Artificial Intelligence Review (2024) 57:61
https://doi.org/10.1007/s10462-023-10684-0

Machine learning methods for service placement: a systematic review

Parviz Keshavarz Haddadha[1] · Mohammad Hossein Rezvani[1] · Mahdi MollaMotalebi[1] · Achyut Shankar[2,3,4]

Accepted: 20 December 2023 / Published online: 17 February 2024

Abstract
With the growth of real-time and latency-sensitive applications in the Internet of Everything (IoE), service placement cannot rely on cloud computing alone. In response to this need, several computing paradigms, such as Mobile Edge Computing (MEC), Ultra-dense Edge Computing (UDEC), and Fog Computing (FC), have emerged. These paradigms aim to bring computing resources closer to the end user, reducing delay and wasted backhaul bandwidth. One of the major challenges of these new paradigms is the limitation of edge resources and the dependencies between different service parts. Some solutions, such as microservice architecture, allow different parts of an application to be processed simultaneously. However, due to the ever-increasing number of devices and incoming tasks, the problem of service placement cannot be solved today by relying on rule-based deterministic solutions. In such a dynamic and complex environment, many factors can influence the solution. Optimization and Machine Learning (ML) are two well-known tools that have been used most for service placement. Both methods typically use a cost function. Optimization is usually a way to define the difference between the predicted and actual value, while ML aims to minimize the cost function. In simpler terms, ML aims to minimize the gap between prediction and reality based on historical data. Instead of relying on explicit rules, ML uses prediction based on historical data. Due to the NP-hard nature of the service placement problem, classical optimization methods are not sufficient. Instead, metaheuristic and heuristic methods are widely used. In addition, the ever-changing big data in IoE environments requires the use of specific ML methods. In this systematic review, we present a taxonomy of ML methods for the service placement problem. Our findings show that 96% of applications use a distributed microservice architecture. Also, 51% of the studies are based on on-demand resource estimation methods and 81% are multi-objective. This article also outlines open questions and future research trends. Our literature review shows that one of the most important trends in ML is reinforcement learning, with a 56% share of research.

Keywords Internet of Everything (IoE) · Computing paradigms · Distributed microservices · Application perspective · Resource estimation

Extended author information available on the last page of the article

1 Introduction

Service placement is the selection of an appropriate execution zone for service instances. In this regard, the service instances are mounted on the underlying computing resources of the network (Taheri-abed et al. 2023). This is done according to various performance criteria such as Quality of Service (QoS), energy consumption, latency, availability, etc. Today, with the advent of the Internet of Everything (IoE), most services include latency-sensitive and computation-sensitive components (Zabihi et al. 2023). Some of these important services include virtual reality, augmented reality, healthcare, museum monitoring, smart transportation, weather monitoring, e-health, and the Internet of Vehicles (IoV) (Gasmi et al. 2022). A huge amount of data is expected to be collected by sensors/connected objects, which have previously been processed centrally by large data centers in traditional ways. These new services cause more latency and energy consumption than before (Alenazi et al. 2022). The Mobile Cloud Computing (MCC) paradigm that emerged nearly two decades ago is not sufficient to fulfill these applications alone. The most important problem with MCC is the long delays due to the remoteness of the cloud data centers from the end devices. In response to this need, several computing paradigms have emerged, such as Mobile Edge Computing (MEC), Ultradense Edge Computing (UDEC), and Fog Computing (FC). The purpose of these paradigms is to bring computing resources closer to the end user and thus reduce the delay and waste of backhaul bandwidth.

Service placement is based on various parameters such as functional layout, service type, workload type, and application model. There are also other restrictions on resource management. The most important constraints are interaction method, resource type, resource orientation, and resource estimation. In addition, since IoT services experience workload fluctuations over time, it is important to automatically provision a sufficient number of resources. In this regard, overprovisioning/underprovisioning should be avoided to meet QoS (Etemadi et al. 2020). Also, if the edge resources are not enough for processing, we need to offload the service to fog devices. This, in turn, raises new issues. Among the most important issues of offloading, we can mention offloading approaches, placement strategy, and workload prediction (Shahraki et al. 2023). In addition to the limitation of edge resources, some services may have interdependencies with each other. This makes service placement more complicated. On the other hand, new applications have a distributed microservice architecture (Shahraki et al. 2023). As an advantage, the microservice allows the simultaneous processing of several parts of an application.

1.1 Motivation

A network service can consist of multiple functions that are placed in a specific order as a Service Function Chain (SFC). Service placement is the selection of appropriate execution zones for SFCs. One of the key challenges in service placement is when an application service is to be delivered in partnership with edge and cloud servers. Today, one of the most important technologies used for SFC is the use of containers. This is a well-known way to virtualize application services. Some well-known container technologies are Docker, Kubernetes, and Rocket (John 2023). The main purpose of service placement is to maintain "service chaining," where multiple services follow a hierarchical order depending on their predefined goals. For example, Netflix may distribute key elements of its service, such as parental controls, traffic encryption, and acceleration, in a particular order (Mohan et al. 2016).

Figure 1 shows a service chain with two separate services in the edge-fog-cloud infrastructure consisting of two Edge Providers (EPs). Suppose one of the EPs has three servers and the other has two servers. Presumably, because EP1 is closer to the user, it can provide better QoS than EP2. Suppose the price of EP1 is higher than that of EP2. This chain ends at the fog platform, which consists of two Fog Servers (FSs). There are several combinations of service placement. Figure 1 shows two of the possible chains, SFC1 and SFC2. Because SFC1 provides both services on the EP1 platform, it has lower latency but higher cost. In contrast, because SFC2 distributes the services over both providers, it has higher experienced latency but lower cost. The reason for the lower cost of SFC2 is that it allows the program owner to limit its placement costs. However, SFC2 suffers from a lack of flexibility to adapt to changing user requirements due to the high management costs of combining two different providers.

There are two categories of distributed microservice programs for service placement. Some of them include parts with low interdependencies (loosely-coupled microservices), while others have high interdependencies (tightly-coupled microservices) (Mahmud et al. 2020). For tightly-coupled services, static methods are inefficient. To place the service, requirements such as the mobility of nodes and the limitation of energy and capacity of the underlying resources should be considered. Nowadays, due to the ever-increasing number of devices and input tasks, it is not possible to solve the service placement problem by relying on deterministic, rule-based solutions. In such a dynamic and complex environment, many factors can influence the solution.

So far, many optimization methods have been used to handle service placement. Due to the NP-hard nature of the problem, classical optimization methods are not sufficient. It cannot be solved on a large scale by methods such as Mixed Integer Programming (MIP) (Taka et al. 2022) or game theory (Shakarami et al. 2020). Instead, meta-heuristic and heuristic methods are widely used (Zabihi et al. 2023; Taka et al. 2022). These methods define a state space for the problem. The search starts from an initial state and continues until the closest solution is found (Tavakoli-Someh and Rezvani 2019). In heuristic algorithms, the search domain is limited by using a specific heuristic to reach the solution faster. A heuristic is an approximation to the problem. The main problem with heuristic methods is that they get stuck in local optima (Maia et al. 2021). Again, getting rid of local optima in metaheuristic

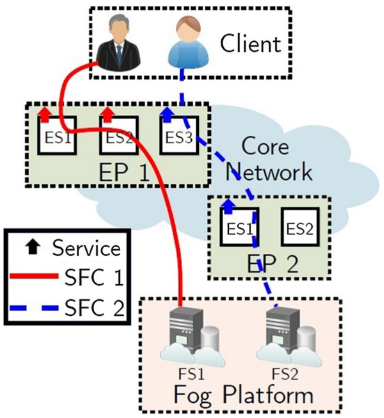


Fig. 1 An example of service placement at the edge-fog-cloud (Mohan et al. 2016)

algorithms is one of the major concerns (Sarrafzade et al. 2022). Unlike heuristic methods, which are usually tailored to a specific problem, metaheuristic methods do not depend on a specific problem. They define a general strategy according to which all problems should be solved (Lu et al. 2022). For example, the authors in Canali and Lancellotti (2019) use the genetic algorithm to reduce the convergence time to the near-optimal solution. In another study, Sarrafzade et al. (2022) propose a penalty-based approach to reduce response time in the cloud. Embedding a penalty mechanism helps to explore a larger space initially. The penalty effect is gradually increased in the next iterations. This is done through a chromosome selection process using a priority value. The closer the dependent modules are to the user, the higher their selection probability. Despite the advantages, metaheuristic methods have very sensitive hyperparameters that are not easy to tune.

Recently, the use of Machine Learning (ML) methods to predict service placement has attracted the attention of researchers. Both optimization and ML often use a cost function. Optimization is usually a way to define the difference between predicted and actual value, while ML aims to minimize the cost function. Simply put, the goal of ML is to minimize the gap between prediction and reality based on historical data. Instead of relying on explicit rules, ML uses predictions based on historical patterns (Shahraki et al. 2023). One of the reasons for the popularity of ML is its ability to handle increasingly large data volumes in IoE environments.

ML is divided into four important categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning (RL). RL methods can exploit environmental experiences to enable the agent to make the best decision for short-term future predictions (Sutton and Barto 2018). These methods have the characteristics of self-learning and self-adaptation, which reduce the sensitivity to hyperparameters. RL methods generally perform global searches effectively. This makes RL ideal for modeling high-dimensional problems in real-world scenarios. RL has a diverse set of techniques that make it ideal for service placement. Unlike MDPs, where the exact state of the system is always fully observable by the agent, sometimes it may be difficult to detect the exact state of the edge/fog system. Processes in which the decision-maker may have incomplete knowledge of the system state are called Partially-observable Markov Decision Process (POMDP). Here, keeping the information of the previous states in memory helps the agent to understand more about the nature of the environment and find the optimal policy. Observability affects the computational complexity of an optimal policy. For example, in MDPs, dynamic programming can be used to calculate an optimal policy for a finite horizon in polynomial time. Also, one of the methods used in POMDP environments is the Deep Recurrent Q-learning (DRQN). Here, recurrent neural networks are used to understand and retain information about past states.

Despite many advantages, ML methods also have limitations. For example, in supervised learning methods, the selection of training data is a challenge (Sutton and Barto 2018). Similarly, in unsupervised learning methods, the learning rate is a challenge. Usually, the biggest challenge of RL methods is establishing a tradeoff between exploration and exploitation. The agent cannot practically explore in an infinite state/action space. The agent should try to exploit the potential of current points by limiting exploration to new points (Sutton and Barto 2018). Note that ML methods may have some optimization operations at their core. For example, one of these operations may be to compute the minimum value of the cost function. In such cases, ML methods may use metaheuristic/heuristic techniques to find the optimal solution. For example, the authors in Shen et al. (2023) use an evolutionary algorithm to reduce the search overhead during the construction of the neural network model. Recently, the combination of evolutionary methods with ML, especially neural networks, has received much attention from researchers. For example, Shen et al. (2023) use evolutionary methods to prune Sparse Neural Networks (SNNs) to have no additional connections after training. The pruning and regeneration of synaptic connections in SNNs evolve dynamically during learning, but the structure dispersion is maintained at a certain level. Since our main focus in this survey is on ML methods in service placement, we refrain from further explanation of evolutionary research. Interested readers can refer to Shen et al. (2021), Natesha and Guddeti (2022), Hu et al. (2023).

In this systematic review, we present a taxonomy for the service placement problem using ML. This review includes a description of the most important performance criteria for each method. We will also outline open issues and future research trends.

1.2 Comparison with previous studies

A literature review reveals that researchers have studied the problem of service placement with different objectives. Some have ignored important issues that conflict with real-world needs. Some of these include not considering mobility (Gasmi et al. 2022; Donyagard Vahed et al. 2019), heterogeneity and dynamism (Torabi et al. 2022; Santos et al. 2022a), amount of resources, and environment (Haibeh et al. 2022). None of these surveys have been included in the field of the IoE, and the paradigms related to this field, i.e., UDEC (Eyckerman et al. 2022; Fang et al. 2022) and FRAN (Xiao et al. 2020; Yu et al. 2020), have not been studied.

To increase the efficiency of the methods, it is necessary to be able to analyze their applications at the software level, and this requires understanding the architecture of the programs. Unfortunately, only a few surveys (Mahmud et al. 2020; Salaht et al. 2020) have addressed the types of program architecture. As we know, service placement can be used for various purposes, such as offloading (Mahmud et al. 2020; Salaht et al. 2020), resource management (Gallego-Madrid et al. 2022; Nayeri et al. 2021), and scheduling (Malazi et al. 2022; Shuja et al. 2021). Some studies have investigated service placement from the perspective of optimization theories (Wang et al. 2022a; Shao et al. 2021).

In this paper, we examine service placement from an ML perspective. As will be explained in Sect. 3, we present 8 key requirements for service placement. These include ML technique, resource estimation, objective, application architecture, paradigm, and simulator used. The tables in the Appendix explain all these requirements in detail. All major MCC complementary computing paradigms, including FC, MEC, EC, UDEC, and FRAN, are covered in the context of IoT and IoT. Table 1 shows a comprehensive comparison of our survey with the most important previous articles in terms of paradigms and key features. As can be seen, the focus of previous studies is mainly on static and dynamic methods. Some of them focus exclusively on heuristic/metaheuristic optimization methods. Among the 19 surveys, only 10 are about ML methods. None of them has comprehensively addressed all computing paradigms. In addition, as shown in Table 1, none of them provides a comprehensive overview of the type of application and mobility.

1.3 Contributions

Our most important contributions to this survey are:

- We make a comprehensive classification of the research conducted on service placement using ML methods. It includes the four well-known categories of supervised

Table 1 A comprehensive comparison of our survey with previous studies

References	Year	MCC	FC	MEC	EC	UDEC	FRAN	IoT	IoE	Service placement	Mobility	App type	Discussing ML strategies	Future directions
Ghobaei-Arani et al. (2020)	2019	X	✓	X	X	X	X	✓	X	X	✓	X	X	✓
Mahmud et al. (2020)	2020	X	✓	X	X	X	X	✓	X	✓	✓	✓	X	✓
Salaht et al. (2020)	2020	X	✓	X	✓	X	X	✓	X	✓	✓	✓	X	X
Rodrigues et al. (2019)	2020	✓	X	✓	X	X	X	✓	X	X	✓	X	✓	✓
Shuja et al. (2021)	2021	X	X	✓	✓	X	X		X	X	✓	X	✓	X
Gallego-Madrid et al. (2022)	2022	X	✓	X	X	X	X	✓	X	✓	X	X	✓	✓
Gasmi et al. (2022)	2021	✓	X	X	X	X	X		X	X	X	X	X	X
Donyagard Vahed et al. (2019)	2019	X	✓	X	X	X	X	✓	X	X	✓	X	X	✓
Torabi et al. (2022)	2022	X	X	X	X	X	X	✓	X	X	X	X	X	X
Santos et al. (2022a)	2022	X	✓	X	X	X	X	✓	X	X	X	X	✓	✓
Nayeri et al. (2021)	2021	X	✓	X	X	X	X	✓	X	✓	✓	X	✓	✓
Malazi et al. (2022)	2022	X	X	✓	✓	X	X	X	X	✓	✓	X	✓	✓
Haibeh et al. (2022)	2022	X	✓	X	X	X	X	X	X	X	X	X	X	X
Kumar et al. (2022)	2021	X	X	X	X	X	X	X	X	X	X	X	X	X
Wang et al. (2022a)	2022	✓	X	X	✓	X	X	X	X	✓	✓	✓	✓	✓
Dimililer et al. (2021)	2021	X	X	X	X	X	X	✓	X	X	X	X	✓	✓
Kar et al. (2023)	2023	✓	✓	X	✓	X	X	X	X	X	X	X	✓	✓
Shakarami et al. (2021)	2022	X	✓	X	✓	X	X	X	X	X	✓	X	✓	✓
Shao et al. (2021)	2021	X	X	X	✓	X	X	✓	X	✓	X	X	X	X
Our Paper	2023	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Descriptive statistics of the use of each method in each ML category are presented along with the research trends.

- Based on a systematic review, we propose seven basic requirements for service placement. These requirements address important issues such as resource estimation techniques, performance metrics, application architecture, interface paradigm, simulation tools, and objective algorithms.
- In describing each technique, we have paid special attention to the application architecture and resource estimation method. Solving the problem of service placement in environments where resources are constantly changing requires short-term and medium-term resource estimation. In addition, distributed microservice architecture can be effective in choosing a problem-solving algorithm. We have highlighted these requirements in the attached tables.

The organization of this paper is shown in Fig. 2. Section 2 provides an overview of service placement and ML fundamentals; Sect. 3 describes the research methodology and basic questions; Sect. 4 describes critical ML techniques used for service placement; Sects. 5 and 6 are devoted to the discussion of results and future directions; Finally, Sect. 7 concludes the survey. Also, Table 2 shows a summary of the abbreviations.

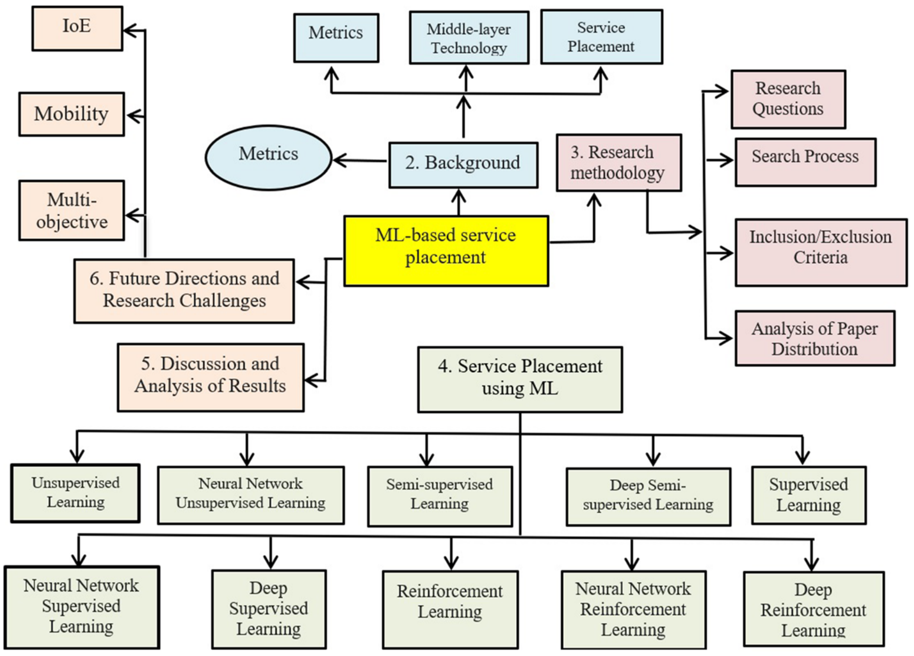

IoE
Mobility
Multi-objective
Metrics
Middle-layer Technology
Service Placement
Metrics
2. Background
3. Research methodology
Research Questions
Search Process
Inclusion/Exclusion Criteria
Analysis of Paper Distribution
ML-based service placement
6. Future Directions and Research Challenges
5. Discussion and Analysis of Results
4. Service Placement using ML
Unsupervised Learning
Neural Network Unsupervised Learning
Semi-supervised Learning
Deep Semi-supervised Learning
Supervised Learning
Neural Network Supervised Learning
Deep Supervised Learning
Reinforcement Learning
Neural Network Reinforcement Learning
Deep Reinforcement Learning

Fig. 2 The organization of this paper

Table 2 List of abbreviations

Acronym	Description	Acronym	Description
SPP	Service Placement Problem	SLA	Service-level Agreement
EC	Edge Computing	I/O	Input/Output
MEC	Mobile Edge Computing	RL	Reinforcement Learning
FRAN	Fog Radio Access Networks	MORL	Multi-objective Reinforcement Learning
UDEC	Ultra Dense Edge Computing	KNN	k-Nearest Neighbor
IoT	Internet of Thing	DNN	Deep Neural Network
IoE	Internet of Every Thing	NGWN	Next Generation Wireless Network
App	Application	NGCN	Next Generation Cellular Network
ML	Machine Learning	BS	Base Station
NGWM	The Next Generation Wireless Network	DL	Deep Learning
QoS	Quality of Service	ESL	Evolutionary Structure Learning
QoE	Quality of Experience	SNN	Spark Neural Network
REST	Representational State Transfer	IID-Data	Independent and Identically Distributed Data
RNN	Recurrent Neural Network	GRU	Gated Recurrent Unit
FSM	Frequent Subgraph Mining		

2 Background

This section first explains the basics of service placement. It then discusses service placement based on machine learning.

2.1 Service Placement

In this survey, we look for machine learning methods that have been used to solve the problem of service placement. Figure 3 shows the main computing paradigms related to service placement. End devices do not have enough processing power to handle an independent task, so there is a need to offload the processes and place the service in the fog or cloud. Usually, two metrics of delay and energy consumption are present in the service placement. A service delivery method should consider both user deadlines and minimize the energy consumption of the entire network.

Typically, end devices in Fog Computing (FC) and Mobile Edge Computing (MEC) are mobile (Taheri-abed et al. 2023). They also differ in terms of the operating system, hardware, and processing power. In addition, the interface connections for offloading user requests are heterogeneous. All these factors make the service placement problem the basis of a multi-objective optimization problem. A balance must be struck between multiple objectives and metrics. The general goal is to determine which request should be placed on which underlying physical node to achieve the best performance. For example, if we want to optimize delay, we cannot simply look for the physically closest nodes. If we want to optimize delay, we cannot ignore network availability (Eyckerman et al. 2022).

Service placement methods are provided according to different factors. Some important factors are node mobility, resource availability, network dynamics, application architecture,

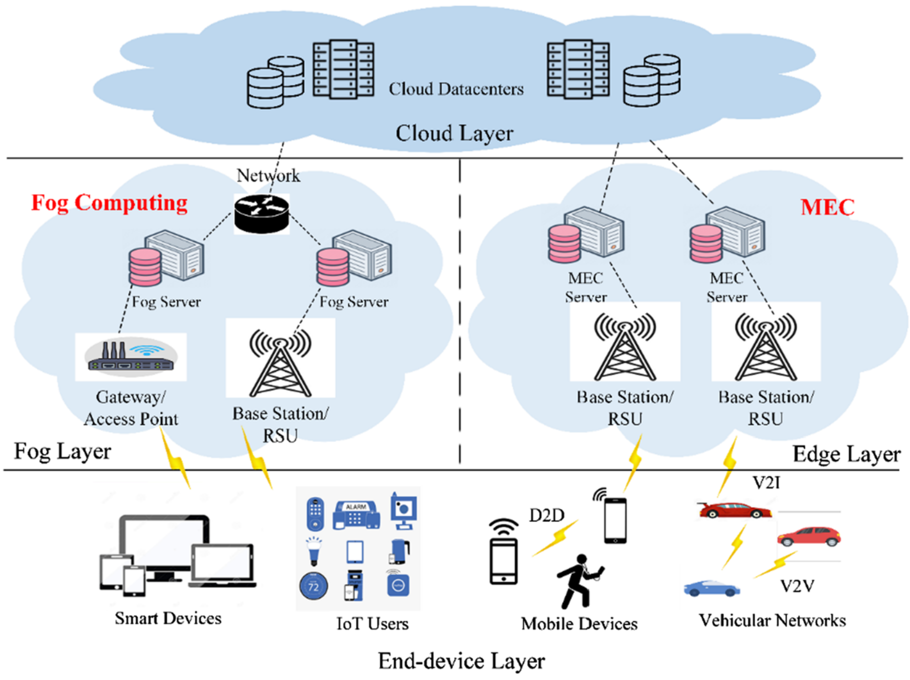


Fig. 3 The main computing paradigms related to service placement (Zabihi et al. 2023)

network resource arrangement, and placement technique. Available resources can be measured by profiling, predictive, on-demand, and dynamic strategies and can be static or dynamic. Existing application architectures may be monolithic or interdependent. They can be distributed or centralized, depending on how network resources are arranged, the type of online/offline placement, and how service placement is controlled (Salaht et al. 2020).

Some of the most important methods for solving the service placement problem are integer programming, constraint function optimization, game theory, and machine learning. Typically, each of these methods can be integrated with other optimization methods, such as metaheuristics, as needed (Salaht et al. 2020). As an example, we consider one of the most common problems of typical service provisioning. In the service placement problem based on the pigeon nest principle, we want to understand how to place different M service components on N underlying nodes. An important constraint is that the capacity of the underlying nodes is not less than the total number of services. The optimization problem (e.g., minimization) of the objective function $F(i,j)$, $i \in M$, $j \in N$ is defined as follows

$$\min \ F(x) = [f_1(x),\ f_2(x),\ \ldots,\ f_o(x)] \tag{1}$$

$$\text{s.t.} g_i(x) \geq 0, \quad i = 1,\ 2, \ldots,\ k \tag{2}$$

$$h_i(x) = 0, \quad j = 1,\ 2, \ldots,\ l \tag{3}$$

In the above formula, the goal is to minimize an objective function F with O different parameters, while we have k number of different constraints in the form of functions g and

Table 3 Important metrics and sub-metrics in service placement

Metrics	Sub-metrics
Reducing energy consumption	Power (energy) consumption, resource utilization, user quantity, mobility, completion time, resource need, resource usage,
Reducing processing time	Latency, time, response time, transmission efficiency, distance, processing delay, task waiting time, time interval, computational overhead,
Reducing I/O operation time	Storage availability, network utility, transmission time, congestion avoidance, VM capacity, availability of resources, service size, task scheduling
Optimal use of resources	Average job scheduling, scheduling time, energy (power) consumption, available resource, network utility, resource utility, congestion avoidance, service size, communication, task scheduling, resource provisioning
Mobility support	Mobility, distance
Reducing costs	Total cost, operational cost of run time
Improving the Quality of Service (QoE) and improving the Quality of Experience (QoE)	SLA, Qoe awareness, QOS parameter,
Expandability	Prediction accuracy, service instability

h. Each of the functions $f_1(x)$, $f_2(x)$, … and $f_o(x)$, describes one of the important metrics such as energy, delay, etc. The constraints *g* and *h* can also be things like computing capacity, available memory, available bandwidth, and application limits like the maximum possible delay between two services (Eyckerman et al. 2022).

2.2 Metrics

The main challenges in communications in the Internet of Things (IoT) include the heterogeneity of devices in terms of the amount of resources, operating system, and mobility (Oliveira et al. 2023). This network is used for purposes such as real-time applications (streaming video, gaming, virtual reality, and augmented reality), processing-based applications, and storage-based applications (Ghobaei-Arani et al. 2018). To overcome the limitations between the end-device layer and the cloud layer, improved intermediate layers have emerged, including the edge layer, and fog layer. In general, the goal of these intermediate paradigms is to improve Quality of Service (QoS) and Quality of Experience (QoE) based on a predefined scenario. We have identified eight important metrics: energy consumption, processing time, I/O operation time, resource utilization, mobility support, cost, QoS and QoE improvement, and scalability. Important categories such as component placement, system integration, application elasticity, and remediation, are also included in these topics (Duc et al. 2019). Table 3 shows important metrics and sub-metrics in service placement.

Service placement can be combined with other independent goals such as offloading, and cashing. Our literature review shows that among the above eight areas, four areas are more important and well-known. For service placement purposes, the following abbreviated list can be provided:

- Service placement for latency reduction
- Service placement for (optimal) resource utilization
- Service placement to reduce energy consumption
- Service placement to reduce cost

Two general methods for service placement can be imagined: (a) on a virtual machine in response to a user request and (b) on different instances of physical machines (Sami et al. 2020). For example, in Sami et al. (2020), service switching is used to optimize the use of resources of each fog layer cluster. In previous research, it has been used for dynamic management through orchestration and containerization technology. On the contrary, in this research, a proactive method is used for horizontal scaling of resources according to workload fluctuations in the fog layer. The authors also consider the cost metric for scalable placement of the service in an instance. According to Table 3, they optimize the response time by preventing instability, detecting failures, and inspecting maintenance. They model the processing of user requests and available resources using a Markov Decision Process (MDP). The technique used in this research is SARSA.

Another classification of service placement is done according to the dynamics of fog computing. In general, there are two main methods: static and dynamic. The fog layer is created to overcome the limitations of the distant cloud layer. Here, computing and storage resources are closer to the end user. A flexible solution is to use the virtual machine to facilitate service hosting, which is supported by many operating systems. Unfortunately, as the number of user requests increases, the cost of resources, especially in the fog layer, increases dramatically. This necessitates the need for dynamic service provisioning. In such situations, containers are now being used instead of virtual machines. Container technology is much lighter than a virtual machine because, unlike a virtual machine, it uses the device's operating system instead of creating a copy of it. Therefore, it is very easy to manage a large number of containers and resources in fog clusters using the orchestrator. To manage the large number of requests in the dynamic management style, horizontal scalability is used. Horizontal scalability means adding or removing container instances to achieve the response rate desired by the user or to free up resources that were being used by other resources. Unfortunately, a potential challenge with horizontal scalability is the heterogeneity of fog resources. In such an environment, using automated methods may be better than other methods and more adaptable to real-world requirements.

Sami et al. (2020) propose a container-based service placement. They use reinforcement learning to proactively monitor horizontal resources. Their goal is to simultaneously minimize response time and cost due to the heterogeneity of resources. In such a heterogeneous environment, the network can become unstable. In the fog environment, not only do user requests change randomly, but the workload also changes randomly. The placement of the service is highly dependent on the availability of resources. When a device in the fog layer receives a request from a user, it first sends performance-related information to a central controller. Then, taking into account the information from all devices, the controller determines which device is best suited to meet the response time and cost requirements. In addition to response time and cost, the system also ensures that the workload of the fog devices is not overloaded by the addition of a new service (Sami et al. 2020).

Consider another example of the importance of latency and energy in healthcare applications (Eyckerman et al. 2022). Here, the patient's vital signs need to be monitored in real-time and continuously. The cloud network is not conducive to low-latency applications, and end devices suffer losses due to missed deadlines. This challenge can be solved with the advent of the 5G network. The 5G network is ultra-reliable, with low

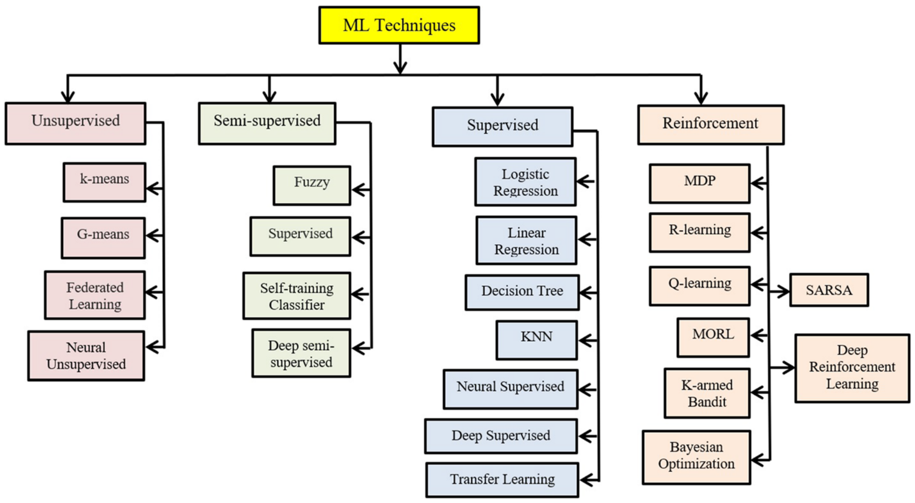


Fig. 4 Taxonomy of machine learning methods for service placement

latency and high throughput. One of the problems with 5G is the heterogeneity of the environment, which in turn causes a lack of bandwidth and increasing latency. To solve this problem and bring end devices closer to network resources, the 5G network is used in combination with fog computing. Fog computing is an intermediate paradigm that is compute-intensive, latency-aware, and even energy-aware. However, to achieve acceptable performance, the best placement of the service should be done according to the desired strategy and important parameters.

2.3 Service placement and machine learning

The most important reason for the success of machine learning methods is the dynamism in solving prediction problems (Rodrigues et al. 2019). Machine learning methods do not guarantee optimality, but they have relatively low complexity in solving problems with many variables. Figure 4 shows a taxonomy of machine learning methods for service placement. In a general classification, machine learning methods fall into four categories: supervised, semi-supervised, unsupervised, and reinforcement learning. In supervised learning, an optimal training set is used to learn how to interact with the environment. In semi-supervised learning, the machine's response is based on a combination of an optimal training set and previous experience with the environment. In unsupervised learning, the machine learns how to interact with the environment based on previous experience. Reinforcement learning has two main phases: exploration and exploitation. Here, the agent first explores the dimensions of the problem in the environment and then exploits its findings (Rodrigues et al. 2019).

3 Research methodology

The goal of this study is to systematically review machine learning algorithms for service placement. The methodology of this Systematic Literature Review (SLR) is based on the guidelines of Zabihi et al. (2023). This section includes research questions, the search process, inclusion/exclusion criteria, and quality assessment.

- Research questions

To plan the review process, it is first necessary to extract the research questions. These questions determine the motivation for the research. We address seven research requirements, which are

RQ1:	Most of the algorithms used in service placement are related to which area of machine learning?
Motivation:	The answer to this question indicates which machine learning techniques are commonly used among the algorithms used in service placement.
RQ2:	What methods are used to estimate the amount of current network resources, and which resource estimation method is more popular?
Motivation:	The answer to this question determines which of the resource estimation methods (dynamic, profiling, or on-demand) are used in service placement and which of these methods is used more often.
RQ3:	The purpose of service placement is optimization/trade-off between which parameters and metrics?
Motivation:	The answer to this question determines what percentage of the algorithms seek to optimize one metric and what percentage seek to find a compromise between multiple metrics.
RQ4:	Can the type of application architecture influence the choice of method and its efficiency?
Motivation:	Architecture refers to the type of distribution of the application and its components. Depending on whether it is monolithic or distributed, it can be useful in the resource estimation algorithm.
RQ5:	What is the dominant paradigm used in the interface layer?
Motivation:	The answer to this question determines what percentage of interface paradigms can be useful in increasing the functionality of the cloud network in different technologies, including IoT and IoV.
RQ6:	What simulation tools have been used in previous research to evaluate efficiency?
Motivation:	The answer to this question indicates what tools researchers have used more for simulation. This can shed light on young researchers and save them from confusion.
RQ7:	What were the main objectives of previous research?
Motivation:	The answer to this question indicates what the purpose of the previous research was and what algorithms the researchers used to achieve which goal.
RQ8:	What are the most important open issues in the future?
Motivation:	Answering this question is the key for young researchers. It allows them to save valuable time in identifying hot research topics.

We will answer the above questions in detail in the following sections.

- Search process

The most important scientific databases in the world used in this survey are Wiley (www.interscience.wiley.com), Springer (www.springerlink.com), ACM Digital Library (www.acm.org/dl), ScienceDirect (or Elsevier) (www.sciencedirect.com), IEEExplore (ieeexplore.ieee.org), MDPI (www.mdpi.com). Keywords used to describe the search string include terms such as "machine learning," "reinforcement learning," "service delivery," etc. By using keywords and combining them with Boolean "AND" and "OR" operators, the search strings were ultimately defined as follows: ("machine learning" or "ML") ("reinforcement learning" or "RL") ("supervised learning" or "unsupervised learning") ("semi-supervised learning").

- Inclusion and exclusion criteria

Our literature review shows that after 2018, research on service placement and related techniques such as offloading, service scheduling, resource management, and resource provisioning has gained momentum. For this reason, we have chosen to focus on the period from the beginning of 2018 to the end of 2023. The article selection process is shown in Fig. 5. Note that we removed the non-English articles, even though their number was very small. Also, in the quality assessment phase, conference articles that were not indexed or were not highly relevant were removed. Table 4 in the Appendix shows the characteristics that were finally extracted from the articles.

- Distribution of articles

Figures 5b and 6a show the distribution of articles by year and by publisher, respectively. As shown in the figure, IEEE has the largest share over other publishers with more than 50%. Other publishers follow with almost the same percentage. Figure 7 shows the number of publications per year. This figure shows a significant increase in the research

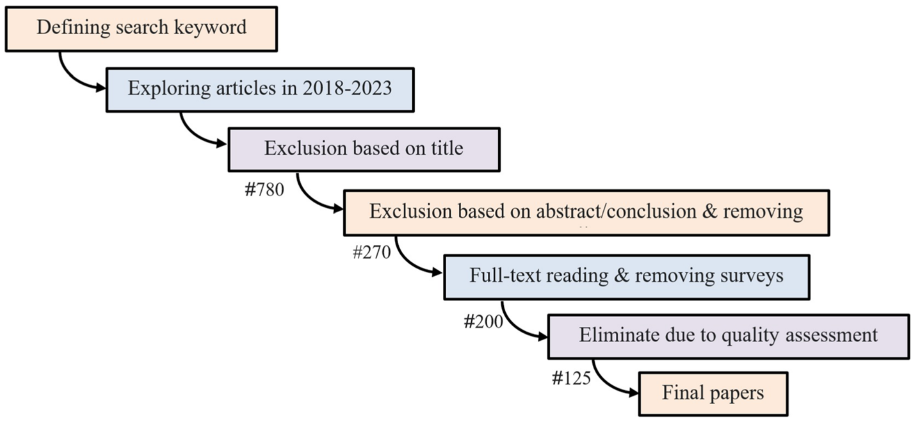

Defining search keyword
Exploring articles in 2018-2023
Exclusion based on title
#780
Exclusion based on abstract/conclusion & removing
#270
Full-text reading & removing surveys
#200
Eliminate due to quality assessment
#125
Final papers

Fig. 5 The process of selecting articles for our systematic review

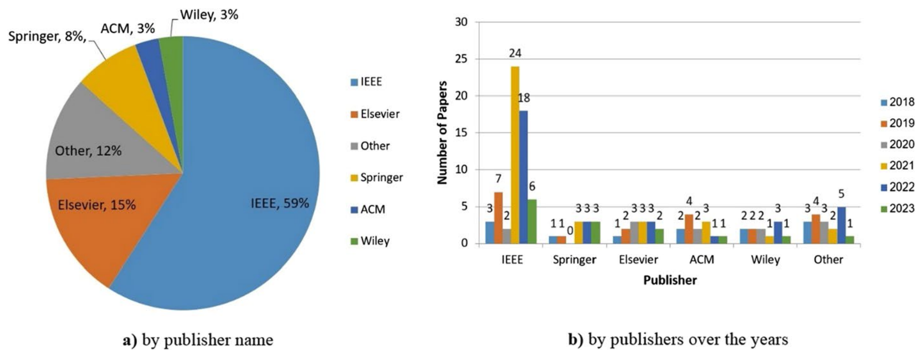
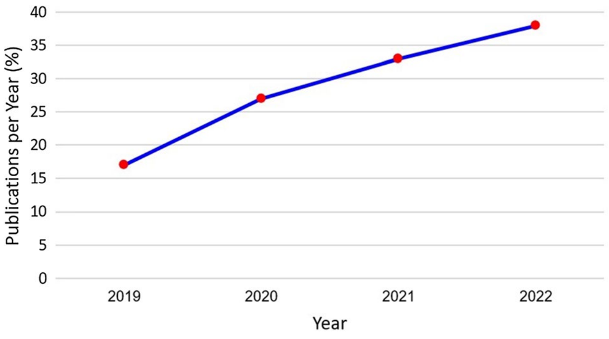


Fig. 6 Distribution of papers

Fig. 7 Percent of publication per year



done since 2018. In addition, our literature review shows 99 journal articles and 26 conference papers. To answer the research questions, the text of each article was carefully analyzed.

4 Service placement using machine learning

Middle-layer computing paradigms such as FC and MEC can solve the computational and network bottlenecks in large-scale IoE applications. These technologies complement cloud computing by providing processing and storage power at the edge of the network. Recently, there has been a growing trend to use ML to resolve middle-layer application problems such as resource management, security, latency, and power consumption. In this section, we examine the types of algorithms and technologies using machine learning algorithms. Figure 8 shows the different aspects according to the literature.

4.1 Unsupervised learning algorithms

Unsupervised learning is used to analyze and cluster unlabeled datasets (Taheri-abed et al. 2023). It can automatically discover hidden patterns or groupings of data. Items within a cluster should be similar and simultaneously dissimilar to other clusters' items.

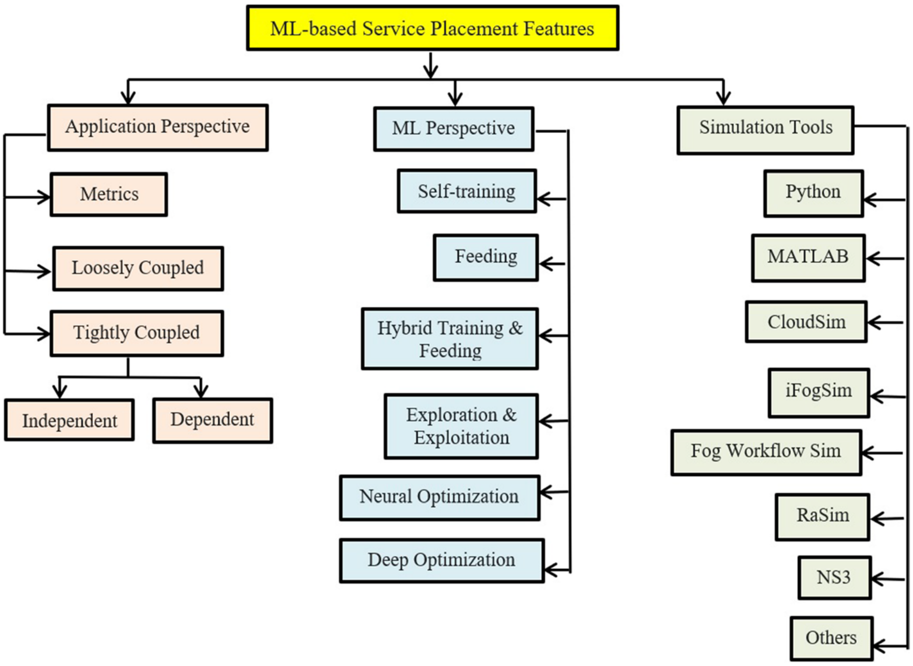


Fig. 8 Different aspects of service placement with machine learning

Definition 1 Suppose we represent a dataset containing N records with $\mathrm{X}=\{\mathbf{X}_1, \mathbf{X}_2, \ldots \mathbf{X}_N\}$. We also represent attributes of this dataset with $\mathbf{A} = \left[a_1\ a_2\ \ldots\ a_D\right]$. Therefore, each record $\mathbf{X}_i$ is in the form $\mathbf{X}_i = \left[a_{i,1}\ a_{i,2}\ \ldots\ a_{i,D}\right]$. The purpose of clustering is to classify records into Kclusters so that each cluster C_j meets the following conditions:

(a) $\bigcup_{j=1}^{K} C_j = \mathrm{X}$
(b) $C_j \neq \emptyset, \qquad j = 1, 2, \ldots, K$
(c) $C_j \bigcap C_k = \emptyset$

4.1.1 K-means

Traditionally, K-means has been used for clustering (Memon 2019). The K-means pseudo-code is shown in Algorithm 1. In line 1, the number of initial centers of the clusters is given to the algorithm as a predefined parameter K. In line 3, each record $\mathbf{X}_i$ is assigned to a cluster with the closest center. For this purpose, the distance between $\mathbf{X}_i$ and each cluster center $\mathbf{m}_j$ is measured. Then, the nearest cluster center to the record $\mathbf{X}_i$ is found as follows:

$$\min_j \left\{ \left\| \mathbf{X}_i - \mathbf{m}_j \right\| \right\}, \qquad \forall j \in \mathrm{M} \tag{4}$$

In line 4, the center of each cluster is updated as follows:

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{X}_k \in C_j} \mathbf{X}_k, \qquad j = 1, 2, \ldots, K \tag{5}$$

where N_j denotes the number of records in the cluster C_j. The above operation is repeated until the centers $\mathrm{M} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K\}$ do not change.

Algorithm 1 Pseudo-code of K-means clustering algorithm

```
Input: the dataset $X = \{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N\}$, number of records $N$, number of attributes $D$, number of clusters $K$
Output: the set of clusters $C = \{C_1, C_2, ..., C_K\}$
1:  Select $K$ items randomly from the dataset X as the initial cluster centers $M = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_K\}$
2:  repeat
3:      Place each record $\mathbf{X}_i$ in the cluster with the closest center to it.
4:      Recompute the centroid of each cluster.
5:  until (centroid does not change)
6:  return C
```

In some research (Li et al. 2019a), clustering is used to create several clusters of fog nodes. Each of these clusters is managed by a cluster head. It is determined based on various metrics such as residual energy. Table 5 in the Appendix shows clustering research using k-means. Farhat et al. (2022) propose an online mechanism for dynamic service chain deployment to optimize operational costs in a limited time frame. In some studies such as (Raghavendra et al. 2021; Yuan et al. 2020), a node is assumed to be a member of more than one cluster due to energy consumption and response time considerations.

4.1.2 G-means

In G-means, the number of clusters is determined iteratively using geometric evaluation and arithmetic analysis. de Oliveira et al. (2023) use G-means to reduce the load of parallel position-based query processing. The position of the source node in the wireless search zones is automatically determined by G-means. This research is a single-objective optimization problem.

4.1.3 Federated learning

In this method, a central server is used to coordinate the participating nodes during the learning process (Qian et al. 2019). It aggregates model updates after node selection. The weakness of this method is that the links leading to the server may become a bottleneck. Also, some complex topologies may affect the performance of the learning process. On the contrary, it has the advantage of avoiding single-point failures, since model updates are performed only between connected nodes, without the intervention of a central server.

In federated learning, the local models are assumed to have the same architecture as the global model. Recently, a new federated learning framework called HeteroFL (Mei et al. 2022) has been developed to handle heterogeneous clients with very different computing and communication capabilities. It can train heterogeneous local models with a global inference model. Federated learning provides the ability to analyze the social behavior of users based on their preferences on an hourly, daily, or weekly basis (Qian et al. 2019). Balasubramanian et al. (Balasubramanian et al. 2021) use a federated learning algorithm to manage the placement of content files in EDC and MDC sites. Federated learning has new applications, especially with the help of blockchain (Kim et al. 2018).

4.1.4 Unsupervised neural network

Neural networks are used to predict the reputation of nodes in combination with clustering (Hallappanavar et al. 2021). The reputation value of a new service is determined based on the correlation of various features and the reputation value of previous services. The features used to estimate service reputation are very diverse and depend on the application used. Neural networks consist of three layers: input, hidden, and output layers. Each neuron represents one service parameter. The output of the network is the prediction value of the service reputation. If the number of neurons in the hidden layer is small, the convergence of the neural network will be disturbed. On the contrary, if the number of neurons is large, overfitting may occur. Table 6 in the Appendix shows unsupervised neural network research.

4.2 Semi-supervised learning

Semi-supervised learning is a type of machine learning that can make predictions about other unlabeled samples based on a small number of labeled samples (Salaht et al. 2020).

4.2.1 Fuzzy systems

In fuzzy clustering, each sample can belong to more than one cluster. Three types of systems are mostly used in service placement: neural fuzzy system, fuzzy neural network, and neuro-fuzzy hybrid system. In neural fuzzy systems, neural networks are used to determine fuzzy rules. They change the weights during training to minimize the cost function. In other words, the fuzzy functions and rules act as the weights of the neural network. Conversely, in a fuzzy neural network, the inputs to the neural network are non-fuzzy. Finally, in hybrid neuro-fuzzy systems, each technique is used independently to perform a task (Alli and Alam 2019). Neuro-fuzzy systems are used to manage the parameters of a fuzzy system. They are used to predict and classify problems. They combine features of neural networks and fuzzy techniques. Here, fuzzy models work with if–then-else rules. In Son and Huh (2019), a fuzzy semi-supervised algorithm is used to meet the user's needs.

4.2.2 Semi-supervised classifier

Suppose a dataset contains N samples $N = N_1 + N_2$. Of these, N_1 samples have a label and the rest (N_2 samples) are unlabeled (Mohammed et al. 2022). To build a semi-supervised classifier, first, the training phase is performed on a small set of labeled samples. Then, a semi-supervised learning process is performed iteratively. In the test phase, probability theory is used to identify unlabeled samples. Labels with the highest probability prediction

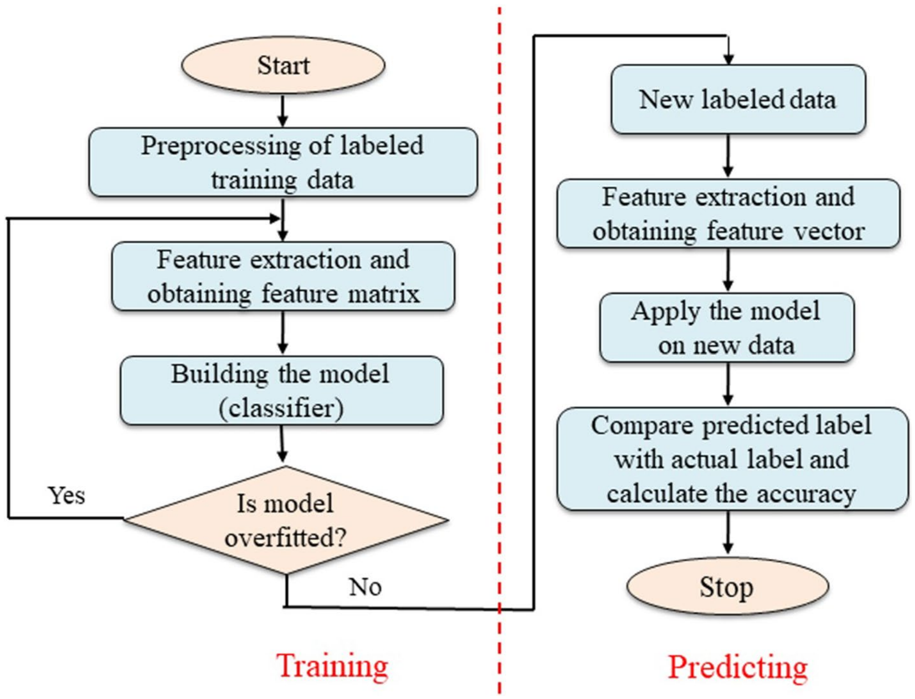

Start
Preprocessing of labeled training data
Feature extraction and obtaining feature matrix
Building the model (classifier)
Is model overfitted?
Yes
No
New labeled data
Feature extraction and obtaining feature vector
Apply the model on new data
Compare predicted label with actual label and calculate the accuracy
Stop
Training
Predicting

Fig. 9 Main operations in supervised learning

values are added to the dataset as new labels. Also, in some research (Mohammadi et al. 2017), deep semi-supervised learning is used to improve the performance and accuracy of the learning agent. The details of the semi-supervised learning algorithm and its characteristics are shown in Table 7 in the Appendix.

4.3 Supervised learning

Generally, supervised learning is divided into classification and regression (Taheri-abed et al. 2023). Our literature review shows that classification has been used more than regression for service placement. It is used when records of a dataset are labeled. Each sample i has a feature vector $\mathbf{X}_i$ and a judgment label y_i. Let us denote the number of samples by N and the number of attributes of each sample by D. For example, in binary classification, each sample i has two possible values for the judgment label.

Definition 2 Suppose we represent a dataset containing N records with $\mathrm{X}=\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\}$. We also represent attributes of this dataset with $\mathbf{A} = \left[a_1 \ a_2 \ \ldots \ a_D \right]$. Therefore, each record $\mathbf{X}_i$ is in the form $\mathbf{X}_i = \left[a_{i,1} \ a_{i,2} \ \ldots \ a_{i,D} \right]$. Each sample i has a label y_i. Our goal in the classification is to build a model that can predict the label y_j.

There are various classification methods, each of which has advantages and disadvantages. Interested readers can refer to Tan et al. (2016) for further study. Some of the most important classification methods are K-nearest Neighbors (KNN), Support Vector Machine

(SVM), Decision Tree, Random Forest, naïve Bayes, *Bayesian belief network*, Adaptive Boosting (AdaBoost), Extreme Gradient Boosting (XGBoost), the ensemble of classifiers.

According to the above explanations, supervised learning has two phases: training and predicting. As shown in Fig. 9, based on the labeled data X, a model, also known as a classifier, is built in the training phase. In the second phase, the model's accuracy is evaluated by injecting new labeled data. For each sample, if the label predicted by the model is the same as the actual label, it indicates that the model is working correctly.

4.3.1 Linear regression

We demonstrate, for example, linear regression, which is the most straightforward regression (Kochovski et al. 2019). Here we define a hypothesis weight vector $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \dots & \theta_D \end{bmatrix}$. Now, for a data record $\mathbf{X}_i = \begin{bmatrix} a_{i,1} & a_{i,2} & \dots & a_{i,D} \end{bmatrix}$, we describe the hypothesis function $h_\theta(\mathbf{X}_i)$ as follows:

$$h_\theta(\mathbf{X}_i) = \mathbf{X}_i.\boldsymbol{\theta}^T = a_{i,1} * \theta_1 + a_{i,2} * \theta_2 + \dots + a_{i,D} * \theta_D \tag{6}$$

Our goal is to minimize the cost function $J(\boldsymbol{\theta})$ (Baek et al. 2019). There are many ways to define a cost function. One of the most well-known cost functions is the Sum of Squared Error (SSE), which is defined as follows:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(h_\theta(\mathbf{X}_i) - y_i)^2 \tag{7}$$

As seen from Eq. (7), it calculates the squared difference of the hypothesis $h_\theta(\mathbf{X}_i)$ with the actual label y_i for all records (Yan et al. 2020). The ultimate goal of all regression methods is to minimize the cost function $J(\boldsymbol{\theta})$. There are many methods for this, the most famous of which is the descent gradient. In this method, we must first obtain the derivative of the cost function, $\partial J(\boldsymbol{\theta})/\partial\theta_j$, concerning each weight θ_j. If we assume that the cost function is in the SSE form of Eq. (7), the derivative is obtained as follows (Mahmud et al. 2020):

$$\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta}) = \sum_{i=1}^{N}(h_\theta(\mathbf{X}_i) - y_i).a_{i,j}, \qquad j = 1,\ 2,\ \dots,\ D \tag{8}$$

Then we update each weight θ_j based on the following equation:

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta}), \qquad j = 1, 2, \dots, D \tag{9}$$

where α is called the learning rate and is a hyperparameter. Many methods in ML use descent gradient, the most known of which are logistic regression and neural network. Algorithm 2 shows the linear regression pseudo-code using the descent gradient.

So far, a lot of research has been done with regression methods, including logistic regression for service placement. For example, in Bashir et al. (2022), using logistic regression, a request is sent to a suitable fog node that can fulfill it. The designed system can take into account the maximum utilization of resources of fog nodes. Table 8 in the Appendix shows the supervised algorithms for service placement. Since 2012, the growth in the use of neural networks has accelerated. This has led to less use of traditional methods. Interested readers can refer to relevant references to study the details of methods such as decision trees (Manikandan et al. 2020), Support Vector Machines (SVMs) (Adege et al. 2018), and K-nearest Neighbor (KNN) (Kim et al. 2018). For example, in Manikandan et al. (2020), the decision tree is used to place the service in e-health. Different parties (e.g. patient, hospital, etc.) register at the key generation center. It issues a pair of public–private keys to the users. The service placement operation is then performed. In Kim et al. (2018), KNN is also used to place the service in the IoT. Here, the KNN algorithm only returns the number of the nearest k providers. If the provider is not able to provide the service, the user has to find another provider. One of the most well-known types of neural networks is the Recurrent Neural Network (RNN). In Memon and Maheswaran (2019), it is used to optimize delivery in vehicular fog computing. The output of the RNN is fed into a feed-forward neural network with only one hidden layer containing 20 neurons with a sigmoid activation function. In recent years, the use of deep neural networks has become very popular (Ran et al. 2019). Table 9 in the Appendix shows the major neural network research in service placement. In addition, Table 10 shows the most important research on deep supervised learning.

Algorithm 2 Pseudo-code of linear regression using descendent gradient

```
Input: the dataset $X = \{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_N\}$, number of records $N$, number of attributes $D$, hypothesis weight vector $\boldsymbol{\theta}$, learning rate
    $\alpha$, hypothesis function $h_\theta(.)$, each record label $y_i$, the cost function $J(\boldsymbol{\theta})$
Output: Updated hypothesis weight vector $\boldsymbol{\theta}$
1:    Randomly initialize the vector $\boldsymbol{\theta}$
2:    repeat
3:        Update the weight vector $\boldsymbol{\theta}$ using Eq. (9)
4:    until (The value of $\boldsymbol{\theta}$ does not change)
5:    Return $\boldsymbol{\theta}$
```

4.3.2 Decision tree

A decision tree is a supervised learning approach. In a tree structure, leaves represent class labels and branches represent combinations of features. A special type of decision tree in which the target variable can take continuous values is called a regression tree. In a decision tree, data comes in records as $(\mathbf{X}_i, y_i) = (x_{i,1},\ x_{i,2}, ...,\ x_{i,D}, y_i)$. Here, y_i is called the dependent variable and is a label to be predicted. Also, $\mathbf{X}_i$ represents the feature vector for instance i, which was previously explained in Definition 2.

In some research, the decision tree has been used for service placement. For example, Manikandan et al. (2020) address the integration of Smart Health Care (SHC) systems with the cloud environment. To reduce the response time to patients, they propose an IoT-based scheduling method using a decision tree. It consists of three stages: registration, data collection, and scheduling. Using entropy and information gain, they include the decision tree in the third stage for scheduling. The results of their experiments show that the use of decision trees reduces the computational overhead and improves the scheduling efficiency. Interested readers can refer to Alsaffar et al. (2016), Noulas et al. (2012), Sriraghavendra et al. (2022) to study other decision tree research. The details of these studies are described in Table 8 in the Appendix.

4.3.3 Transfer learning

The Transfer Learning (TL) technique uses the knowledge gained during the construction of an initial model to create a new model for a secondary task. TL can dramatically reduce training time. It also reduces the amount of data needed to train the secondary model. As a result, it improves performance, especially in neural networks. In TL terminology, the primary (input) model is called the *source domain* and the secondary (output) model is called the *target domain*. Before proceeding, the reader is advised to review Definition 1 in Sect. 4.1 and Definition 2 in Sect. 4.3. First of all, let us provide the definitions of the *"domain"* and the *"task"*.

Definition 3 (Domain) Let A and $P(\mathrm{X})$ denote the feature space and the marginal distribution, respectively (Zhuang et al. 2020). Now we represent the domain by $\mathrm{D}=\{\mathrm{A}, \mathrm{P(X)}\}$. Recall from Definition 1 that we represent a dataset containing N records with $\mathrm{X}=\{\mathbf{X}_1, \mathbf{X}_2, \ldots \mathbf{X}_N\}$. We also represent attributes of this dataset with $\mathbf{A} = \left[a_1 \ a_2 \ \ldots \ a_D \right]$. Therefore, each record $\mathbf{X}_i$ is in the form $\mathbf{X}_i = \left[a_{i,1} \ a_{i,2} \ \ldots \ a_{i,D} \right]$. Each instance i has a label y_i.

Definition 4 (Task) Let T and Y represent the task and label space, respectively (Zhuang et al. 2020). Also, f represents the decision function. The task space T is now represented by $\mathrm{T} = \{\mathrm{Y}, f\}$. In other words, each task T depends on a label and a decision function. As in other ML methods, f is an implicit function learned from the datasets in the training phase. The decision function f, like some ML models (e.g., Bayesian), outputs a predicted conditional distribution for each input instance $\mathbf{X}_i$. This output is defined as

$$f(\mathbf{X}_i) = \left\{ p\left(y_j \mid \mathbf{X}_i\right) \middle| y_j \in \mathrm{Y}, \quad j = 1, 2, \ldots, |\mathrm{Y}| \right\} \tag{10}$$

Suppose we have different sources of data. For example, let $\mathbf{D}_S$ and T_S denote the domain and the task space of a source S, respectively. Now we represent an observation with instance-label pairs as follows

$$\mathbf{D}_S = \left\{ \left(\mathbf{X}_i, y_i\right) | \mathbf{X}_i \in \mathrm{X}^S, y_i \in \mathrm{Y}^S, \quad i = 1, 2, \ldots, N^S \right\} \tag{11}$$

Note that these instances can be labeled or unlabeled.

Definition 5 (Transfer Learning (TL)) Suppose we have some observations about $m^S \in \mathbb{N}^+$ source domains and tasks (Zhuang et al. 2020). These observations are represented by $\left\{ \left(\mathrm{D}_{S_i}, \mathrm{T}_{S_i}\right) \middle| \quad i = 1, \ldots m^s \right\}$. Also, suppose we have some observations about $m^T \in \mathbb{N}^+$ target domains and tasks. These observations are represented by $\left\{ \left(\mathrm{D}_{T_j}, \mathrm{T}_{T_j}\right) | \ j = 1, \ldots, m^T \right\}$. Note that m^T denotes the number of TL tasks. The purpose of TL is to use the knowledge of the source domains to improve the performance of the decision functions in the target domain, i.e., $f^{T_j}\left(j = 1, \ldots, m^T\right)$.

In the special case where $m^S = 1$, the scenario is called single-source transfer learning. Also, the literature review shows that most researchers currently focus on studying scenarios with $m^T = 1$ (Liu et al. 2021; Chen et al. 2019; Girelli Consolaro et al. 2023; Anwar and Raychowdhury 2020). This assumes that many labeled instances are available and that the model is well-trained in the source domain. On the contrary, there are

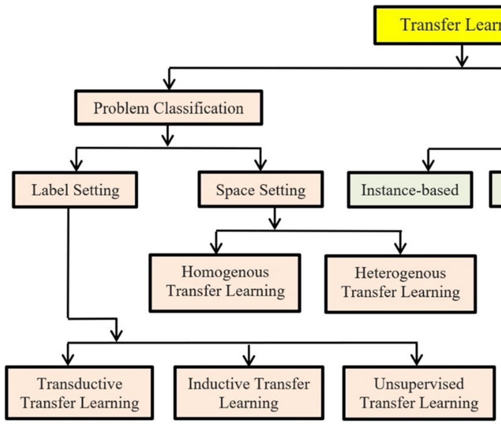

Transfer Learning
Problem Classification
Solution Classification
Label Setting
Space Setting
Instance-based
Parameter-based
Feature-based
Relation-based
Homogenous Transfer Learning
Heterogenous Transfer Learning
Symmetric Transformation
Asymmetric Transformation
Transductive Transfer Learning
Inductive Transfer Learning
Unsupervised Transfer Learning

Fig. 10 Transfer learning categories

usually limited instances available in the target domain. In these scenarios, resources such as instances and models are observations. The goal of TL is to obtain a decision function *f* with maximum accuracy in the target domain.

Another important concept in TL is *domain adaptation* (Zhuang et al. 2020; Hou et al. 2018). Here, the goal is to reduce differences between source domains to maximize knowledge transfer and improve learner performance.

TL problems are divided into three categories: Transductive, inductive, and unsupervised (Zhuang et al. 2020). If the instances of the target domain have labels, we have an inductive TL. If the source and target instances do not have labels, we have an unsupervised TL. If $X^2 = X^T$ and $Y^2 = Y^T$, this scenario is a homogeneous TL. Otherwise, if $X^S \neq X^T$ or/and $Y^S \neq Y^T$, this scenario is a heterogeneous TL. As shown in Fig. 10, TL approaches are also divided into four groups: instance-based, feature-based, parameter-based, and relation-based. In instance-based approaches, instance-weighting techniques are used. Also, in feature-based methods, transformations of original features are used to obtain new features. It has two types, asymmetric and symmetric. In the asymmetric approach, it is tried to change the features of the source to match the features of the target. On the contrary, in symmetric approaches, the commonality between the source and the target is calculated and new features are created based on it. Parameter-based approaches try to transfer knowledge based on the model. The relational approach tries to extend the rules learned in the source domain to the target domain.

TL has been used in some important research. Hou et al. (2018) propose a proactive caching mechanism using TL. Their goal is to minimize the transmission cost while improving the QoE. They use TL to estimate the popularity of the content. Due to the NP-hardness of the problem, they devise a greedy algorithm to solve the cache content placement problem.

Typically, the training of the “generic” model is done offline in the cloud data center because it is computationally intensive. The resulting model is then deployed on the local nodes to make corrections through incremental training. For example, for real-time vision applications (such as face recognition), a “general” model is first trained on millions of faces. This model is then copied to the edge nodes so that users can customize the model according to their preferences (Yu et al. 2020). The “generic” model may be retrained from time to time to improve the initial accuracy. In such cases, updates should be sent periodically to the edge servers (Nisha 2018).

4.4 Reinforcement learning

Reinforcement learning (RL) mainly relies on the Markov Decision Process (MDP). It is a stochastic process, where the transition between states depends only on the current state of the process, not on a previous sequence of events. So far, a lot of research has been done on MDP-based computer networks. Some examples are estimation of service prices (Naghdehforoushha et al. 2022), and allocation of network resources (Besharati et al. 2023; Rawashdeh et al. 2021). It allows for a description of the potential system behavior with an automatic model. In its most general form, an MDP is a tuple $(\mathrm{S},\ \mathrm{A},\ \mathrm{P},\ \mathrm{R},\ \gamma)$ where $\mathrm{S} = \{s_0,\ s_1,\ \dots,\ s_N\}$ and $\mathrm{A} = \{a_0,\ a_1,\ \dots,\ a_N\}$ represent a finite set of states and actions, respectively. Also, $\mathrm{P} = \{s_{t+1} = s' |,\ s_t = s,\ a_t = a\}$ is the probability of transition from state s in stage t to state s' in the next stage. Finally, the expected reward received after the transfer is represented by $R(s,\ s')$. Here, to show the difference in importance between current and future rewards, the discount factor γ is used, which is a value between 0 and 1. An example of a state is the benefit a certain QoS may bring to the network (Kochovski et al. 2019). Usually, the use of RL for service placement leads to a significant improvement in performance (Baek et al. 2019). For example, in Sefati and Navimipour (2021), MDP is used to increase reliability in combination with Ant Colony Optimization (ACO). The authors prove that it is more efficient than previous studies (Liu et al. 2022).

Recently, Multi-objective Reinforcement Learning (MORL) has gained much attention in research (Eyckerman et al. 2022). In addition to the elements of a normal MDP, it has an additional parameter called the preference space. It takes a preference as input and returns a utility scalar for each action in a given state. The goal is to maximize utility.

4.4.1 Q-learning

Q-learning is a model-free RL algorithm in the sense that it does not require a model of the environment. It finds an optimal policy by maximizing the expected value of the total reward in all successive steps, where *"Q"* refers to the function computed by the algorithm (Gasmi et al. 2022). The agent tries to add the weighted reward of future states to the reward of the current state to maximize the total reward. The weight of each step is represented by $\gamma.\Delta t$. It has a function to compute the quality of the state-action, which is defined as Q:S × A → R. At any time t, the agent chooses an action α, observes a reward r_t, and then enters a new state. This updates the value of Q. Based on the Bellman equation, Q-learning updates the Q-value as follows:

$$Q^{new}(s_t,\ a_t) = Q^{old}(s_t,\ a_t) + \alpha\left(r_t + \gamma . \max_a Q(s_{t+1},\ a_t)\right) \tag{12}$$

In the above equation, α is the learning rate and is a value between 0 and 1. The Q-learning algorithm continues until it reaches a terminal state.

Appendix A-8 shows the Q-learning algorithms used for service placement. For example, Wang et al. (2019) use Q-learning to solve the online service tree placement problem. They show that for an edge network of size n and a service tree with m nodes, the action space has an exponential time complexity of $O(n^m)$. To resolve this problem, they propose an efficient hierarchical placement strategy to significantly reduce the action space. In another research (Yan et al. 2020), Q-learning is used for content placement (cache) to maximize the efficiency of the F-RAN. Also, Nsouli et al. (2022) address the accessibility of cars in VEC. In another study (Ghobaei-Arani et al. 2018), hybrid resource provisioning for cloud networks is addressed.

4.4.2 R-learning

R-learning has a similar concept to the Q-learning method. Both use Q-tables, states, actions, rewards, and losses. However, the main difference is that R-learning uses the concept of average reward rather than discount reward. It calculates the average total transfer reward between all states and actions. R-learning provides total rewards sometimes, but punishments whenever the model fails to meet expectations. Farhat et al. (2020) use R-learning to maximize voluntary resources. The predictions of this algorithm help to increase QoS and also keep the pressure on cloud resources low. This is done by studying user behavior during service requests.

4.4.3 Meta-learning

The data of a meta-learning system usually comes from different domains. It is based on learning biases. Here, bias refers to the assumptions that influence the choice of explanatory hypotheses. Meta-learning is concerned with two aspects of learning biases, which are

Declarative bias:	It determines how the hypothesis space is represented and has a major impact on the search space (e.g., representing hypotheses using only linear functions).
Procedural bias:	It imposes restrictions on the ordering of inductive hypotheses (e.g., favoring smaller hypotheses).

Recently, meta-learning systems have been equipped with the power of neural networks. It has a tremendous effect on increasing accuracy by injecting too much training data. However, since it involves second-order gradient computation, it can increase the execution time, especially for large-scale problems. Therefore, in problems such as service placement, which inherently have a large search space, meta-learning can greatly affect the portability of the model and the level of decision-making. To overcome this problem, some research (Chen et al. 2022a) uses heuristics. For example, we can mention the elimination of the second-order gradient calculations. For further reading, you can refer to Chen et al. (2020), Zhang et al. (2021), Arif et al. (2020).

4.4.4 K-armed Bandit

In this classic problem, you have the opportunity to select K levers in a game city. Each lever fires an action. After each choice, you receive a numerical reward, which corresponds to a probability. The goal of this method, like other RL methods, is to maximize the total expected reward (action value) in a given period. For each action taken, we denote its value by $q(a)$. Now, the action reward is calculated as follows

$$q^*(a) = E[r_t | a_t = a] \tag{13}$$

The most important point in this process is that you do not know the value of an action in advance. Instead, you try to greedily choose the action that you think is likely to be worth more than the others (Borelli et al. 2022). For this, you may have estimates of an action a, which we denote by $Q_t(a)$. Let us denote the best estimate by $q^*(a)$. Therefore, we want the value of $Q_t(a)$ to be as close as possible to the value of $q^*(a)$.

Some research has used the K-Armed Bandit for service placement. For example, in Huang et al. (2022), service storage at the edge is solved by considering edge servers as agents. Here, services are considered as arms, service placement decisions are considered as actions, and system tools are considered as rewards.

4.4.5 Bayesian optimization

In Bayesian Optimization (BO), operations are performed based on previous observations and the posterior distribution of the solution vector (Eyckerman et al. 2022). We assume that the utility function $T_{resp}^{\max}(A)$ follows a normal distribution. After d iterations, BO modifies the prior belief function based on the posterior distributions. It uses an "acquisition function" (EI) to select the settings of the iterations. Let $D_d = \{(a_1,\ T_{resp}^{\max}(a_1)),\ ...,\ (a_1,\ T_{resp}^{\max}(a_d))\}$ be the set of prior observations after d iterations. We assume that it follows a normal distribution with mean $\mu(0)$ and covariance kernel function $K(0, 0)$, i.e., $p(T_{resp}^{\max}(A)) = N(\mu, K)$. We also formulate the mean and variance as follows:

$$\mu(a_u) = \mu(T_{resp}^{\max}(a_u)) \tag{14}$$

$$K(a_u,\ a_v) = E[T_{resp}^{\max}(a_u) - \mu(a_u)T_{resp}^{\max}(a_v) - \mu(a_v)] \tag{15}$$

Roberts et al. (2018) use BO to automate the placement of time series models for IoT healthcare applications. For details of other research, see Table 11 in the Appendix.

4.4.6 SARSA

Like Q-learning, SARSA is one of the well-known reinforcement learning methods. Here, the agent tries to find the optimal policy $\pi^*(s) : \mathrm{S} \to \mathrm{A}$ by minimizing a cost function (Sami et al. 2020). After checking the previous rewards, the next action a_{t+1} is selected using the state value function $V\pi(s)$. Since the SARSA method assumes that the agent does not know the environment, the action value function $Q(s, a)$ is used. Action selection is greedy based on $Q(s, a)$. In SARSA, the optimal policy π^* is defined as follows:

$$\pi^*(s) = \arg\min_a\ Q^*(s, a), \qquad \forall s \in S \tag{16}$$

Sami et al. (2020) perform optimal scaling and placement in each cluster of fog nodes by finding π^*. For the details of other research, see Table 11 in the Appendix.

4.4.7 Deep RL

One of the drawbacks of Q-learning is low scalability due to the large number of state-action pairs. This can mislead the agent in choosing the optimal policy (Zabihi et al. 2023).

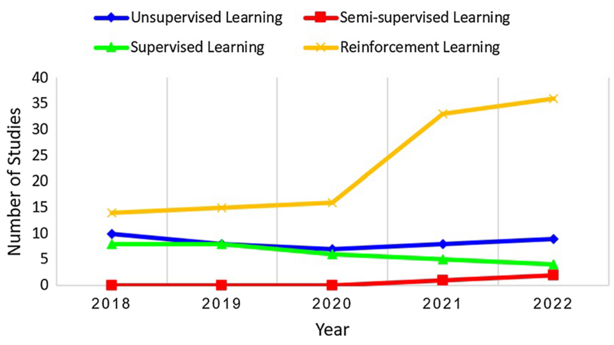


Fig. 11 The trend of different types of machine learning in service placement

A deep Q-network (DQN) is used to overcome this problem. Previous research has used several variants, which we do not review here (Liu et al. 2020; Qi et al. 2020). DQN uses a neural network to generate a value function vector. Despite its many advantages, DQN may suffer from overfitting due to the high correlation between inputs. To overcome this problem, a replay buffer is typically used (Taheri-abed et al. 2023). The loss function in DQN is defined as follows:

$$L(\theta) = E[(y - Q(s,\ a|\theta))^2] \tag{17}$$

In the above equation, the target value $y = r + \gamma \max_{a'} Q(s',\ a'|\theta'_i)$ is obtained from the Bellman equation. It minimizes the error between the target Q-values and the predicted value. Table 12 shows the major studies on service placement using deep RL. For example, Memon and Maheswaran (2019) use DDPG for SFC-DOP in dynamic and complex IoT. It can handle high-dimensional input and continuous output actions. Therefore, it is suitable for more complex IoT network scenarios with large and continuous operation space. Qi et al. (2020) use deep RL for the policy and value function. Here, the input to the neural network is a temporal state. The parameters of the shared layers can understand the resource differences between multiple tasks and learn to minimize the total task execution time in each application.

5 Discussion and analysis of results

In this section, a complete analysis of questions RQ1 to RQ7 will be given based on the information obtained. The future research process will also be highlighted.

RQ1:	Most of the algorithms used in service placement are related to which area of machine learning?
Analysis result:	As can be seen in Fig. 11, the use of reinforcement learning methods is associated with a significant jump starting in 2020. At almost the same time, the number of clustering studies has decreased slightly. Due to the emergence of new reinforcement learning methods combined with deep learning, this area is expected to expand further in the coming years. The emergence of new methods such as adversarial RL, actor-critic, and Soft Actor-Critic (SAC) may also accelerate this growth. Interested readers are advised to refer to Zabihi et al. (2023) for a more in-depth study.

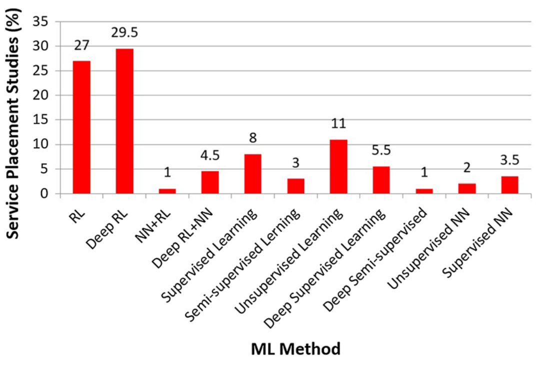


Fig. 12 Distribution of machine learning algorithms in service placement

Figure 12 also shows that deep RL methods have been used almost 30% of the time in service placement research. The details of the deep RL method are shown in Table 12 in the Appendix. As can be seen from Fig. 12, after deep RL, about 27% of service placement studies are conducted with pure RL methods. As mentioned above, most of the learning that has happened in the world of artificial intelligence is related to the field of reinforcement learning. From 2012, after deep and neural network technologies were added to these methods, they became more popular. The main reason for this popularity is the compatibility of these methods with model-free environments. As shown in Fig. 12, among other machine learning methods, unsupervised methods are more popular (11%) than supervised and semi-supervised methods (8 and 3%, respectively). As we can see from the figure, the combination of deep and neural network technologies has created a new trend in the methods of solving the problem of service placement. It is expected that in the coming years, we will see an increase in these algorithms combined with newer technologies.

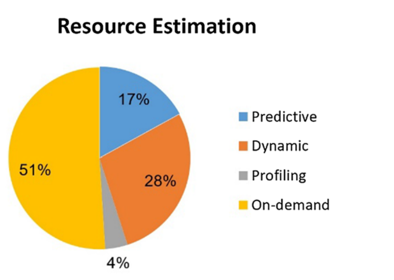
RQ2: What methods are used to estimate the amount of current network resources, and which resource estimation method is more popular?

Analysis result: As we can see in Fig. 13a, among the resources studied, 51% use the on-demand method to estimate system resources. About 28% of them use the dynamic method to estimate the available resources in the system. Also, 17% of the studies use the predictive method and about 4% use the profiling method. Profiling methods are mostly used in static environments to estimate resources.

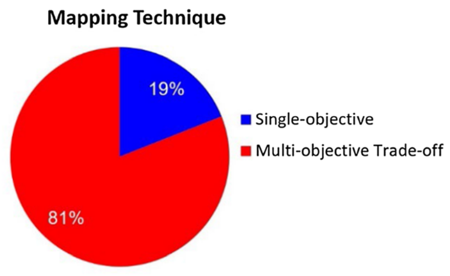
RQ3: The purpose of service placement is optimization/trade-off between which parameters and metrics?

Analysis result: The answer to this question determines what percentage of the algorithms try to optimize a single metric and what percentage tries to find a compromise between several metrics. As we can see in Fig. 13b, about 81% of the methods try to find a compromise between several parameters, and about 19% of them try to optimize a single parameter. Compromising among multiple parameters is more in line with real-world requirements.

RQ4: Can the type of application architecture influence the choice of method and its efficiency?

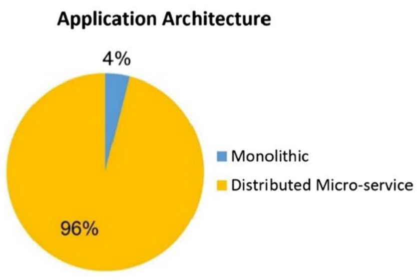


a) Distribution of computing paradigms in service deployment

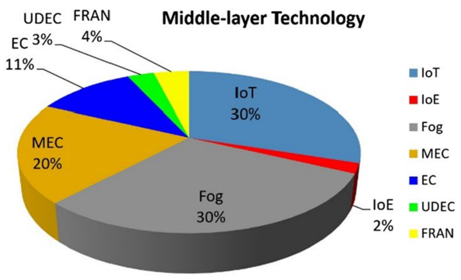


b) Mapping techniques used in service placement research

Fig. 13 Distribution of computing paradigms and mapping techniques in service placement



a) Application architectures in service placement



b) Distribution of resource estimation techniques

Fig. 14 Application architectures and distribution of resource estimation techniques in service placement

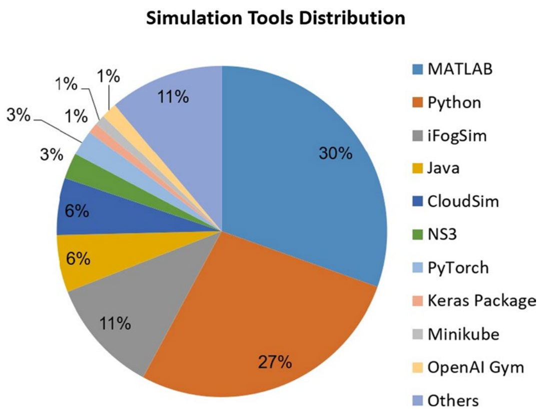
Analysis result: In the real world, most applications today are developed in a distributed manner. Managing separate parts of programs that have interdependencies requires methods that can dynamically perform resource management. According to our findings in Fig. 14a, about 96% of applications and related programs are developed in a distributed fashion, and a small number, about 4%, are developed in a monolithic fashion. Managing service placement in monolithic applications is very simple. However, proper performance may not be achieved.

RQ5: What is the dominant paradigm used in the interface layer?

Analysis result: The answer to this question determines the percentage of interface paradigms that can play a major role in increasing network efficiency. As shown in Fig. 14b, IoE is an emerging technology that will gradually impact human applications in the network. Soon, complementary technologies such as Fog Radio Access Network (FRAN), FC, and MEC will be integrated. Currently, IoT and FC technologies are the most popular in studies. The emergence of cloud computing was in 2012 by Cisco. However, with the increase of mobile wireless devices with high processing power, it seems that technologies such as FRAN and UDEC will be more popular in the near future.

RQ6: What simulation tools have been used in previous research to evaluate efficiency?

Fig. 15 Distribution of simulation tools in service placement



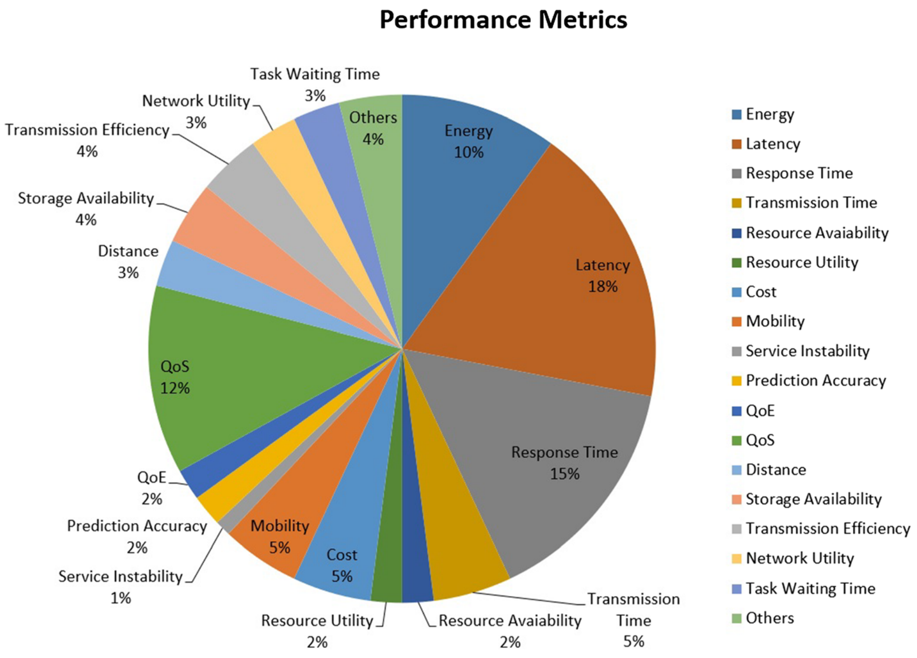


Fig. 16 Major metrics used in previous research

Analysis result: As can be seen in Fig. 15, 20% of the studies have performed simulations using the Python language. Also, about 30% of them have used MATLAB tools to analyze the results. Well-known tools such as CloudSim and iFogSim are both used in the analysis of the results with the same amount of 11%. Reinforcement learning methods often use MATLAB and Python tools for simulation.

RQ7: What were the main objectives of previous research?

Analysis result: Fig. 16 shows the main metrics used in previous research. As can be seen from the figure, delay, energy, QoS, and response time are the most important metrics used in the research.

6 Future directions and research challenges

In general, the fundamental challenges in service placement are task correlation, dynamic environment, server heterogeneity, resource management, mobility, scheduling handling, destination node selection for offloading, energy consumption, delay, and efficient resource allocation. The eighth question of this research deals with the challenges and open issues and the research process of service placement.

RQ8: What are the most important open issues in the future?
Analysis result: Based on the literature review, we have identified several open issues, which we describe below:

7 IoE

Recently, in addition to IoT, a new paradigm called the Internet of Everything (IoE) has emerged. IoT emerged to meet the needs of communication and connectivity between things, while IoE emerged as a more general paradigm. In addition to communication between things, it seeks to address human-to-human communication, processes, and data to create a digital world. In simpler terms, IoT discusses issues such as data collection, sending, and receiving, while IoE addresses the collaboration of intelligent systems, and advanced analytics for processing the big data. Therefore, service placement in heterogeneous and dynamic environments is more related to IoE. In new network applications, the heterogeneity of nodes can cause many inconsistencies in inter-communication.

7.1 Mobility management

One example of the heterogeneity of nodes in IoT and IoE is mobility. The network topology may change rapidly due to the movement of nodes. In such a situation, a service placement strategy may be useless in the near future. Mobility management is closely related to issues such as power consumption, latency, and security. Next Generation Critical Communications (NGCC) has a strong relationship with IoT and IoE. It provides emergency services to the community. The main application areas of NGCC are disaster management, smart city, citizen security, protection of human life, and e-health. It relies heavily on cellular communication, MEC, and FC. It is expected that an important part of IoE research will flow into this area. Suppose a resource has been detected in a position a few seconds ago. The algorithm must be able to work intelligently and quickly so that the resource does not become unavailable due to mobility. The problem becomes more difficult when often conflicting objectives are to be optimized by considering mobility. Therefore, it may be necessary to run the service placement algorithm almost continuously. Simply put, service placement algorithms must be intelligent enough to perform offloading based on a snapshot of the state of available resources.

7.2 Multi-objective trade-offs

Service placement in the real world is a multi-objective issue. This may impose more processing load on the system. In such a decision, it is important to pay attention to the limitations of resources and the environment. The trade-off between multiple metrics in a heterogeneous and dynamic network is likely to be much more complicated than in single-objective problems.

7.3 Processing overhead

Applications that generate large volumes of data are proliferating in the IoE. As a result, the consumption of processing resources becomes an issue in service placement. On the other hand, advanced ML methods such as deep neural networks themselves require huge processing resources. Handling such large volumes of data, especially for multi-objective problems, adds to the difficulties. Performing such tasks, even if they are not continuously iterated, requires a centralized entity. Currently, among the computing paradigms, only cloud data centers can handle such a volume of computations for training ML models. Moreover, cloud data centers have higher availability because they are constantly powered, unlike edge/fog servers.

Some ML methods, such as RL, require interaction with the environment. This can make service placement more difficult. Remember that in RL, the environment may be constantly changing. In addition, the existence of heterogeneity in resources can complicate the situation. For this reason, the execution frequency of RL algorithms can be much higher than other ML methods, which increases the cost. Currently, a large part of RL research is focused on finding a trade-off between exploration and exploitation (Xiao et al. 2022).

7.4 Environmental heterogeneity

One of the major limitations that computing paradigms impose on ML is environmental heterogeneity. For example, one of the emerging ML methods for service placement is Federated Learning (FL) (Brecko et al. 2022). It is a distributed technique for building a global model by learning from multiple decentralized edge clients. The main advantages of FL are scalability and user privacy (Qian et al. 2019). However, the inherently heterogeneous environment of the IoE can affect the computational complexity of FL (Dworzak et al. 2023). End devices and edge/fog servers may have very different hardware/software infrastructures. For example, they may have significant differences in CPU, RAM, link quality, and operating system. As mentioned earlier, one idea may be to perform the ML training phase using engines such as Spark and TensorFlow on the central cloud (Xu et al. 2023).

As a rule of thumb, the training phase can be run with large data on cloud data centers, and then the test/validation phase can be run with small data on edge/fog servers (Zhou et al. 2019). Recently, the use of pre-trained models to reduce the burden of the training phase has received a lot of attention from researchers (Sufian et al. 2020; Hsu et al. 2022). First, the model is trained with a lot of data in the cloud data center. Then, the pre-trained model is copied to the edge nodes and used many times. However, some entities/users may want to incrementally change the built model from time to time. Therefore, there is a need to formulate standards for integrated deployment of code and models based on different deployment policies. Also, debugging the models created by ML in a decentralized way may be another future challenge (Nisha 2018).

8 Conclusion and future trends

In this study, machine learning-based service placement research was addressed. A review of the literature showed that the use of reinforcement learning methods has increased significantly since 2020. The main reason for this popularity is the compatibility of these methods with model-free environments. Due to the emergence of new reinforcement learning methods combined with deep learning, this field is expected to expand further in the coming years. The emergence of new methods such as adversarial RL, Actor-Critic, and Soft Actor-Critic (SAC) may also accelerate this growth. In addition, deep RL methods have been used in service placement research for nearly 30% of the time. Among other machine learning methods, unsupervised methods are more popular (11%) than supervised and semi-supervised methods (8 and 3%, respectively).

Among the resources studied, 51% use the on-demand method to estimate system resources. About 19% of them use the dynamic method to estimate the available resources in the system. Also, 12% of the studies use the predictive method and about 7% use the profiling method. About 81% of the methods try to find a compromise between several parameters and about 19% of them try to optimize a single parameter. About 20% of the studies have done simulations with Python language. Also, about 30% of them have used MATLAB tools to analyze the results. Well-known tools such as CloudSim and iFogSim are both used in the analysis of the results with an equal amount of 11%. The literature review also revealed that delay, energy, QoS, and response time are the most important metrics used in service placement research.

Based on the literature review, we identified several open issues, including IoE, mobility, multi-objective trade-offs, processing overhead, and environmental heterogeneity.

This study showed that most ML methods require a lot of computing resources to perform training phase operations. This operation should mainly be performed in cloud data centers. Here, the single point of failure is a serious concern. Therefore, an important part of future studies will continue to focus on reliability. On the other hand, the central controller needs to communicate with all network nodes, which may increase the computational complexity. One of the research trends may be to cluster a set of nodes and leave their leadership to a cluster head. The use of metaheuristic/heuristic methods continues to be of interest to researchers as a solution to reduce computational complexity. Another trend in recent research is to find a trade-off between exploration and exploitation in ML methods. It is possible to increase the speed of convergence by reducing the number of states, while not degrading much efficiency. Considering the extreme heterogeneity of end devices and edge/fog servers, it seems that the main focus of researchers should be on using model-free ML methods.

Another way to reduce the computational burden on a centralized entity is to estimate the resources needed in the near future. By identifying the underloaded/overloaded servers and combining this with the requirements of the ML model, a short-term prediction of the resources can be achieved. This can also be used for load balancing. Transfer learning methods can be used here. In this way, the knowledge learned in one problem can be used for the next steps with only a small change. It is only necessary to make small changes in the weights of the neural network. Thus, prior knowledge can greatly reduce the computational overhead. This, in combination with pre-trained models, can open up new areas of research for the future.

One of the future research trends may be federated learning. Here, in addition to the above concerns, the imbalance of aggregated data from different sources is a serious concern. Local data generated by some users may be biased towards a particular class label. This can significantly affect the speed of the training process. The use of data balancing methods can help alleviate this problem.

Recent advances in ML have made it possible to combine online and offline training methods to improve accuracy. This is particularly useful for edge devices. Users may run the same centrally trained model, but then evolve it online to address specific edge scenarios. Debugging an ML model as a single entity rather than a collection of independent entities can improve performance.

Appendix: This appendix contains some tables and sub-sections of the original article that were moved here due to space limitations

See Tables 4, 5, 6, 7, 8, 9, 10, 11, and 12.

Table 4 Attributes that were extracted from articles

Publication date	Year of publication
Bibliographic date	Name of authors, the title of the paper, and publication venue
Paper type	Conference or journal
DOI	Digital object identifier
Objective	Purpose of the paper
AI method	The AI algorithm that used
Paradigm	The complementary paradigm to which this research relates
Analytic tools	Simulation or analyzer used for evaluation
Case study	The proposed method
Functional layout	Indicates whether the app architecture is loosely-couple or tightly-couple
Service type	Indicates whether the SP is used for storage actuation or display
Resource type	Indicates whether the resource used is a VM or container
Resource orientation	Indicates the arrangement of resources in the environment
Resource estimation	Indicates whether the resource estimation is predictive or dynamic or profiling
Placement approach	Indicates whether the SP is bottom-up, up-bottom, or hybrid
Objective	Multi-objective trade-off or optimization
Aim	The main goal of the article
Mobility	Indicates whether the mobility is supported or not

Table 5 Unsupervised algorithms that were used for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Farhat et al. (2022)	Unsupervised/K-Means	Predictive	Resource Management & Availability	Distributed microservice	MEC	Java	Large-scale container placement
Li et al. (2019a)	Unsupervised/K-Means	On-demand	Latency & power consumption	Distributed microservice	Fog	Java	Efficiently realized the requirement of low-latency communication
Oliveira et al. (2023)	Unsupervised/Clustering	Dynamic	Latency	Distributed microservice	IoT	Java	Facilitate the reproduction of the work, all the code and data produced are publicly available
Aznavouridis et al. (2022)	Unsupervised/Partitioning	Dynamic	Traffic Load & Cost	Distributed microservice	IoT	Kubernet	Handles the problem of service placement as a graph clustering one
Sangaiah et al. (2019)	Unsupervised/KNN	Dynamic	Network Latency and computing latency	Distributed microservice	MEC	Java	Confidentiality of roaming PBS users
Kochovski et al. (2019)	Unsupervised/K-Means	Dynamic	response time & bandwidth	Distributed microservice	Fog	iFogSim	Improve the matching between service requirements and the characteristics of resources
Raghavendra et al. (2021)	Unsupervised/K-Means	Dynamic	Energy Consumption & Response Time	Distributed microservice	Fog	iFogSim	Allow a node to be included in more than one cluster
Yuan et al. (2020)	Unsupervised/K-Means	Dynamic	Delay & Cost & High QoS	Distributed microservice	EC	MATLAB	Allocate the service resources at low cost conveniently

Table 5 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Qian et al. (2019)	Unsupervised/Federated Learning	Dynamic	Bandwidth & Storage	Distributed microservice	MEC	MATLAB	Privacy-aware service placement
Nouri et al. (2021)	Unsupervised/K-Means	On-demand	Maximize Energy Efficiency and minimize the Number of Needed Drones Minimize & the Cost	Distributed microservice	Unmanned Aerial Vehicle Cloud Computing	Python	Jointly optimize the 3-D UAV placements, transmit power, and cloud resources
Balasubramanian et al. (2021)	Unsupervised/Federated Learning	Dynamic	Cache Size & Latency	Distributed microservice	EC	Python	Manage the content file placement at EDC and MDC sites
Wahab et al. (2019)	Unsupervised/K-Means	Dynamic	Latency & Hardware Utilization	Distributed microservice	Fog	MATLAB	VNF placement

Table 6 Unsupervised neural network algorithms for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Kim et al. (2016)	Unsupervised/Neural Network	On-demand	Resource Scheduling & Response Time	Distributed micro-service	IoE	Python	Providing context-aware service during unpredicted situations in the IoE environment
Hallappanavar et al. (2021)	Unsupervised/Neural Network	Profiling	Response time	Distributed micro-service	Fog	iFogSim	Estimation of the reputation of newcomer web services in fog computing

Table 7 Semi-supervised algorithm for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Alli and Alam (2019)	Semi-Supervised/Fuzzy	Dynamic	Response time & Energy & Latency	Distributed micro-service	Fog-Cloud	NS3	Secure computation offloading scheme in Fog-Cloud-IoT environment
Mohammed et al. (2022)	Semi-supervised/Self-Training Classifier	Dynamic	User Speed & Number of Requests & Spread of Communication Range	Distributed micro-service	IoT	Python	Combining a small number of labeled data sets with a large number of unlabeled data sets
Son and Huh (2019)	Semi-Supervised/Fuzzy	Dynamic	Maximize Energy Efficiency & Maximize Performance & Reduce Cost	Distributed micro-service	Small Cloud	NS3	Satisfy the user's requirements based on the service
Ramezani Shahidani et al. (2023)	Semi-Supervised/Fuzzy	Dynamic	response time, energy, latency	Distributed micro-service	Fog-Cloud	NS3	Propose a secure computation offloading scheme in a Fog-Cloud-IoT environment (SecOFF-FCIoT)

Table 8 Supervised algorithms for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Tuli et al. (2021)	Supervised/Back-Propagation of Gradients	On-demand	Low Energy Consumption & Response Time	Distributed micro-service	Fog	CloudSim	Create a hybrid simulation-driven decision approach, to optimize Quality of Service (QoS) parameters
Kim et al. (2018)	Supervised/KNN	On-demand	Security & Optimized Assignment	Distributed micro-service	Fog	iFogSim	Managing location data of IoT service provider
Bukhari et al. (2022)	Supervised/Logistic Regression	Profiling	Time & Latency & Energy Consumption, Bandwidth & Total Network Utilization	Distributed micro-service	Fog-cloud	iFogSim	Task offloading in a fog-cloud
Teoh et al. (2021)	Supervised/Logistic Regression	On-demand	Time & cost & Energy	Distributed micro-service	Fog-Cloud	FogWorkflowSim	Industrial Internet of Things (IIoT) eases information visibility and process automation
Alsaffar et al. (2016)	Supervised/Decision Tree	Profiling	Services Size & Completion Time & VMs Capacity	Distributed micro-service	Fog	CloudSim	IoT service delegation and resource allocation
Noulas et al. (2012)	Supervised/Linear Regression and M5 Trees	Dynamic	Power	Distributed micro-service	IoT	Mathematics analysis	Predicting the next venue a mobile user will visit
Sriraghavendra et al. (2022)	Supervised/Decision Tree	Dynamic	Response time latency and resource Utilization	Distributed micro-service	Fog	iFogSim	Multi-tier fog computing architecture called Deadline-oriented Service Placement (DoSP)

Table 8 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Taka et al. 2022)	Supervised/Linear Regression	Dynamic	Efficiency & reliability	Distributed micro-service	MEC	iFogSim	Preventive start-time optimization against a single BS failure in MEC networks
Santos et al. (2022b)	Supervised/Regression	Dynamic	Service Provisioning & Latency	Distributed micro-service	Fog	MATLAB	Improve the positioning of multimedia services
Su et al. (2022)	/supervised/regression	Dynamic	Dynamic	Distributed micro-service	MEC	MATLAB	joint service placement and request scheduling
Panadero et al. (2021)	Supervise Logestic Regression	Dynamic	Bandwidth	Distributed micro-service	EC	Python	Presents the Multi-Criteria Optimal Placement method, a novel and fast two-stage multi-objective method to place services in decentralized community network edge micro-clouds
Hou et al. (2018)	Transfer Learning	On-demand	Latency & transition cost	Distributed micro-service	MEC	Python	Proposing an active storage mechanism based on transfer learning to reduce transmission cost while improving QoE for future mobile networks

Table 9 Supervised neural network algorithms for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Peng et al. (2022)	Supervised/Back Propagation Neural Network	Dynamic	Reduce Spectrum-Sensing & Energy Consumption & Increase Spectrum Utilization	Distributed micro-service	NGWN	Python	Expanding a securing radio resources allocation scheme with Deep Reinforcement Learning for IoE services in NGWN
Danish et al. (2021)	Supervised/Neural Network	On-demand	SLA & Storage Availability	Distributed micro-service	Fog-Cloud	javascript	A neural network-based middleware designed for intelligent selection of storage technology for IoT applications
Memon and Maheswaran (2019)	Supervised/Three-Layer Feed-Forward Neural Network	Dynamic	Resource Utilization	Distributed micro-service	Fog	JAMScript	Assist the smooth transition of device connections and offloaded tasks between fog nodes
Lim (2022)	Supervised/Neural Network	On-demand	Response Time & Latency	Distributed micro-service	Fog	iFogSim	Propose a latency-aware task scheduling method for IoT applications based on artificial intelligence in small-scale fog computing environments

Table 9 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Abdelaziz et al. (2018)	Supervised/Hybrid Neural Network & Logistic Regression	Dynamic	Accuracy & Data Retrieval Efficiency	Distributed micro-service	Cloud	RL analyzer	The prediction model of Chronic Kidney Disease (CKD) is implemented using linear regression and neural network

Table 10 Deep supervised algorithms for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Piccialli et al. (2020)	Supervised/Deep Learning	Profiling	QoS	Distributed micro-service	IoT	Java	Present and discuss a deep learning methodology applied to data coming from a non-invasive Bluetooth IoT monitoring system deployed inside a cultural space
Huang et al. (2020)	Supervised/Deep Learning	Dynamic	Time & Accuracy	Distributed micro-service	MEC	Python	Dynamic computation tasks
Hou et al. (2023)	Supervised/Deep Learning	Dynamic	Time & User Quantity	Distributed micro-service	MEC	Python	Offload decision-making
Ran et al. (2019)	Supervised/Deep Q-Learning	Dynamic	Energy Consumption	Distributed micro-service	IoT	MATLAB	Dynamically schedule tasks with precedence relationship to cloud servers
Dong et al. (2020)	Supervised/Deep Q-Learning	Dynamic	Task Scheduling & Convergence Time	Distributed micro-service		MATLAB Cloud	Prediction model of Chronic Kidney Disease (CKD)
Sufian et al. (2020)	Deep Transfer Learning	Dynamic	Location	Distributed micro-service	IoT	N/A	Improving prediction performance in new cities with insufficient data for optimal store location by transferring prior knowledge learned from multiple data-rich cities

Table 11 Reinforcement learning algorithms for service placement

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Kochovski et al. (2019)	RL/MDP	Predictive	QoS Requirements	Monolithic	IoT	MATLAB	Database container placement
Baek et al. (2019)	RL/MDP	Predictive	Load Balancing Under Minimum Latency	Distributed micro-service	fog	MATLAB	Optimal offloading decision
Yan et al. (2020)	RL/Q-Learning	Predictive	Low-latency & High-Energy & Efficiency & Spectrum Effi-ciency	Distributed micro-service	F-RAN	Python	Content (cache) placement aiming to maximize the F-RAN network payoff
Farhat et al. (2020)	RL/R-Learning	Predictive	Load	Distributed micro-service	IoT	MATLAB	Decrease the cloud's load by utilizing the maximum available fog resources over different locations
Eyckerman et al. (2022)	RL/MORL	Predictive	Energy & Latency & Bandwidth	Distributed micro-service	fog	MATLAB	Reduces the resource consumption of the placement algorithm
Nsouli et al. (2022)	RL/Q-Learning	On-demand	QoE & Latency & Response Time	Distributed micro-service	IoT	Minikube	Real-time vehicular applications for self-driving cars to maintain service availability and reach-ability
Sami et al. (2020)	RL/SARSA	On-demand	response time, avoid service instability	Distributed micro-service	Fog	Fscaler	Schedules the place-ment of new instances based on pre-defined cost functions

Table 11 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Sulimani et al. (2022)	RL/Optimization	On-demand	Latency & Network Usage, Computing Usage	Distributed micro-service	Fog	Ifog sim	Decentralized service placement policy
Huang et al. (2022)	K-Armed Bandit	On-demand	Minimize the Service Cost While Minimizing the Service Latency	Distributed micro-service	IoT	MATLAB	Formulate the collaborative service caching problem
Wang et al. (2019)	RL/Q-Learning	Dynamic	Network Utility & Congestion Avoidance	Distributed micro-service	EC	Python	Optimize the utility, defined as achieved utility minus network congestion
Chen et al. (2022a)	RL/Meta	Dynamic	Latency & Energy Consumption & QoE-Aware Utility	Distributed micro-service	EC	Python	QoE-aware utility maximization
Ghobaei-Arani et al. (2018)	RL/Q-Learning	Dynamic	Total Cost & Resource Utilization	monolithic	Cloud	Cloud sim	Hybrid resource provisioning approach for cloud service
Arif et al. (2020)	RL/Meta	Dynamic	Scalability & Computational Cost & flexibility	Distributed micro-service	Fog	MATLAB	Places fog-enabled IoT applications at the most suitable fog node
Hao et al. (2022)	RL/MDP	Dynamic	Security & Latency & Storage	Distributed micro-service	Fog	MATLAB	Closed-form expressions for the storage size, transmission latency, and tampering time

Table 11 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Roberts et al. (2018)	RL/Bayesian	Dynamic	Energy Efficiency & Bandwidth	Distributed micro-service	IOT	MATLAB	Automatically partitioning stream processing across a set of components
Rawashdeh et al. (2021)	RL/MDP	Dynamic	Computational Time & Error Rate	Distributed micro-service	MEC	MATLAB	Model the migration process in an integrated architecture to resolve cross-layering issues
Sefati and Navimipour (2021)	RL/MDP	Dynamic	Availability & Response Time & Cost & Reliability & Energy Consumption	Distributed micro-service	IoT	MATLAB	Service composition issue by enhancing the QoS
Zhang et al. (2018)	RL/Q-Learning	Dynamic	Acceptance Ratio & Latency	Distributed micro-service	IoT	MATLAB	Guaranteed performance and convergence rate
Han et al. (2021)	RL/MORL	Dynamic	Power Consumption & Acceptance Ratio	Distributed micro-service	EC	MATLAB	Interference-aware online multi-component service placement
Hudson et al. (2021)	RL/K-Armed Bandit	Dynamic	Accuracy & the Average Delay	Distributed micro-service	MEC	Python	Maximize QoS for end-users
Tao et al. (2021)	RL/Multi-Armed Bandit	Dynamic	Latency & Cost	Distributed micro-service	MEC	Python	Adaptive user-managed service placement mechanism

Table 11 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Asghari et al. (2021)	RL/SARSA	Dynamic	Load Balancing & Resource Utiliza-tion	Distributed micro-service	IoT	Python	State-action-reward-state-action (SARSA) learning
Liu et al. (2022)	RL/MDP	Dynamic	Bandwidth Delay & Resource	Distributed micro-service	IoT	Pycharm	Multi-objective optimi-zation service func-tion chain placement
Santos et al. (2022c)	RL/CAND	Dynamic	Energy consumption, bandwidth, cost	Distributed micro-service	Cloud	OpenAI Gym	based on proximal policy optimization
Wang et al. (2020)	RL/Meta	Dynamic	Latency	Distributed micro-service	Multi-Access EC	Python	Propose a task offload-ing method based on meta-reinforcement learning, which can adapt fast to new environments with a small number of gradient updates and samples

Table 11 (continued)

References	ML technique (RQ1)	Resource estimation (RQ2)	Objective (RQ3)	App architecture (RQ4)	Paradigm (RQ5)	Simulator (RQ6)	Main objective (RQ7)
Wang et al. (2022b)	RL/SARSA	Dynamic	QoE & Fairness	Distributed micro-service	IoT	Python	Leverage MAPG-finite, a policy gradient algorithm designed for multi-agent learning problems with nonlinear objectives. It allows us to optimize bandwidth distributions among multiple agents and to maximize QoE and fairness objectives on video streaming rewards
Nakanoya et al. (2019)	RL/Mont Carlo	Dynamic	Bandwidth & Latency	Distributed micro-service	Fog	MATLAB	Propose an accelerated RL method that can learn proper VNF sizing and placement on a real network under various environments

Table 12 Deep reinforcement learning neural network algorithms for service placement

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Zhao et al. (2022)	RL/Deep Q-Learning	Predictive	Energy Consumption & Response Time & Latency & Bandwidth	Distributed micro-service	Fog	MATLAB	Service placement using multi-objective reinforcement learning
Fang et al. (2022)	RL/Deep Q-Learning	On-demand	Optimal Resource Allocation Problem as a Minimal Delay Model	Distributed micro-service	F-RAN	Python	Content distribution
Poltronieri et al. (2021)	RL/Deep Q-Learning	On-demand	Value-Based Utility	Distributed micro-service	Fog	MATLAB	Value-based management of fog services over a pool of fog nodes
Sami et al. (2021a)	Deep RL	Dynamic	QoS	Distributed micro-service	Cloud	CloudSim	Making placement decisions before demands occurrence
Lu et al. (2020)	RL/Deep Q-learning	On-demand	Network Load & Latency	Distributed micro-service	Fog	iFogSim	Clustering mobile tasks in large-scale heterogeneous MEC
Goudarzi et al. (2021)	Deep RL	On-demand	Cost	Distributed micro-service	Fog	iFogSim	Distributed application placement using the Actor-critic method

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Yu et al. (2020)	Deep RL & Federated learning	On-demand	Computation offloading, resource allocation, and service caching placement	Distributed microservice	UDEC	MATLAB	minimize the total offloading delay and network resource usage by jointly optimizing computation offloading, resource allocation, and service caching placement
Santos et al. (2021)	Deep RL	On-demand	Resource provisioning (CPU usage & latency)	Distributed microservice	Fog	Gym-fom	service function chaining allocation in fog computing focused on energy efficiency
Pham et al. (2021)	Deep RL	On-demand	Prediction accuracy, spectrum utilization, energy consumption	Distributed microservice	Fog	MATLAB	joint routing and placement problem for the IoT service chain to scale in/out the number of VNF instances
Sami et al. (2021b)	MDP/Deep RL	On-demand	Resource usage, available resources	Distributed microservice	MEC-supported 6G	MA	predicting the users' demand or the usage of resources by various applications
Jha et al. (2022)	Deep RL	On-demand	Energy, resource, cost& QoE, memory	Distributed microservice	Fog	Python	time-critical applications cannot rely on the availability of cloud servers since they are in a remote location

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Arri et al. (2021)	Deep RL	Dynamic	QOS	Distributed micro-service	Fog	Python	task group aggregation overflow handling system for fog computing environments
Sami et al. (2021a)	Deep RL	Dynamic	Resource availability	Distributed micro-service	Fog	Python	perform instantaneous placement decisions proactively
Zhou et al. (2020)	Deep Risk-Sensitive RL	mixed	Scheduling & power consumption	Distributed micro-service	IoT	VIS Sim	developing a novel deep risk-sensitive reinforcement learning algorithm
Zeng et al. (2019)	Deep RL	Dynamic	Mobility & operational cost of run time	Distributed micro-service	EC	Python	efficiently manage the resources at the network edge
Liu et al. (2020)	Deep RL	Dynamic	Transmission time & processing delay	Distributed micro-service	MEC	NetworkX	SFC dynamic orchestration framework for IoT deep reinforcement learning
Chen et al. (2022b)	Deep RL	Dynamic	Resource& latency requirement	Distributed micro-service	MEC	Pytorch	edge-enabled IoT system and investigates the joint caching and computing service placement
Qi et al. (2020)	RL/Deep RL & Neural Network	Dynamic	Latency, load, total cost	Distributed micro-service	IoV	Python	multi-task deep reinforcement learning approach

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Chen et al. (2020)	Meta & Deep RL	Dynamic	Energy consumption	Distributed micro-service	MEC	Python	improving the migration ability of the whole model, and a deep reinforcement learning (DRL) model
Chen et al. (2017)	Deep RL	Dynamic	Average job scheduling	Distributed micro-service	IoT	Python	improving a recently proposed job scheduling algorithm using deep reinforcement learning and extending it to multiple server clusters
Li et al. (2019b)	Deep RL	Dynamic	Scheduling	Distributed micro-service	IoT	Python	introduce a framework enabling mobile crowdsensing in fog environments with a hierarchical scheduling strategy
Rjoub et al. (2021)	Deep RL	dynamic	Resource computation, task waiting time	Distributed micro-service	cloud	MATLAB	automate the process of scheduling large-scale workloads onto cloud computing resources

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Hao et al. (2020)	RL/Deep Q-Learning	Dynamic	Average service response time	Distributed micro-service	IoT	Python	joint optimization of service placement, workload scheduling, and resource allocation
Bi et al. (2021)	Deep MORL	Dynamic	Acceptance ratio & latency	Distributed micro-service	IoT	Gurobi solver	Chebyshev actor-critic SFCS placement algorithm, to overcome the limitations of traditional heuristic and evolutionary algorithms
Luo et al. (2022)	RL/Deep Q-Network	Dynamic	Access delay & workload balance	Distributed micro-service	MEC	MATLAB	achieve optimal placements without relying on previous placement experience
Talpur and Gurusamy (2021)	Deep RL	Dynamic	Average service delay, fairness, service instance utilization, edge resource usage	Distributed micro-service	IoV	SUMO and MATLAB	minimizing the maximum edge resource usage and service delay while considering the vehicle's mobility
Li et al. (2021)	Deep RL	Dynamic	Network size & arriving rate	Distributed micro-service	MEC	MATLAB	Maximize the profit of the service provider by admitting service requests for the monitoring period

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Abdellah et al. (2023)	Deep RL	Dynamic	Latency	Distributed micro-service	IoT	RASim	Enabling mobile crowdsensing in fog environments with a hierarchical scheduling strategy
Zare et al. (2023)	Deep RL	Dynamic	Cost, latency under deadline, resource	Distributed micro-service	Fog	OpenAI Gym	Minimizing cost and latency under deadline and resource constraints
Talpur and Gurusamy (2023)	Deep RL	Dynamic	Latency under resource constraint	Distributed micro-service	IoV	MATLAB	Improving the user experience while minimizing the delay and simultaneously considering efficient utilization of limited edge resources
Chen et al. (2020)	RL/Neural Network	Dynamic	Response time	Distributed micro-service	IoT	Python	Reduce the average waiting time of IoT devices in the hybrid environment

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Bansal et al. (2022)	RL/multi-action Deep Reinforcement Learning	On-demand	Compute time, communication time	Distributed micro-service	MEC-Cloud Collaboration	Not mention	UrbanEnQoSMDP – formulation for energy and QoS (latency) optimized service placement for a set of applications in the 'Urban IoT-Federated MEC-Cloud' architecture to satisfy applications' compute, per-flow communicate
Bensalem et al. (2020)	RL/deep Neural Network	Dynamic	Latency & load	Distributed micro-service	EC	Not mention	propose the first model of optimal placement of Deep Neural Network (DNN) Placement and Inference in edge computing
Amidzadeh et al. (2021)	Deep RL	Dynamic	Storage & latency	Distributed micro-service	IoT	Not mention	design the CPD policy while taking into account the user Quality of Service and the backhaul load, and using an Actor-Critic RL framework with two neural networks

Table 12 (continued)

References	ML technique RQ1	Resource estimation RQ2	Objective RQ3	App architecture RQ4	Paradigm RQ5	Simulator RQ6	Main aim RQ7
Premsankar and Ghaddar (2022)	RL/Deep Neural Network	Dynamic	Energy Efficiency by keeping latency blew a threshold	Distributed micro-service	EC	Python	model the problem of energy-efficient placement of services (namely, DNN models) for AI applications as a multiperiod optimization problem
Wang et al. (2021)	RL/Double Deep Q Network	Dynamic	Acceptance ration & service availability & throughput	Distributed micro-service	Fog	Python	propose a double deep Q-networks-based online SFC placement scheme DDQP. Specifically, DDQP uses deep neural networks to deal with large continuous network state space
Mezni et al. (2023)	RL/Deep Learning	Dynamic	Quality of Service & Performance	Distributed micro-service	Fog-Cloud	Java	propose a Frequent Subgraph Mining algorithm that is reinforced with a tuning method to increase the probability of executing the past placement schemes

Author contributions All authors contributed to the concept and design of the study. PKH collected and analyzed the data. The survey of the project and the validation of the results were done by MHR. MM and AS were remote scientific advisors. PKH wrote the first draft of the manuscript, and all authors commented on earlier versions. All authors read and approved the final manuscript. This article reports part of the scientific findings of an academic Ph.D. thesis. The thesis was presented by PKH as the student and MHR as the thesis supervisor.

Funding The authors did not receive support from any organization for the submitted work.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states no conflict of interest.

Informed consent Informed consent was obtained from all participants included in the study.

References

Abdelaziz A, Elhoseny M, Salama AS, Riad AM (2018) A machine learning model for improving healthcare services on cloud computing environment. Measurement 119:117–128

Abdellah KACI, Ait-Chellouche S, Hadjadj-Aoul Y, Bagaa M (2023) RAP-G: reliability-aware service placement using genetic algorithm for deep edge computing. 2023 IEEE 20th consumer communications & networking conference (CCNC). IEEE, pp 255–260

Adege AB, Lin HP, Tarekegn GB, Munaye YY, Yen L (2018) An indoor and outdoor positioning using a hybrid of support vector machine and deep neural network algorithms. J Sens 2018:1–12

Alenazi MM, Yosuf BA, Mohamed SH, El-Gorashi TE, Elmirghani JM (2022) Energy Efficient placement of ML-based services in IoT networks. 2022 IEEE international mediterranean conference on communications and networking (MeditCom). IEEE, pp 19–24

Alli AA, Alam MM (2019) SecOFF-FCIoT: machine learning based secure offloading in fog-cloud of things for smart city applications. Internet of Things 7:100070

Alsaffar AA, Pham HP, Hong CS, Huh EN, Aazam M (2016) An architecture of IoT service delegation and resource allocation based on collaboration between fog and cloud computing. Mob Inf Syst 2016:1–15

Amidzadeh M, Al-Tous H, Tirkkonen O, Zhang J (2021) Joint cache placement and delivery design using reinforcement learning for cellular networks. 2021 IEEE 93rd vehicular technology conference (VTC2021-Spring). IEEE, pp 1–6

Anwar A, Raychowdhury A (2020) Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning. IEEE Access 8:26549–26560

Arif M, Azam F, Anwar MW, Rasheed Y (2020) A model-driven framework for optimum application placement in fog computing using a machine learning based approach. In: Information and Software Technologies: 26th International Conference, ICIST 2020, Kaunas, Lithuania, 15–17 October 2020, Proceedings, vol. 26. Springer, pp. 102–112

Arri HS, Singh R, Jha S, Prashar D, Joshi GP, Doo IC (2021) Optimized task group aggregation-based overflow handling on fog computing environment using neural computing. Mathematics 9(19):2522

Asghari A, Sohrabi MK, Yaghmaee F (2021) Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm. J Supercomput 77:2800–2828

Aznavouridis A, Tsakos K, Petrakis EG (2022) Micro-service placement policies for cost optimization in Kubernetes. International conference on advanced information networking and applications. Springer, Cham, pp 409–420

Baek JY, Kaddoum G, Garg S, Kaur K, Gravel V (2019) Managing fog networks using reinforcement learning based load balancing algorithm. 2019 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, pp 1–7

Balasubramanian V, Aloqaily M, Reisslein M (2021) FedCo: A federated learning controller for content management in multi-party edge systems. 2021 International Conference on Computer Communications and Networks (ICCCN). IEEE, pp 1–9

Bansal M, Chana I, Clarke S (2022) UrbanEnQoSPlace: a deep reinforcement learning model for service placement of real-time smart city IoT applications. IEEE Trans Serv Comput 16:3043–3060

Bashir H, Lee S, Kim KH (2022) Resource allocation through logistic regression and multicriteria decision making method in IoT fog computing. Trans Emerg Telecommun Technol 33(2):e3824

Bensalem M, Dizdarevć J, Jukan A (2020) Modeling of deep neural network (DNN) placement and inference in edge computing. 2020 IEEE international conference on communications workshops (ICC workshops). IEEE, pp 1–6

Besharati R, Rezvani MH, Gilanian Sadeghi MM (2023) An auction-based bid prediction mechanism for fog-cloud offloading using Q-learning. Complexity 2023:1–20

Bi Y, Meixner CC, Bunyakitanon M, Vasilakos X, Nejabati R, Simeonidou D (2021) Multi-objective deep reinforcement learning assisted service function chains placement. IEEE Trans Netw Serv Manage 18(4):4134–4150

Borelli H, Costa FM, Carvalho ST (2022) Use of multilevel resource clustering for service placement in fog computing environments. 2022 IEEE/ACM 15th international conference on Utility and Cloud Computing (UCC). IEEE, pp 360–365

Brecko A, Kajati E, Koziorek J, Zolotova I (2022) Federated learning for edge computing: a survey. Appl Sci 12(18):9124

Bukhari MM, Ghazal TM, Abbas S, Khan MA, Farooq U, Wahbah H, Ahmad M, Adnan KM (2022) An intelligent proposed model for task offloading in fog-cloud collaboration using logistics regression. Comput Intell Neurosci 2022:1–25

Canali C, Lancellotti R (2019) Gasp: genetic algorithms for service placement in fog computing systems. Algorithms 12(10):201

Chen Q, Zheng Z, Hu C, Wang D, Liu F (2019) On-edge multi-task transfer learning: model and practice with data-driven task allocation. IEEE Trans Parallel Distrib Syst 31(6):1357–1371

Chen L, Xu Y, Lu Z, Wu J, Gai K, Hung PC, Qiu M (2020) IoT microservice deployment in edge-cloud hybrid environment using reinforcement learning. IEEE Internet Things J 8(16):12610–12622

Chen S, Rui L, Gao Z, Li W, Qiu X (2022a) Cache-assisted collaborative task offloading and resource allocation strategy: a metareinforcement learning approach. IEEE Internet Things J 9(20):19823–19842

Chen Y, Sun Y, Yang B, Taleb T (2022b) Joint caching and computing service placement for edge-enabled iot based on deep reinforcement learning. IEEE Internet Things J 9(19):19501–19514

Chen W, Xu Y, Wu X (2017) Deep reinforcement learning for multi-resource multi-machine job scheduling. arXiv preprint arXiv:1711.07440

Danish SM, Zhang K, Jacobsen HA (2021) BlockAIM: a neural network-based intelligent middleware for large-scale IoT data placement decisions. IEEE Trans Mob Comput 22(1):84–99

de Oliveira GW, Nogueira M, dos Santos AL, Batista DM (2023) Intelligent VNF placement to mitigate DDoS attacks on industrial IoT. IEEE Trans Netw Service Manag 20:1319–1331

Dimililer K, Dindar H, Al-Turjman F (2021) Deep learning, machine learning and internet of things in geophysical engineering applications: an overview. Microprocess Microsyst 80:103613

Dong T, Xue F, Xiao C, Li J (2020) Task scheduling based on deep reinforcement learning in a cloud manufacturing environment. Concurrency Comput: Practice Exp 32(11):e5654

Donyagard Vahed N, Ghobaei-Arani M, Souri A (2019) Multiobjective virtual machine placement mechanisms using nature-inspired metaheuristic algorithms in cloud environments: a comprehensive review. Int J Commun Syst 32(14):e4068

Duc TL, Leiva RG, Casari P, Östberg PO (2019) Machine learning methods for reliable resource provisioning in edge-cloud computing: a survey. ACM Comput Surv (CSUR) 52(5):1–39

Dworzak M, Großmann M, Le DT (2023) Federated autonomous orchestration in fog computing systems. In International congress on information and communication technology. Springer Nature Singapore. Vancouver, Singapore, pp 639–649

Etemadi M, Ghobaei-Arani M, Shahidinejad A (2020) Resource provisioning for IoT services in the fog computing environment: an autonomic approach. Comput Commun 161:109–131

Eyckerman R, Reiter P, Latré S, Marquez-Barja J, Hellinckx P (2022) Application placement in fog environments using multi-objective reinforcement learning with maximum reward formulation. NOMS 2022–2022 IEEE/IFIP network operations and management symposium. IEEE, pp 1–6

Fang C, Xu H, Yang Y, Hu Z, Tu S, Ota K, Yang Z, Dong M, Han Z, Yu FR, Liu Y (2022) Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks. IEEE Internet Things J 9(18):16874–16883

Farhat P, Sami H, Mourad A (2020) Reinforcement R-learning model for time scheduling of on-demand fog placement. J Supercomput 76:388–410

Farhat P, Arisdakessian S, Wahab OA, Mourad A, Ould-Slimane H (2022) Machine learning based container placement in on-demand clustered fogs. 2022 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp 1250–1255

Gallego-Madrid J, Sanchez-Iborra R, Ruiz PM, Skarmeta AF (2022) Machine learning-based zero-touch network and service management: a survey. Digit Commun Netw 8(2):105–123

Gasmi K, Dilek S, Tosun S, Ozdemir S (2022) A survey on computation offloading and service placement in fog computing-based IoT. J Supercomput 78(2):1983–2014

Ghobaei-Arani M, Jabbehdari S, Pourmina MA (2018) An autonomic resource provisioning approach for service-based cloud applications: a hybrid approach. Futur Gener Comput Syst 78:191–210

Ghobaei-Arani M, Souri A, Rahmanian AA (2020) Resource management approaches in fog computing: a comprehensive review. J Grid Comput 18(1):1–42

Girelli Consolaro N, Shinde SS, Naseh D, Tarchi D (2023) Analysis and performance evaluation of transfer learning algorithms for 6G wireless networks. Electronics 12(15):3327

Goudarzi M, Palaniswami MS, Buyya R (2021) A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. IEEE Trans Mob Comput 20:1298–1311

Haibeh LA, Yagoub MC, Jarray A (2022) A survey on mobile edge computing infrastructure: design, resource management, and optimization approaches. IEEE Access 10:27591–27610

Hallappanavar VL, Bulla CM, Birje MN (2021) ANN based estimation of reputation of newcomer web services in fog computing. 2021 International Conference on Computer Communication and Informatics (ICCCI). IEEE, pp 1–7

Han P, Liu Y, Guo L (2021) Interference-aware online multicomponent service placement in edge cloud networks and its ai application. IEEE Internet Things J 8(13):10557–10572

Hao Y, Chen M, Gharavi H, Zhang Y, Hwang K (2020) Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system. IEEE Trans Ind Inf 17(8):5552–5561

Hao X, Yeoh PL, Ji Z, Yu Y, Vucetic B, Li Y (2022) Stochastic analysis of double blockchain architecture in IoT communication networks. IEEE Internet Things J 9(12):9700–9711

Hou T, Feng G, Qin S, Jiang W (2018) Proactive content caching by exploiting transfer learning for mobile edge computing. Int J Commun Syst 31(11):e3706

Hou J, Chen M, Geng H, Li R, Lu J (2023) GP-NFSP: Decentralized task offloading for mobile edge computing with independent reinforcement learning. Futur Gener Comput Syst 141:205–217

Hsu TH, Wang ZH, See AR (2022) A cloud-edge-smart IoT architecture for speeding up the deployment of neural network models with transfer learning techniques. Electronics 11(14):2255

Hu Y, Huang T, Yu Y, An Y, Cheng M, Zhou W, Xian W (2023) An energy-aware service placement strategy using hybrid meta-heuristic algorithm in iot environments. Clust Comput 26(5):2913–2919

Huang L, Zhang L, Yang S, Qian LP, Wu Y (2020) Meta-learning based dynamic computation task offloading for mobile edge computing networks. IEEE Commun Lett 25(5):1568–1572

Huang B, Liu X, Xiang Y, Yu D, Deng S, Wang S (2022) Reinforcement learning for cost-effective IoT service caching at the edge. J Parallel Distrib Comput 168:120–136

Hudson N, Khamfroush H, Lucani DE (2021) QoS-aware placement of deep learning services on the edge with multiple service implementations. 2021 international conference on computer communications and networks (ICCCN). IEEE, pp 1–8

Jha AK, Patel MP, Pawar TD (2022) Computation offloading using K-nearest neighbour time critical optimisation algorithm in fog computing. Int J Wireless Mob Comput 23(3–4):281–292

John VPM (2023) A study on cloud container technology. i-Manager's J Cloud Comput 10(1):7

Kar B, Yahya W, Lin YD, Ali A (2023) Offloading using traditional optimization and machine learning in federated cloud-edge-fog systems: a survey. IEEE Commun Surv Tutor 25:1199–1226

Kim EJ, Kwon S, Kang H, Jun JA, Kim NS (2016) A study on knowledge-based context aware framework using machine learning. Adv Sci Technol Lett 139:90–94

Kim BY, Choi SS, Jang JW (2018) Data managing and service exchanging on IoT service platform based on blockchain with smart contract and spatial data processing. In Proceedings of the 1st international conference on information science and systems, pp. 59–63

Kochovski P, Sakellariou R, Bajec M, Drobintsev P, Stankovski V (2019) An architecture and stochastic method for database container placement in the edge-fog-cloud continuum. 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, pp 396–405

Kumar D, Baranwal G, Shankar Y, Vidyarthi DP (2022) A survey on nature-inspired techniques for computation offloading and service placement in emerging edge technologies. World Wide Web 25(5):2049–2107

Li Y, Zhang Y, Liu Y, Meng Q, Tian F (2019a) Fog node selection for low latency communication and anomaly detection in fog networks. 2019 International Conference on Communications, Information System and Computer Engineering (CISCE). IEEE, pp 276–279

Li H, Ota K, Dong M (2019b) Deep reinforcement scheduling for mobile crowdsensing in fog computing. ACM Trans Internet Technol 19(2):1–18

Li Y, Liang W, Li J (2021) Profit maximization for service placement and request assignment in edge computing via deep reinforcement learning. In Proceedings of the 24th international acm conference on modeling, analysis and simulation of wireless and mobile systems, pp. 51–55

Lim J (2022) Latency-aware task scheduling for IoT applications based on artificial intelligence with partitioning in small-scale fog computing environments. Sensors 22(19):7326

Liu Y, Lu H, Li X, Zhang Y, Xi L, Zhao D (2020) Dynamic service function chain orchestration for NFV/MEC-enabled IoT networks: a deep reinforcement learning approach. IEEE Internet Things J 8(9):7450–7465

Liu Y, Guo B, Zhang D, Zeghlache D, Chen J, Zhang S, Zhou D, Shi X, Yu Z (2021) MetaStore: a task-adaptative meta-learning model for optimal store placement with multi-city knowledge transfer. ACM Trans Intell Syst Technol 12(3):1–23

Liu H, Ding S, Wang S, Zhao G, Wang C (2022) Multi-objective optimization service function chain placement algorithm based on reinforcement learning. J Netw Syst Manage 30(4):58

Lu H, Gu C, Luo F, Ding W, Liu X (2020) Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Futur Gener Comput Syst 102:847–861

Lu J, Zhao W, Zhu H, Li J, Cheng Z, Xiao G (2022) Optimal machine placement based on improved genetic algorithm in cloud computing. J Supercomput:1–29

Luo F, Zheng S, Ding W, Fuentes J, Li Y (2022) An edge server placement method based on reinforcement learning. Entropy 24(3):317

Mahmud R, Ramamohanarao K, Buyya R (2020) Application management in fog computing environments: a taxonomy, review and future directions. ACM Comput Surv (CSUR) 53(4):1–43

Maia AM, Ghamri-Doudane Y, Vieira D, de Castro MF (2021) An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing. Comput Netw 194:108146

Malazi HT, Chaudhry SR, Kazmi A, Palade A, Cabrera C, White G, Clarke S (2022) Dynamic service placement in multi-access edge computing: a systematic literature review. IEEE Access 10:32639–32688

Manikandan R, Patan R, Gandomi AH, Sivanesan P, Kalyanaraman H (2020) Hash polynomial two factor decision tree using IoT for smart health care scheduling. Expert Syst Appl 141:112924

Mei Y, Guo P, Zhou M, Patel V (2022) Resource-adaptive federated learning with all-in-one neural composition. Adv Neural Inf Process Syst 35:4270–4284

Memon SA (2019) Leveraging machine learning for efficient mobility management and data transmission in fog computing. McGill University, Canada

Memon S, Maheswaran M (2019) Using machine learning for handover optimization in vehicular fog computing. In Proceedings of the 34th ACM/SIGAPP symposium on applied computing, pp. 182–190

Mezni H, Hamoud FS, Charrada FB (2023) Predictive service placement in cloud using deep learning and frequent subgraph mining. J Ambient Intell Humaniz Comput 14(9):11497–11516

Mohammadi M, Al-Fuqaha A, Guizani M, Oh JS (2017) Semisupervised deep reinforcement learning in support of IoT and smart city services. IEEE Internet Things J 5(2):624–635

Mohammed LB, Anpalagan A, Khwaja AS, Jaseemuddin M (2022) Semi-supervised learning with self-training classifier for cache placement in mobile edge networks. 30th Biennial symposium on communications 2021. Springer, Cham, pp 197–210

Mohan N, Kangasharju J (2016) Edge-fog cloud: a distributed cloud for internet of things computations. 2016 cloudification of the Internet of Things (CIoT). IEEE, pp 1–6

Naghdehforoushha M, Fooladi MDT, Rezvani MH, Sadeghi MMG (2022) BLMDP: a new bi-level Markov decision process approach to joint bidding andtask-scheduling in cloud spot market. Turk J Electr Eng Comput Sci 30(4):1419–1438

Nakanoya M, Sato Y, Shimonishi H (2019) Environment-adaptive sizing and placement of NFV service chains with accelerated reinforcement learning. 2019 IFIP/IEEE symposium on integrated network and service management (IM). IEEE, pp 36–44

Natesha BV, Guddeti RMR (2022) Meta-heuristic based hybrid service placement strategies for two-level fog computing architecture. J Netw Syst Manage 30(3):47

Nayeri ZM, Ghafarian T, Javadi B (2021) Application placement in fog computing with AI approach: taxonomy and a state of the art survey. J Netw Comput Appl 185:103078

Nisha T (2018) ECO: harmonizing edge and cloud with ml/dl orchestration. In: USENIX workshop on hot topics in edge computing (HotEdge 18). Boston

Noulas, A., Scellato, S., Lathia, N. and Mascolo, C., 2012, December. Mining user mobility features for next place prediction in location-based services. In *2012 IEEE 12th international conference on data mining* (pp. 1038–1043). IEEE.

Nouri N, Abouei J, Sepasian AR, Jaseemuddin M, Anpalagan A, Plataniotis KN (2021) Three-dimensional multi-UAV placement and resource allocation for energy-efficient IoT communication. IEEE Internet Things J 9(3):2134–2152

Nsouli A, Mourad A, El-Hajj W (2022) Reinforcement learning based scheme for on-demand vehicular fog formation and micro services placement. 2022 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp 1244–1249

Panadero J, Selimi M, Calvet L, Marquès JM, Freitag F (2021) A two-stage multi-criteria optimization method for service placement in decentralized edge micro-clouds. Futur Gener Comput Syst 121:90–105

Peng Y, Xue X, Bashir AK, Zhu X, Al-Otaibi YD, Tariq U, Yu K (2022) Securing radio resources allocation with deep reinforcement learning for IoE services in next-generation wireless networks. IEEE Trans Netw Sci Eng 9(5):2991–3003

Pham C, Nguyen DT, Tran NH, Nguyen KK, Cheriet M (2021) Optimized IoT service chain implementation in edge cloud platform: a deep learning framework. IEEE Trans Netw Serv Manage 18(1):538–551

Piccialli F, Giampaolo F, Casolla G, Di Cola VS, Li K (2020) A deep learning approach for path prediction in a location-based IoT system. Pervasive Mob Comput 66:101210

Poltronieri F, Tortonesi M, Stefanelli C, Suri N (2021) Reinforcement learning for value-based placement of fog services. 2021 IFIP/IEEE international symposium on integrated network management (IM). IEEE, pp 466–472

Premsankar G, Ghaddar B (2022) Energy-efficient service placement for latency-sensitive applications in edge computing. IEEE Internet Things J 9(18):17926–17937

Qi Q, Zhang L, Wang J, Sun H, Zhuang Z, Liao J, Yu FR (2020) Scalable parallel task scheduling for autonomous driving using multi-task deep reinforcement learning. IEEE Trans Veh Technol 69(11):13861–13874

Qian Y, Hu L, Chen J, Guan X, Hassan MM, Alelaiwi A (2019) Privacy-aware service placement for mobile edge computing via federated learning. Inf Sci 505:562–570

Raghavendra MS, Chawla P, Narasimhulu Y (2021) A probability based joint-clustering algorithm for application placement in fog-to-cloud computing. 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (trends and future directions)(ICRITO). IEEE, pp 1–5

Ramezani Shahidani F, Ghasemi A, Toroghi Haghighat A, Keshavarzi A (2023) Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. Computing 105(6):1337–1359

Ran Y, Hu H, Zhou X, Wen Y (2019) Deepee: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning. 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, pp 645–655

Rawashdeh M, Al Zamil MG, Samarah SM, Obaidat M, Masud M (2021) IOT-based service migration for connected communities. Comput Electr Eng 96:107530

Rjoub G, Bentahar J, Abdel Wahab O, Saleh Bataineh A (2021) Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems. Concurrency Comput: Practice Exp 33(23):e5919

Roberts L, Michalák P, Heaps S, Trenell M, Wilkinson D, Watson P (2018) Automating the placement of time series models for iot healthcare applications. 2018 IEEE 14th international conference on e-science. IEEE, pp 290–291

Rodrigues TK, Suto K, Nishiyama H, Liu J, Kato N (2019) Machine learning meets computation and communication control in evolving edge and cloud: challenges and future perspective. IEEE Commun Surv Tutor 22(1):38–67

Salaht FA, Desprez F, Lebre A (2020) An overview of service placement problem in fog and edge computing. ACM Comput Surv (CSUR) 53(3):1–35

Sami H, Mourad A, Otrok H, Bentahar J (2020) Fscaler: automatic resource scaling of containers in fog clusters using reinforcement learning. 2020 International Wireless Communications and Mobile Computing (IWCMC). IEEE, pp 1824–1829

Sami H, Mourad A, Otrok H, Bentahar J (2021a) Demand-driven deep reinforcement learning for scalable fog and service placement. IEEE Trans Serv Comput 15(5):2671–2684

Sami H, Otrok H, Bentahar J, Mourad A (2021b) AI-based resource provisioning of IoE services in 6G: a deep reinforcement learning approach. IEEE Trans Netw Serv Manage 18(3):3527–3540

Sangaiah AK, Medhane DV, Han T, Hossain MS, Muhammad G (2019) Enforcing position-based confidentiality with machine learning paradigm through mobile edge computing in real-time industrial informatics. IEEE Trans Ind Inf 15(7):4189–4196

Santos J, Wauters T, Volckaert B, De Turck F (2021) Resource provisioning in fog computing through deep reinforcement learning. 2021 IFIP/ieee international symposium on integrated network management (IM). IEEE, pp 431–437

Santos GL, Bezerra DDF, Rocha ÉDS, Ferreira L, Moreira ALC, Gonçalves GE, Marquezini MV, Recse Á, Mehta A, Kelner J, Sadok D (2022a) Service function chain placement in distributed scenarios: a systematic review. J Netw Syst Manage 30(1):4

Santos F, Immich R, Madeira ER (2022b) Multimedia services placement algorithm for cloud–fog hierarchical environments. Comput Commun 191:78–91

Santos GL, Endo PT, Lynn T, Sadok D, Kelner J (2022c) A reinforcement learning-based approach for availability-aware service function chain placement in large-scale networks. Futur Gener Comput Syst 136:93–109

Sarrafzade N, Entezari-Maleki R, Sousa L (2022) A genetic-based approach for service placement in fog computing. J Supercomput 78(8):10854–10875

Sefati S, Navimipour NJ (2021) A qos-aware service composition mechanism in the internet of things using a hidden-markov-model-based optimization algorithm. IEEE Internet Things J 8(20):15620–15627

Shahraki A, Ohlenforst T, Kreyß F (2023) When machine learning meets network management and orchestration in edge-based networking paradigms. J Netw Comput Appl 212:103558

Shakarami A, Shahidinejad A, Ghobaei-Arani M (2020) A review on the computation offloading approaches in mobile edge computing: a game-theoretic perspective. Software: Practice Exp 50(9):1719–1759

Shakarami A, Shahidinejad A, Ghobaei-Arani M (2021) An autonomous computation offloading strategy in mobile edge computing: a deep learning-based hybrid approach. J Netw Comput Appl 178:102974

Shao ZL, Huang C, Li H (2021) Replica selection and placement techniques on the IoT and edge computing: a deep study. Wireless Netw 27(7):5039–5055

Shen J, Zhao Y, Liu JK, Wang Y (2021) HybridSNN: combining bio-machine strengths by boosting adaptive spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems

Shen J, Xu Q, Liu JK, Wang Y, Pan G, Tang H (2023) ESL-SNNs: an evolutionary structure learning strategy for spiking neural networks. arXiv preprint arXiv:2306.03693

Shuja J, Bilal K, Alasmary W, Sinky H, Alanazi E (2021) Applying machine learning techniques for caching in next-generation edge networks: a comprehensive survey. J Netw Comput Appl 181:103005

Son AY, Huh EN (2019) Multi-objective service placement scheme based on fuzzy-AHP system for distributed cloud computing. Appl Sci 9(17):3550

Sriraghavendra M, Chawla P, Wu H, Gill SS, Buyya R (2022) DoSP: A deadline-aware dynamic service placement algorithm for workflow-oriented IoT applications in fog-cloud computing environments. Energy conservation solutions for fog-edge computing paradigms. Springer, Singapore, pp 21–47

Su L, Wang N, Zhou R, Li Z (2022) Dynamic service placement and request scheduling for edge networks. Comput Netw 213:108997

Sufian A, Ghosh A, Sadiq AS, Smarandache F (2020) A survey on deep transfer learning to edge computing for mitigating the COVID-19 pandemic. J Syst Architect 108:101830

Sulimani H, Sajjad AM, Alghamdi WY, Kaiwartya O, Jan T, Simoff S, Prasad M (2022) Reinforcement optimization for decentralized service placement policy in IoT-centric fog environment. Trans Emerg Telecommun Technol 34:e4650

Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT Press

Taheri-abed S, Eftekhari Moghadam AM, Rezvani MH (2023) Machine learning-based computation offloading in edge and fog: a systematic review. Cluster Comput 26:1–32

Taka H, He F, Oki E (2022) Service placement and user assignment in multi-access edge computing with base-station failure. 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS). IEEE, pp 1–10

Talpur A, Gurusamy M (2021) DRLD-SP: a deep-reinforcement-learning-based dynamic service placement in edge-enabled internet of vehicles. IEEE Internet Things J 9(8):6239–6251

Talpur A, Gurusamy M (2023) On attack-resilient service placement and availability in edge-enabled iov networks. IEEE Trans Intell Transp Syst 24:6244–6256

Tan PN, Steinbach M, Kumar V (2016) Introduction to data mining. Pearson Education India

Tao O, Chen X, Zhou Z, Li L, Tan X (2021) Adaptive user-managed service placement for mobile edge computing via contextual multi-armed bandit learning. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2021.3106746

Tavakoli-Someh S, Rezvani MH (2019) Multi-objective virtual network function placement using NSGA-II meta-heuristic approach. J Supercomput 75(10):6451–6487

Teoh YK, Gill SS, Parlikad AK (2021) IoT and fog computing based predictive maintenance model for effective asset management in industry 4.0 using machine learning. IEEE Internet Things J 10(3): 2087–2094

Torabi E, Ghobaei-Arani M, Shahidinejad A (2022) Data replica placement approaches in fog computing: a review. Clust Comput 25(5):3561–3589

Tuli S, Poojara SR, Srirama SN, Casale G, Jennings NR (2021) COSCO: container orchestration using co-simulation and gradient based optimization for fog computing environments. IEEE Trans Parallel Distrib Syst 33(1):101–116

Wahab OA, Kara N, Edstrom C, Lemieux Y (2019) MAPLE: a machine learning approach for efficient placement and adjustment of virtual network functions. J Netw Comput Appl 142:37–50

Wang Y, Li Y, Lan T, Choi N (2019) A reinforcement learning approach for online service tree placement in edge computing. 2019 IEEE 27th International Conference on Network Protocols (ICNP). IEEE, pp 1–6

Wang J, Hu J, Min G, Zomaya AY, Georgalas N (2020) Fast adaptive task offloading in edge computing based on meta reinforcement learning. IEEE Trans Parallel Distrib Syst 32(1):242–253

Wang L, Mao W, Zhao J, Xu Y (2021) DDQP: a double deep Q-learning approach to online fault-tolerant SFC placement. IEEE Trans Netw Serv Manage 18(1):118–132

Wang Y, Wang J, Zhang W, Zhan Y, Guo S, Zheng Q, Wang X (2022a) A survey on deploying mobile deep learning applications: a systemic and technical perspective. Digit Commun Netw 8(1):1–17

Wang Y, Agarwal M, Lan T, Aggarwal V (2022b) Learning-based online QoE optimization in multi-agent video streaming. Algorithms 15(7):227

Xiao T, Cui T, Islam SR, Chen Q (2020) Joint content placement and storage allocation based on federated learning in F-RANs. Sensors 21(1):215

Xiao D, Chen S, Ni W, Zhang J, Zhang A, Liu R (2022) A sub-action aided deep reinforcement learning framework for latency-sensitive network slicing. Comput Netw 217:109279

Xu Z, Li D, Liang W, Xu W, Xia Q, Zhou P, Rana OF, Li H (2023) Energy or accuracy? Near-optimal user selection and aggregator placement for federated learning in MEC. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2023.3262829

Yan S, Jiao M, Zhou Y, Peng M, Daneshmand M (2020) Machine-learning approach for user association and content placement in fog radio access networks. IEEE Internet Things J 7(10):9413–9425

Yu S, Chen X, Zhou Z, Gong X, Wu D (2020) When deep reinforcement learning meets federated learning: intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network. IEEE Internet Things J 8(4):2238–2251

Yuan X, Sun M, Lou W (2020) A dynamic deep-learning-based virtual edge node placement scheme for edge cloud systems in mobile environment. IEEE Trans Cloud Comput 10(2):1317–1328

Zabihi Z, Moghadam AME, Rezvani MH (2023) Reinforcement learning methods for computing offloading: a systematic review. ACM Comput Surv 56:1–41

Zare M, Sola YE, Hasanpour H (2023) Towards distributed and autonomous IoT service placement in fog computing using asynchronous advantage actor-critic algorithm. J King Saud Univ—Comput Inf Sci 35(1):368–381

Zeng D, Gu L, Pan S, Cai J, Guo S (2019) Resource management at the network edge: a deep reinforcement learning approach. IEEE Network 33(3):26–33

Zhang Z, Ma L, Leung KK, Tassiulas L, Tucker J (2018) Q-placement: Reinforcement-learning-based service placement in software-defined networks. 2018 IEEE 38th international conference on distributed computing systems (ICDCS). IEEE, pp 1527–1532

Zhang Z, Wang N, Wu H, Tang C, Li R (2021) MR-DRO: a fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments. IEEE Internet Things J 10(4):3165–3178

Zhao D, Zou Q, Boshkani Zadeh M (2022) A QoS-aware IoT service placement mechanism in fog computing based on open-source development model. J Grid Comput 20(2):12

Zhou Z, Chen X, Li E, Zeng L, Luo K, Zhang J (2019) Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762

Zhou C, Wu W, He H, Yang P, Lyu F, Cheng N, Shen X (2020) Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN. IEEE Trans Wireless Commun 20(2):911–925

Zhuang F, Qi Z, Duan K, Xi D, Zhu Y, Zhu H, Xiong H, He Q (2020) A comprehensive survey on transfer learning. Proc IEEE 109(1):43–76

Authors and Affiliations

Parviz Keshavarz Haddadha[1] · **Mohammad Hossein Rezvani**[1] · **Mahdi MollaMotalebi**[1] · **Achyut Shankar**[2,3,4]

✉ Mohammad Hossein Rezvani
rezvani@qiau.ac.ir

Parviz Keshavarz Haddadha
p.haddadha@qiau.ac.ir

Mahdi MollaMotalebi
motalebi@qiau.ac.ir

Achyut Shankar
ashankar2711@gmail.com

1 Department of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran

2 Department of Cyber Systems Engineering, WMG, University of Warwick, Coventry CV74AL, UK

3 Center of Research Impact and Outreach, Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India

4 School of Computer Science Engineering, Lovely Professional University, Phagwara, 144411 Punjab, India