Cloud-Native Scheduling and Resource Orchestration: A Deep Dive into AI-Driven Approaches

Tomás Dias[1], Luís Ferreira[1], Diogo Fevereiro[1], Luis Rosa[1], Luis Cordeiro[1], and João Fernandes[1]

OneSource, Coimbra, Portugal
{tomas.dias,luis.ferreira,duarte.fevereiro,luis.rosa,luis.cordeiro,joao.fernandes}@onesource.pt

Abstract. Cloud-native computing has transformed modern application development, deployment, and management by enabling scalability and flexibility. However, the increasing complexity of workloads and dynamic resource demands challenge traditional scheduling and resource provisioning techniques, often leading to inefficiencies. This paper explores AI-driven approaches to optimizing cloud-native scheduling and resource provisioning. By leveraging machine learning, deep reinforcement learning, and predictive analytics, AI enhances decision-making, automates scaling, and improves workload distribution. We present a comprehensive review of recent AI techniques applied to container orchestration, and Kubernetes-based scheduling, analyzing their impact on cost reduction, performance optimization, and resource efficiency. Additionally, we discuss key challenges such as model interpretability, real-time adaptability, and integration with existing cloud and edge infrastructures. Ultimately, this paper provides insights into the future of intelligent cloud and edge resource management, emphasizing the necessity of AI-augmented strategies to meet the growing demands of next-generation applications.

Keywords: Cloud-native computing · AI-driven scheduling · Resource provisioning · Machine learning · Kubernetes · Serverless computing

1 Introduction

The cloud-native approach has become one of the most widely used strategies for building and hosting microservice-based applications. Such applications are more scalable, portable, and cost-effective than those based on traditional monolithic architectures. However, their lifecycle requires careful monitoring and management. Orchestration frameworks such as Kubernetes [3], Docker Swarm [4], and Cloudify [5] include tools to manage cloud and edge workloads and provide features such as scheduling, load balancing, and auto-scaling. Such features rely on standard algorithms, which are based on heuristics (scheduling) and thresholds (scaling, migration). Currently, AI techniques are being explored [32] and

integrated for monitoring and management of 5G cloud-native and edge environments. By leveraging AI-driven intelligence, critical decisions can be automated, such as provisioning the necessary infrastructure and optimising component placement for performance, cost, and efficiency. Artificial Intelligence (AI) plays a crucial role in enhancing the capabilities of orchestration frameworks. The ETSI ZSM standard [1] aims to automate network management, enabling the creation of highly autonomous networks. These networks can self-configure, self-monitor, self-heal, and self-optimise without needing further human intervention. In this regard, this paper reviews AI-driven techniques for optimising resource orchestration in edge and cloud environments, with a particular focus on Kubernetes-based scheduling. Unlike existing studies, which often analyze AI applications in isolation, this work provides a comprehensive, comparative assessment of multiple AI methodologies, highlighting their impact on cost reduction, performance optimization, and resource efficiency. This research offers valuable insights for academia and industry, paving the way for more intelligent, autonomous cloud management solutions.

The remainder of the paper is structured as follows: Section 2 revisits the role of AI in resource orchestration. Section 3 reviews the literature on AI-based resource provisioning. Section 4 presents the scheduling as a multi-criteria optimization problem. Section 5 draws the conclusions.

2 AI-driven Cloud-Native Resource Orchestration

Cloud-native applications commonly utilise a microservices architecture, where their deployment requires managing a complex, distributed system. Despite the advantages of microservice-based architectures, these benefits come with challenges, including their orchestration. Service orchestration is crucial for efficiently coordinating and managing interactions among microservices. It facilitates effective scaling, deployment, and maintenance of complex systems [33]. This includes challenges related to the deployment, scalability, scheduling, management, and networking of microservices. Moreover, ETSI ZSM discusses the necessity of using closed-loop concepts to build autonomous systems that can be self-managed, removing human error. One subjacent idea is the leverage of AI/ML into such workflows. The concept of self-management through closed loops such as OODA loops, which stands for "observe, orient, decide, act", was first created by the military strategist Colonel John Boyd. The relevance of the loops was then discovered by other fields and can be applied in the context of cloud-native orchestration. AI-driven resource orchestration sub-fields can be summarized as follows [34]. **Scheduling** consists of deciding the initial placement of a containerised task unit. Scheduling methods may be aware of the application architecture and task structure. **Scaling** comprises the ability to scale containerised applications or compute nodes in response to fluctuations in workloads, ensuring that applications have the resources to fulfil their resource requirements (e.g. memory, CPU, storage). **Migration** involves moving one or more tasks from one node to another. It consists of two key concepts: rescheduling and offloading. Rescheduling

refers to relocating tasks between nodes for load balancing or offloading compute-intensive tasks in hybrid clouds. This is often necessary due to poor scheduling decisions, which can result in either resource overload or underload.

3 Literature Review

3.1 AI in Edge and Cloud Native

AI/ML techniques are increasingly applied in cloud-native environments to automate tasks, predict resource needs, and enhance security. ML's ability to handle large data volumes and uncover insights through iterative learning makes it a key tool for decision-making. Such techniques are divided into Generative AI, which creates new content and Predictive AI, which forecasts outcomes [20–22]. Other sub-fields of AI have also been investigated for their potential to address a range of optimisation challenges. For instance, evolutionary computation is a subfield where algorithms are inspired by biological evolution. One branch within this family is Swarm Intelligence (SI), which has been the subject of research due to its effectiveness in addressing a diverse range of optimisation problems, including those related to scheduling. These algorithms are based on the collective behaviour observed in organised groups of animals, such as birds, ants or bees, which strive to survive collectively [23].

RL have been attracting researchers' interest in the scaling field. Khaleq et al. [29] proposed an approach to improve Kubernetes auto-scaling that uses RL agents to learn and identify the auto-scaling threshold values based on the resource demand and Quality of Service (QoS). This approach showcased that auto-scaling based on the pod resource could lower the response time by a factor of 20% compared to default K8s CPU-based auto-scaling (HPA). To achieve that factor, a simulation environment was created in MATLAB [8] to train various RL algorithms, including Q-learning, Deep Q-Network, SARSA, among others. The goal was to evaluate their performance in that environment in two different scenarios: CPU-Intensive and Memory-Intensive. Although Reinforcement Learning (RL) is often referenced in the literature, other ML algorithms can have a crucial role in cloud-native orchestration. A proof of concept by [30] demonstrates that LSTM-based proactive auto-scaling can improve end-to-end latency for cloud-native applications. This approach uses the Digital Ocean public cloud platform [9], which provides management options for Kubernetes master nodes [30].

Dang-Quang and Yoo [31] combined the strengths of the two families, Deep RL, which integrates RL with Recurrent Neural Network (RNN) capabilities of time-analysis. The authors proposed a Proactive Custom Auto-scaler (PCA) framework focused on the analysis and planning phases. For the former was selected a Bi-LSTM which extends the LSTM models applying two LSTMs to the input data - the first trained on the input sequence in its original direction (forward) and the second on the reverse form of the input sequence (backward). The experimental phase was conducted using two different datasets for training and evaluation of models. Following this, Dang-Quang and Yoo [31] developed a simulated environment using Python [10] and ran on the Google Colab environment

[11] to assess resource provision accuracy and elastic speed-up. In this evaluation, Bi-LSTM demonstrated the highest overall accuracy and predicted error. Finally, Dang-Quang and Yoo proposed an architecture based on K8s and Proactive Custom Auto-scaler which was compared with the Kubernetes auto-scaler (HPA). The results showcased that PCA outperformed the Kubernetes auto-scaler in accuracy, speed and also efficiency, particularly when dealing with a burst of workload. Lin et al. [24] and Chen and Xiao [25] explored two SI algorithms, ACO and PSO, to optimise microservices container-based scheduling and over-perform the Spread algorithm of Docker Swarm [4]. To validate the proposed algorithm, both used real data from the Alibaba Cluster TraceV2018 cluster dataset in the experiments [6]. Lin et al. [24] proposed Ant Colony Optimization algorithm for Multi-objective of Container-based Microservice Scheduling (ACO_MCMS), a ACO multi-objective optimisation algorithm, to enhance container-based microservice scheduling. This algorithm combines multi-objective heuristic information with a feedback mechanism to avoid falling into the local optimal solution. ACO_MCMS tried to achieve three objective improvements: reducing the network transmission overhead among microservices, balancing the cluster load, and improving cluster services' reliability. The authors compared the algorithm with different multi-objective algorithms such as GA-MOCA [26], the Docker Swarm spread algorithm and Multiopt [27]. The ACO_MCMS successfully met all the proposed objectives and demonstrated the best overall performance in all categories. Chen and Xiao [25] analysed two container-based microservice scheduling multi-objective optimization algorithms - (GA-MOCA) [26] and ACO_MCMS [24]) - highlighting their drawbacks on large network transmission costs, unbalanced clusters and individual loads, and long optimization times. To address these challenges, the authors proposed MOPPSO-CMS, a multi-objective optimization algorithm based on PSO. Further evaluation showcased that MOPPSO-CMS outperformed the previous approaches in terms of network transmission overhead, local and global load balancing, standard deviation of cluster resources, service reliability, and algorithm running speed. Santos et al. [28] proposed an RL-based Global Topology Manager (GTM) for the deployment of applications in multi-cluster infrastructure. The RL algorithm uses a Deep Set (DS) approach. DS is a family of neural networks tailored for processing data-structured assets. Once trained, DS networks can perform inference to sets of arbitrary size without the need for retraining [2]. In order to facilitate the training of the reinforcement learning algorithm, an OpenAI Gym-based framework [7] was developed to enhance scalability and cost-effectiveness during the training process [28]. The developed framework demonstrates potential advantages for training RL algorithms in this environment.

3.2 Challenges-behind AI

Swarm Intelligence Over the past year, SI has also emerged as a significant case study in optimization problems [23]. These algorithms are classified as a series of heuristic, nature-inspired methods used to solve optimization problems where mathematical or traditional approaches are inadequate. They rely

on stochastic search methods that iteratively exchange heuristic information to enhance subsequent iterations [23]. Brezočnik et al. [35] highlighted the benefits of SI and provided an overview of its framework. One key advantage is autonomy, where each agent in the swarm independently controls its behaviour without the need for centralized management. Another benefit is self-organisation, where each agent represents a potential solution to a problem. While the collective approach of algorithms leads to solutions that emerge from the swarm, it does not solely focus on individuals. Additionally, scalability represents a benefit as each agent monitors its behaviour and adjusts the solution based on the swarm, ensuring there is no single point of failure. These factors allow the swarm to consist of a few to up to thousands of agents without changing the control architecture. Despite recognizing that SI algorithms offer an alternative and unconventional approach to designing complex systems without centralized control or extensive pre-programming, Ahmed and Glasgow [36] points out several limitations of this approach. One significant limitation is that SI algorithms are unsuitable for time-critical applications requiring real-time system control, prompt decision-making, and adequate solutions within strict time constraints. This is because the solutions produced by SI systems are emergent and not pre-programmed. Additionally, parameter tuning is a significant drawback of SI approaches. The parameters needed for these algorithms are often problem-dependent and must be either empirically pre-selected based on trial and error or adaptively adjusted during execution. Another limitation is the potential for stagnation or premature convergence to local optima, which can occur due to the lack of central control. Brezočnik et al. [35] define the standard SI workflow as a sequence of tasks: Initialisation population; Definition of stop condition; Evaluation of fitness function; Update and move agents; Return the global best solution. The first task involves defining the initial candidate solutions, referred to as the Swarm. The next step is to evaluate these solutions after each iteration. In addition to defining the initial candidate solutions, it is essential to establish a stopping condition. If the stopping condition is not met, a new iteration will occur, requiring an update of the Swarm and its agents. Once the condition is met, the algorithm should return the global best solution. Yang et al. [37] explored the key challenges in the development of SI algorithms. The first challenge outlined is a limitation mentioned by Ahmed and Glasgow [36], specifically concerning parameter tuning and control. As previously mentioned, the algorithms have interrelated parameters, and appropriately setting these parameters can significantly impact their performance. Tuning these parameters is essential for maximizing the effectiveness of the algorithm. Typically, tuning approaches involve parametric studies, while parameter control uses stochastic adaptivity, allowing specific parameters to vary randomly within a predefined range. In conclusion, Ahmed and Glasgow [36] argue that it is crucial to have an effective method for parameter tuning that is both automatic and adaptive. The second challenge relates to finding the right balance between optimal exploration and exploitation. Excessive exploration combined with insufficient exploitation can slow down the search process, while too much exploitation with insufficient

exploration can lead to premature convergence. Identifying the optimal balance can be difficult, and empirical observations suggest this balance may be problem-specific [37]. Another challenge is scalability. Current literature on optimization problems often deals with a variable count ranging from just a few to a few hundred, which is relatively low compared to real-world applications. Furthermore, there is no unified framework that provides a comprehensive view of an algorithm concerning convergence, rate of convergence, stability, ergodicity, repeatability, and scalability [37]. Lastly, controlling the rate of convergence is crucial because we want to find the best solution to a problem quickly and with minimal computational costs. The authors argue that it can be challenging to maximize search efficiency in this regard [37].

Machine Learning On the other hand, Paleyes et al. [38] examined the challenges associated with ML deployment, utilizing the workflow proposed by Ashmore et al. [39]. Ashmore et al. [39] categorize the deployment process of an ML-based solution in the industry into four stages: Data Management, Model Learning, Model Verification, and Model Deployment. Data management focuses on data preparation and consists of four sub-stages: Data collection, Data preprocessing, Data augmentation and Data analysis. Paleyes et al. [38] identified the main challenges in the data management phase as discovering and understanding what data is available, as well as how to organize convenient storage for it in the data collection sub-stage. In the data analyses sub-stage, the challenge is behind the visualisation of data profiling because there are still too few tools that enable the efficient execution of these data mining tasks [39]. In the Data preprocessing sub-stage, Emmanuel et al. [40] analysed the literature techniques for handling missing values and made a comparison between two imputation methodsk-Nearest-Neighbor (KNN) and an iterative imputation method based on the random forest neighbour. The imputation process involves filling in missing values with predicted values. Techniques for imputation can be categorised into various types, such as simple methods, regression methods, and those inspired by machine learning. Among these, regression is one of the preferred techniques [40], as it replaces missing values with predictions generated from a regression model. KNN is a technique based on ML that classifies the nearest neighbours of missing values and uses those neighbours for imputation, applying a distance measure between instances. According to Emmanuel et al. [40], the data type analysed influences the precision and accuracy of machine learning imputation algorithms. There is no clear evidence favouring one method over another because the results may vary depending on the evaluation metrics used, such as the Mean Absolute Error (MAE), Mean Squared Error (MSE) and Root Mean Square Error (RMSE). According to the literature, RMSE is commonly used as an evaluation metric [40]. However, Emmanuel et al. [40] point out that a limitation in the literature is the failure to use metrics in conjunction. The three most common metrics use the number of samples in the dataset m, the actual value of the i^{th} sample y_i and the predicted value of the i^{th} sample, $\hat{y}_i$.

Although there is a difference between them in how they calculate the values, the formulas are described below

Mean Absolute Error (MAE): measures the average difference between imputed values y_i and true values $\hat{y}_i$, defined as:

$$\text{MAE} = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i| \tag{1}$$

Mean Squared Error (MSE): It is equal to the sum of variance and the squared predicted missing values, defined by the following equation:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2}$$

Root Mean Square Error (RMSE): computes the difference between imputed value applying a root on the MSE.

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2} \tag{3}$$

Zebari et al. [41] explore dimensionality reduction techniques for feature selection and feature extraction. The authors analysed the advantages and constraints of dimension reduction techniques like feature extraction and selection. Feature extraction methods produce a dataset that is a linear combination of features, while feature selection methods yield a sub-list of the most relevant features [41]. Both methods offer distinct advantages. Feature selection helps maintain the original dataset while reducing its size, whereas feature extraction preserves the relative distances between features and retains the inherent structure of the data. However, each method has its limitations. Feature extraction may not be effective if there are too many irrelevant features, while feature selection can struggle if the underlying structure of the data is not adequately represented.

Sarker et al. [42] analysed the data augmentation sub-stage in terms of dimensionality reduction. The authors argue that dimensionality reduction is essential because it leads to better human interpretation, lower computational cost and the avoidance of over-fitting and redundancy. Based on machine learning and data science literature, the authors provided an overview of the most commonly used techniques for feature selection, such as Pearson Correlation, Analysis of variance (ANOVA), and feature extraction techniques, such as Principal component analysis (PCA).

Pearson Correlation provides insight into the relationship between the two features. The resulting correlation coefficient can range from -1 to 1. A negative value indicates a negative correlation, while a positive value signifies a positive correlation. A value of 0 indicates that the two variables do not have a linear correlation. For two features, X and Y, the correlation coefficient between them is defined as [48]:

$$r(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{4}$$

The symbol $r(X, Y)$ represents Pearson's correlation coefficient between the variables X and Y. The symbols X_i and Y_i denote individual data points of the variables X and Y, while $\bar{X}$ and $\bar{Y}$ represent their respective means (averages). The total number of data points is given by n. Additionally, the sum of the product of deviations from the mean is expressed as $\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})$. The denominator of the correlation coefficient formula includes the terms $\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}$ and $\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$, which correspond to the standard deviation components of X and Y, respectively.

ANOVA is a statistical tool used to determine whether there are significant differences between the mean values of two or more groups. This method assumes a linear relationship between the variables and the target, as well as a normal distribution of the variables. ANOVA tests can be classified as either one-way or two-way, depending on the number of variable groups being analysed [47].

The one-way formula is presented below [47]

$$F^* = \frac{\sum_{j=1}^{k} n_j (M_j - M)^2}{\sum_{j=1}^{k} \left(1 - \frac{n_j}{N}\right) s_j^2} \tag{5}$$

The symbol k represents the number of groups. The symbol n_j denotes the number of observations in group j. The mean of the group j is represented by M_j, while M denotes the overall mean. The total number of observations is given by N. Finally, s_j^2 represents the variance of group j.

PCA is a feature extraction method that transforms a set of correlated variables into a set of uncorrelated variables. It is an unsupervised learning technique commonly used in the MLfield [48]. PCA identifies the components with the highest eigenvalues from a covariance matrix and uses these components to project the data into a new subspace, which can have an equal number of dimensions or fewer. The resulting dimensions can vary from one-dimensional (1D) to three-dimensional (3D).

In the Model learning stage, the focus is on model selection and training. This stage includes three sub-stages: Model Selection, Training and Hyper-parameter selection. In the model training, Sharir et al. [43] observe that the training dataset size, number of model parameters, and number of operations used by the training procedure all contribute towards the overall cost. The mentioned factors are a challenge that all model implementations must be aware of to avoid expensive development. During the model learning stage, Yu et al. [44] reviewed existing hyper-parameters and highlighted their importance in algorithms. Yu et al. analysed hyper-parameters such as learning rate and optimizer, as well as those related to model design. Additionally, Yang et al. [45] argue that hyper-parameter tuning approaches require users to define the entire search space. Setting hyper-parameter optimization remains a challenge, as it is practically impossible to have complete knowledge of the problem. The subsequent phase, known as model verification, ensures the model follows the functional and performance requirements. The challenge behind this stage lies in the test-based verification,

which should be done in real-life settings. However, due to challenges related to safety, security, and scalability, testing is frequently replaced with simulation environments [38]. Polyzotis et al. [46] proposed a data validation system for ML, arguing that dataset validation is essential in the machine learning pipeline to prevent common data issues that arise when data generation is separate from the pipeline. The final stage, also known as model deployment, consists of the model integration in the system or infrastructure where it should run. Russell and Norvig [19] classify ML algorithms based on the learning approach, the input and output data type, and the problem solved: in **Supervised, Unsupervised, and RL** and a few hybrid approaches. Supervised learning algorithms require a training set of labelled examples provided by an external supervisor. The data consists of input-output pairs, where the label represents the correct action the system should take. The goal of this learning type is to generalize responses effectively, ensuring appropriate actions in situations beyond the training set. Unsupervised learning algorithms, in contrast, do not require labelled training data. Instead, they model underlying patterns in the data to uncover its characteristics. The most common unsupervised learning approach is clustering, which groups data points based on similarity. This method is often applied in tasks such as customer segmentation and anomaly detection. Reinforcement learning (RL), also known as Reinforcement Learning from Human Feedback (RLHF), is a dynamic programming technique that trains algorithms through a reward and punishment system. During the learning process, an artificial agent receives rewards or penalties based on its actions, aiming to maximize the total accumulated reward. This approach is particularly useful in decision-making processes and dynamic environments.

Russell et Norvig [19] argue that despite **supervised learning** excels at predicting specific outputs based on labelled data, unsupervised learning can also contribute by analysing historical patterns in resource utilization data. **Unsupervised models** can identify clusters or outliers that might represent potential bottlenecks. Furthermore, both **supervised and unsupervised** learning excels at leveraging historical data to inform decision-making. However, the real world necessitates dynamic adjustments based on constantly evolving workloads and system conditions. **RL** offers a compelling approach for this dynamic environment.

4 Scheduling – a multi-criteria optimization problem

Improper scheduling can lead to resource waste and degradation of in-service performance [12]. The goal of scheduling is to find the best edge/cloud resources for upcoming end-user applications to improve QoS parameters such as latency, throughput, response time, and resource utilisation. Improving all these QoS parameters involves significant complexity. In the literature, the scheduling problem is usually an NP-complete problem or an NP-hard problem without a polynomial-time algorithm [13]. The scheduling depends on the nature of resources that should be scheduled. The literature focuses on the difference between

static and dynamic scheduling algorithms. Kumar et al. [16] concluded that, in edge environments, due to highly dynamic workloads and system behaviour, dynamic solutions can approximate the NP-hard scheduling problem Authors such as [15–18] argue that scheduling in cloud-native environments can be modelled as a multi-objective optimisation problem. Zhang et al. [17] present implementations that modelled the problem as a bi-objective to minimize time and cost, or even a tri-objective to optimize the makespan, cost, and energy consumption. Rjoub et al. [15] also support that idea, arguing that the scheduler should consider multiple objectives simultaneously, such as minimising response time, maximising resource utilisation, and reducing energy consumption. Another key aspect that authors such as [12, 14] focused on is the nature of scheduling; this means the nature of resources that need to be allocated. Kumar et al. [12] concluded that scheduling algorithms can be categorised into two groups based on the task structure as independent or dependent (Workflow scheduling). Zhang et al. [17] explored the differences between scheduling algorithms for dependent (Workflow) and independent tasks in multi-domain environments. Solutions proposed for each type of scheduling algorithm have different constraints. Independent task scheduling constraints are related to reliability, security, and, for some, priority for user-specific requirements. Workflow scheduling constraints are more related to deadline, reliability and budget constraints. Based on the referenced papers, solutions proposed for both scheduling types try to achieve two main objectives: makespan and cost. In independent task scheduling, the users and cloud providers need the algorithms to pay attention to three other objectives: load balance, resource utilisation and QoS. Workflow scheduling solutions also pay more attention to resource utilisation because they are designed to deal with tasks with complex dependencies and requirements. Another distinction made by authors is the difference between multi-cloud and single-cloud environments. The main difference is in the distinct characteristics of providers and clusters. Zhang et al. [17] concluded that most of the existing multi-objectives work in workflow scheduling for single-cloud environments are tailored for specific scheduling models. This means that they cannot be used directly in multi-cloud environments. Besides that comparison, Zhang et al. [17] concluded that workflow scheduling solutions in multi-cloud and edge systems are tested in simulated environments because real cloud environments are hard to achieve.

Table 1. Summary of author analysis

Author	Multi-objective	Multi-cloud	Static/Dynamic	Workflow
Quin et al. [14]	✓	✓	Dynamic	✓
Senjab et al. [18]	✓	¬	Dynamic	✓
Rjoub et al. [15]	✓	¬	Discrete	¬
Kumar et al. [16]	✓	¬	Static	¬
Zhang et al. [17]	✓	✓	Static	¬
Zhou et al. [13]	✓	¬	Dynamic	✓

In the literature, several classifications of scheduling algorithms appear. Senjab et al. [18] classified scheduling techniques into generic scheduling, multi-objective optimization-based scheduling, AI-focused scheduling, and autoscaling-enabled scheduling. Rjoub et al. [15] classified. Existing approaches were categorised as traditional or intelligent. Traditional approaches focus on improving conventional scheduling approaches such as FIFO or First Fit. On the other hand, intelligent approaches capitalise on AI techniques. The main conclusion was that traditional solutions are not best suited for edge/cloud environments because they often assume a static or semi-static environment while the environment is dynamic. Traditional scheduling methods tend to be centralised, suffering from scalability limitations and high computational costs. Zhou et al. [13] classified existing scheduling solutions into three categories: heuristic, meta-heuristic and hybrid. Furthermore, Zhou et al. [13] provided an overview of existing solutions and concluded that heuristic algorithms are problem-dependent, which means they perform well only in specific domains and cannot solve hard optimisation problems [16]. Meta-heuristic algorithms combine heuristic algorithms with randomness. Unlike heuristic algorithms, they are not problem-dependent and efficiently explore the search space to find (near) optimal or sub-optimal solutions to the NP-complete scheduling problem. Despite this advantage, the meta-heuristic algorithms also have challenges, such as the randomness present and not providing a way to predict system status. Hybrid solutions can combine the advantages of multiple algorithms, combining two or more scheduling algorithms of two other categories (meta-heuristic and heuristic) to produce better solutions. However, a hybrid algorithm with multiple heuristics or meta-heuristics as elemental algorithms cannot exceed the scenarios for which the elemental algorithms are suitable.

5 Conclusion

Today's cloud-native orchestration needs to surpass the capabilities of traditional scheduling and resource provisioning techniques, which often rely on heuristic-based approaches. These methods are usually problem-specific and cannot be generalized across all workflows. Cloud-native environments present challenges such as managing data heterogeneity and their dynamic nature. To effectively tackle these complex issues, the fields of resource provisioning require a more dynamic and proactive approach. This paper highlighted the potential of AI in cloud-native computing, particularly in the decision-making processes related to scaling and scheduling (through RL) and in the optimisation scheduling decisions (using SI). AI-based scheduling has proven to be more effective than traditional heuristic-based algorithms in several areas: it reduces network transmission overhead among microservices, balances cluster load, and enhances the reliability of cluster services. AI-based approaches have demonstrated superior performance compared to Kubernetes HPA in terms of accuracy, speed, and efficiency, particularly when managing sudden increases in workload. Furthermore, AI algorithms exhibit overall performance when dealing with diverse workloads

and providers. Despite the well-defined AI techniques and frameworks, there are still limitations and challenges that need to be addressed, such as response time, memory usage, and training costs. The area of AI-based migration has not been fully explored, but it could yield new insights related to scaling and scheduling.

Acknowledgements This work was supported in part by the European Union's SNS JU research and innovation programme HORIZON-JU-SNS-2022 under the 6G-PATH project (Grant No. 101139172) and by the European Union's HE Research and Innovation programme HORIZON-CL4-2024-DATA-01 under the COP-PILOT project (Grant No. 101189819). The views expressed in this contribution are those of the authors and do not necessarily represent the project nor the Commission.

References

1. European Telecommunications Standards Institute Zero touch network and Service Management, https://www.etsi.org/technologies/zero-touch-network-service-management. Last accessed 2 Oct 2024.
2. Cicco, N. D., Pittalà, G. F., Davoli, G., Borsatti, D., Cerroni, W., Raffaelli, C., Tornatore, M.: DRL-FORCH: A Scalable Deep Reinforcement Learning-based Fog Computing Orchestrator. In: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), pp. 125-133. IEEE, (2023).
3. Kubernetes Official Documentation, https://kubernetes.io/. Last accessed 31 Mar 2025.
4. Docker Official Documentation, https://docs.docker.com. Last accessed 31 Mar 2025.
5. Cloudify Documentation, https://docs.cloudify.co/. Last accessed 31 Mar 2025.
6. Alibaba Cluster Trace V2018, https://github.com/alibaba/clusterdata. Last accessed 2018.
7. OpenAI Gym Beta, https://openai.com/index/openai-gym-beta/. Last accessed 2 Nov 2024.
8. MATLAB, https://www.mathworks.com/products/matlab.html. Last accessed 2 Nov 2024.
9. DigitalOcean, https://www.digitalocean.com/. Last accessed 2 Nov 2024.
10. Python, https://www.python.org/. Last accessed 20 Nov 2024.
11. Google Colab, https://colab.google/. Last accessed 16 Sep 2024.
12. Kumar, M., Sharma, S.C., Goel, A., Singh, S.P.: A comprehensive survey for scheduling techniques in cloud computing. In: Journal of Network and Computer Applications, vol. 143, pp. 1–33. Elsevier (2019).
13. Zhou, G., Tian, W., Buyya, R., Xue, R., Song, L.: Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. In: Artificial Intelligence Review, vol. 57, no. 5, pp. 124. Springer (2024).
14. Qin, S., Pi, D., Shao, Z., Xu, Y., Chen, Y.: Reliability-aware multi-objective memetic algorithm for workflow scheduling problem in multi-cloud system. In: IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 4, pp. 1343–1361. IEEE (2023).

15. Rjoub, G., Bentahar, J., Abdel Wahab, O., Saleh Bataineh, A.: Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. In: Concurrency and Computation: Practice and Experience, vol. 33, no. 23, pp. e5919. Wiley Online Library (2021).
16. Kumar, M., Sharma, S.C., Goel, A., Singh, S.P.: A comprehensive survey for scheduling techniques in cloud computing. In: Journal of Network and Computer Applications, vol. 143, pp. 1–33. Elsevier (2019).
17. Zhang, Q., Geng, S., Cai, X.: Survey on task scheduling optimization strategy under multi-cloud environment. In: CMES-Computer Modeling in Engineering & Sciences, 135(3), pp. 1863–1900. Editora, Local (2023)
18. Senjab, K., Abbas, S., Ahmed, N., Khan, A. U. R.: A survey of Kubernetes scheduling algorithms. In: Journal of Cloud Computing, 12(1), p. 87. Springer, Local (2023)
19. Russell, S. J., Norvig, P.: Artificial Intelligence: A Modern Approach. 3ª ed. Prentice Hall, Upper Saddle River, NJ (2010)
20. Mallikarjunaradhya, V., Pothukuchi, A. S., Kota, L. V.: An Overview of the Strategic Advantages of AI-Powered Threat Intelligence in the Cloud. In: Journal of Science & Technology, 4(4), pp. 1–12. Ahmedabad, India (2023)
21. Qayyum, A., Ijaz, A., Usama, M., Iqbal, W., Qadir, J., Elkhatib, Y., Al-Fuqaha, A.: Securing machine learning in the cloud: A systematic review of cloud machine learning security. In: Frontiers in Big Data, 3, pp. 587139. Frontiers Media SA (2020)
22. Butt, U. A., Mehmood, M., Shah, S. B. H., Amin, R., Shaukat, M. W., Raza, S. M., Suh, D. Y., Piran, M. J.: A Review of Machine Learning Algorithms for Cloud Computing Security. In: Electronics, 9(9), pp. 1379. (2020). DOI: 10.3390/electronics9091379
23. Tang, J., Liu, G., Pan, Q.: A Review on Representative Swarm Intelligence Algorithms for Solving Optimization Problems: Applications and Trends. In: IEEE/CAA Journal of Automatica Sinica, 8(10), pp. 1627–1643. (2021). DOI: 10.1109/JAS.2021.1004129
24. Lin, M., Xi, J., Bai, W., Wu, J.: Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud. In: IEEE Access, pp. 83088-83100 (2019)
25. Chen, X., Xiao, S.: Multi-Objective and Parallel Particle Swarm Optimization Algorithm for Container-Based Microservice Scheduling. In: Sensors, vol. 21, artigo 6212 (2021)
26. Guerrero, C., Lera, I., Juiz, C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. In: Journal of Grid Computing, vol. 16, pp. 113–135. Springer (2018)
27. Liu, B., Li, P., Lin, W., Shu, N., Li, Y., Chang, V.: A new container scheduling algorithm based on multi-objective optimization. In: Soft Computing, vol. 22, pp. 7741–7752. Springer (2018)
28. Santos, J., Zaccarini, M., Poltronieri, F., Tortonesi, M., Sleianelli, C., Di Cicco, N., De Turck, F.: Efficient Microservice Deployment in Kubernetes Multi-Clusters through Reinforcement Learning. In: NOMS 2024-2024 IEEE Network Operations and Management Symposium, pp. 1-9 (2024)
29. Khaleq, A. A., Ra, I.: Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications. In: IEEE Access, vol. 9, pp. 35464-35476. IEEE, (2021).
30. Marie-Magdelaine, N., Ahmed, T.: Proactive Autoscaling for Cloud-Native Applications using Machine Learning. In: GLOBECOM 2020 - 2020 IEEE Global Communications Conference, pp. 1-7. IEEE, (2020).

31. Dang-Quang, N.-M., Yoo, M.: Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes. In: Applied Sciences, vol. 11, no. 9, article no. 3835. MDPI, (2021). Available at: https://www.mdpi.com/2076-3417/11/9/3835.
32. Abouelyazid, M., Xiang, C.: Architectures for AI Integration in Next-Generation Cloud Infrastructure, Development, Security, and Management. In: International Journal of Information and Cybersecurity, vol. 3, no. 1, pp. 1–19. Available at: https://publications.dlpress.org/index.php/ijic/article/view/92 (Jan. 2019).
33. Newman, S.: Building microservices. O'Reilly Media, Inc., Sebastopol (2021)
34. Zhong, Z., Xu, M., Rodriguez, M. A., Xu, C., Buyya, R.: Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions. ACM Comput. Surv. **54**(10s), Article 217, 35 pages (2022). doi:10.1145/3510415
35. Brezočnik, L., Fister, I., Podgorelec, V.: Swarm Intelligence Algorithms for Feature Selection: A Review. Applied Sciences **8**(9), Article 1521 (2018). doi:10.3390/app8091521
36. Ahmed, H., Glasgow, J.: Swarm intelligence: concepts, models and applications. School of Computing, Queens University Technical Report **2012** (2012)
37. Yang, X.-S., Deb, S., Zhao, Y.-X., Fong, S., He, X.: Swarm intelligence: past, present and future. Soft Computing **22**, 5923–5933 (2018)
38. Paleyes, A., Urma, R.-G., Lawrence, N. D.: Challenges in Deploying Machine Learning: A Survey of Case Studies. ACM Comput. Surv. **55**(6), Article 114, 29 pages (2022). doi:10.1145/3533378
39. Ashmore, R., Calinescu, R., Paterson, C.: Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges. ACM Comput. Surv. **54**(5), Article 111, 39 pages (2021). doi:10.1145/3453444
40. Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., Tabona, O.: A survey on missing data in machine learning. Journal of Big Data **8**, 1–37 (2021)
41. Zebari, R., Abdulazeez, A., Zeebaree, D., Zebari, D., Saeed, J.: A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction. Journal of Applied Science and Technology Trends **1**(1), 56–70 (2020)
42. Sarker, I. H.: Machine learning: Algorithms, real-world applications and research directions. SN Computer Science **2**(3), 160 (2021)
43. Sharir, O., Peleg, B., Shoham, Y.: The cost of training NLP models: A concise overview. arXiv preprint arXiv:2004.08900 (2020)
44. Yu, T., Zhu, H.: Hyper-Parameter Optimization: A Review of Algorithms and Applications. arXiv preprint arXiv:2003.05689 (2020). Available at: https://arxiv.org/abs/2003.05689
45. Yang, L., Shami, A.: On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing **415**, 295–316 (2020)
46. Polyzotis, N., Zinkevich, M., Roy, S., Breck, E., Whang, S.: Data Validation for Machine Learning. In: Proceedings of Machine Learning and Systems, pp. 334–347. A. Talwalkar, V. Smith, M. Zaharia (eds.), vol. 1 (2019)
47. Liu, Q., Wang, L.: t-Test and ANOVA for data with ceiling and/or floor effects. Behavior Research Methods **53**(1), 264–277 (2021)
48. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques. 3rd edn. Morgan Kaufmann, Waltham, MA (2011)