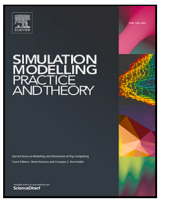
Simulation Modelling Practice and Theory 130 (2024) 102858

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



Integrating OpenAI Gym and CloudSim Plus: A simulation environment for DRL Agent training in energy-driven cloud scaling

Siti Nuraishah Agos Jawaddi [b], Azlan Ismail [a,b,*]

[a] Institute for Big Data Analytics and Artificial Intelligence (IBDAAI), Kompleks Al-Khawarizmi, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia

[b] School of Computing Sciences, College of Computing, Informatics and Mathematics, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia

ARTICLE INFO

Keywords:
Deep reinforcement learning
Energy-driven cloud scaling
Power consumption
Cloud simulation

ABSTRACT

Experimentation in real cloud environments for training Deep Reinforcement Learning (DRL) agents can be costly, time-consuming, and non-repeatable. To overcome these limitations, simulation-based approaches are promising alternatives. This paper introduces a specialized simulation environment that integrates OpenAI Gym, a popular platform for reinforcement learning, with CloudSim Plus, a versatile cloud simulation framework. The proposed simulator specifically focuses on the case study of energy-driven cloud scaling. By leveraging the strengths of both Python-based OpenAI Gym and Java-based CloudSim Plus, the simulation environment offers a flexible and extensible platform for DRL-Agent training. The integration is facilitated through a gateway that enables seamless interaction between the two frameworks. The simulation environment is designed to support the training process of DRL agents, enabling them to tackle the complexities of cloud scaling in an energy-aware context. It provides configurable settings that represent various cloud scaling scenarios, allowing researchers to explore different parameter configurations and evaluate the performance of DRL agents effectively. Through extensive experimentation, the proposed simulation environment demonstrates its functionality and applicability in measuring the performance of DRL agents with respect to energy-driven cloud scaling. The results obtained from the case study validate the effectiveness and potential of the simulation environment for training DRL agents in cloud scaling scenarios. Overall, this work presents a novel simulation environment that bridges the gap between DRL-Agent training and cloud scaling challenges, offering researchers a valuable tool for advancing the field of energy-driven cloud scaling through reinforcement learning.

1. Introduction

Cloud computing has revolutionized the IT industry by providing scalable and on-demand access to computing resources. Effectively managing and dynamically scaling cloud infrastructures to meet changing workloads remains a critical challenge for service providers. Traditional approaches to cloud scaling often fall short of fully utilizing the underlying infrastructure's potential or adapting to evolving demands. DRL, a subfield of artificial intelligence, offers promising capabilities for intelligent decision-making and automation in complex and dynamic environments. However, training DRL agents in real cloud environments is expensive, time-consuming, and non-repeatable. To address these limitations, simulation-based modeling has gained popularity in cloud computing

* Corresponding author at: Institute for Big Data Analytics and Artificial Intelligence (IBDAAI), Kompleks Al-Khawarizmi, Universiti Teknologi MARA (UiTM), 40450 Shah Alam, Selangor, Malaysia.
E-mail addresses: aishahagos96@gmail.com (S.N. Agos Jawaddi), azlanismail@uitm.edu.my (A. Ismail).

https://doi.org/10.1016/j.simpat.2023.102858
Received 18 May 2023; Received in revised form 6 October 2023; Accepted 29 October 2023
Available online 31 October 2023
1569-190X/© 2023 Elsevier B.V. All rights reserved.

research [1–3]. It enables developers or researchers to model and evaluate various algorithms in a controlled and customized environment within the scope of their studies. For instance, in existing works, only Bitsakos et al. [4] evaluate the implementation of DRL-based cloud scaling in a real environment while other works utilized several simulation frameworks to evaluate their proposed DRL-based approach including CloudSim [5,6], TensorFlow [7] and WorkflowSim [8]. While existing works have utilized simulation frameworks to evaluate proposed scaling approaches, none have implemented a concurrent simulation that trains the RL agent's brain while simultaneously simulating the decision-making process in two distinct environments. This concurrent simulation and training approach is considered necessary as it offers improved training efficiency, real-time evaluation, robustness testing, transfer learning, and validation against predefined metrics and requirements.

Therefore, we aim to support the research of DRL for cloud scaling by addressing its evaluation perspective. Specifically, we propose a simulation method that offers a specialized simulator for investigating DRL performance in relation to cloud scaling configurations. The scope of the simulation is motivated by the following key questions, namely:

- What is the metric that has not been given much attention within the context of RL in cloud scaling?
- What is the specialized context of the simulation for DRL in cloud scaling?
- How to realize the cloud simulation method effectively?
- Which DRL agent types are commonly used to address cloud scaling problems?

The first question leads us to the energy consumption metric [9]. Energy-aware has been one of the significant topics for cloud scaling. However, little attention has been given from the DRL point of view. Tackling energy consumption has a significant contribution to the operational costs of the data center [10]. The second question results in the adoption of DRL. This decision is made after reviewing the related works namely, [4–6] which have implemented DRL for the cloud scaling, where [4,6] specifically applied the experience replay technique to achieve better results. Beside cloud scaling, [7] also implemented this approach to address the problem of resource provisioning and task scheduling. In addition, our review of the literature of DRL for cloud scaling found that only several works [5,6,11] have actually utilized existing cloud simulators, to be specific the CloudSim [12]. Along with this direction, however, we have decided to adopt CloudSim Plus which has been reported as an improved version of CloudSim in terms of the engineering aspects, such as maintainability, reusability and extensibility [13]. Meanwhile, two types of DRL-Agent training approaches have been identified, namely, value-function iteration and policy optimization [14]. Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) have been selected to represent the first approach whilst Proximal Policy Optimization (PPO) has been chosen to represent policy optimization.

Thus, our contributions encompass three main aspects:

- **Simulation Method Development:** We have created a simulation method that combines a Python-based DRL framework (Keras) with a Java-based cloud simulation tool (CloudSim Plus), using Py4J as a bridge. This method is designed to evaluate how effectively DRL can optimize energy-driven cloud scaling, with a specific focus on enhancing Power Usage Effectiveness (PUE). It operates within a hybrid Python-Java environment.
- **Proof-of-Concept Implementation:** We have put our simulation method into practice as a proof-of-concept to demonstrate its practicality. This showcases its capability to assess and compare the performance of three different DRL agents, all trained using the same simulated environment.
- **Comparative Analysis:** We have conducted a detailed comparative analysis. This involves evaluating the simulator's performance during the training of each DRL agent and assessing the learning performance of these agents. We examine factors like how quickly they converge to effective solutions, their decision-making processes for scaling actions, and their task completion rates based on when they decide to stop training. These analyses provide insights into the strengths and weaknesses of each DRL agent in our context.

The rest of this paper is organized as follows. Section 2 summarizes the related works and highlights the key findings. Section 3 introduces the energy-driven cloud scaling problem and provides the fundamental knowledge of the adapted techniques. Section 4 presents the proposed simulation approach, while Section 5 elaborates the algorithm design. Section 6 describes the implementation of the GUI and the visualization outputs. Section 7 discusses the performance evaluation of our simulator and the three DRL agents trained within it. Finally, Section 8 concludes the paper with several suggestions for future work.

2. Related works

A few surveys have been published in the area of cloud scaling, specifically by Qu et al. [15],Garí et al. [16]. Hence in this section, we only highlight the related works that addressed the implementation of RL, DRL, and energy-driven cloud scaling. We then summarize the simulation works for the cloud scaling.

2.1. RL-based cloud scaling

RL has been one of the most studied techniques for realizing cloud scaling besides other techniques such as threshold-based policies, queuing theory, and control-theoretic approach. RL provides the advantage in terms of addressing the cloud scaling problem without any prior knowledge or model [17].

Several works have been published in the area of cloud scaling using RL techniques. One such work by Dutreilh et al. [18] used Q-learning and Markov Decision Processes (MDP) to address the resource allocation problem. The proposed method was evaluated by simulating a web application on the cloud. Another work by Veni and Bhanu [19] proposed a neuro-fuzzy model-based RL technique to scale resources vertically. The resource scaling module was implemented on a Xen virtualized environment to evaluate the proposed approach. Similarly, Arabnejad et al. [20] combined fuzzy logic control and RL algorithms, SARSA and Q-Learning, for decision-making of cloud scaling. The OpenStack platform was used to test the proposed approach. Ghobaei-Arani et al. [11] used Q-Learning to generate scaling policies that minimize resource rental costs and meet the SLA of the cloud application. The approach was evaluated on a simulated environment developed using the CloudSim toolkit.

Horovitz and Arian [21] adapted and improved the Q-Learning approach to dynamically adjust utilization thresholds, minimizing resource allocation and preventing SLA violation. The algorithm was evaluated by simulating a web app on a rented virtual machine and running an actual web app on a Node.js web server. Bibal Benifa and Dejey [22] aimed for cloud scaling that minimizes resource utilization and response time using the SARSA algorithm. The algorithm was implemented on three case studies and evaluated in a Xen-virtualized environment. Wei et al. [23] adapted Q-Learning to generate resource renting policies for SaaS providers to adapt to the uncertainty of the market. The approach was simulated using Matlab R2014b on an on-premise machine. Nouri et al. [24] proposed a decentralized architecture for provisioning and managing resources to meet SLA targets while minimizing infrastructure costs using RL-based elasticity controllers. The approach was evaluated on several web apps running on public cloud virtual machines.

Recently, Schuler et al. [25] addressed workload-based auto-scaling in a serverless environment using an RL-based model to determine the optimal concurrency for an individual workload. The approach has been experimented with on IBM Cloud Kubernetes Service (IKS). However, none of the existing works have implemented a concurrent simulation that trains the RL agent's brain while simultaneously simulating the decision-making process in two distinct environments. Implementing this concurrent simulation and training approach is necessary as it offers several benefits, including improved training efficiency, real-time evaluation of the agent's performance, robustness testing, transfer learning, and validation against predefined metrics and requirements.

2.2. DRL-based cloud scaling

The complexity and dimensionality of the state/action space in the cloud scaling problem contribute to the performance degradation of RL. Consequently, the recent development of DRL aims to enhance RL's efficiency in exploring larger state/action spaces, surpassing the evaluation of state values and the necessity for novel actions [6].

The number of works addressing cloud scaling using DRL remains limited. Wang et al. [5] investigated the utilization of DRL in generating cloud scaling policies based on cost and performance, employing the CloudSim simulator. Bitsakos et al. [4] introduced a DRL agent for cloud elasticity, aiming to balance cluster throughput, latency, and user costs, utilizing a real cloud environment for experimentation. Kardani-Moghaddam et al. [6] employed DRL in a resource scaling framework that combines vertical and horizontal scaling to respond to performance anomalies in resource provisioning, and the framework's evaluation was conducted using the CloudSim simulator.

Although various works have addressed cloud resource management rather than cloud scaling, a few notable ones deserve mention. Liu et al. [26] introduced a framework that utilizes deep reinforcement learning (DRL) for global decision-making and model-free RL for power management. They employed a simulation setup to evaluate their approach. Cheng et al. [7] proposed a DRL method, specifically Deep Q-Learning, for resource provisioning and task scheduling, minimizing energy costs for large-scale Cloud Service Providers through experience replay. They employed Tensor Flow for simulation. Tong et al. [8] implemented a DRL technique combining back-propagation network and Q-Learning to address task scheduling in the cloud, evaluated using WorkflowSim. Nevertheless, the potential of CloudSim Plus remains unexplored for evaluating DRL-based cloud scaling methods.

2.3. Energy-driven cloud scaling

Several studies have explored energy-efficient cloud scaling by introducing resource provisioning mechanisms that allocate excess capacity based on customer utilization patterns. For example, Moreno and Xu [27] proposed a customer-aware overallocation algorithm that employs statistical techniques to provision resources in real-time cloud data centers. Their study investigated the effect of the proposed mechanism on energy efficiency and SLA fulfillment, and a simulator based on the CloudSim framework was developed to evaluate the efficiency of the cloud scaling model.

Tesfatsion et al. [28] presented a cloud scaling solution that aimed to minimize energy consumption while meeting performance objectives. Their approach utilized a control-theoretic approach to address the dynamic resource-provisioning problem, and experimentation was performed in a testbed environment. Similarly, Paya and Marinescu [29] proposed energy-aware scaling algorithms that maximize the number of servers in different operating scenarios (energy-optimal regimes) using load balancing and application scaling. They simulated the process in VMs rented on the Amazon cloud.

Moreover, Berkane et al. [30] proposed an architecture for scaling and energy efficiency based on the MAPE-K [31] and Feature Model [32]. They used threshold-based policies to model the scaling policies and evaluated the approach on several internet browsers of an on-premise machine. However, none of these studies combined the OpenAI Gym library and CloudSim Plus framework to provide a simulated environment to assess the performance of energy-driven cloud scaling algorithms.

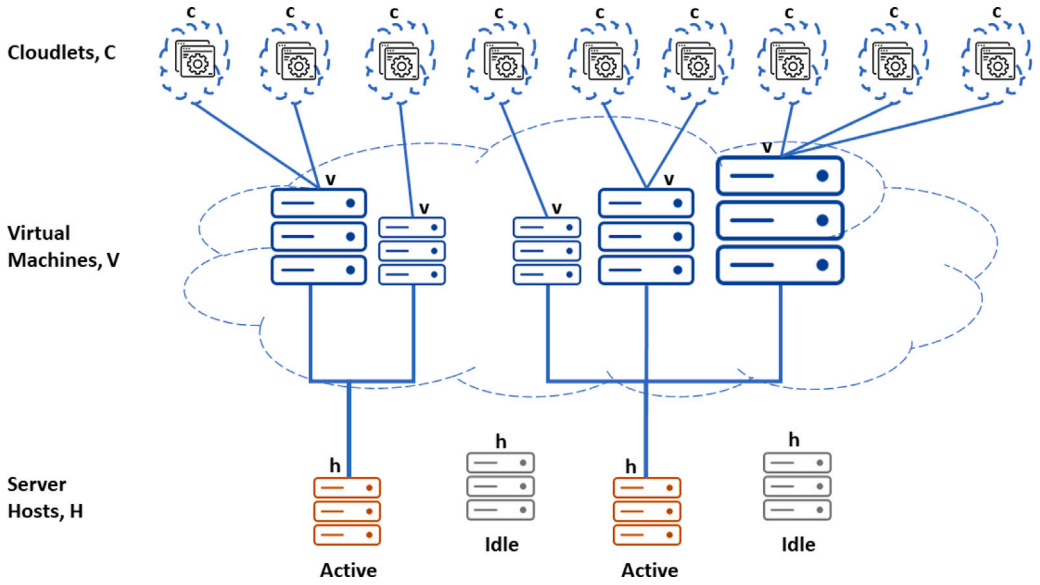


Fig. 1. Simplified illustration of cloud scaling environment.

2.4. Cloud scaling simulation environment

Several cloud simulation frameworks have been proposed in the literature to evaluate various configurations and algorithms for cloud computing. A few reviews have summarized and categorized these simulators based on different dimensions, such as [1–3]. Each proposed simulator can be used for similar or different purposes. Some of them provide a broad context of simulation capabilities, such as CloudSim [33], CloudSim Plus [13], and GreenCloud [34], which enable different kinds of cloud simulation settings with some customization efforts. Meanwhile, some specialized simulators focus on a particular cloud simulation aspect, such as the visual modeler [35] or a specific area of cloud computing problem, such as energy-aware cloud scheduling [36].

Previous works on cloud scaling simulation can be found in [37–39]. In particular, Kim et al. [37] presented a simulator that focuses on the cloud user's perspective and supports horizontal and vertical scaling to evaluate resource usages. Vondra et al. [38] proposed a simulation method for cloud scaling using queuing network models based on the R language and the PDQ open-source queuing network analyzer library [40]. Aslanpour et al. [39] introduced AutoScaleSim, which extends the existing CloudSim simulator to support auto-scaling of web applications in cloud environments in a customizable and extendable manner, emphasizing control theory based on MAPE for the cloud scaling technique.

Our work addresses specialized cloud simulation for cloud scaling using DRL with energy efficiency concerns. We extend CloudSim Plus by integrating it with the OpenAI Gym environment through Py4J as the gateway, forming the core components of our proposed simulator. To the best of our knowledge, no previous studies have addressed the specific scope presented in this paper.

3. Background

In this section, we first introduce the energy-driven cloud scaling problem to be focused on in this paper. We then provide the fundamental background of the applied technique which is DRL.

3.1. Energy-driven cloud scaling decision

A simplified illustration of the cloud scaling environment is presented in Fig. 1. The cloud scaling decision is a problem of deciding what is the best action to be executed at a certain point of time. The need to decide the best action is determined by several factors, such as CPU utilization. Meanwhile, the energy-driven cloud scaling emphasizes the power consumption of the cloud resources as one of the important factors to determine the best action. Furthermore, there are two types of scaling mechanisms, vertical and horizontal scaling [15]. To motivate this work, we limit to the vertical scaling of VM in relation to the *CPU utilization*. Hence, for simplicity, we use *utilization* and *CPU utilization* interchangeably.

3.1.1. Problem formulation

In the context of the Energy-driven Cloud Scaling Decision Problem (ECSDP), the goal is to identify an optimal scaling decision that simultaneously reduces host energy consumption within the data center and minimizes the PUE. This involves taking into account the observed cloud metrics while adhering to the established scaling actions and rules.

Definition 3.1. Let ECSDP be a set of components denoted as ECSDP = EN, CM, A, G, θ, T_n, where:

1. $E_n = \{H, V, C\}$ represents the elements of the cloud environment, consisting of a set of finite hosts, a set of finite VMs a set of finite cloudlets where a single cloudlet $c_i \in C$ is used to execute a cloud application.
2. $M = \{T_C, U_C, P_V, P_H, R_V, R_R, B_V, B_R, E_H^U, E_H^C\}$ is a set of observed cloud metrics, including:
   - $T_C = \{t_{c1}..t_{cn}\}$: a set of cloudlets' execution times.
   - $U_C = \{u_{c1}..u_{cn}\}$: a set of cloudlets utilization.
   - $P_V = \{p_{v1}..p_{vn}\}$: processing elements (PE) of a set of finite VMs.
   - $P_H = \{p_{h1}..p_{hn}\}$: PE of a set of finite hosts.
   - $R_V = \{ram_{v1}..ram_{vn}\}$: RAM capacities of a set of finite VMs.
   - $R_R = \{ram_{r1}..ram_{rn}\}$: a set of RAM requested by the cloudlets.
   - $B_V = \{bw_{v1}..bw_{vn}\}$: bandwidth capacities of a set of finite VMs.
   - $B_R = \{bw_{r1}..bw_{rn}\}$: a set of bandwidth requested by the cloudlets.
   - $E_H^U = \{e_{h1}^u..e_{hn}^u\}$: a set of host energy consumption in relation to the VM utilization.
   - $E_H^C = \{e_{h1}^c..e_{hn}^c\}$: a set of energy consumed by the cooling devices.
3. $A = \{a_{dw}, a_{up}, a_{dn}\}$ is a set of scaling actions, comprising:
   - a_{dw} : decreasing the number of processors
   - a_{up} : increasing the number of processors
   - a_{dn} : maintain the number of processors (i.e., do nothing)
4. $G = \min_x(PUE)$, where x refers to E_H^U. The objective function of the cloud scaling decision aim to minimize the PUE of a data center by adjusting the host energy consumption.
5. $\theta = \{\theta_0, \ldots, \theta_n\}$ is a set of rules to determine the scaling decision, where each $\theta_i \in \theta$ constrains the maximum or minimum utilization and power consumption of the host.
6. $T_n = \{t_1..t_n\}$: Number of steps for each training episode.

The cloud metrics M are essential in monitoring the operation within the cloud environment E_n. Through the metrics, a few analyses can be performed to estimate the impact of scaling decisions on the cloud system. In this work, all metrics grouped in M are computed using the functions implemented in CloudSim Plus [13]. We employed a pair of built-in functions that capture both the initiation and completion times of execution, enabling us to calculate the cloudlet execution time (T_C). Furthermore, we leveraged an additional built-in function to retrieve the number of PEs assigned to a VM (P_V), aggregating these values to determine the total processing elements available on a host (P_H). Additionally, the built-in power model was harnessed to compute the energy consumption of a host (e_h^u), relying on the CPU utilization of said host.

$$
\begin{aligned}
e_h^u &= \begin{cases} stat(e_H^u) + dyn(e_h^u), & \text{if } u_h^v > 0 \\ stat(e_H^u), & \text{if } u_h^v <= 0 \end{cases} \\
&Where; \quad stat(e_H^u) = (k \times e_H^{max}), \\
&dyn(e_h^u) = ((1-k) \times e_H^{max} \times u_h^v)
\end{aligned} \tag{1}
$$

Eq. (1) presents the formula of the power consumption model adopted from [13] which defines the power consumption of a single host (e_h^u) derives from the VM utilization (u_h^v) and is influenced by the types of the host status, namely, idle and active. The elements involved are as follows:

- $stat(e_H^u)$ is the static power of the host;
- $dyn(e_h^u)$ is the dynamic power of the host;
- $u_h^v > 0$ refers to the resource utilized by the host is more than 0%;
- $u_h^v <= 0$ refers to the resource utilized by the host that is less or equal to 0%
- k is a constant value that lies between 0 and 1. When k is set to 0, the static power becomes zero, and the power consumption is solely determined by the dynamic power. This implies that the power consumption is entirely dependent on the host's utilization, and there is no baseline power consumption when the host is idle. When k is set to 1, the dynamic power component becomes zero, and the power consumption is solely determined by the static power. This implies that the power consumption remains constant regardless of the host's utilization.

Table 1
Rules for scaling decision process.

Rule	Condition	Description
Rule 1	$u_c > SLA(u_C^{\max})$	Cloudlet utilization is greater than max utilization defined in SLA
Rule 2	$u_c < SLA(u_C^{\min})$	Cloudlet utilization is smaller than min utilization defined in SLA
Rule 3	$e_h^u > SLA(e_H^{\max})$	Host energy consumption is greater than max host power defined in SLA
Rule 4	$e_h^u < SLA(e_H^{\min})$	Host energy consumption is smaller than min host power defined in SLA

$$PUE = \frac{(\sum_{h=0}^{H} e_h^u) + (\sum_{h=0}^{H} e_h^c)}{\sum_{h=0}^{H} e_h^u} \tag{2}$$

PUE is formulated based on the fraction of two parameters, which are the total facility power and total IT equipment power [41]. Eq. (2) shows the formula used to compute PUE in this project. The facility power encompasses the combined energy consumption of data center hosts and the devices employed for host cooling ($(\sum_{h=0}^{H} e_h^u) + (\sum_{h=0}^{H} e_h^c)$). In contrast, the total power of IT equipment pertains to the collective energy consumption of data center hosts alone ($\sum_{h=0}^{H} e_h^u$). This project employs the calculation approach introduced in a prior research work [42] .

Finally, the rules θ determine when the scaling should be made. They constrain two dimensions, namely, the utilization and the power consumption where the thresholds are obtained from the Service Level Agreement (SLA) file. The details are presented in Table 1.

3.2. Deep Reinforcement Learning (DRL)

DRL is a combined implementation of Deep Learning (DL) and Reinforcement Learning (RL) which can solve a wide range of complex decision-making tasks using human-like intelligence [14]. RL is constructed upon foundational principles, including agent-based trial-and-error learning, the formulation of stochastic problems through the lens of the Markov Decision Process (MDP), the application of the Bellman equation, and the utilization of temporal difference methods. Meanwhile, DL harnesses the potential of deep and expansive neural networks to emulate the cognitive processes of the human brain when handling data and constructing patterns for informed decision-making.

Both techniques possess their own strengths and weaknesses when contributing to the composition of DRL. For instance, RL empowers an agent to learn through its interactions and experiences, allowing it to discern optimal strategies based on the rewards garnered during training [43]. However, a drawback of RL emerges in cases where the potential states expand, leading to limitations in memory for storing Q-values within its lookup table. Conversely, DL offers the capability to handle extensive datasets using neural network approximations, but it exhibits susceptibility to adversarial attacks, yielding unstable outcomes in network behavior [14]. Consequently, DRL emerges as a fusion to leverage the merits of both techniques while mitigating their drawbacks, specifically in terms of employing Q-networks to approximate and retain Q-values within a continuous state space.

Nevertheless, DRL also has weaknesses in terms of sampling data for online learning and Catastrophic interference [14]. This is because the sampled data is highly correlated and the network tends to forget its previous learning after learning new things. Therefore, the implementation of the target network, as well as experience replay, has been introduced to stabilize the Q-network [44]. Experience replay involves updating the Q-network within a fixed-size mini-batch buffer, enabling the agent to archive its training episodes. This buffer concurrently disrupts data similarity to address the correlation concern [45]. Consequently, the agent retains past learning and can systematically draw upon its experiential tuples to counteract the oscillation or divergence of Q-values. On the other hand, a target network used to estimate the targeted Q-values is separated from the network that estimates the action policy. Algorithms such as Deep Q-Network (DQN), Double DQN, and Proximal Policy Optimization (PPO) implement these strategies. In contrast to DQN, Double DQN integrates two target networks (Network A and Network B) to compute Q-values, with alternating roles after each training iteration. Meanwhile, PPO deviates from the preceding algorithms by additionally prioritizing policy optimization alongside value optimization.

4. Approach design

In this section, we elaborate on the proposed simulation approach in detail. We started by introducing the system architecture followed by the simulation process.

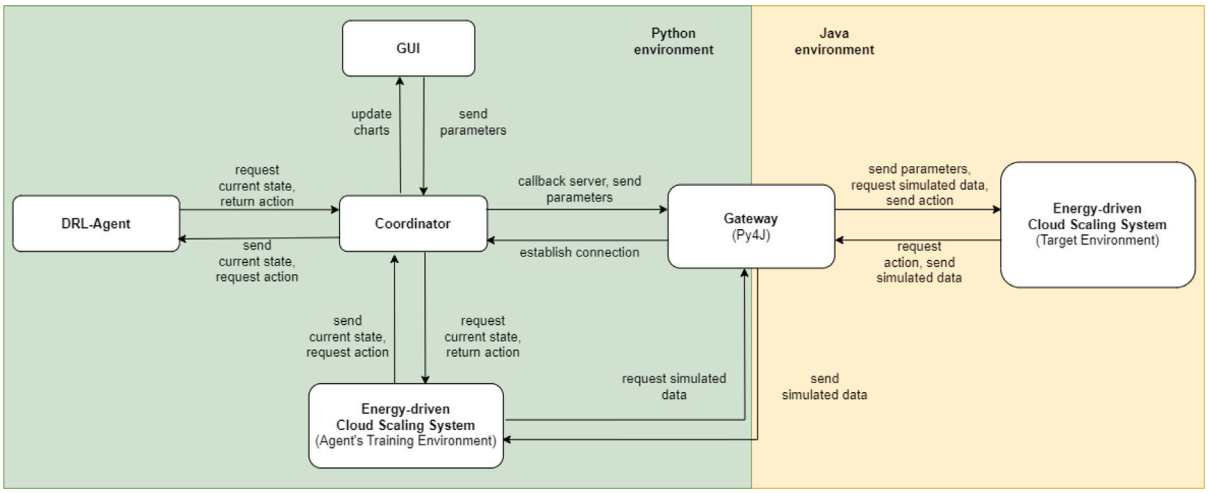


Fig. 2. System architecture of the simulator.

4.1. Overview

In this study, we create two environments: the targeted environment (using CloudSim Plus [13]) and the agent's training environment (using OpenAI Gym [46]). The interaction between these environments has been illustrated as in Fig. 2. They are integrated, with the targeted environment simulating the cloud scaling process in advance to provide initial states for training episodes in the training environment. This integration offers benefits as the simulation closely resembles real-world cloud scenarios, enabling the agent to interact realistically and optimize actions accordingly. This leads to more reliable and accurate performance evaluations for energy-driven cloud scaling processes. The learned policies become better suited to handle complexities and uncertainties in real cloud environments, making evaluation results more meaningful. Additionally, the agent can leverage power-aware packages to optimize actions not only based on rewards but also on power consumption, crucial for evaluating energy-driven cloud scaling processes. Thus, the evaluation assesses the impact of different scaling strategies on overall power consumption, a critical factor in modern cloud environments.

4.2. System architecture

We have developed an energy-driven cloud scaling simulator using Python and Java programming frameworks. In building the simulator, we utilized a set of tools including Keras [47], OpenAI Gym [48], CloudSim Plus [49], and Py4J [50]. The architecture of the simulator consists of six main components: the *Coordinator*, *Graphical User Interface (GUI)*, *DRL-Agent*, *Gateway*, and two instances of the *Energy-driven Cloud Scaling System*. These instances are implemented using distinct frameworks.

The *Coordinator* component serves the purpose of facilitating communication between the *DRL-Agent* and *Energy-driven Cloud Scaling System*. This is achieved by merging the training process with cloud simulation, resulting in a unified system. During this process, we utilize OpenAI Gym to create a controlled training environment for the agent. Simultaneously, the cloud scaling simulation occurs within the target environment, powered by the CloudSim Plus framework. Additionally, we have developed a *Graphical User Interface (GUI)* that permits users to interact with the simulator. When the button function is activated, the *Coordinator* plays a pivotal role in enabling data exchange among these three components.

The *GUI* component includes graphical components that allow users to input simulation parameters and view results after completion. These parameters are essential for starting the simulation. The results are presented through Kernel Density Estimation (KDE) plots and a summarized list of information. These graphical representations depict the agent's performance in terms of both reward values and cumulative rewards obtained during each episode. These values are tracked in real-time within the OpenAI Gym environment. Additionally, the GUI includes graphs illustrating the influence of the agent's decision-making on host energy consumption and PUE within the CloudSim Plus environment. KDE plots excel at visualizing the probability distribution of the performance metrics, including rewards, cumulative rewards, host energy consumption, and PUE, across episodes. This visualization aids in understanding the range and variability of performance. KDE plots also offer users a quick, accessible overview of the data, eliminating the need for them to grapple with complex statistical measures. They allow users to perceive the distribution's shape and spot any potential outliers or patterns. These features provide users with insights into the relationship between specific simulation configurations and the agent's training performance. As the simulation wraps up, users have the option to save the collected results.

The *DRL-Agent* component holds the responsibility of constructing a neural-network model and defining the behaviors of the agent. The training process begins by predicting an appropriate scaling action based on the state provided by the training environment. This chosen action is then relayed to the target environment through the *Coordinator*. The scaling action, performed

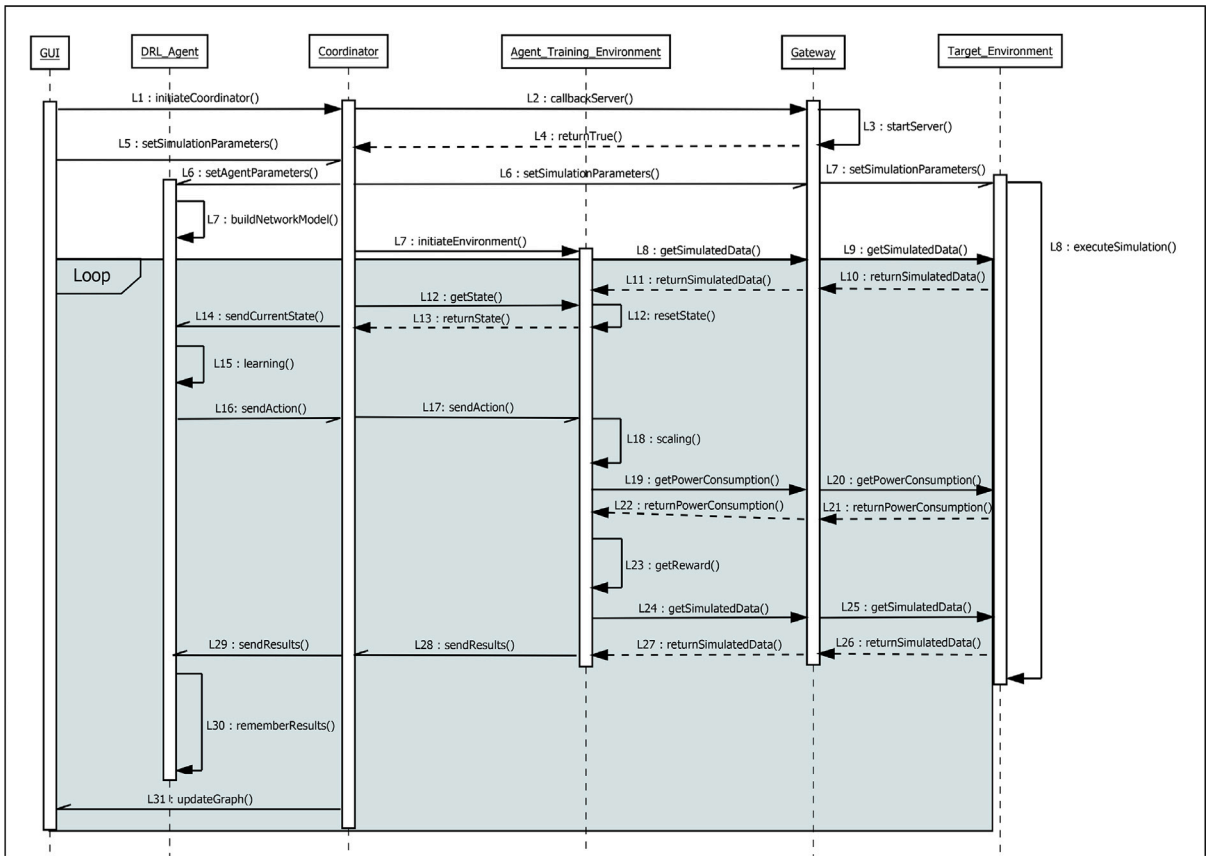


Fig. 3. Sequence diagram of the simulation in training environment.

on the resources of the target environment, triggers changes in host energy consumption. Subsequently, the agent receives an updated state. Before proceeding to train in the subsequent episode, the agent stores various simulation outcomes, such as state, next state, action, and reward, in its storage buffer. For agents that rely on value-based methods, these outcomes are stored in a minibatch buffer. This collected data is used for experience replay, a mechanism that aids in continuous learning. On the other hand, policy-based agents store outcomes to estimate the advantage value associated with each policy.

The *Gateway* component manages the connection between the *Coordinator* and the DRL components (Python-based *DRL-Agent* and *Energy-driven Cloud Scaling System*) with the Java-based *Energy-driven Cloud Scaling System*. It facilitates interaction between these components, which are built using different programming frameworks. To establish this connection, a listener interface is created on both sides of the program, enabling data exchange when the *Gateway* server is activated. This interface encompasses various methods for communication and simulation parameter configuration. The main class initializes the Gateway server and defines the structure of the datacenter. These methods serve distinct purposes, such as checking connection status, monitoring simulation progress (start, completion, readiness, or conclusion), and exchanging simulation inputs and outputs between the training and target environments. For effective data exchange, it is essential that the methods transferring updated observations between the Java and Python environments are consistent. This requires using the same method/function names in both environments. In this project, the environment built upon CloudSim Plus updates values for the OpenAI Gym-based environment, while the OpenAI Gym-based environment conveys potential scaling actions determined through agent training. By aligning these method/function names, communication between the two environments becomes seamless and efficient, ensuring effective collaboration between the distinct frameworks.

The final components comprise two instances of the *Energy-driven Cloud Scaling System*. The first instance, implemented using the Python framework, is dedicated to establishing an energy-driven training environment for the *DRL-Agent*. We will also refer to this environment as *ECSS-DRL Environment*. It considers the host's energy consumption conditions (E_H^U) to formulate states and calculate rewards throughout the training phase. Additionally, to create a state that closely resembles an actual cloud system, it assigns simulated data values obtained from the target environment to its state parameters. This process is detailed in Section 5.2.

The second instance of the *Energy-driven Cloud Scaling System* is built using Java and serves as the target environment, encompassing the execution of the simulation and the cloud scaling process. CloudSim Plus is employed to model and facilitate the functionalities of the elements within this simulated cloud environment. This environment will also be referred to as *ECSS-Environment* in this paper. The cloud scaling process involves three core phases: observing data, analyzing data, and making scaling

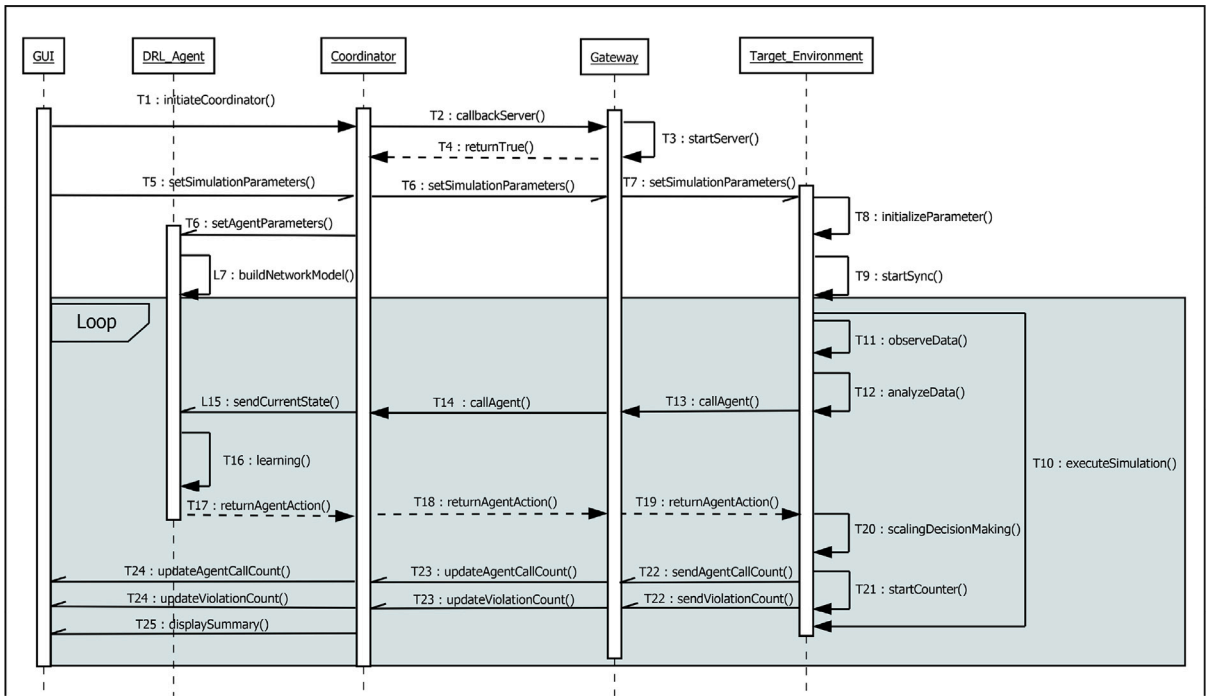


Fig. 4. Sequence diagram of the simulation in target environment.

decisions. The observed and analyzed data pertaining to the utilization of the running cloudlet and the energy consumption of the active host. The scaling decision-making phase entails executing the scaling decisions provided by the *DRL-Agent*. Upon completion of each episode, the total number of agent calls and instances of energy violations are sent back to the *Coordinator* for display on the *GUI*.

4.3. Simulation process

Fig. 3 shows the sequence diagram for the simulation process involving the Agent's training environment, while Fig. 4 depicts the target environment's simulation process. The training environment consists of six components, and the target environment comprises five components, representing the architecture described in Section 4.2. Some of the processes conducted by the components may occur simultaneously. In this section, we explain the functions and interactions of these components, detailing how they collectively achieve the energy-driven cloud scaling simulation.

In Fig. 3, the simulation begins when the user enters simulation parameters, prompting the *GUI* to start the *Coordinator* (seq. L1). The *Coordinator* then triggers the initialization of the *Gateway* server (seq. L2). Upon receiving confirmation from the *Gateway* ('true'), the previously entered parameters are dispatched via the Gateway to the target environment, initiating its simulation process (seq. L3–L7). Simultaneously, the agent's training environment is activated to acquire simulated data from the target environment (seq. L7–L11). This new simulated data resets the training environment's initial state (seq. L12). Next, the *DRL-Agent* starts by obtaining parameter values and constructing the Q-network model (seq. L6, L7). It then acquires the state from its training environment, learning to make optimal scaling decisions for each state (seq. L14–L16). The chosen action is forwarded to the target environment, initiating a scaling simulation involving VM resource adjustment and agent reward computation (seq. L18, L23). Various data, including host energy consumption, is collected during this training process (seq. L24–L27). Subsequently, the *DRL-Agent* stores the results in a buffer for future use (seq. L30, L31). Some results are also relayed to the *Coordinator*, which then collaborates with the *GUI* to generate graphs (seq. L28, L32)".

Simultaneously, both the training and target environments initiate simulations through the *Coordinator* (steps T1–T7). The target environment initializes variables to represent cloud infrastructures, syncing them with CloudSim Plus components before starting the simulation (steps T8–T10). Within this process, scaling occurs, involving three sub-activities: data observation, analysis, and scaling decision-making (steps T11, T12, T20). This entails observing and analyzing cloudlet utilization and host energy consumption. If specific conditions are met, the *DRL-Agent* is invoked to make scaling decisions (steps T13, T14). The *DRL-Agent* learns within its own environment to make these decisions and then applies them in the target environment, adjusting resources for running VMs using the selected action (steps T15–T17). Each episode tracks the agent calls and energy consumption violations, updating the *Coordinator*. The *GUI* displays the cumulative count for each counter once the entire simulation concludes (steps T21–T25).

Table 2
Simulation parameters (SP).

Parameter	Description
E_p	Number of training episodes for the *DRL-Agent*
E_n	A set of cloud environments consisting of finite numbers of hosts, VMs, and cloudlets
e_h^{max}	Maximum power threshold of an active host
$stat(e_h^u)$	Power consumption of an idle host
T_{safe}	Safe temperature of a host
T_n	Simulated time steps for each training episode

Table 3
Agent hyperparameters (AHP).

Parameter	Description
ϵ	Exploration versus exploitation trade-off parameter for action selection
ϵ_{decay}	Rate of exploration reduction over time
ϵ_{min}	Minimum exploration rate during the exploration schedule
γ	Discount factor weighing future rewards in Q-learning updates
α_{policy}	Coefficient regulating policy network adaptation speed for reward maximization
α_{value_v}	Coefficient modulating value function estimation speed according to rewards (i.e. for value-function based agent)
α_{value_p}	Coefficient governing update pace for enhanced value function accuracy in advantage estimation (i.e. for policy-based agent)
$\mathbb{C}_{entropy}$	Coefficient that encourages exploration by controlling policy randomness
$\mathbb{C}_{value}$	Coefficient that balances policy and value function updates
$clip$	Clip ratio
$\mathbb{B}_{size}$	Size of buffer
$\mathbb{L}_{size}$	Size of neural network layers

5. Algorithm design

This section presents the algorithm design for the *Coordinator* as well as two *Energy-driven Cloud Scaling System* built as *Agent's Training Environment* and *Target Environment*.

5.1. Coordinator

The *Coordinator* initializes the Python-side gateway to establish a connection between Python and the Java environment. With this connection in place, a series of steps unfold. Initially, the GUI becomes accessible, enabling users to input values for a predefined set of simulation parameters. Subsequently, these parameters are transmitted to the targeted environment via the gateway. Concurrently, the *Coordinator* triggers the activation of the *DRL-Agent*, which involves passing hyperparameter values and initializing its training environment. Table 2 and Table 3 present the simulation parameters and agents' hyperparameters respectively. Policy-based agents use all the listed hyperparameters except for α_{value_v}. On the other hand, value-based agents use only seven hyperparameters (ϵ, ϵ_{decay}, ϵ_{min}, γ, α_{value_v}, $\mathbb{B}size$, and $\mathbb{L}size$). This environment facilitates training by gathering simulated data from the targeted environment through the established gateway connection.

The *DRL-Agent* undertakes various tasks including neural network creation, prediction of scaling actions and corresponding Q-values, storage of simulation outputs derived from predictions (namely, *action*) and Python environment interactions (specifically, *state, reward, done status, log probability, Q-values*). This stored information which can be referred to as Sim_{output}, is harnessed throughout the training process, sustaining agent comprehension until the maximum step limit is attained. Finally, the *Coordinator* concludes by retrieving data from the training environment. This data serves a dual purpose: it dynamically updates the GUI's charts and simulation summary as well as cataloged as data frames. Moreover, the *Coordinator* empowers users to preserve the GUI charts on their personal devices.

The *Coordinator* starts by determining the type of *DRL-Agent* selected by the user. If a value-based agent is chosen, the value-based training as in Algorithm 1 will be selected. Meanwhile, if a policy-based agent is chosen, Algorithm 2 which represents the policy-based training will be triggered. In contrast to value-based agents, policy-based agents involve the implementation of trajectories, which capture the agent's entire journey from the initial state to a terminal state. This trajectory encompasses all the actions taken and the rewards obtained along the way. A terminal state, in this context, signifies a state where the agent has either accomplished its objective or reached a point where it cannot proceed further. For this project, the term *value-based agent* refers to DQN and DDQN, while the *policy-based agent* corresponds to PPO.

Algorithm 1: Value-based Training Algorithm

Data: AHP, SP
Result: Sim_{output}
for $e \in E_p$ **do**
  Get s from *ECSS-DRL Environment*
  if *simulation started* **then**
    for $t_n \in T_n$ **do**
      Send s to *DRL-Agent* and get a
      Send a to *ECSS-DRL Environment* and get Sim_{output}
      Send Sim_{output} to *DRL-Agent*
      Get s' from *ECSS-DRL Environment*
      if *target environment activated* **then**
        Send s to *DRL-Agent* and get a
        Send a to *ECSS-Environment*
      if $\mathbb{B}_{size} > 1$ **then**
        Trigger replay of *DRL-Agent*
      Send Sim_{output} to *DRL-Agent*

Algorithm 2: Policy-based Training Algorithm

Data: AHP, SP
Result: Sim_{output}
for $e \in E_p$ **do**
  Get s from *ECSS-DRL Environment*
  if *simulation started* **then**
    for $t_n \in T_n$ **do**
      Send s to *DRL-Agent*'s actor, get a and logits
      Send a to *ECSS-DRL Environment* and get Sim_{output}
      Send s to *DRL-Agent*'s critic, get Q-value
      Send a and logits to *DRL-Agent* and get $\log_{\text{probability}}$
      Send Sim_{output} to *DRL-Agent*
      if *terminal done* **then**
        finish trajectory
      Get Sim_{output} stored in the buffers **for** *each policy iteration* j **do**
        optimize policy
      for *each value iteration* k **do**
        optimize value function
      Get s' from *ECSS-DRL Environment*
      if *target environment activated* **then**
        Send s to *DRL-Agent* and get a
        Send a to *ECSS-Environment*

5.2. Agent training environment

The agent's training environment is modeled as a Markov Decision Process, in relation to the ECSDP addressed in Section 3.1.

Definition 5.1. Let the MDP denoted as $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}\}$, where:

- $\mathcal{S}$ refers to the states of cloud environment E_n during cloud scaling simulation in relation to cloud metrics M such as power consumption of a host (e_h^u) and utilization of a VM (u_h^v);
- $\mathcal{A}$ is a set of scaling actions equivalent to $A = \{a_{dw}, a_{up}, a_{dn}\}$ which labeled as integer $0, 1, 2$;
- $\mathcal{R}$ is the reward computed based on the objective function G;
- $\mathcal{T}$ is the transition probability represented by the decimal value of the actions $\mathcal{A}$;

Table 4
Simulated data.

Parameter	Source
M	CSS-Environment
e_H^{max}	User input
$stat(e_H^u)$	User input
T_n	User input
$SLA_d(u_C^{\max \mid \min})$	SLA file

This component (built on OpenAI Gym-based environment) performs key activities, including generating an environment with state conditions mirroring the present state in the CloudSim Plus-based environment. It captures variations in host energy consumption when the *DRL-Agent* selects scaling actions, rewarding the agent for decisions, and continually tracking updated state conditions.

Creating the training environment entails two primary activities, namely, retrieving the simulated data from the target environment and capturing the changes in the training environment state during agent training. Simulated data which have been defined in Section 3.1.1 and listed in Table 4, can be gained from user input, computed within cloud scaling simulation, or defined in the SLA file. These data establish the state of the training environment and compute rewards.

Once the training environment is established, the state information is passed to the *DRL-Agent* to predict which scaling action is appropriate to counter the current state of the simulated cloud system. In our case, actions are predicted by the agent as integers (e.g., 0, 1, 2). The action results from the prediction are then returned to the training environment to observe the possible impact of the decision on the host in the target environment. In our case, the observation will only be performed if the selected scaling action is either 0 or 1. Otherwise, this observation will be skipped. To observe the impact, the running VM in the target environment is cloned in advance. Then, the resources in the cloned VM are allocated (e.g. increasing utilized PEs) or deallocated (e.g. reducing utilized PEs) based on a constant value, followed by returning the cloned host's power consumption.

The third step involves rewarding the agent according to the energy consumption of the cloned host from the previous step. Energy consumption is compared against static and dynamic power thresholds. Static thresholds assess if host power aligns with standard ranges, while dynamic thresholds prevent the *DRL-Agent* from overly simplifying its environment, thus avoiding extreme scaling decisions. Eqs. (3) and (4) compute static and dynamic power thresholds, where input values are from the user and SLA file respectively. The value for static power threshold is computed as a product of $e_H^{\max}$ with $SLA_s(u_C^{\max})$. Meanwhile, the dynamic power threshold is computed as a product of $e_H^{\max}$ with $SLA_d(u_C^{\max \mid \min})$. The "×" operator denotes the multiplication operation in these calculations. Herein, we differentiate the values from SLA with s and d to emphasize that they can be different depending on the perspective. We assume that the threshold values related to s define a safe region (e.g., between 40% to 80%), while the values related to d define the critical region (e.g., between 30% and 90%).

$$stat(e_U^{\max}) = e_H^{\max} \times SLA_s(u_C^{\max}) \tag{3}$$

$$dyn(e_U^{\max \mid \min}) = e_H^{\max} \times SLA_d(u_C^{\max \mid \min}) \tag{4}$$

Before we calculate the reward using Eq. (5), it is crucial to determine the value of $Modifier$ by applying the rules outlined in Eq. (6) until (10). In the initial step, we establish the initial value of $Modifier$ by sequentially evaluating the state conditions described in Eqs. (6). Importantly, we select only one value at a time. For example, if the first condition is not met, we proceed to assess the second condition. If the second condition is satisfied, we assign a value of '1' to the $Modifier$. In the subsequent step, we update the $Modifier$ value by considering the remaining conditions specified in Eqs. (7) through (10). Finally, in the last step, we use the final $Modifier$ value obtained, as outlined in Eq. (10), to compute the reward. This calculation also involves a fixed constant value k which holds the expected reward values. In our specific case, we have set k to 1, as it aligns with the optimal PUE of 1.0 [41].

$$reward = (k - PUE) * Modifier \tag{5}$$

$$Modifier = \begin{cases} -1, & \text{if } e_h^u > e_H^{\max} \\ 1, & \text{if } e_h^u \leq SLA(e_H^{\max}) \\ 1, & \text{if } e_h^u \geq SLA(e_H^{\min}) \text{ and } e_h^u \leq SLA(e_H^{\max}) \\ 1, & \text{if } ram_v \leq ram_r \\ 1, & \text{if } bw_v \leq bw_r \\ 1, & \text{if } u_c < SLA(u_C^{\max}) \text{ and } u_c > SLA(u_C^{\min}) \end{cases} \tag{6}$$

$$Modifier = Modifier - 2, \quad \text{if } e_h^u > SLA(e_H^{\max}) \text{ or } e_h^u < SLA(e_H^{\min}) \tag{7}$$

$$Modifier = Modifier - 2, \quad \text{if } u_c > SLA(u_C^{\max}) \text{ or } u_c < SLA(u_C^{\min}) \tag{8}$$

$$Modifier = Modifier - 2, \quad \text{if } ram_v > ram_r \tag{9}$$

$$Modifier = Modifier - 2, \quad \text{if } bw_v > bw_r \tag{10}$$

Lastly, the current state information of the target environment is updated. The environment captures new values until maximum training steps are reached, enabling the agent to predict meaningful actions. State information passed to the agent includes host energy consumption, CPU utilization, RAM, and bandwidth capacity.

5.3. Target environment

The target environment is meant to simulate the scaling decision process within a cloud environment setup as presented in Algorithm 3.

Algorithm 3: Cloud Scaling Simulation Algorithm

Data: Cloud scaling simulation parameters CSS
Result: Total of agent calls ag, Total of VM utilization violation vio

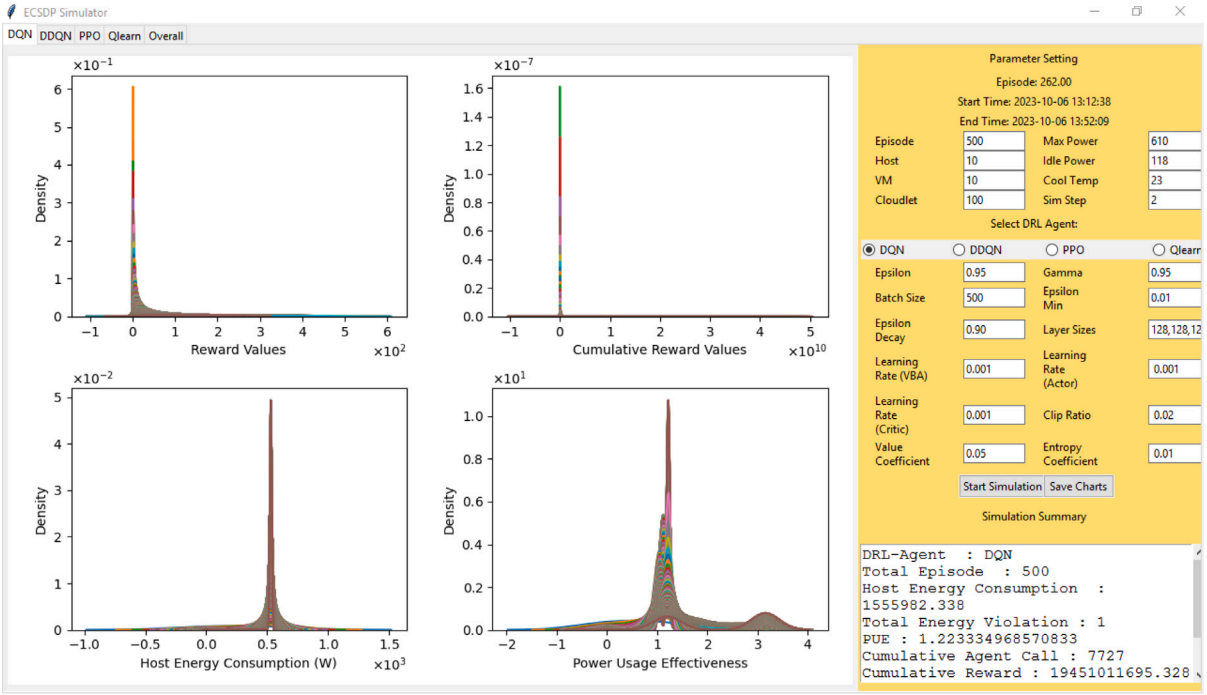
1 **while** $t_n \in T_n$ **do**
2 **while** $v \in V$ **do**
3 **while** $c \in C$ **do**
4 Observe u_c and e_h^u of current running v
5 **if** $u_c > SLA(u_C^{\max})$ || $u_c < SLA(u_C^{\min})$ || $e_h^u \geq SLA(e_H^{\max})$ || $e_h^u \leq SLA(e_H^{\min})$ **then**
6 $a_i \in \mathcal{A} \leftarrow callAgentAction()$
7 $ag = \sum AgentNum + 1$
8 **switch** a_i **do**
9 **case** a_{dw} **do**
10 $p_v = p_v + \delta^-$
11 $r_v = r_v + \delta^-$
12 $b_v = b_v + \delta^-$
13 **case** a_{up} **do**
14 $p_v = p_v - \delta^+$
15 $r_v = p_v - \delta^+$
16 $b_v = b_v - \delta^+$
17 **case** a_{dn} **do**
18 $p_v = p_v \; r_v = r_v \; b_v = b_v$
19 **if** $u_c > SLA(u_C^{\max})$ || $u_c < SLA(u_C^{\min})$ **then**
20 $vio = \sum v_u^{vio} + 1$

The algorithm involves a set of simulation parameters denoted as CSS = { V, C, U_C, E_H^U, $SLA(e_h^{\max})$, $SLA(u_C^{\max})$, P_V, R_V, B_V, T_n} where each of them has been introduced earlier (line 1). It then traverses the lists of VMs and cloudlets to observe data (i.e. $u_c \in U_C$ and the $e_h^u \in E_H^U$ during the simulation cycle) (lines 2–4). The u_c is generated by taking the value of the current simulated time as the input of the CPU utilization model. Then, the value of u_c is used as the input of the power model to compute the e_h^u. After that, the observed data are analyzed to determine any violation by comparing the data with the threshold values (i.e. $SLA(e_H^{\max})$, $SLA(e_H^{\min})$, $SLA(u_C^{\max})$, $SLA(u_C^{\min})$ (line 5). If the conditions are satisfied, a scaling decision needs to be made and the *DRL-Agent* will be called to provide an action a_i to trigger resource scaling of the VM (line 7).

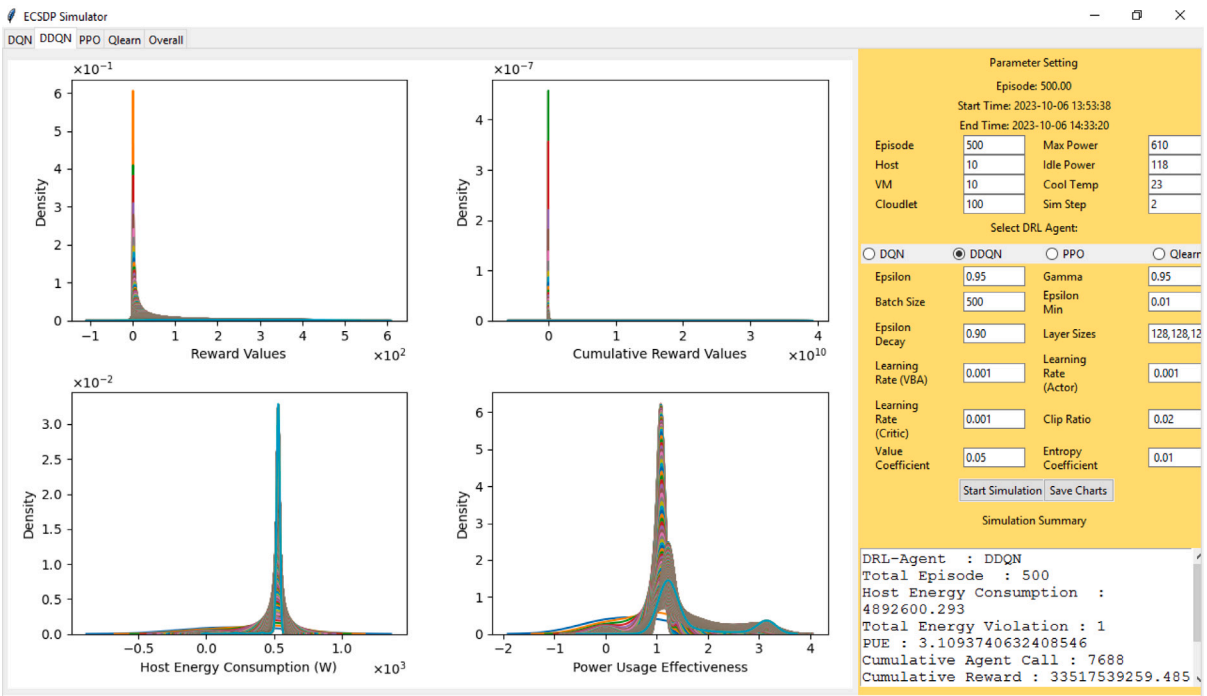
The number of agent calls will be counted for each call and with action a_i, the $p_v \in P_V$ may either be scaled down a_{dw}, scaled up a_{up} or remain the same a_{dn} (lines 8–14). The p_v, r_v and b_v to be de-allocated and allocated are represented with δ^- and δ^+ respectively. When performing a_{dw}, the p_v will be de-allocated from the cloudlet for δ^- which increases the number of available PEs in the VM. Meanwhile, during a_{up} the p_v will maximumly be utilized by the cloudlet for δ^+ which reduces the number of available PEs in the VM. Once the scaling process is completed, the u_c will be re-analyzed and the violation in the VM v_u^{vio} will be counted if there is any (lines 15–16). Finally, when the simulation cycle ends, the total of agent calls, ag, and the total of VM utilization violation, vio will be returned to the *Coordinator* to be displayed as simulation summary by the *GUI*.

6. Implementation and visualization

In this section, we discuss the implementation of the GUI and how to interpret the visualized results.

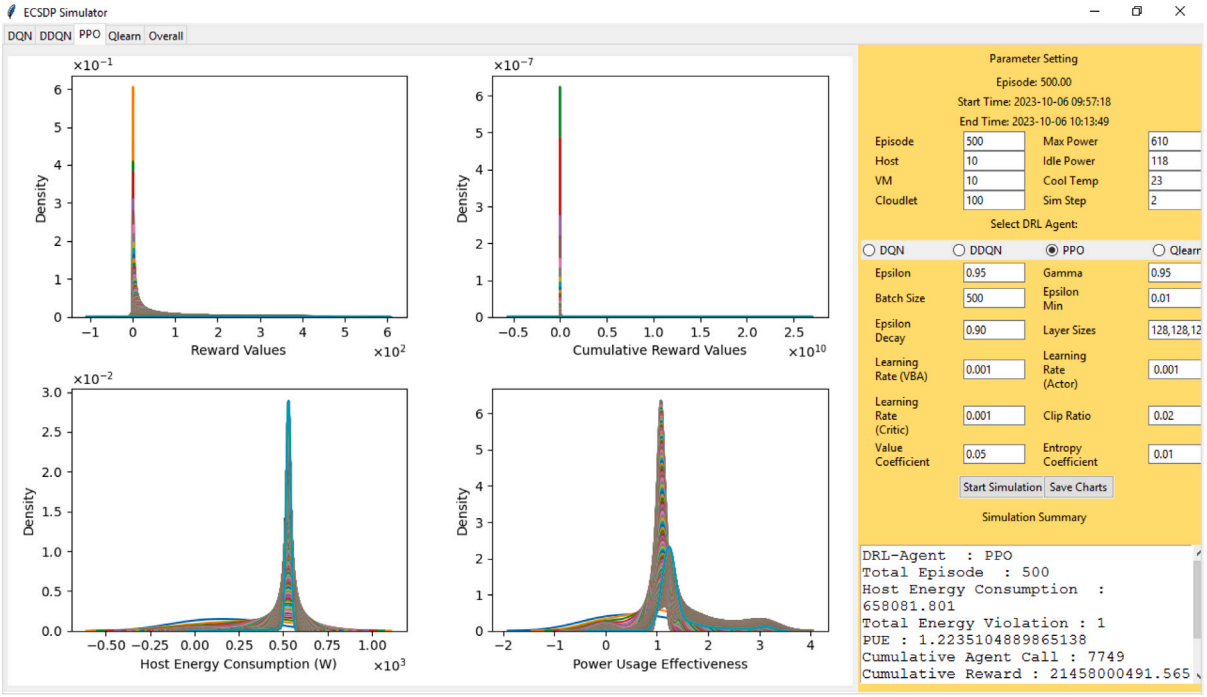


(a) Tab 1 - DQN

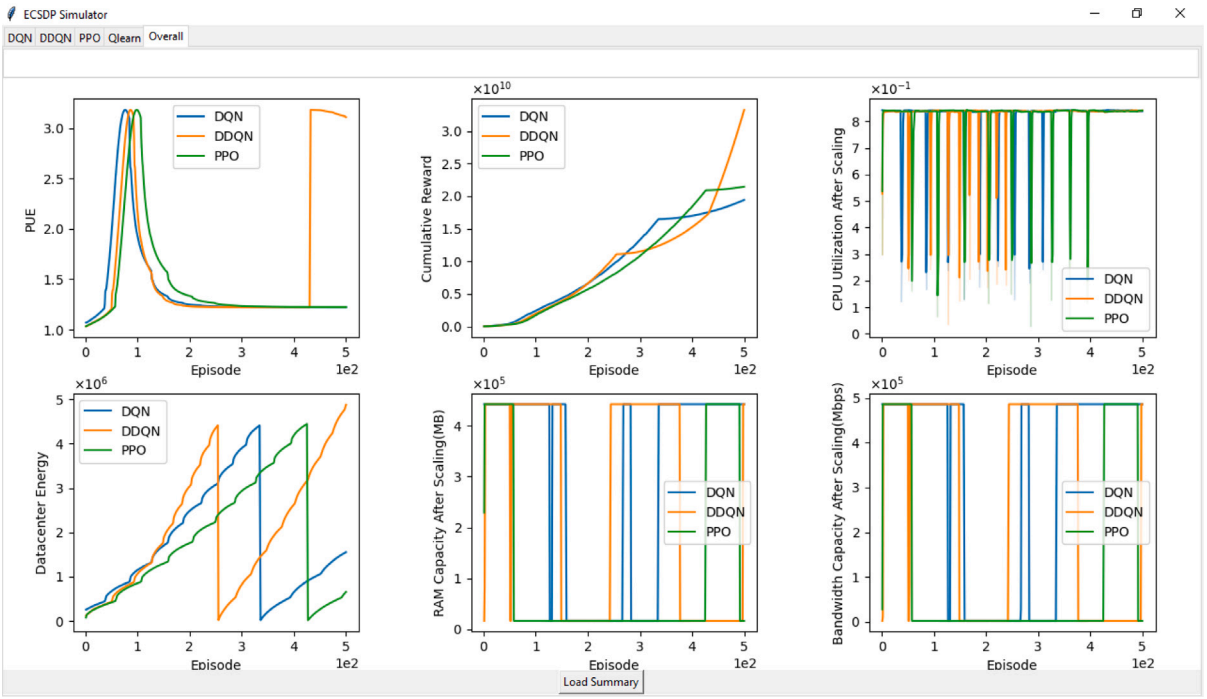


(b) Tab 2 - DDQN

Fig. 5. Simulation output.



(c) Tab 3 - PPO



(d) Tab 4 - Overall

Fig. 5. (*continued*).

6.1. GUI implementation

A GUI has been developed to facilitate user interaction with the simulation. This interaction involves providing initial values for the target environment. To build the GUI, a library called Tkinter was utilized in the development process.

The implementation of tabs has been employed to organize the display of charts that represent the progress of various DRL agents. Each tab is designated for a specific agent: DQN Agent, DDQN Agent, and PPO Agent. Additionally, the final tab is dedicated to presenting a comparative analysis of datacenter energy consumption and PUE across all three agents. Within each tab corresponding to a DRL-Agent, there exists a main section as well as a sidebar. The main section is equipped with four subplots, while the sidebar incorporates components such as a text area, radio buttons, and trigger buttons. These input controls offer users the ability to input simulation parameters and agent's hyperparameters through the text area as well as select a desired DRL-Agent type using the radio buttons. In addition, to ease the user to track the duration of the running simulator, two timestamps (that hold date and time information) represent the start time and the end time of the simulation have been included.

Fig. 5 depicts the output of the three simulations. The upper pair of charts in the main section depicts the agent's progression in terms of rewards and cumulative rewards over the designated training episodes. In contrast, the lower two charts showcase how the agents influence host energy consumption within a datacenter, alongside the PUE of the datacenter itself. Conversely, the final tab encompasses six smaller plots that facilitate a comprehensive comparison of how the three agents' decision-making impacts host energy consumption and datacenter PUE. KDE plots are employed to illustrate the simulation output values captured during runtime. These plots are effective in handling missing data and outliers, offering a clearer visualization of runtime data patterns. Meanwhile, each episode number is represented by a different colored line. Using different colors for each episode can help users distinguish between episodes and observe trends or changes over time, which can be useful for diagnosing training progress. On the other hand, line plots are utilized to provide a post-simulation comparison analysis. These line plots help elucidate the relationship between various parameters and their effects on host energy consumption and datacenter PUE.

6.2. Visualization output

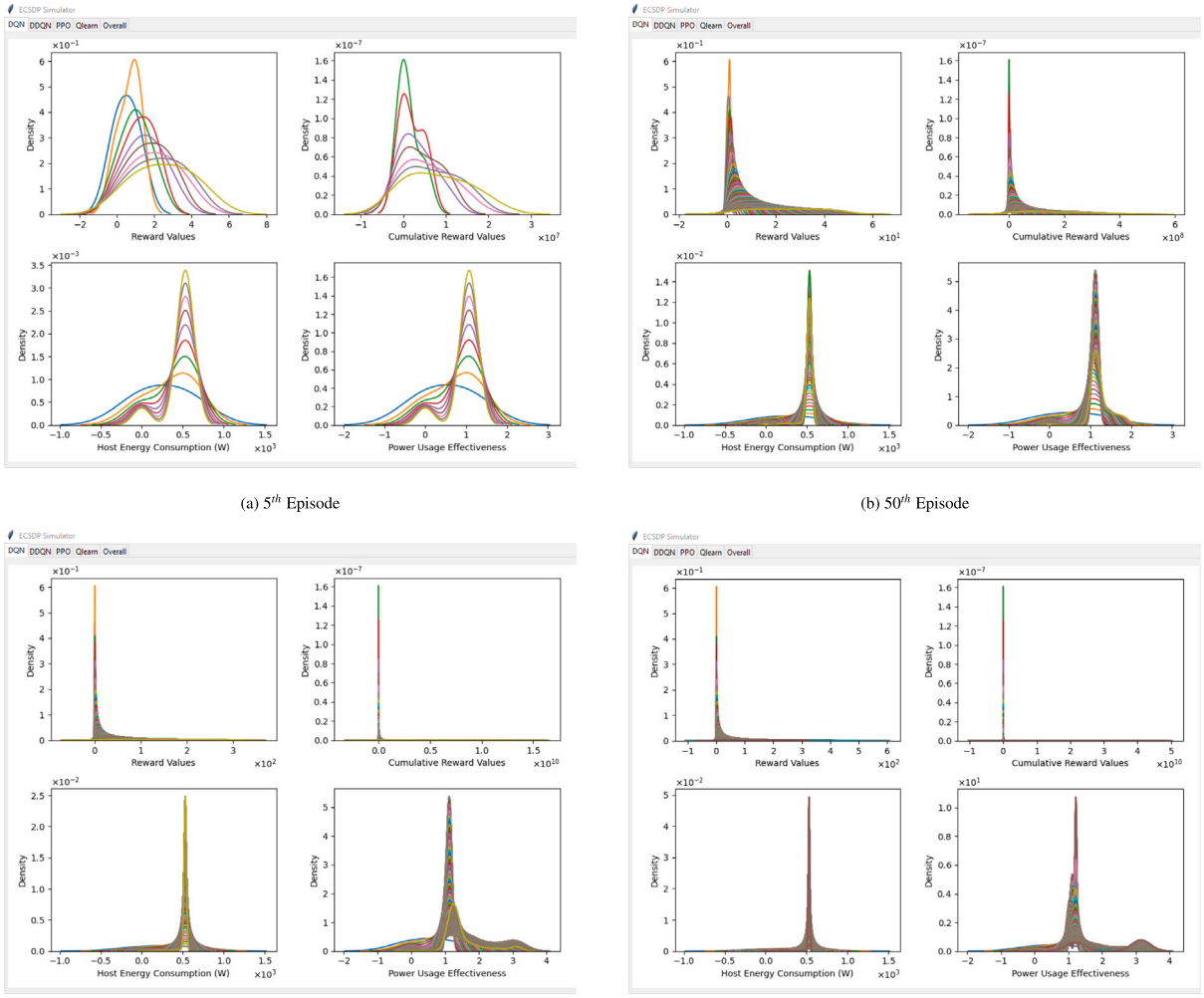
To run the simulator, the Python-based *Coordinator* file must be executed first, followed by the Java main file. Once the *GUI* appears, the user can input initial parameter values and start the simulation. During the simulation, three visualization libraries (i.e. GraphMaker, Matplotlib, and Seaborn) have been used to update the charts. At the end of the simulation, a list of simulation summaries is displayed. The summary provides several key pieces of information:

- *DRL-Agent* - It indicates the type of DRL agent selected for the training process.
- *Host Energy Consumption* - It indicates the final total energy consumed by hosts in a datacenter.
- *Total Energy Violation* - It indicates the cumulative number of events where the total energy consumed by hosts in a datacenter exceeds or below the threshold defined in the SLA contract.
- *PUE* - It indicates the final power usage effectiveness of the data center. The standard value for PUE is 1.0 and above whilst the power usage of the datacenter is considered effective if the PUE value is closer to 1.0.
- *Cumulative Agent Call* - It indicates the cumulative total for the number of agent calls during the simulation.
- *Cumulative Reward* - It indicates the cumulative reward values received by the agent throughout the simulation.
- *Max Reward* - It indicates the maximum reward values received by the agent throughout the simulation.

6.2.1. Runtime data visualization

In this work, runtime data visualization pertains to the real-time visualization of data during the ongoing agent's training. In the example provided, we refer to the charts presented in Fig. 5(a). These charts offer insights into the DQN agent's training progress and the outcomes of specific decisions made over 500 episodes. The initial chart presents the distribution of reward values received by the agent across different episodes, with rewards ranging from −1 to 600, most frequently falling between 0 and 100. Furthermore, the second chart displays the density distribution of cumulative reward values achieved by the agent throughout each episode, covering a wide data distribution from −1 to 50,000,000,000. Additionally, the third chart illustrates the distribution of energy consumption values, exhibiting variance from −1 to 1500 watts. Lastly, the final chart captures the variations in PUE values, with the highest density of data values falling within the range of 1.0 to 2.0.

Furthermore, in Fig. 6, we present a visual representation of the progress made by the DQN agent at different stages of simulation, specifically at the 5th, 50th, 300th, and 500th episodes, with the final episode being 500. The convergence of the DRL-Agent becomes apparent as the variability in cumulative reward data decreases, as observed through the density of the KDE plots. Initially, within the first five episodes (see Figure 6(a)), the data distribution exhibits a wide range, with cumulative rewards spanning from −1 to 30,000,000. However, the agent typically received cumulative rewards within the range of −1 to 10,000,000 during this early stage. To ensure that the agent is making appropriate decisions, we also monitor its energy performance, especially the PUE. A PUE value closer to 1.0 indicates that the agent is making suitable scaling decisions. By the fiftieth episode, the data range expands to approximately 100,000,000, with the density of PUE values primarily concentrated between 1 and 2, as depicted in Figure 6(b). As the simulation progresses to the 500th episode, the cumulative rewards data range continues to increase. Consequently, the plot gradually flattens out, as seen in Figure 6(d). This flat line signifies that the agent's policy and value estimates stabilize, showing minimal variation in subsequent iterations. However, it is essential to note that convergence does not necessarily imply that the agent has found the optimal policy; it may have merely settled into a local minimum.



(a) 5^{th} Episode

(b) 50^{th} Episode

(c) 300^{th} Episode

(d) 500^{th} Episode

Fig. 6. Sample of visualization progress.

6.2.2. Batch data visualization

In this study, batch data visualization refers to the process of visualizing data collected throughout a simulation after its completion. Fig. 5(d) provides a comparative analysis of three DRL agents in the post-simulation phase.

The first chart illustrates that all three agents achieve an optimal PUE value between 1.0 to 1.5 during training in an integrated environment, which is generally considered acceptable in practical scenarios. This chart also highlights the consistency in learning patterns among the agents, with PPO and DQN showing greater consistency compared to DDQN.

Moving to the second chart, displays the cumulative rewards earned by the agents as episodes progress. We observe that DDQN's rewards start to rise rapidly after around the 400th episode, outperforming PPO and DQN. This suggests that DDQN is excelling in the given task or environment, possibly due to a more effective exploration strategy, allowing it to discover better actions and learn more efficiently from experiences.

The third chart examines the relationship between CPU utilization after scaling actions and the number of episodes. CPU utilization ranges from 1% to 80%. Notably, PPO and DDQN's decisions lead to a further reduction in CPU utilization compared to DQN.

In the fourth chart, we analyze the agents' impact on datacenter energy. All agents successfully maintain data center energy between 4,000 kW and 5,000 kW. DDQN is the first to achieve this, followed by DQN and PPO. Additionally, the chart reveals that DDQN occupies the second host in the data center faster than DQN and PPO, as indicated by the restart of datacenter energy in the line chart.

The fifth and sixth charts showcase the RAM and bandwidth capacity allocated for the simulated cloud system over the specified episodes. The system's highest allocated RAM and bandwidth capacity reach approximately 400,000 megabytes and 450,000 megabits per second, respectively.

Table 5
Setting configuration for Experiment 1.

Parameter	Value
Episodes	1000 & 3000
Number of host (H)	10
Number of VMs (V)	8
Number of cloudlets (C)	10
Time step	2
Max Safe temperature (T_{safe})	23 °C
Max Active host energy (e_H^{max})	610 W
Average Idle host energy	118 W

Table 6
Setting configuration for Experiment 2.

Parameter	Setting A	Setting B	Setting C	Setting D
Episodes	500	500	500	500
Number of host (H)	10	10	10	20
Number of VMs (V)	10	15	20	20
Number of cloudlets (C)	100	100	100	100
Time step	2	2	2	2
Max Safe temperature (T_{safe})	23 °C	23 °C	23 °C	23 °C
Max Active host energy (e_H^{max})	610 W	610 W	610 W	610 W
Average Idle host energy	118 W	118 W	118 W	118 W

Table 7
Parameter setting for component.

Component	Parameter	Value
Host	CPU Core (PE)	64
	MIPS	1,000,000
	RAM (MB)	524,288
	Bandwidth (Mbps)	2,000
	ROM (MB)	3,840,000
VM	CPU Core (PE)	8
	MIPS	4,000
	RAM (MB)	8,192
	Bandwidth (Mbps)	10,000
	ROM (MB)	512,000

Overall, these visualizations aim to provide valuable insights into the performance and decision-making of DRL agents within the simulator, aiding users in assessing agent behavior and optimizing datacenter management strategies.

7. Evaluation

In this section, we provide sample simulation setups to evaluate both the simulator's performance and the training of DRL agents within the simulated environment. Following that, we analyze the results of our performance assessments.

7.1. Simulation setup

In order to initiate the simulation, several parameters require specific values. For Experiment 1 and Experiment 2, we have established distinct simulation settings, as outlined in Tables 5 and 6 respectively. Experiment 1 has been crafted to assess the simulator's performance, with a primary focus on evaluating simulation duration. To this end, we conducted simulations for agent training over two different episode counts: 1000 and 3000. We set up the simulation with the following parameters: 10 hosts, each hosting 8 virtual machines (VMs), a total of 10 cloudlets. The active hosts were limited to a maximum energy consumption of 610 watts, and when idle, they consumed 118 watts. Additionally, we maintained a permissible host temperature of 23 °C. Due to cost considerations associated with renting VMs, we capped the maximum number of active VMs at 8. This choice aligns with practical constraints faced by developers and researchers who often cannot afford a large number of VMs for small-scale applications. The host power settings were derived from the SPECpower benchmark report found in [51], where active server energy consumption ranged from 232 watts to 610 watts, while idle servers consumed 118 watts. Consequently, we set the maximum power to 610 watts and the idle power to 118 watts. Furthermore, Experiment 2 was conducted to explore how each DRL agent performs in an integrated environment, while also leveraging data collected and exported from the simulator itself. The settings for Experiment 2 were specifically designed to gauge the impact of various scenarios on the agents' training performance.

Meanwhile, the hosts and VMs had specified computing resources and processing capabilities outlined in Table 7. Furthermore, the cloudlet requirements, including CPU capacity, application file size, and maximum and minimum instruction lengths, were

Table 8
Parameter setting for cloudlet.

Component	Parameter	Value
Cloudlet	CPU Core (PE)	2
	Input File Size (MB)	100
	Output File Size (MB)	100
	Max Instruction Length (MIPS)	1000
	Min Instruction Length (MIPS)	100

Table 9
Parameter setting for DRL-Agent.

Parameter	Value
ϵ	0.95
ϵ_{min}	0.01
ϵ_{decay}	0.90
γ	0.95
α_{policy}	0.001
α_{value_v}	0.001
α_{value_p}	0.001
$clip$	0.2
$\mathbb{C}_{entropy}$	0.01
$\mathbb{C}_{value}$	0.5
$\mathbb{B}_{size}$	500
$\mathbb{L}_{size}$	128, 128, 128

detailed in Table 8. Keeping the maximum and minimum instruction lengths constant ensured that the experiment was completed within the specified deadline, as they directly influenced the cloudlets' execution time.

The DQN and DDQN agents we built upon a feedforward neural network. The hidden layers use the Rectified Linear Unit (ReLU) activation function, which is a common choice for DRL networks. The output layer, which corresponds to action selection, uses the softmax activation function. Softmax is often used when dealing with discrete action spaces as it converts the network's output into a probability distribution over available actions. Furthermore, the PPO agent also implements a feedforward approach for its actor-critic networks. However, the hidden layers use the *tf.tanh* (Hyperbolic Tangent) activation function while there is no specific activation function attached to the output layer.

The *DRL-Agent*'s parameters, as shown in Table 9, were configured to support state transitions. This enabled the agent to observe the current state conditions and select actions leading to the next state. The agents have been trained for 1000 and 3000 episodes with 2 steps/iterations per episode. In addition, these simulations have been run on Windows Server 2016 Datacenter installed with 64 GB RAM and Intel Xeon E (8 processor, 2.40 GHz).

The agent observes a continuous state space, which includes CPU utilization and host energy consumption values. It then compares these values against predefined thresholds. On the other hand, the action space is discrete, comprising three scaling actions: adding, removing, or maintaining (i.e., do nothing) processing elements, RAM capacity, and bandwidth allocated for a specific VM on a host. For this project, users are not allowed to input any actions beyond these three choices. This limitation aims to maintain control and stability while minimizing the PUE of the data center. By restricting the actions to predefined choices, the training process focuses on efficient resource management within a controlled scope, ensuring that the agent makes more meaningful decisions to achieve the primary goal of minimizing host energy consumption and PUE of a data center. Moreover, this approach mitigates unnecessary complexities and risks associated with unrestricted actions.

7.2. Simulation results

In this section, our focus shifts to analyzing the performance of the simulator as well as the performance of the three DRL-Agents when training with the proposed simulation environment.

7.2.1. Performance of simulator

The simulator performance is measured based on time-based metrics, mainly, the time taken to complete the overall simulation and the average duration to complete a single episode. To gauge the time taken by each agent to complete its training, we have tracked a parameter termed start time and end time to compute the duration of each episode. The variation in duration per episode is mainly caused by three factors: the agents' training process, processing and visualizing data at runtime as well as collecting data for batch processing and visualization. Thus, Table 10 presents the details on the average time duration taken to complete the processes.

The agents underwent training for both 1000 and 3000 episodes within the proposed simulated environment. Across 1000 episodes of simulation, the PPO method demonstrated superior performance compared to DQN and DDQN in terms of overall simulation time. It accomplished this in approximately 1 h and 8 min to complete all episodes, with each individual simulation

Table 10
Simulator performance results.

Total episodes	Agent	Overall simulation time (s)	Average duration per episode (s)	Average training time (s)	Average runtime processing & Visualization time (s)
1000	DQN	4150.2	4.0	1.0	3.0
	DDQN	8137.8	8.0	3.0	3.0
	PPO	4093.2	4.0	1.0	2.6
3000	DQN	27 120.0	9.0	1.0	4.0
	DDQN	42 574.8	14.0	4.0	8.0
	PPO	26 547.0	9.0	1.0	7.0

episode taking about 4 s to conclude. As previously mentioned, the average episode duration is influenced by factors such as the average training time, real-time data processing for visualization, and data collection for batch processing.

Regarding the average training duration, both DQN and PPO agents completed their training in approximately 1 s per episode. Conversely, DDQN took roughly 3 s to finish its training process. When it comes to processing and visualizing data in real-time, PPO outperformed DQN and DDQN, taking an average of 2.6 s to perform these tasks. Notably, DQN and DDQN exhibited shorter average data collection times, needing less than 0.1 s to store essential information (including state, action, next state, reward, and completion status) in the minibatch buffer. In contrast, PPO required about 0.1 and 0.3 s to collect its buffer data during 3000 and 1000 episodes, respectively. This is due to PPO's utilization of multiple buffers to store additional information like log probabilities and advantages, beyond what DQN and DDQN store. This extra information is integral to PPO's implementation strategy as an actor-critic approach. Additionally, DDQN's longer training duration compared to DQN and PPO can be attributed to its use of two separate networks: one for selecting the optimal action based on the highest Q-value, and another for addressing Q-value overestimation.

These insights apply similarly to the second simulation setting, which involved 3000 total episodes. Upon careful observation, it can be concluded that the training of these agents within this simulator aligns with expected behavior.

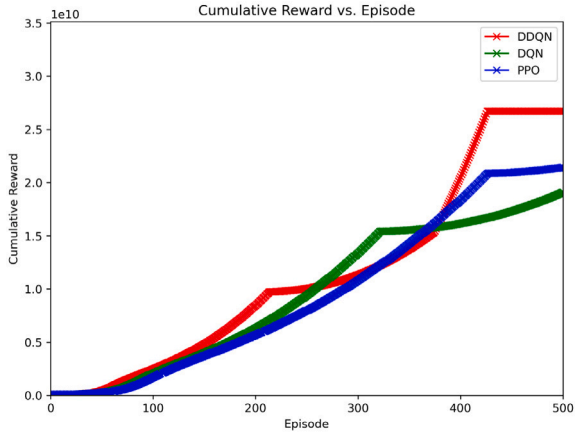
7.2.2. Performance of DRL-Agents

In this section, we discuss the performance of DRL agents involving three main aspects: convergence speed, selection of scaling action, and number of executed tasks based on the termination status. To perform this analysis we utilized the data collected using the proposed simulator. The simulator enables the users to export the data in CSV format which gives them flexibility to visually present the data using their own approaches. Before visualizing the data into charts, we performed basic data processing steps including stripping whitespace from the column title to ensure consistent and clean column names, making it easier to access and manipulate data without encountering unexpected errors due to spacing inconsistencies. We also applied winsorization which helps mitigate the impact of extreme outliers on statistical analysis and visualization by capping or flooring extreme values, making the data more robust.

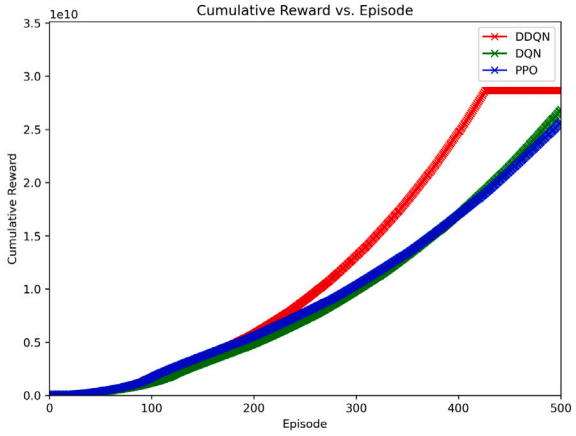
Fig. 7 illustrates how quickly each DRL agent learns, based on the rewards it receives as it goes through different episodes. Our observations show that DDQN performs better than both DQN and PPO in most scenarios, except for Setting D, where we increased the resources for the simulated cloud system. In this case, the agent needed more episodes to figure out the best actions for convergence. When we give the agent more resources, it can encounter more variability in the results of its actions. This variability makes it trickier for the agent to tell which actions are good or bad, which can slow down its learning process.

Fig. 8 illustrates the decision outputs of different agents in various settings. In Setting A, we observe distinct scaling behaviors. The PPO agent tends to scale down frequently and scale up infrequently, as it is highly sensitive to the PUE-based reward signal. This aligns with the goal of achieving a PUE value of 1.0 by reducing active resources and, consequently, energy consumption. In the same setting, the DQN agent also frequently scales down but scales up more often than the PPO. This indicates DQN's cautious approach to ensure resource availability while aiming to minimize energy consumption. DDQN follows a similar pattern, favoring scaling down and scaling up in its pursuit of optimal PUE. In Setting B, which is similar to Setting A but with 15 VMs per host, PPO scales down even less and rarely scales up. DQN's behavior remains similar to PPO but with a slightly higher scaling-up frequency, reflecting its responsiveness to resource demands. However, DDQN exhibits a strong preference for scaling up, possibly overcompensating for the increased VM count. Setting C, with 20 VMs per host, maintains PPO's cautious scaling approach. DQN's pattern aligns with PPO but with more scaling down and less scaling up. DDQN still leans towards scaling down, aimed to optimize the existing resources. In Setting D, with 20 hosts and 20 VMs each, PPO scales down less and scales up more, indicating its willingness to accommodate resource demands while prioritizing energy efficiency. DQN scales down significantly more and scales up less, emphasizing energy efficiency over resource availability. Conversely, DDQN reverses this pattern, scaling down less and scaling up more, prioritizing resource provisioning.

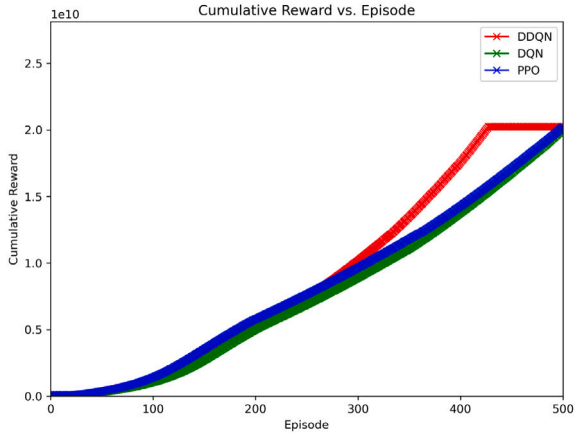
The consequences of these scaling decisions can be observed based on the number of executed tasks shown in Fig. 9. The SUCCESS and FAIL status represents the tasks by the cloudlets that successfully completed or failed to be completed within a specified duration. In this case, the duration referred to the number of training episodes (i.e. 500 episodes). For Setting A, the decision made by DQN has outperformed the decision from PPO and DDQN. This is because the simulated cloud system occupied with the DQN-based autoscaler provides enough resources to process the tasks. For Setting B, the decision made by DDQN surpassed the other two agents' decision. Although in previous observation, it seems like the decision made by DDQN will lead to a bad decision as it will



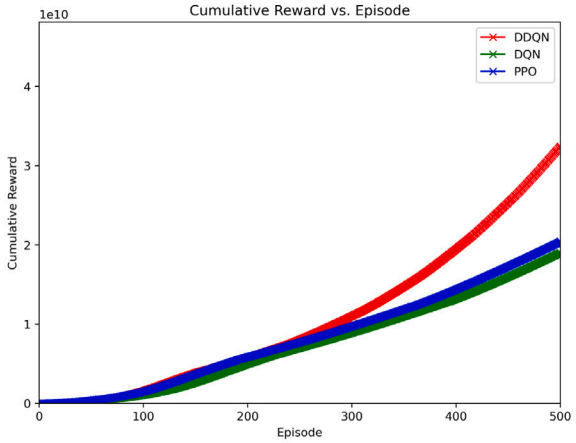
(a) Setting A



(b) Setting B



(c) Setting C



(d) Setting D

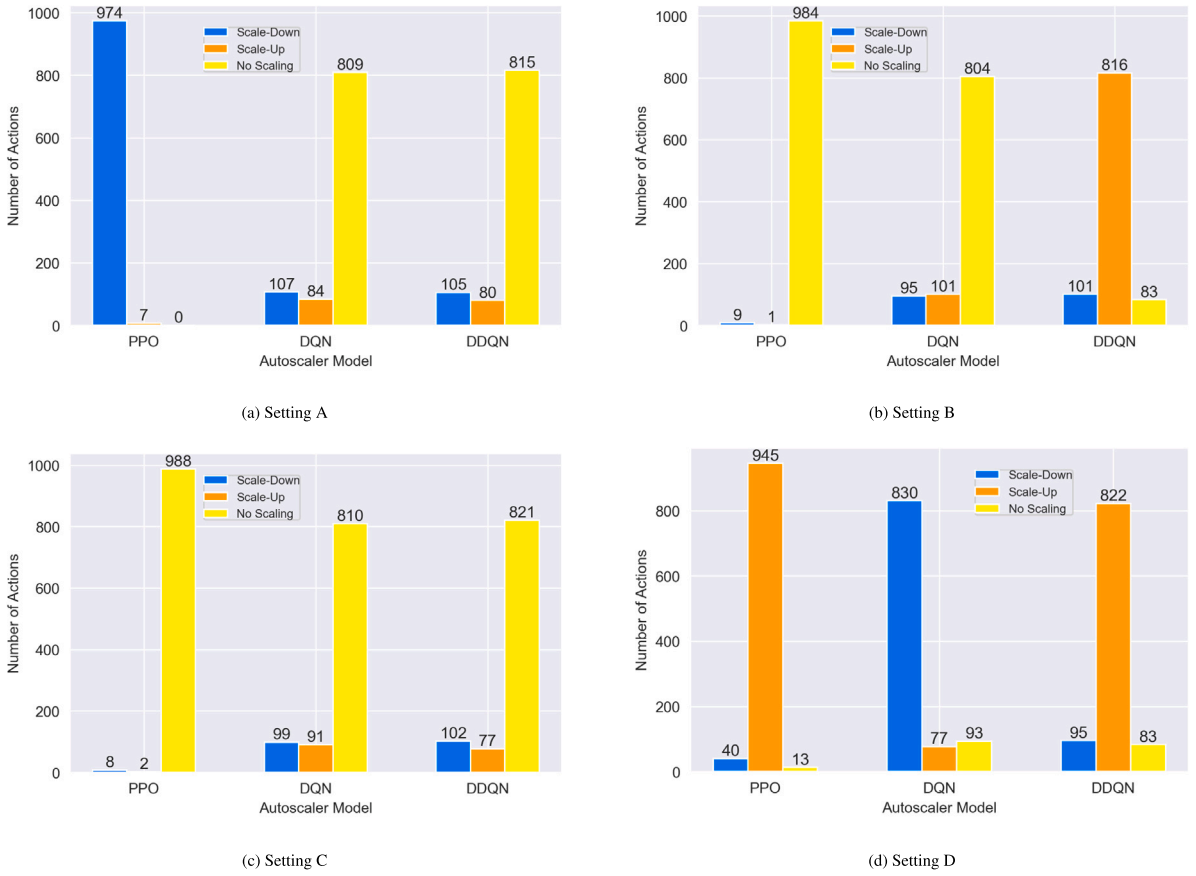
Fig. 7. Convergence of DRL-Agents based on cumulative rewards.

increase energy consumption, however, the decision is still relevant in order to support the processing of the task. In contrast with Setting C, DDQN is able to adapt to the increase in VMs number. Instead of adding more resources, it optimized resource utilization. As a result, it is able to successfully complete more tasks than DQN and PPO. For Setting D, PPO outperforms DQN and DDQN as it is able to maximize the number of SUCCESS tasks and minimize FAIL tasks better.

Each of the agents demonstrates optimal performance with respect to specific objectives: energy efficiency, task completion, or a balance between both. DDQN stands out as a robust choice when the primary concern is energy efficiency, prioritizing resource utilization optimization. It particularly excels in scenarios where resource constraints are relatively relaxed. Conversely, if the primary goal is to maximize task completion, PPO emerges as the preferred option. PPO exhibits exceptional performance in situations where achieving a high number of successful tasks is imperative. Its cautious and accommodating nature makes it well-suited for this purpose. In cases where striking a balance between energy efficiency and task completion is the foremost concern, DQN proves to be the optimal choice. DQN adeptly manages both resource availability and energy efficiency, making it a sensible selection for those seeking a compromise between these two objectives. Moreover, DQN's versatility enables it to perform admirably across a range of settings, further solidifying its position as a well-rounded option.

8. Conclusion

This paper presents a simulation-based approach to evaluate the performance of deep reinforcement learning (DRL) in energy-driven cloud scaling. The proposed method combines the Python-based DRL framework and the extensible Java-based cloud simulator, CloudSim Plus, using the Py4J gateway framework to facilitate interaction between the two environments. The training environment is built using OpenAI Gym, while the target environment is implemented in CloudSim Plus. The simulation method employs a DRL-Agent that learns the current state of the cloud scaling environment and makes scaling decisions based on PUE



(a) Setting A

(b) Setting B

(c) Setting C

(d) Setting D

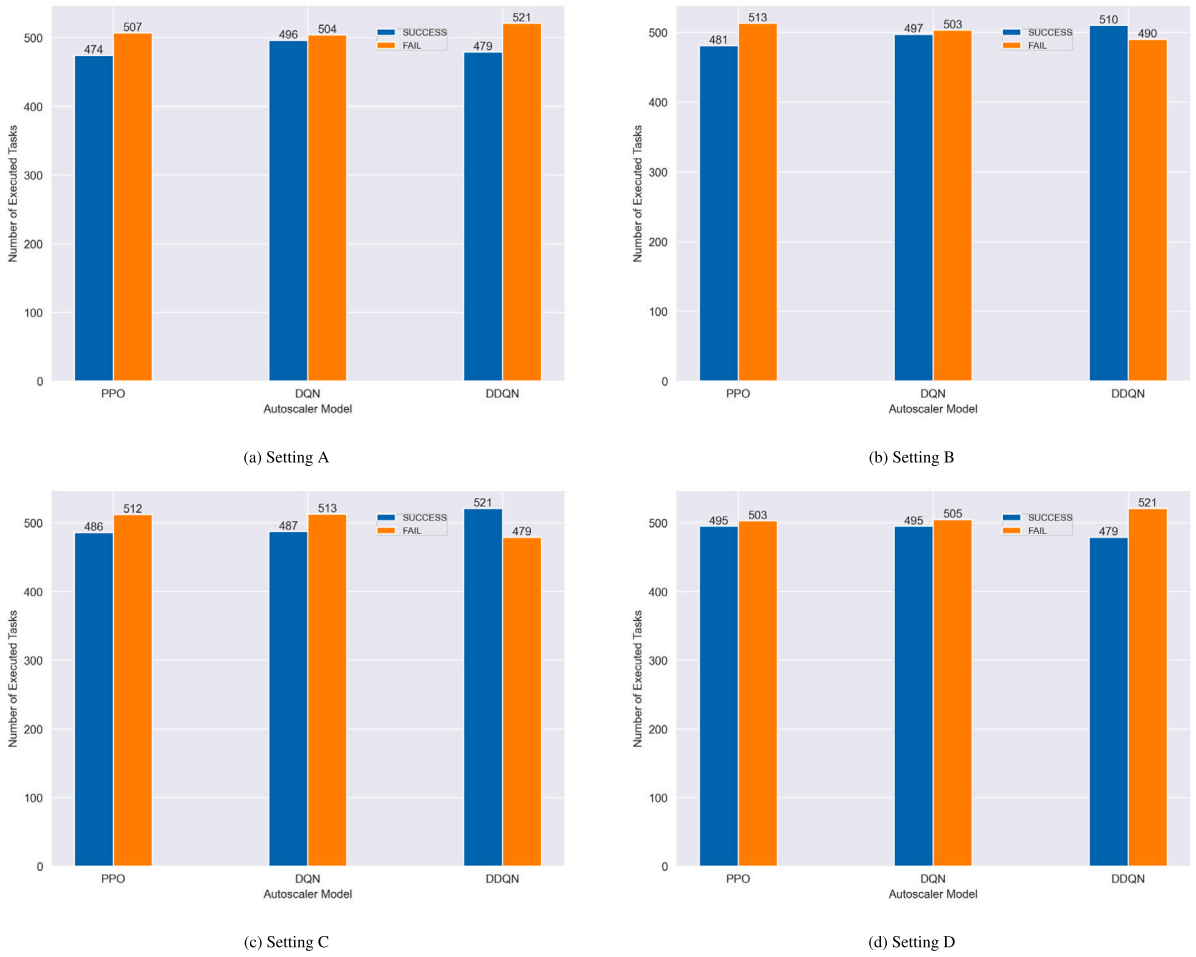
Fig. 8. Selection of scaling actions by DRL-Agents.

to achieve energy efficiency. The graphical outputs of the simulator provide valuable insights into the DRL-Agent's performance across various configuration settings. Furthermore, it specifically provides a group of charts that compare the impact of the agents' decision-making process on the host energy consumption, the PUE of a data center, and the targeted computing resources.

This simulation approach offers several advantages. Firstly, it enables continuous learning and adaptation of the DRL agent's brain, allowing for faster convergence and improved training efficiency. Secondly, it facilitates real-time evaluation of the agent's performance in a more realistic setting, providing valuable insights into its behavior and ability to handle dynamic and unpredictable scenarios. Thirdly, it helps identify potential weaknesses or vulnerabilities that may not be apparent in the training environment alone. Fourthly, it supports transfer learning, allowing knowledge gained in one environment to be effectively applied to another, leading to improved performance and adaptability. Lastly, it allows for rigorous validation and verification of the agent's performance against predefined metrics and requirements.

Future work could improve the simulator by incorporating multi-objective support and other scaling dimensions. Multi-objective functions can be integrated to analyze trade-offs between energy factors and other performance metrics. The simulator can be extended to include horizontal scaling as an additional dimension, further exploring its impact on PUE. Additionally, the capability for comparative analysis will allow users to evaluate the effectiveness of different DRL approaches for simulating energy-driven cloud scaling. Besides that, we also acknowledge the need to address the optimization concern. So that, these future improvements will contribute to a more comprehensive evaluation platform for DRL-based cloud scaling research.

In summary, the proposed simulation approach addresses the limitations of expensive and non-repeatable real-world training environments, providing researchers and developers with a powerful tool to explore and evaluate DRL performance in energy-driven cloud scaling scenarios. By leveraging Python-based deep learning frameworks and the extensibility of Java-based cloud simulators, this approach contributes to the evaluation perspective of DRL in cloud scaling, ultimately advancing the field of intelligent cloud resource management.



(a) Setting A

(b) Setting B

(c) Setting C

(d) Setting D

Fig. 9. Number of tasks executed based on termination status.

Data availability

Data will be made available on request.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used ChatGPT in order to improve sentence structure and fix grammatical issues. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Acknowledgments

Azlan Ismail acknowledges the support of the Fundamental Research Grant Scheme, FRGS/1/2018/ICT01/UITM/02/3, funded by Ministry of Education Malaysia. We also acknowledge the contribution of Khairul Azzim Long Ahmad Adli in implementing some parts of the simulator of the earlier version.

References

[1] A.T. Makaratzis, K.M. Giannoutakis, D. Tzovaras, Energy modeling in cloud simulation frameworks, Future Gener. Comput. Syst. 79 (2018) 715–725.
[2] N. Mansouri, R. Ghafari, B.M.H. Zade, Cloud computing simulators: A comprehensive review, Simul. Model. Pract. Theory 104 (2020) 102144.
[3] A. Ismail, Energy-driven cloud simulation: existing surveys, simulation supports, impacts and challenges, Cluster Comput. (2020) 1–17.
[4] C. Bitsakos, I. Konstantinou, N. Koziris, DERP: A deep reinforcement learning cloud system for elastic resource provisioning, in: 2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, 2018, pp. 21–29.
[5] Z. Wang, C. Gwon, T. Oates, A. Iezzi, Automated cloud provisioning on aws using deep reinforcement learning, 2017, arXiv preprint arXiv:1709.04305.

[6] S. Kardani-Moghaddam, R. Buyya, K. Ramamohanarao, ADRL: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds, IEEE Trans. Parallel Distrib. Syst. 32 (3) (2020) 514–526.
[7] M. Cheng, J. Li, S. Nazarian, DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers, in: 2018 23rd Asia and South Pacific Design Automation Conference, ASP-DAC, 2018, pp. 129–134.
[8] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep Q-learning, Inform. Sci. 512 (2020) 1170–1191.
[9] M. Dayarathna, Y. Wen, R. Fan, Data center energy consumption modeling: A survey, IEEE Commun. Surv. Tutor. 18 (1) (2016) 732–794.
[10] M. Zakarya, L. Gillam, Energy efficient computing, clusters, grids and clouds: A taxonomy and survey, Sustain. Comput. Inform. Syst. 14 (2017) 13–33.
[11] M. Ghobaei-Arani, S. Jabbehdari, M.A. Pourmina, An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach, Future Gener. Comput. Syst. 78 (2018) 191–210.
[12] R.N. Calheiros, R. Ranjan, C.A.F. De Rose, R. Buyya, Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services, 2009, arXiv preprint arXiv:0903.2525.
[13] M.C. Silva Filho, R.L. Oliveira, C.C. Monteiro, P.R.M. Inácio, M.M. Freire, CloudSim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness, in: IEEE International Symposium on Integrated Network Management, IM2017, 2017, pp. 400–406.
[14] Y. Fenjiro, H. Benbrahim, Deep reinforcement learning overview of the state of the art, J. Autom. Mob. Robot. Intell. Syst. 12 (2018).
[15] C. Qu, R.N. Calheiros, R. Buyya, Auto-scaling web applications in clouds: A taxonomy and survey, ACM Comput. Surv. 51 (4) (2018).
[16] Y. Garí, D.A. Monge, E. Pacini, C. Mateos, C. García Garino, Reinforcement learning-based application autoscaling in the cloud: A survey, Eng. Appl. Artif. Intell. 102 (2021) 104288.
[17] T. Lorido-Botran, J. Miguel-Alonso, J.A. Lozano, A review of auto-scaling techniques for elastic applications in cloud environments, Int. J. Grid Util. Comput. 12 (4) (2014) 559–592.
[18] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, I. Truck, Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow, in: ICAS 2011: The Seventh International Conference on Autonomic and Autonomous Systems, 2011, pp. 67–74.
[19] T. Veni, S.M.S. Bhanu, Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach, Int. J. Big Data Intell. 3 (3) (2016) 145–153.
[20] H. Arabnejad, C. Pahl, P. Jamshidi, G. Estrada, A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling, in: Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017, 2017, pp. 64–73.
[21] S. Horovitz, Y. Arian, Efficient cloud auto-scaling with SLA objective using Q-learning, in: Proceedings - 2018 IEEE 6th International Conference on Future Internet of Things and Cloud, FiCloud 2018, IEEE, 2018, pp. 85–92.
[22] J.V. Bibal Benifa, D. Dejey, RLPAS: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment, Mob. Netw. Appl. 24 (2019) 1348–1363.
[23] Y. Wei, D. Kudenko, S. Liu, L. Pan, L. Wu, X. Meng, A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment, Math. Probl. Eng. 2019 (2019).
[24] S.M.R. Nouri, H. Li, S. Venugopal, W. Guo, M.Y. He, W. Tian, Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications, Future Gener. Comput. Syst. 94 (2019) 765–780.
[25] L. Schuler, S. Jamil, N. Kühl, AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments, in: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing, CCGrid, 2021, pp. 804–811.
[26] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, Y. Wang, A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, 2017, pp. 372–382.
[27] I.S. Moreno, J. Xu, Customer-aware resource overallocation to improve energy efficiency in realtime cloud computing data centers, in: 2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA, IEEE, 2011, pp. 1–8.
[28] S.K. Tesfatsion, E. Wadbro, J. Tordsson, A combined frequency scaling and application elasticity approach for energy-efficient cloud computing, Sustain. Comput. Inform. Syst. 4 (4) (2014) 205–214.
[29] A. Paya, D.C. Marinescu, Energy-Aware load balancing and application scaling for the cloud ecosystem, IEEE Trans. Cloud Comput. 5 (1) (2017) 15–27.
[30] M.L. Berkane, M. Boufaida, N.E.H. Bouzerzour, Modelling elastic scaling of cloud with energy-efficiency: Application to smart-university, J. King Saud Univ. Comput. Inf. Sci. (2020).
[31] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50.
[32] K. Czarnecki, S. Helsen, U. Eisenecker, Formalizing cardinality-based feature models and their specialization, Softw. Process Improv. Pract. 10 (1) (2005) 7–29.
[33] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (1) (2011) 23–50.
[34] D. Kliazovich, P. Bouvry, S.U. Khan, GreenCloud: A packet-level simulator of energy-aware cloud computing data centers, J. Supercomput. 62 (3) (2012) 1263–1283.
[35] B. Wickremasinghe, R.N. Calheiros, R. Buyya, CloudAnalyst: A CloudSim-Based visual modeller for analysing cloud computing environments and applications, in: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, IEEE, 2010, pp. 446–452.
[36] D. Fernández-Cerero, A. Jakóbik, A. Fernández-Montes, J. Kołodziej, GAME-SCORE: Game-based energy-aware cloud scheduler and simulator for computational clouds, Simul. Model. Pract. Theory 93 (2019) 3–20.
[37] I.K. Kim, W. Wang, M. Humphrey, PICS: A public IaaS cloud simulator, in: 2015 IEEE 8th International Conference on Cloud Computing, 2015, pp. 211–220.
[38] T. Vondra, J. Šedivý, Cloud autoscaling simulation based on queueing network model, Simul. Model. Pract. Theory 70 (2017) 83–100.
[39] M.S. Aslanpour, A.N. Toosi, J. Taheri, R. Gaire, AutoScaleSim: A simulation toolkit for auto-scaling web applications in clouds, Simul. Model. Pract. Theory 108 (2021) 102245.
[40] N.J. Gunther, Analyzing Computer System Performance with Perl::PDQ, Springer Science & Business Media, 2011.
[41] C. Belady, A. Rawson, J. Pflueger, T. Cader, Green Grid Data Center Power Efficiency Metrics: PUE and DCIE, The Green Grid, White Paper 6, 2008.
[42] T. Renugadevi, K. Geetha, N. Prabaharan, P. Siano, Carbon-efficient virtual machine placement based on dynamic voltage frequency scaling in geo-distributed cloud data centers, Appl. Sci. 10 (8) (2020) 2701.
[43] E. Akanksha, N. Sharma, K. Gulati, et al., Review on reinforcement learning, research evolution and scope of application, in: 2021 5th International Conference on Computing Methodologies and Communication, ICCMC, IEEE, 2021, pp. 1416–1423.
[44] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[45] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, W. Dabney, Revisiting fundamentals of experience replay, 2020, arXiv preprint arXiv:2007.06700.
[46] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, 2016, arXiv preprint arXiv:1606.01540.
[47] Team, Keras, Keras.io, 2020, Keras, URL: https://keras.io/. (Accessed 21 January 2022).
[48] OpenAI, 2015-2022, Gym, URL: https://gym.openai.com/. (Accessed 21 January 2022).

[49] M.C. Silva Filho, R.L. Oliveira, C.C. Monteiro, P.R.M. Inácio, M.M. Freire, 2017, CloudSim Plus, URL: https://cloudsimplus.org/. (Accessed 21 January 2022).
[50] B. Dagenais, Welcome to Py4J — Py4J, 2009-2015, https://www.py4j.org/. (Accessed 22 November 2021).
[51] Standard Performance Evaluation Corporation, Team, Benchmark Result Summary (Lenovo Global Technology Think System SR645 V3), Lenovo Global Technology, 2023, SPECpower_ssj2008, URL: https://www.spec.org/power_ssj2008/results/res2023q3/power_ssj2008-20230523-01265.html.