dependencies, as required with LSTM or GRU. An

experimental dataset was built from multiple traffic scenarios, enabling the model to learn how these three factors behave under different load conditions. By focusing on a multi-dimensional signature rather than forecasting future load, the approach maintained model simplicity while ensuring effective performance in resource allocation. Using more complex models, such as LSTM or GRU, could have added implementation complexity and computational costs without providing proportional benefits in this specific scenario, as the chosen approach already responds to load changes based on current behavior.

6 Comparison with the state of the art

The ANN-HS proposal presents significant advantages compared to the intelligent autonomous autoscaling approaches [14] and autoscaling for serverless platforms [6] based on reinforcement learning. Although reinforcement learning solutions effectively identify thresholds and autonomously adapt scaling policies, they often require considerable time to learn and adjust to workload changes. In contrast, ANN-HS utilizes pre-trained artificial neural networks to provide faster and more accurate load forecasts. This capability allows for dynamic and continuous real-time resource adjustments, resulting in a more agile and efficient response to load variations. Consequently, ANN-HS ensures service quality maintenance and optimizes resource consumption more effectively than reinforcement learning-based approaches.

Regarding the AI-based auto-scaling prediction [15] and elastic scaling using Holt-Winter and GRU [16] approaches, ANN-HS stands out for its more accurate load forecasting and dynamic resource adaptation. Although AI-based prediction integrates multiple methods such as AR, HTM, and LSTM, and the Holt-Winter and GRU method utilizes time series techniques, both approaches can present limitations in scenarios with highly dynamic and complex load patterns. In contrast, ANN-HS uses an artificial neural network specifically trained to predict complex and variable workloads, allowing continuous and precise real-time adjustments in the number of replicas. This results in more effective resource use optimization and service quality maintenance, demonstrating ANN-HS's efficiency and adaptability in Kubernetes environments.

When compared to deep learning-based autoscaling approaches using Bi-LSTM [17, 18] and the double tower deep learning architecture [19], ANN-HS presents distinct advantages. While Bi-LSTM approaches use bidirectional extended short-term memory networks to predict future resource demands and dynamically adjust the number of replicas, and the double tower architecture processes local and global metrics independently before combining their outputs, both can be more complex to implement and adjust. Using a MLP artificial neural network for continuous load forecasting, ANN-HS offers a more straightforward and less complex solution, maintaining high accuracy in load forecasting and resource adaptation. This simplicity, combined with ANN-HS's real-time response capability, results in more efficient resource management and more consistent service quality maintenance in Kubernetes environments, demonstrating its superiority in terms of ease of implementation and operational efficiency.

7 Conclusion

The proposed approach, ANN-HS, significantly improves the horizontal scaling of Kubernetes clusters by utilizing artificial neural networks to predict workloads and dynamically adjust resources. Experimental results show that ANN-HS outperforms traditional methods, such as the HPA, regarding resource consumption efficiency and replica allocation, better adapting to variable demands, and maintaining latency levels within required limits. Additionally, the ANN-HS approach offers a scalable and flexible solution for managing microservices in cloud environments, contributing to the advancement of intelligent resource management in cluster computing. Compared to other state-of-the-art approaches in the literature, ANN-HS stands out for its accuracy in predicting complex and variable workloads, providing continuous and precise real-time adjustments in the number of replicas. This real-time responsiveness, combined with a more straightforward and less complex implementation, makes ANN-HS a superior solution for ease of implementation and operational efficiency. Consequently, ANN-HS represents a significant advancement in optimizing horizontal scaling in K8s, ensuring service quality and effective resource utilization. It demonstrates its applicability and advantages in dynamic and challenging workload scenarios.

ANN-HS contributes to Kubernetes cluster management and horizontal scaling by combining improved accuracy, reduced resource consumption, and simplicity. Its neural network-based approach provides more precise load predictions, enabling better resource optimization and real-time adjustments to fluctuating demands. Additionally, ANN-HS simplifies implementation compared to more complex techniques, making it a practical and effective solution for enhancing the efficiency and reliability of cloud-based microservices. Future research directions could focus on extending the ANN-HS model to support a broader range of workload types, including batch processing and data-intensive applications, to enhance its versatility further. Integrating ANN-HS with other cloud-native technologies, such as service meshes or serverless

computing frameworks, could also be explored to improve interoperability and resource management across diverse environments. Additionally, hybrid scaling strategies that combine horizontal and vertical scaling can be investigated to optimize resource usage by adjusting the number of replicas and tuning the resource limits of individual pods, thereby providing a more comprehensive scaling approach.

Acknowledgements The authors would like to express their gratitude to the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for providing financial support. This research was also made possible with the support of the InovAI Laboratory at UFRN and the Santa Cruz Campus of IFRN.

Author Contributions All authors have contributed in various degrees to ensure the quality of this work (e.g., L.M.D.d.S., P.V.A.A., S.N.S, and M.A.C.F. conceived the idea and experiments; L.M.D.d.S., P.V.A.A., S.N.S, and M.A.C.F. designed and performed the experiments; L.M.D.d.S., P.V.A.A., S.N.S., and M.A.C.F. analyzed the data; L.M.D.d.S., S.N.S, and M.A.C.F wrote the paper. L.M.D.d.S and M.A.C.F. coordinated the project). All authors have read and agreed to the published version of the manuscript.

Funding Not applicable

Data availability The datasets generated and/or analysed during the current study are available in the Mendeley Data repository, https://data.mendeley.com/datasets/ks9vbv5pb2/1.

Materials availibility Not applicable

Code availibility Not applicable

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical approval Not applicable

Consent to participate Not applicable

Consent for publication All authors agreed with the content and gave explicit consent to submit.

References

1. Tran, M.-N., Vu, D.-D., Kim, Y.: A survey of autoscaling in kubernetes. In: 2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 263–265 (2022). IEEE
2. Huo, Q., Li, S., Xie, Y., Li, Z.: Horizontal pod autoscaling based on kubernetes with fast response and slow shrinkage. In: 2022 International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC), pp. 203–206 (2022). IEEE
3. Kuranage, M.P.J., Hanser, E., Nuaymi, L., Bouabdallah, A., Bertin, P., Al-Dulaimi, A.: Ai-assisted proactive scaling solution for cnfs deployed in kubernetes. In: 2023 IEEE 12th International Conference on Cloud Networking (CloudNet), pp. 265–273 (2023). IEEE
4. Augustyn, D.R., Wyciślik, Ł, Sojka, M.: Tuning a kubernetes horizontal pod autoscaler for meeting performance and load demands in cloud deployments. Appl. Sci. **14**(2), 646 (2024)
5. Senjab, K., Abbas, S., Ahmed, N., Khan, A.U.R.: A survey of kubernetes scheduling algorithms. J. Cloud Comp. **12**(1), 87 (2023)
6. Zafeiropoulos, A., Fotopoulou, E., Filinis, N., Papavassiliou, S.: Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms. Sim. Modell. Prac. Theory **116**, 102461 (2022)
7. Tamiru, M.A., Tordsson, J., Elmroth, E., Pierre, G.: An experimental evaluation of the kubernetes cluster autoscaler in the cloud. In: 2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 17–24 (2020). https://doi.org/10.1109/CloudCom49646.2020.00002
8. Balla, D., Simon, C., Maliosz, M.: Adaptive scaling of kubernetes pods. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–5 (2020). https://doi.org/10.1109/NOMS47738.2020.9110428
9. Nguyen, H.T., Van Do, T., Rotter, C.: Scaling upf instances in 5g/6g core with deep reinforcement learning. IEEE Access **9**, 165892–165906 (2021)
10. Horizontal Pod Autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/. Accessed: 2024-06-13 (2024). https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
11. Nguyen, T.-T., Yeom, Y.-J., Kim, T., Park, D.-H., Kim, S.: Horizontal pod autoscaling in kubernetes for elastic container orchestration. Sensors (2020). https://doi.org/10.3390/s20164621
12. Shim, S., Dhokariya, A., Doshi, D., Upadhye, S., Patwari, V., Park, J.-Y.: Predictive auto-scaler for kubernetes cloud. In: 2023 IEEE International Systems Conference (SysCon), pp. 1–8 (2023). https://doi.org/10.1109/SysCon53073.2023.10131106
13. Silva, S.N., Goldbarg, M.A.S.d.S., Silva, L.M.D.d., Fernandes, M.A.C.: Application of fuzzy logic for horizontal scaling in kubernetes environments within the context of edge computing. Future Internet **16**(9) (2024) https://doi.org/10.3390/fi16090316
14. Khaleq, A.A., Ra, I.: Intelligent autoscaling of microservices in the cloud for real-time applications. IEEE Access **9**, 35464–35476 (2021)
15. Toka, L., Dobreff, G., Fodor, B., Sonkoly, B.: Adaptive ai-based auto-scaling for kubernetes. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pp. 599–608 (2020). https://doi.org/10.1109/CCGrid49817.2020.00-33
16. Yuan, H., Liao, S.: A time series-based approach to elastic kubernetes scaling. Electronics (2024). https://doi.org/10.3390/electronics13020285
17. Dang-Quang, N.-M., Yoo, M.: Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. Appl. Sci. (2021). https://doi.org/10.3390/app11093835
18. Yan, M., Liang, X., Lu, Z., Wu, J., Zhang, W.: Hansel: Adaptive horizontal scaling of microservices using bi-lstm. Appl. Soft Comp. **105**, 107216 (2021). https://doi.org/10.1016/j.asoc.2021.107216
19. Violos, J., Tsanakas, S., Theodoropoulos, T., Leivadeas, A., Tserpes, K., Varvarigou, T.: Intelligent horizontal autoscaling in edge computing using a double tower neural network. Comp. Netw. **217**, 109339 (2022). https://doi.org/10.1016/j.comnet.2022.109339
20. Zerwas, J., Krämer, P., Ursu, R.-M., Asadi, N., Rodgers, P., Wong, L., Kellerer, W.: KapetÁnios: Automated kubernetes adaptation through a digital twin. In: 2022 13th International Conference on Network of the Future (NoF), pp. 1–3 (2022). https://doi.org/10.1109/NoF55974.2022.9942649

21. Toka, L., Dobreff, G., Fodor, B., Sonkoly, B.: Machine learning-based scaling management for kubernetes edge clusters. IEEE Trans. Netw. Ser. Manag. **18**(1), 958–972 (2021). https://doi.org/10.1109/TNSM.2021.3052837
22. MicroK8s: Lightweight Kubernetes. https://microk8s.io/. Acesso em: 18-07-2023
23. The Apache Software Foundation: Apache JMeter. https://jmeter.apache.org/. Acesso em: 18-07-2023 (2023)
24. The Prometheus Authors: Prometheus. https://prometheus.io/. Acesso em: 18-07-2023 (2023)
25. Fernandes, M.: Horizontal Scaling in Kubernetes Dataset Using Artificial Neural Networks for Load Forecasting (2024). https://doi.org/10.17632/ks9vbv5pb2.1
26. Red Hat: Fabric8 Kubernetes-Client. https://github.com/fabric8io/kubernetes-client. Acesso em: 18-07-2023 (2023)
27. Xiao, Z., Hu, S.: Dscaler: A horizontal autoscaler of microservice based on deep reinforcement learning. In: 2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS), pp. 1–6 (2022). https://doi.org/10.23919/APNOMS56106.2022.9919994

Lucileide M. D. da Silva received a dual BS degree in Electronic Engineering through a joint program between the École Nationale Supérieure d'Électronique, d'Electrotechnique, d'Informatique et d'Hydraulique et des Télécommunications (ENSEEIHT) in Toulouse, France, and the Federal University of Rio Grande do Norte (UFRN), Natal, Brazil, completed in 2011 and 2012, respectively. She earned her MS and Ph.D. in Electrical and Computer Engineering from UFRN in 2016 and 2023. During her doctoral studies, she served as a Visiting Researcher at King's College London's Centre for Telecommunications Research, UK. Currently, she is an Assistant Professor at the Federal Institute of Rio Grande do Norte (IFRN) in Santa Cruz, Brazil, actively involved with the IFRN Robotics Group and the InovAI research lab at UFRN. Her research interests include AI, reconfigurable hardware, and educational technology.

Pedro V. A. Alves received a B.S. degree in Science and Technology in 2018 and a B.S. degree in Computer Engineering in 2021, both from the Federal University of Rio Grande do Norte (UFRN), Natal, Brazil. He completed his M.S. degree in Electrical and Computing Engineering in 2023 at UFRN. Currently, he is currently pursuing a Ph.D. in Electrical and Computing Engineering at UFRN. He is also a member of the InovAI Lab. His research interests include Tactile Internet, Embedded Systems, Generative AI, Machine Learning, and Artificial Intelligence.

Sérgio N. Silva received a BS degree in Science and Technology in 2012 and a BS degree in Computer Engineering in 2014, both from the Federal University of Rio Grande do Norte (UFRN), Natal, Brazil. He completed his MS in Computer Engineering in 2016 and his Ph.D. in Computer Engineering in 2021, both at UFRN. Currently, he holds a post-doctoral position with the Graduate Program in Electrical and Computer Engineering at UFRN. He is also a member of the InovAI Lab. His research interests include Generative AI, Deep Learning, Machine Learning, Artificial Intelligence, Tactile Internet, Embedded Systems, Data Analysis, and Reconfigurable Computing.

Marcelo A.C. Fernandes received BS degree in Electrical Engineering in 1997, MS degree in Electrical Engineering in 1999, from the Federal University of Rio Grande do Norte, Natal, Brazil, and Ph.D. degree in Electrical Engineering in 2010, from the University of Campinas, Campinas, SP, Brazil. He is an Associate Professor in the Department of Computer Engineering and Automation, Federal University of Rio Grande do Norte, Natal, Brazil. From 2015 to 2016, he worked as a visiting researcher in Centre Telecommunication Research (CTR) at King's College London, in London, UK. From 2019 to 2021, he worked as a visiting scholar in the John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, USA. He is the leader of the InovAI Lab and senior researcher at the Leading Advanced Technologies Center of Excellence (LANCE). His research interests include artificial intelligence, deep learning, digital signal processing, embedded systems, reconfigurable and hardware.