Neural Network-Enhanced Self-Organization for Efficient Resource Allocation in Distributed Edge Computing Environments

Samira Hayat
Lakeside Labs GmbH
Klagenfurt, Austria
hayat@lakeside-labs.com

Melanie Schranz
Lakeside Labs GmbH
Klagenfurt, Austria
schranz@lakeside-labs.com

Loris Cannelli
Dalle Molle Institute for Artificial Intelligence
University of Applied Sciences of Switzerland
Lugano, Switzerland
loris.cannelli@supsi.ch

2025 IEEE 9th International Conference on Fog and Edge Computing (ICFEC) | 979-8-3315-9457-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICFEC65699.2025.00015

***Abstract*—Containerization has transformed application deployment across diverse environments in the edge-cloud computing continuum, providing lightweight, portable solutions that integrate seamlessly across various cloud-native environments. However, default scheduling often leads to resource underutilization due to overestimated requests. This results in inaccurate demand prediction and static allocation strategies. This paper enhances a bottom-up, self-organizing approach with an NN-based peer selection mechanism, enabling intelligent resource allocation while maintaining decentralization. The proposed NN-based approach enables a data-driven selection strategy that significantly improves resource utilization and system performance; a 21% improvement in resource utilization is reported in a high-traffic scenario compared to the predecessor framework. The findings contribute to the broader research community by demonstrating the potential of combining agent-based modeling with machine learning techniques, paving the way for more adaptive and efficient edge orchestration systems.**

***Index Terms*—Edge computing, workload allocation, agent-based modeling, NN-based decision**

I. Introduction

The rise of containerization has transformed application deployment, providing lightweight and portable solutions that integrate seamlessly across diverse environments in the edge-cloud computing continuum. Containers enable developers to optimize native cloud architectures through continuous integration and deployment practices that abstract the underlying infrastructure. Kubernetes[1], the leading open source container orchestration framework, automates the deployment, scaling, and management of containerized applications. However, its default scheduling mechanisms often result in resource underutilization, primarily due to overestimated resource requests, resulting in inaccurate demand prediction and static allocation strategies. Edge environments face unique challenges due to dynamic workloads, constrained resources, and real-time processing demands [2], [5]. While edge computing offers benefits like enhanced security, reliability, and reduced latency, managing the distributed edge computing environment is complex. Autonomy and local decision-making are crucial for real-time applications [15], [16], with stringent QoS requirements like low latency across multiple sites [1].

This paper is based on [14], which proposes a novel distributed scheduling approach that uses self-organization principles to implement a bottom-up strategy that addresses these limitations. The approach prioritizes the allocation of rigid (QoS-sensitive) pods while exploiting the slack resources for elastic (QoS-insensitive) pods, leveraging agent-based modeling (ABM) to simulate and optimize scheduling decisions. Although effective, the original method used a random peer selection mechanism for elastic pod allocation, which limits its ability to fully capitalize on available resources.

Recent ML advances have improved resource allocation in edge computing, challenging centralized strategies. This paper enhances the bottom-up framework with an NN-based peer selection mechanism, using a two-hidden-layer MLP trained on data from the best-performing algorithm in [14]. This approach replaces random selection, enabling smarter resource allocation while maintaining decentralization. The research improves scheduling efficiency while preserving edge environment agility and scalability, demonstrating the potential of combining ABM with ML for adaptive edge orchestration systems. We compare our extended framework against the baseline best, original bottom-up, and a purely NN-based approach. Results show significant improvements in resource utilization and workload balancing, demonstrating the benefits of ML-driven peer selection in edge computing. This integration of learning-based methods with ABM bridges the gap between theoretical strategies and practical implementation, advancing edge resource management.

The paper is organized as follows. Sec. II describes the state-of-the-art, and how the proposed solution differs from existing work. Sec. III provides insights into the system model, followed by a description of NN-based peer selection in Sec. IV. Sec. V presents the results and Sec. VI concludes the paper.

Funded by the European Union, project ACES, by grant No. 101093126. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

[1]https://kubernetes.io/ [Online, last access 30-12-2024]

2694-3255/25/$31.00 ©2025 IEEE
DOI 10.1109/ICFEC65699.2025.00015

II. Related Work

Containerization, a core of cloud-native architectures, enables efficient application deployment and resource allocation. Kubernetes, the leading orchestration framework, uses pods for resource management [4]. However, its default rule-based scheduling often leads to inefficiencies, particularly in dynamic, resource-constrained edge environments [8].

To address these limitations, advanced ML techniques have been applied to optimize Kubernetes scheduling. For example, NNs dynamically adjust pod resources in [6], while RL enhances container placement in [5] and improves workload adaptation in [9]. Convolutional NNs in [8] focus on real-time resource prediction. Despite these advancements, most approaches emphasize global optimization, often neglecting the interplay between local and global decisions in peer selection.

In latency-critical edge environments, mismanagement intensify resource contention. ROSE [7] introduces speculative oversubscription but risks QoS violations. Forecasting methods like ARIMA [10] and hybrid models (e.g., ARIMA-LSTM [18]) improve accuracy but struggle with microbursts and workload volatility highlighting the need for robust prediction frameworks tailored to heterogeneous edge scenarios.

Advances in resource allocation leverage deep RL and user-centric scheduling. For example, [2] demonstrates deep RL's potential for QoE-driven resource allocation in heterogeneous edge-IoT environments. Similarly, [3] uses dynamic NNs for predictive resource allocation, balancing workloads in high-demand scenarios. Comparative analyses of RL methods like A3C and PPO [12] highlight performance-resource efficiency trade-offs, aiding model selection based on application needs. Additionally, [13] emphasizes deep NNs' adaptability for resource allocation in mobile edge computing environments under evolving network conditions.

The methodology proposed enhances bottom-up resource orchestration [14] by integrating a two-hidden-layer MLP for peer selection, improving adaptability over prior work. Unlike [6]'s global scaling focus, our approach emphasizes local edge-node slack resource utilization. While [5] applies RL for container placement, it neglects vertical resource scaling within pods. Our MLP algorithm enables dynamic peer assignment with QoS guarantees, contrasting ROSE's speculative methods by using NN-driven slack allocation for lower contention. The hybrid strategy merges local/global optimization strengths, reducing resource waste. Simulations leverage MESA's ABM framework [11] to analyze emergent behaviors, complementing traditional ML approaches.

III. System Model

The system comprises distributed *edge edge micro data centers* with limited CPU/memory/storage resources. Applications run as Kubernetes *pods* (smallest deployable units containing containers), aiming for pod-to-node assignments that optimize resource use while meeting QoS (response time, reliability). Pods are categorized as *rigid pods* (strict QoS requiring overprovisioned resources, creating slack) or *elastic pods* (tolerate allocation variability/delays). This distinction is critical in resource-constrained, dynamic edge environments where computational contention is high.

Each edge node maintains a resource inventory, including the total, allocated, and available CPU and memory resources. Pods arrive at the system dynamically, with their resource demand vectors (CPU and memory demand) and QoS requirements specified in their deployment configurations. The objective of the scheduler is twofold: (i) to ensure the placement of rigid pods while guaranteeing their QoS requirements, and (ii) to utilize slack resources to maximize overall utilization.

A. Workload Allocation Process

The workload allocation process has two sub-processes:

1) **Rigid Pod Allocation:** Incoming rigid pods are prioritized and assigned CPU and memory according to their exact resource requirements (including slack). The scheduler ensures that the QoS constraints of these pods are met by avoiding overloading.
2) **Elastic Pod Allocation:** Elastic pods are then considered for allocation using the slack resources left unutilized by the deployed rigid pod.

In the following, the deployed rigid pod, whose slack resources are utilized by an elastic pod for execution, is called its "peer rigid pod", a collection of these is called "peer pods".

IV. Neural Network-Based Peer Selection

The original bottom-up resource orchestration framework [14] proposes a computationally efficient approach for the resource allocation process described above. The edge node categorizes the deployed rigid pods based on the pod's slack resources into **buckets**. There are four buckets when considering (CPU slack, memory slack): (LL), (LH), (HL), and (HH), with L for low and H for high slack. The node maintains a lookup table of the four buckets containing their deployed pods. When an elastic pod arrives, a bucket is selected on the basis of the incoming pod's resource demand, and a peer rigid pod is chosen uniformly at random from that bucket. This approach scales efficiently while leveraging initial slack estimates for informed assignments. However, while computationally lightweight, random assignment does not exploit the potential for (near-)optimal decision making.

To enhance the resource allocation strategy, this work integrates an NN-based approach for peer pods selection. The NN model uses a data set generated by the *Best* algorithm (presented below), which computes an optimal mapping of elastic pods to rigid peer pods with full system information.

1) Training Data Generation: The *Best* algorithm generates training data by evaluating the fitness of potential peer pods for each incoming elastic pod. This algorithm is very expensive in time and computation as it goes through the entire list of deployed rigid pods to find the most optimal match between the incoming elastic pod and the available rigid pods. The fitness computation considers:

- **Resource Match (f1):** The difference between the slack resources of the deployed rigid pod and the demand vector of the elastic pod.
- **Temporal Match (f2):** The difference between the remaining execution steps of the deployed rigid pod (t_{exe_r}) and the execution steps of the elastic pod (t_{exe_e}).

These metrics ensure a comprehensive evaluation of the suitability of the peer rigid pod. The training data set includes features such as the demand vector of the incoming elastic pod, slack resources of each deployed rigid pod, and the measured fitness scores (f1, f2) for the pairs of rigid-elastic pods. We use λ to represent the arrival rate of the pods, which is tuned to test the system under light and heavy loads.

2) Neural Network Training: For this task, we use an MLP model. The MLP consists of two hidden layers with 64 and 32 neurons, respectively. The model uses rectified linear unit (ReLU) as the neuron activation function and mean squared error (MSE) as the cost function for training. It is trained on a normalized dataset using the following features:

- CPU and memory demand of the new elastic pod.
- Slack resources of the deployed rigid pods.
- Demand steps of the elastic pod and remaining steps of the deployed rigid pods.

The output of the model predicts the fitness scores (f1, f2) of the elastic and deployed rigid pods, enabling informed decision-making.

3) Peer Selection Process: The peer selection process is shown in Algorithm 1. At runtime, the trained MLP model (named *nn_model*) is used to predict the fitness of the potential peer pods (*candidate_peers*) for the incoming elastic pod (*new_pod*). The best rigid peer pod is selected based on the minimum predicted f1 score. For candidates with similar f1 scores, the model uses both f1 and f2 for the final decision.

Input: List of deployed rigid pods $candidate\_peers$, incoming pod new_pod, trained neural network nn_model
Output: Best peer pod $best\_peer$
$best\_peer \leftarrow$ None
$current\_pods \leftarrow$ Deployed pods in $candidate\_peers$
if $current\_pods = \emptyset$ **then**
  return None
end
for *each pod in* $current\_pods$ **do**
  Prepare input vector $X \leftarrow$ [new pod demand, peer pod slack, demand steps]
  Normalize X
  Predict fitness $[f1, f2] \leftarrow$ nn_model.predict(X)
  Store $peer\_fitness[pod] \leftarrow [f1, f2]$
end
$best\_peer \leftarrow$ Pod with min $f1$ (and $f2$, if required)
return $best\_peer$

Algorithm 1: NN-Based Peer Pod Selection Algorithm

A. Algorithm

The algorithm proposed in this paper, based on the Bottom-up framework, is called *Bottom-up with NN* and uses Algorithm 1 for peer pod selection. The edge node maintains a lookup table of all the deployed rigid pods in the four buckets, as in the original bottom-up framework [14]. When a new elastic pod arrives, a bucket is selected according to the resource demands of the pod. Thus, for our solution, the *candidate_peers* in Algorithm 1 is a list that contains potential peer pods in the selected bucket (which is a subset of all deployed rigid pods). The algorithm predicts the fitness of the elastic pod against all the rigid pods in that bucket using the pre-trained NN model. The best peer rigid pod selected from the bucket has the best fitness with the elastic pod based on the predicted values of f1 (and f2). This approach enables a data-driven, adaptive bucket and peer pod selection strategy significantly improving resource utilization and system performance.

B. Performance Metrics

The performance of the proposed framework is evaluated based on the following key metrics:

- **Resource Utilization:** Are the slack resources (CPU and memory) utilized efficiently?
- **Satisfaction Rate:** Is the approach improving the number of deployed (rigid and elastic) pods?

We compare the algorithm implemented in this paper (Bottom-up with NN) with two algorithms presented in [14] (Best and Bottom-up) and a bucketless NN-based peer selection algorithm. The algorithms vary in their elastic pod deployment process. Upon arrival of an elastic pod:

The *Best* algorithm selects its best match among the deployed rigid pods, based on the matching scores f1 (and f2). Although the resultant resource utilization and QoS satisfaction is optimal, this algorithm is computationally demanding, as it requires calculating the matching scores of the elastic pod against all deployed rigid pods to find the optimal match.

The *Bottom-up* algorithm randomly assigns a peer rigid pod from the bucket containing candidate peer pods that may satisfy the elastic pod's demands. The solution is computationally efficient, although using buckets and random selection negatively impacts performance metrics.

The *NN-basic* uses Algorithm 1 to predict the matching scores of the elastic pod against *all* deployed rigid pods for optimal peer pod assignment. The algorithm requires a large amount of training data for a well-trained model. Similar to the *Best* algorithm, this one is computationally demanding, as the matching score prediction is performed against all deployed rigid pods. More specifically, in *NN-basic* the *candidate_peers* list in Algorithm 1 contains all deployed rigid pods, compared to *Bottom-up with NN*, where *candidate_peers* contains only the pods inside the selected bucket, as mentioned previously.

V. Results

For the simulation setup, we use a set of pod profiles (small, medium, large), arrival probabilities (0.4 for small, 0.4 for medium, 0.2 for large), and available CPU/Memory capacity

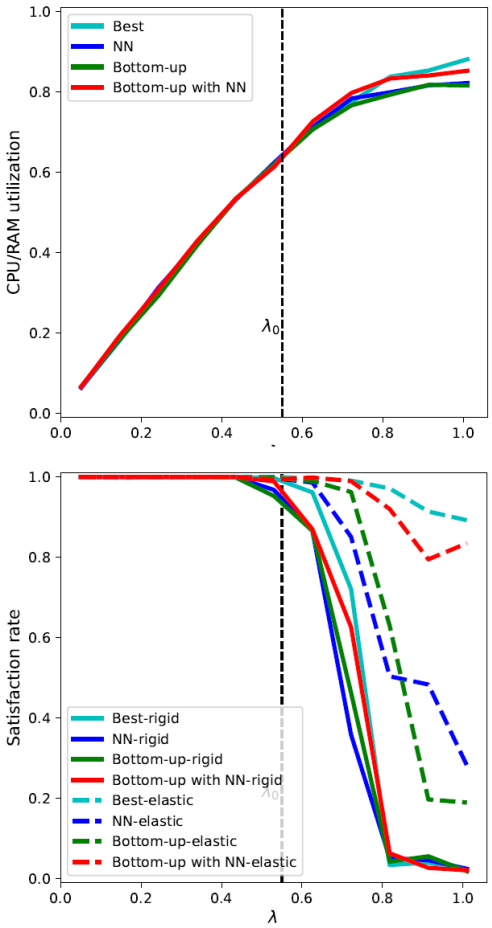

Fig. 1. Performance comparison for different algorithms in terms of resource utilization versus arrival rate (left), and satisfaction rate versus arrival rate (right).

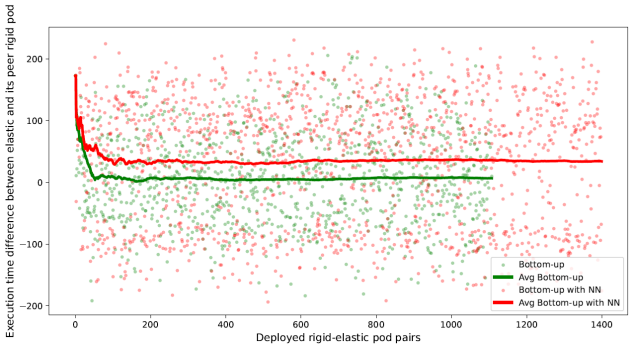

Fig. 2. Difference between deployed rigid-elastic pod pairs execution time versus the number of deployed pod pairs for *Bottom-up* and *Bottom-up with NN* algorithms in lightly-loaded system ($\lambda = 0.4$).

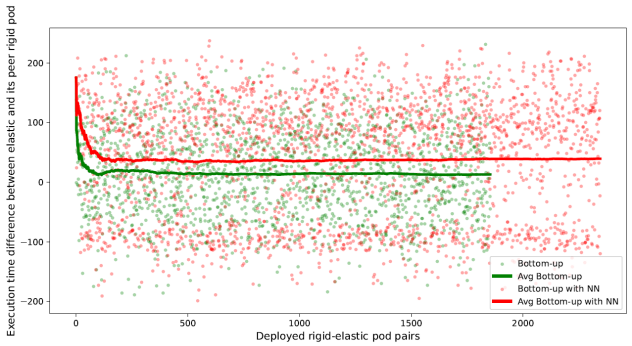

Fig. 3. Difference between deployed rigid-elastic pod pairs execution time versus the number of deployed pod pairs for *Bottom-up* and *Bottom-up with NN* algorithms in heavily-loaded system ($\lambda = 0.8$).

(512×512, details in [14]). With the considered pod profiles and resources, computations in [14] show that beyond the pod arrival rate of $\lambda_0 = 0.55$, the queue will start building up; i.e., at λ_0, the average demand equals the resource capacity. Pod categorization into buckets is done with CPU/Memory slack of less than 5 considered as L, and H otherwise.

We implement the multi-agent system in Python and run the simulations using the MESA library [11] for 12,000 simulation steps. With the "never enough training data" problem of ML in mind, we trained our NN model on 120,000 data points[2]. We expect that NNs more sophisticated than the MLP-based network selected here may perform better. The comparison of different NN models is out of scope for this work.

We start by comparing the performance of the *Bottom-up with NN* algorithm against the three algorithms mentioned in Section IV-B. Fig. 1 shows that, as expected, the *Best* algorithm excels both in resource utilization and satisfaction rate. The original *Bottom-up* algorithm and *NN-basic* algorithm perform similarly in terms of the evaluated metrics. We see a great improvement in performance with the proposed *Bottom-up with NN* algorithm, where peer pod selection from the buckets is facilitated by an NN instead of being randomized. Infact, the resource utilization of *Bottom-up with NN* is much closer to the curve for the *Best* algorithm than the other two algorithms, as shown in Fig. 1. An improvement in the satisfaction rate is observed compared to the original *Bottom-up* algorithm, with a quite drastic improvement for the elastic pods without compromising the rigid pods satisfaction rate.

To further evaluate what causes the improvement in satisfaction rates, we compare the difference in remaining execution time of the deployed *rigid-elastic pod pairs* (i.e., the elastic pod and its peer rigid pod). Fig. 2 and 3 report the results for lightly loaded ($\lambda = 0.4$) and heavily loaded ($\lambda = 0.8$) systems, respectively. The dots represent the difference between the remaining execution time of the peer rigid pod and its elastic pod, i.e., $t_{exe_r} - t_{exe_e}$. Positive values indicate longer execution times for rigid pods compared to their elastic peers, while negative values indicate the opposite. This approach prioritizes rigid pods, potentially tying up resources for future rigid pods,

[2]Simulations showed the NN-basic algorithm trained on 2,000 data points performed worse than random peer selection [14], while training with 12,000 data points achieved performance comparable to the original *Bottom-up* approach.

as elastic pods are deployed only on slack resources. The line shows the average execution time difference between peer rigid and elastic pods.

The figures show that the reported number of deployed pod pairs (x-axis) for both lightly and heavily loaded systems is much higher for the *Bottom-up with NN* algorithm compared to the *Bottom-up* algorithm. For the lightly loaded system, the increase is 18% and for heavily loaded system, it is 21%, as reported in Fig. 2 and 3. This explains the improvement in the satisfaction rate for the *Bottom-up with NN* algorithm reported in Fig. 1. The reason is that our solution selects the optimal predicted peer rigid pod from the selected bucket compared to the *Bottom-up* algorithm, where a peer rigid pod is selected randomly. As bucket-based frameworks suffer from low granularity, there is a high probability that a random peer rigid pod may not satisfy the demands of the elastic pod. In comparison, our solution's choice of predicted optimal peer rigid pod leads to a higher number of deployed rigid-elastic pod pairs. Furthermore, selecting a peer rigid pod from the bucket based on available resources *as well as* execution time has an additional advantage: the probability that the elastic pod will finish execution close to the peer rigid pod's execution time is much higher, i.e., $t_{exe_e} \approx t_{exe_r}$; as a result, even when t_{exe_e} is higher than t_{exe_r}, we expect the elastic pod to finish execution soon after the peer rigid pod, thus releasing the resources for future rigid pods to be deployed.

Fig. 2 and 3 also show that more pod pairs have time difference towards the positive part of the y-axis for *Bottom-up with NN* than for *Bottom-up*. This means that peer rigid pods take longer to execute than their elastic pods for most pod pairs in the *Bottom-up with NN* approach. The freed slack resources are used to deploy additional elastic pods in the next steps. This contributes as a further explanation for the increased number of pod pairs deployed (Fig. 2 and 3), and for the increased satisfaction rate for the elastic pods in Fig. 1 using our proposed solution.

VI. Conclusions

This paper introduces an enhanced resource allocation approach for distributed edge computing environments by augmenting a bottom-up framework with an NN-based peer pod selection mechanism [14]. The proposed method employs a multilayer perceptron with two hidden layers, trained on data from the best-performing algorithm, to improve upon the original random selection strategy. This NN-driven approach maintains decentralization while enhancing adaptive scheduling decisions, resulting in improved resource utilization, workload balancing, and increased processing of rigid-elastic pod pairs. The results of this study pave the way for future research combining ABM with machine learning techniques for edge orchestration systems. Further exploration of more sophisticated learning models and hybrid orchestration strategies could lead to even greater advancements in edge resource management.

In conclusion, this research highlights the potential of NN-driven decision-making to enhance resource allocation in dynamic, resource-constrained edge environments while avoiding computation overload.

References

[1] M. Schranz, K. Harshina, P. Forgacs and F. Buining, "Agent-based Modeling in the Edge Continuum using Swarm Intelligence," *in Proc. Int. Conf. Adapt. and Self-Adapt. Sys. & Apps.*, pp. 1-7, Apr. 2024.

[2] I. Alqerm and J. Pan, "DeepEdge: A New QoE-Based Resource Allocation Framework Using Deep Reinforcement Learning for Future Heterogeneous Edge-IoT Applications," in *IEEE Trans. Netw. and Serv. Mgm.*, vol. 18, no. 4, pp. 3942-3954, Dec. 2021.

[3] L. Ma et al., "Dynamic Neural Network-Based Resource Management for Mobile Edge Computing in 6G Networks," in *IEEE Trans. Cogn. Comm. and Net.*, vol. 10, no. 3, pp. 953-967, June 2024.

[4] K. Senjab, S. Abbas, N. Ahmed, "A survey of Kubernetes scheduling algorithms." in *J. Cloud Comp.*, vol. 12, no. 87, June 2023.

[5] Z. Chen and B. Zhu, "Deep Reinforcement Learning Based Container Cluster Placement Strategy in Edge Computing Environment," in *IEEE Glob. Comm. Conf.*, pp. 2212-2217, Dec. 2022.

[6] X. Wang et al., "Scalable Resource Management for Dynamic MEC: An Unsupervised Link-Output Graph Neural Network Approach," in *IEEE Int. Symp. Personal, Indoor and Mobile Radio Comm.*, pp. 1-6, Oct. 2023.

[7] X. Sun et al., "ROSE: Cluster Resource Scheduling via Speculative Over-Subscription," in *IEEE Int. Conf. Dist. Comput. Sys.*, pp. 949-960, July, 2018.

[8] T. Kamble et. al. "Predictive Resource Allocation Strategies for Cloud Computing Environments Using Machine Learning". in *J. Elect. Sys*. vol. 19. pp. 68–77, Dec. 2023.

[9] Y. Wei, L. Pan, S. Liu, L. Wu and X. Meng, "DRL-Scheduling: An Intelligent QoS-Aware Job Scheduling Framework for Applications in Clouds," in *IEEE Access*, vol. 6, pp. 55112-55125, Sep. 2018.

[10] Y. X. Chia, C. K. Seow, K. Chen and Q. Cao, "Exploring Resource Prediction Models Based on Custom Kubernetes Auto-scaling Metrics," in *Int. Conf. Cloud Comput. & Big Data Analyt.*, pp. 47-52, Apr. 2024.

[11] D. Masad and J. L. Kazil, "Mesa: Simulation frameworks for AI-based Kubernetes orchestration," presented at the SciPy Conf., Austin, TX, USA, Jul. 2023.

[12] A. d. Rio, D. Jimenez and J. Serrano, "Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments," *IEEE Access*, vol. 12, pp. 146795-146806, 2024.

[13] J. Li and T. Lv, "Deep Neural Network Based Computational Resource Allocation for Mobile Edge Computing," in *IEEE Globecom Worksh.*, pp. 1-6, Apr. 2018.

[14] A. Ghasemi and M. Schranz, "Bottom-Up Resource Orchestration in Edge Computing: An Agent-Based Modeling Approach," *IEEE 12th International Conference on Intelligent Systems (IS)*, Varna, Bulgaria, 2024, pp. 1-7.

[15] N. Kochdumper and S. Bak, "Real-time capable decision making for autonomous driving using reachable sets," *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Yokohama, Japan, 2024, pp. 14169-14176.

[16] Q. Li, Y. Deng, X. Liu, W. Sun, W. Li, J. Li, and Z. Liu, "Autonomous smart grid fault detection," *IEEE Commun. Stand. Mag.*, vol. 7, no. 2, pp. 40-47, 2023.

[17] F. Pokou, J. S. Kamdem, and F. Benhmad, "Hybridization of ARIMA with learning models for forecasting of stock market time series," *Comput. Econ.*, vol. 63, no. 4, 2024, pp. 1349-1399.

[18] W. Wang, B. Ma, X. Guo, Y. Chen, and Y. Xu, "A Hybrid ARIMA-LSTM Model for Short-Term Vehicle Speed Prediction," *Energies*, vol. 17, no. 15, 2024, pp. 3736.

[19] J. Li, J. Cai, R. Li, Q. Li, and L. Zheng, "Wavelet transforms based ARIMA-XGBoost hybrid method for layer actions response time prediction of cloud GIS services," *J. Cloud Comput.*, vol. 12, no. 1, 2023, pp. 11.