Article

Skyline-Enhanced Deep Reinforcement Learning Approach for Energy-Efficient and QoS-Guaranteed Multi-Cloud Service Composition

Wenhao Ma and Hongzhen Xu

1 School of Information Engineering, East China University of Technology, Nanchang 330013, China; winstonmwh@foxmail.com
2 School of Software, East China University of Technology, Nanchang 330013, China
3 Jiangxi Key Laboratory of Cybersecurity Intelligent Perception, Nanchang 330013, China
* Correspondence: xuhz@ecut.edu.cn

Abstract: Cloud computing has experienced rapid growth in recent years and has become a critical computing paradigm. Combining multiple cloud services to satisfy complex user requirements has become a research hotspot in cloud computing. Service composition in multi-cloud environments is characterized by high energy consumption, which brings attention to the importance of energy consumption in cross-cloud service composition. Nonetheless, prior research has mainly focused on finding a service composition that maximizes the quality of service (QoS) and overlooks the energy consumption generated during service invocation. Additionally, the dynamic nature of multi-cloud environments challenges the adaptability and scalability of cloud service composition methods. Therefore, we propose the skyline-enhanced deep reinforcement learning approach (SkyDRL) to address these challenges. Our approach defines an energy consumption model for cloud service composition in multi-cloud environments. The branch and bound skyline algorithm is leveraged to reduce the search space and training time. Additionally, we enhance the basic deep Q-network (DQN) algorithm by incorporating double DQN to address the overestimation problem, incorporating Dueling Network and Prioritized Experience Replay to speed up training and improve stability. We evaluate our proposed method using comparative experiments with existing methods. Our results demonstrate that our approach effectively reduces energy consumption in cloud service composition while maintaining good adaptability and scalability in service composition problems. According to the experimental results, our approach outperforms the existing approaches by demonstrating energy savings ranging from 8% to 35%.

Keywords: cloud service composition; multi-cloud; deep reinforcement learning; skyline; energy consumption; QoS aware

1. Introduction

Cloud computing is an elastic and scalable model of providing Information Technology (IT) services to consumers, where they can conveniently access resources, platforms, or software on demand [1]. It has become a critical computing paradigm in recent years because it offers virtualized resources, reduced capital outflow, and improved computing efficiency without revealing platform and implementation details [2]. As a result, cloud service providers can offer parallel computing resources and achieve large-scale storage at a relatively low cost [3]. The global cloud computing market has been steadily expanding since 2022, and cloud-based applications are gaining popularity.

In the cloud environment, users are provided with cloud services based on their requirements. However, existing services often cannot fully satisfy users’ needs, so composite services that combine several services are commonly used to provide value-added functionality. Service composition is a fundamental concept in the field of service-oriented
computing. It involves the integration of multiple services to provide a comprehensive solution that meets the specific needs of customers [4]. These services may originate from a single provider or multiple providers. The purpose of service composition is to provide a more complete and personalized service to better meet the needs of the customers. In practice, service composition takes various forms that cater to different customer needs. For example, it may involve selecting and combining the most appropriate services to create a solution. Alternatively, it may entail providing customers with customized service packages or plans that correspond to their unique preferences and constraints. Personalized recommendation services based on data analytics and behavioral analysis are also becoming increasingly prevalent in modern service composition systems. With the rapid development of cloud computing, numerous cloud service providers offer many services with similar functions but different quality of service (QoS) attributes. QoS is a critical attribute in the service composition process, serving as a means to assess the quality of service delivery by service providers. Typically, QoS pertains to the non-functional attributes of a service, including latency, throughput, reliability, security, and availability, among others [2]. These metrics can accurately describe and measure a service’s performance and ultimately impact the user experience. Therefore, selecting appropriate cloud services and combining them to meet users’ QoS expectations, a process known as QoS-aware cloud service composition (QoS-CSC) [4], has become a significant issue for cloud service providers. Considering the significance and necessity of QoS attributes in multi-cloud service composition, it can provide users with more reliable and efficient services. By taking into account QoS attributes, service composition can provide better quality assurance and meet the users’ needs, thus improving user satisfaction. QoS-CSC is a combinatorial optimization problem. However, due to fierce competition in the market, technical similarity, customer needs, and the growing number of similar cloud services, QoS-CSC has become an NP-hard problem. Over the past decade, QoS-CSC has become a crucial research topic in the academic community. In QoS-CSC, users’ needs are broken down into multiple subtasks, and suitable cloud services are selected from the cloud service pool for each subtask. These services are then combined to create a composite service that meets users’ needs while obtaining the optimal QoS value. However, finding an optimal cloud service composition has become particularly challenging with the proliferation of clouds, the entry of more service providers to the market, and the dynamic nature of multi-cloud environments that cause cloud services to change with time, season, or other complex factors.

Most current research on cloud service composition restricts the cloud environment to a single cloud, assuming that all the cloud services in the composition come from a single cloud environment. However, multi-cloud environments have become a trend because no single cloud provider can establish its data centers in all possible locations worldwide. Moreover, complex user requirements are likely to involve multiple cloud service providers distributed across different clouds [5]. As the number of clouds increases, the availability of better and more diverse services also increases, which may be missed if only one cloud provider is used. Limiting cloud service composition to a single cloud environment hampers the potential to access the best available services offered by other cloud providers. Furthermore, some consumers require high network transmission speeds, making clouds with excellent data transfer rates more suitable for their needs. Other consumers prioritize data privacy and security and may exclude untrusted clouds. Therefore, considering multi-cloud environments is essential when researching cloud service composition, as it provides better support for consumer needs.

Cloud service composition in a multi-cloud environment presents several challenges. For example, multi-cloud environments often generate more data transmission and computation, which increases energy consumption and communication costs for cross-cloud service composition. Specifically, composite services involving multiple clouds will inevitably result in longer response times and higher network bandwidth usage. This is because invoking multiple services across clouds requires a large amount of data transmission and exchange, causing increased energy consumption [5]. Given the current global
environmental situation, reducing carbon dioxide emissions and energy consumption is crucial to promoting the sustainable development of cloud computing and maintaining an ecological environment. Therefore, reducing the energy consumption of cloud service composition is a significant challenge when composing services in a multi-cloud environment. Moreover, the performance, price, and energy consumption of cloud services can frequently fluctuate in a dynamic multi-cloud environment. For example, a service that currently offers the best QoS performance may not remain optimal in the future. Additionally, the QoS attribute value may change, or the cloud service may suddenly become unavailable during runtime. These variables can have an impact on applications and services in a multi-cloud environment, necessitating the need to consider how to adapt to this dynamic nature. To address this, algorithms must have good adaptability and continuously adjust strategies based on environmental changes.

Researchers have proposed various methods to address the cloud service composition problem, including traditional methods, such as 0–1 programming, and metaheuristic methods, such as particle swarm optimization (PSO) [6] and genetic algorithm (GA) [7]. While these approaches have yielded satisfactory results, they lack adaptability in dynamic environments. In other words, when changes occur in the environment, these algorithms may require adjustment or even redesign. Additionally, these methods are not suitable for large-scale space scenarios, which can be challenging given the large number and dimensions of services involved. Recently, deep reinforcement learning-based (DRL-based) methods have gained attention for web service composition [8]. DRL-based methods offer a quick adjustment of strategies based on the current scenario, provide the adaptive updating of weights, and dynamically select optimal service composition solutions. DRL-based methods can adjust the network structure and other means to deal with different scales and types of problems. In contrast, metaheuristic algorithms require manual feature representation and rules, knowledge and experience guidance, and tedious tuning processes and are not ideal for large-scale and high-dimensional optimization problems. Metaheuristic algorithms are often limited by current search spaces and do not readily obtain globally optimal solutions. Given the dynamic nature and a large number of services in a multi-cloud environment, deep reinforcement learning offers better application value. By continuously learning and optimizing, deep reinforcement learning can provide users with higher-quality service composition solutions.

There are several issues with current research on cloud service composition: (1) although some researchers have used DRL-based methods for service composition in web services [8], DRL is rarely used in cloud service composition in a multi-cloud environment; (2) when using deep reinforcement learning for service composition, most existing research has not considered energy consumption. Researchers typically use the characteristics of DRL to find the best service composition plan while overcoming adaptability issues but do not consider service energy consumption; (3) the increasing number of cloud services increases the search space significantly, which substantially increases the computation time of the algorithm. However, few researchers have optimized the time complexity of the deep reinforcement learning algorithm.

To address the energy-aware cloud service composition problem in a multi-cloud environment, we propose the skyline-enhanced deep reinforcement learning approach (SkyDRL). In this algorithm, we consider both energy consumption and dynamic environmental issues in a multi-cloud environment. We added energy consumption terms to both the objective and reward functions to ensure that the service composition has excellent QoS performance while reducing energy usage and achieving a balance between QoS and energy consumption. As previously discussed, while energy consumption is a significant factor in evaluating cloud services, optimizing QoS remains the primary focus of service composition. A singular focus on reducing energy consumption without considering QoS attributes can lead to a decline in service quality and user satisfaction. Therefore, by balancing QoS with energy consumption, data center energy consumption can be further reduced, and resource utilization can be improved while ensuring that users obtain the
required service quality and reliability, thus enhancing user satisfaction. We leverage the advantages of deep reinforcement learning’s adaptive learning policy to solve the dynamic environment problem. At the same time, we combined the skyline algorithm with deep reinforcement learning to effectively narrow the search space, reduce deep reinforcement learning training time, and improve algorithm execution efficiency.

Overall, the main contributions of this paper are as follows:

1. We propose a novel method called SkyDRL based on deep reinforcement learning models to solve the cloud service composition problem in a multi-cloud environment, providing a new solution to this problem;
2. In multi-cloud environments, service composition generates more energy consumption. In addition, cloud services are usually not static. Therefore, we mainly focus on the energy consumption generated during the cloud service composition process in multi-cloud environments and consider dealing with dynamic cloud environments rather than static ones. We propose an energy-efficient QoS-guaranteed cloud service composition algorithm;
3. Additionally, a large amount of data can lead to excessively long training times and increased optimization difficulty. Therefore, we combined the branch and bound skyline algorithm with deep reinforcement learning, using the skyline algorithm to reduce the search space and computational complexity;
4. We compared our proposed method with other methods in experiments to demonstrate its effectiveness.

In the upcoming sections, we will review related work in this research area, including traditional research on cloud service composition and recent machine learning-based service composition research. Section 3 will describe and model the problem studied in this paper. Section 4 will introduce the method used in this paper. Section 5 will compare the proposed method with other methods in experiments and present experimental results. Section 6 will summarize this entire paper and provide insights into future research directions.

2. Related Work

This section will briefly review related research on cloud service composition, including non-heuristic algorithms, heuristic algorithms, and machine learning-based algorithms.

2.1. Cloud Service Composition Based on Non-Heuristic Algorithms

Non-heuristic algorithms can find the optimal solution for a problem and are commonly used for optimization problems [4]. Two well-known non-heuristic algorithms are integer programming and linear programming. Wang et al. [9] proposed a QoS evaluation method that incorporated SLA satisfaction and designed a service selection algorithm based on this method. Bharathan et al. [10] proposed a new penalty-based optimization model in which QoS constraints can be relaxed but incur penalties accordingly. However, adjusting the penalty function parameters is computationally expensive. Zhu et al. [11] combined graph planning and fuzzy logic to solve the QoS-aware service composition problems, developing fuzzy rules to rank services based on user preferences and, thereby, reducing the search space. Wang et al. [12] proposed an FCDC green service composition method, which minimizes energy and network resource consumption on physical servers and switches in cloud data centers to achieve green service composition optimization. However, this method is not suitable for dealing with large-scale search spaces. Badidi et al. [13] proposed a generic model for adjusting context information using service meta-data specifications, which selects appropriate services under the user’s QoS constraints and current context. Poryazov et al. [14] proposed three intuitionistic fuzzy representations of virtual service devices. Six intuitionistic fuzzy estimations that include service device uncertainty were proposed. The proposed uncertainty estimations allow the definition of new QoS metrics and can be used to determine the quality of service composition across wide-ranging service systems. Wang et al. [15] designed a multi-objective algorithm called HypE-C, which employs three correlation-based local search strategies within the frame-
work of the HypE (the hypervolume estimation algorithm for multiobjective optimization) to achieve better trade-offs among multiple conflicting QoS criteria. Chai et al. [16] proposed a fast energy-centered and QoS-aware service composition approach (FSCA-EQ) for the IoT service composition. FSCA-EQ adopts a hierarchical optimization approach that uses the compromise ratio method (CRM) to pre-select services that meet the user’s QoS requirements. Then, to reduce the energy consumption of IoT devices and extend their lifespan, the concept of relative advantage is applied to select the optimal service as the final composite service.

Non-heuristic algorithms are typically straightforward to implement and have relatively low time complexity. However, they require an enumeration of all possible compositions, and the computational complexity increases with the size of the problem, making them unsuitable for large-scale problems. They may also encounter local optimal solutions when facing specific problems.

2.2. Cloud Service Composition Based on Heuristic Algorithms

Heuristic algorithms are a popular solution for specific problems, yielding high-quality feasible solutions within an acceptable timeframe by leveraging problem-specific characteristics [4]. Common heuristic algorithms include ant colony algorithms and simulated annealing. Bhushan et al. [17] utilized PROMETHEE in multi-criteria decision-making to select the best service from an optimal cloud composition based on QoS standards, delivering more efficient QoS services with fewer clouds. Kurdi et al. [18] developed a cuckoo bird-inspired algorithm addressing multi-cloud supply chain issues, utilizing IoT and web services. Ghobaei-Arani et al. [19] proposed the CSA-WSC cuckoo search algorithm, considering both service and network QoS. Yang et al. [20] proposed a dynamic ant colony genetic hybrid algorithm that dynamically controls the execution time of the genetic and ant colony algorithms to maximize the optimization ability and speed up overall convergence. However, this method requires complex tuning and optimization. Naseri et al. [21] identified QoS parameters through an agent-based approach and selected the best service using a particle swarm optimization algorithm. However, inaccurate agent models or large deviations may negatively impact resulting service composition schemes. Yang et al. [22] proposed an improved multi-objective grey wolf optimization algorithm with an enhanced search strategy to avoid local optima. Ibrahim et al. [23] applied the energy-aware mechanism using a hybrid frog-leaping and genetic algorithm (SFGA). However, the definition and processing of cloud service energy consumption are too simple. Zanbouri et al. [24] used bee mating optimization and trust-based clustering algorithms to solve the trust problem. Dahan et al. [25] introduced a hybrid algorithm combining ant colony optimization and genetic algorithms to compose cloud services, but the adaptability of their approach has not been fully verified. Finally, Jin et al. [26] proposed an eagle strategy with a uniform mutation and improved whale optimization algorithm to balance global and local search capabilities. Tarawneh et al. [27] proposed the use of multistage forward search (MSF) to minimize the number of integrated web services in order to enhance the selection and composition process of web services. They also improved the provided services from the aspects of symmetry and variation of the service composition method by adopting the spider monkey optimization (SMO) algorithm. Rajeswari et al. [28] developed a hybrid algorithm that combines a newly developed firefly optimization algorithm with a fuzzy-logic-based service composition model (F3L-WSCM) for location awareness. The firefly algorithm is applied to generate a synthesis plan that minimizes the number of composite plans. The fuzzy subtraction clustering is used to select the optimal composition plan from existing composition plans. Li et al. [29] proposed an improved salp swarm algorithm (SSA) for the QoS service composition selection named CSSA by integrating the chaotic mapping method into the algorithm. The randomness and traversability of chaos are used to reduce the possibility of falling into local optimal solutions and enhance the mining capability of the algorithm. Furthermore, the fuzzy continuous neighborhood search method is adopted to enhance the local search capability.
of the algorithm. Li et al. [30] focused on the energy consumption of file transfer between clouds. Based on the multi-cloud environment, they first standardized the QoS of services. Then, the objective of service composition under a multi-cloud environment is analyzed. Finally, a genetic algorithm is used to solve the service composition problem under a multi-cloud environment. Xiao et al. [31] proposed a combinatorial optimization model that not only considers the transmission and switching energy consumption but also takes into account the execution energy consumption when the device provides services. To balance the QoS attributes and energy consumption, the composition problem is treated as a multi-objective optimization problem and solved using a genetic algorithm. Guzel et al. [32] proposed a multi-objective IoT service composition framework based on a fog computing environment. The generated service composition plans consider QoS, energy consumption, and fairness. The problem is modeled as a multi-objective optimization problem, and an NSGA-II (the non-dominated sorting genetic algorithm II)-based optimization model is used to obtain the composition plans and compose IoT applications.

However, heuristic algorithms are not without their limitations. Firstly, they often require significant time to produce results. Secondly, the design process of meta-heuristic algorithms is typically complex and involves cumbersome tuning processes.

2.3. Service Composition Based on Machine Learning

In recent years, there has been a growing trend in using machine learning methods for service composition. Compared to non-heuristic and heuristic algorithms, machine learning algorithms have the ability to automatically learn and adapt to various data distributions and changes in real time, which allows them to update and optimize their models. Additionally, machine learning algorithms can efficiently process data adaptively. Furthermore, they can be applied to different types of cloud service composition problems and can be adjusted and optimized according to specific situations. This makes them more widely applicable and eliminates the need to define complex models and rules in advance. Currently, there are three main types of machine learning-based service composition methods: deep learning; reinforcement learning; and deep reinforcement learning.

For deep-learning-based service composition, Haytamy et al. [33] proposed a combination of long short-term memory (LSTM) and particle swarm optimization (PSO) algorithms to accurately recommend cloud services based on QoS properties. Bouzary et al. [34] suggested combining traditional machine learning methods with metaheuristic methods to provide a comprehensive approach to service composition problems under cloud manufacturing mode. Additionally, Bouzary et al. [35] proposed using Word2Vec and LSTM-based neural network models to identify suitable candidate sets for each submitted manufacturing subtask in the cloud manufacturing platform, followed by an approach to optimal composition services using genetic algorithms. Ren et al. [36] introduced DeepQSC, a deep supervision learning framework based on graph convolutional networks and attention mechanisms that can form high-QoS composition services within a limited computing time. However, this method does not account for user QoS constraints.

Reinforcement learning has been widely explored for service composition. Wang et al. [37] proposed a multi-agent reinforcement learning model that combines multi-agent technology and reinforcement learning for adaptive service composition, but communication cost reduction between agents remains an issue. Ren et al. [38] introduced the CSSC-MDP model, which accounts for the uncertainty of service quality and behavior. Wang et al. [39] combined reinforcement learning with skyline computation to improve the efficiency of compositional computation. However, without integrating deep learning, this method may not be efficient in dealing with large-scale problems. Alizadeh et al. [40] proposed a reinforcement-learning-based method for service composition in WoT environment, which does not require prior knowledge of user preferences for QoS attributes.

Several researchers have proposed combining reinforcement learning with deep learning for service composition. Wang et al. [8] introduced an adaptive deep reinforcement learning method for service composition, combining recurrent neural networks and heuris-
tic behavior policies. Wang et al. [41] proposed a QoS-prediction-based reinforcement learning method for service composition that combines recurrent neural networks with reinforcement learning, achieving better results in the effectiveness of service composition. Yu et al. [42] introduced the CSSC-DQN model to address dynamic service composition problems. The model handles missing candidate services and fluctuations in QoS values during the execution phase, particularly for services with fine-grained QoS attributes. Liang et al. [43] proposed a DRL-based QoS-aware cloud manufacturing service composition model that considers logistic issues and established a logistics-based DRL model. Neiat et al. [44] proposed a deep reinforcement learning-based combination method for selecting and composing quality parameter-aware mobile IoT services and developed a parallel group-based service discovery algorithm as the basis for measuring the accuracy of the proposed method. Yi et al. [45] introduced a DRL-based service composition solution, PPDRL, which uses pre-training policies and maximum likelihood estimation for adaptive and large-scale service composition. Liu et al. [46] proposed a cloud manufacturing service composition model involving logistics based on deep deterministic policy gradient, which solved the optimal service composition solution through repeated training and learning. Wang et al. [47] constructed tasks and services as graphs, used Graph Neural Networks (GNN) to mine potential correlations, predicted the probability of each service being used to construct the solution corresponding to the task, built the initial service solution based on Petri nets and reinforcement learning, and used the whale optimization algorithm (WOA) to fine-tune the initial solution for obtaining high-quality solutions. Zeng et al. [48] proposed a multi-strategy deep reinforcement learning algorithm that combines a basic DQN algorithm, dueling architecture, a double estimator, and a prioritized-replay mechanism. In addition, strategies, such as instantaneous reward strategy, greedy strategy, and heuristic strategy, are added to the algorithm.

DRL-based methods can handle complex and large-scale service composition problems, learn and adapt to different environments, and have good transferability.

Compared with the works mentioned above, our method has better adaptability than heuristic algorithms and does not require tedious design. Moreover, compared to existing deep reinforcement learning methods, we greatly improved the training efficiency of our method by using the skyline algorithm and achieved a trade-off between energy consumption and QoS. Our method also performs well when facing large-scale problems. In addition, existing service composition methods based on deep reinforcement learning assume a single-cloud environment while we consider a multi-cloud environment.

3. Problem Formulation

This paper aims to address the energy consumption and QoS-aware cloud service composition problem in a multi-cloud environment. In this section, we begin by modeling the research problem and abstracting it into mathematical symbols. We then introduce the energy consumption model and define relevant content on QoS awareness, followed by defining the objective formula.

3.1. Service Composition under Multi-Cloud

Firstly, we need to introduce the basic process of cloud service composition. The process of cloud service composition is shown in Figure 1 and can be decomposed into three stages:

1. Requirements decomposition: a complex requirement is decomposed into several subtasks, all of which constitute a composite task. A subtask can be completed by a cloud service;
2. Service selection: a search is conducted for services in the cloud service data center that match the functional requirements of each subtask, and these services are aggregated to form a candidate service set;
Service composition: an algorithm is used to select appropriate cloud services from the pre-selected candidate service set for each subtask and combine them to form a final cloud service composition plan.

Figure 1. Cloud service composition framework.

However, in our study, the focus is on cloud service composition within a multi-cloud environment. A multi-cloud environment (MCE) consists of multiple clouds, each containing a set of cloud service classes that include multiple cloud services with similar functionality yet varying non-functional attributes. All cloud service information within the cloud environment is registered in the cloud service data center. Users submit requests via a web application or mobile application. When the composition agent receives a user’s request, it first decomposes the request into multiple subtasks. The agent then selects appropriate cloud services for each subtask from the cloud service data center to create a candidate cloud service set. Finally, an algorithm is employed to select the most suitable cloud services from the candidate service set, resulting in the formation of a comprehensive cloud service composition plan. The results are subsequently returned to the user. Figure 2 illustrates an example of this process, in which there are two clouds, \(cd_1\) and \(cd_2\), with three service classes in \(cd_1\), each containing multiple cloud services with varying non-functional attributes. The composition agent serves as an intermediary, responsible for receiving user requests, searching for matching cloud services amongst the service pool, and using an algorithm to combine the identified cloud services, ultimately returning the cloud service composition results to the user.
We use MCECSC = \{ Task, CD, P, W, Cons \} to represent cloud service composition in a multi-cloud environment, where the meanings of the symbols are as follows:

1. **Task** = \{ T_1, T_2, \ldots, T_n \} represents a collection of subtasks. \( T_i (i = 1, 2, \ldots, n) \) refers to the \( i \)-th subtask, and \( n \) signifies the total number of subtasks. These subtasks collectively form the user’s intricate requirement. While a subtask can be completed by one or multiple cloud services, typically, one cloud service is sufficient for completing a single subtask;

2. **CD** = \( \{ cd_1, cd_2, \ldots, cd_k \} \) signifies a group of clouds, with \( cd_j (j = 1, 2, \ldots, k) \) signifying the \( j \)-th cloud in the environment, and \( k \) representing the number of clouds in the multi-cloud setting. A cloud usually contains various service classes, also referred to as abstract services. The definition of a cloud can be summarized as follows:

\[
cd_j = \{ S^j_1, S^j_2, \ldots, S^j_m \}
\]

where \( S^j_l (l = 1, 2, \ldots, m) \) represents the \( l \)-th service class in the \( j \)-th cloud, with \( m \) representing the number of service classes in the \( j \)-th cloud. A service class is a set of cloud services that cater to the requirements, possessing similar functionality yet varying non-functional attributes. The definition of a service class is as follows:

\[
S^j_i = \{ cs^j_{i,1}, cs^j_{i,2}, \ldots, cs^j_{i,c} \}
\]

where \( cs^j_{i,l} (i = 1, 2, \ldots, c) \) represents the \( i \)-th specific service in the \( l \)-th service class in the \( j \)-th cloud, with \( c \) representing the number of cloud services in the service class. Each cloud service possesses a set of QoS attributes. For instance, the QoS attributes of the \( cs^j_{i,l} \) service are defined as follows:

\[
Q^j_{i,l} = \{ q^j_{i,l,1}, q^j_{i,l,2}, \ldots, q^j_{i,l,r} \}
\]
where $q_{i,l,r}$ represents the $r$-th QoS attribute of the $i$-th cloud service in the $l$-th service class of the $j$-th cloud, with $r$ representing the number of QoS attributes under consideration;

(3) $P$ represents the selection of workflow patterns, encompassing sequential, parallel, conditional, and loop structures, each possessing its unique QoS aggregation formula. Since it is straightforward to convert between these four structures, we choose the sequential structure for our study, as depicted in Table 1;

(4) $W$ is a set of user preferences for the weight of each QoS attribute.

\[
W = \{w_1, w_2, \ldots, w_r\}
\]

where $w_r$ represents the weight preference for the $r$-th QoS attribute, with $r$ representing the number of QoS attributes under consideration;

(5) $Cons = \{CT_{\max}, RT_{\max}, RL_{\min}\}$ represents a set of user constraints. $CT_{\max}$ signifies the maximum cost of service composition; $RT_{\max}$ represents the maximum response time of service composition, and $RL_{\min}$ represents the minimum reliability of service composition.

### Table 1. The aggregation formula for QoS attributes based on different workflow patterns. $n$ is the number of abstract services; $k$ is the number of iterations, and $prob$ is the conditional probability [25].

<table>
<thead>
<tr>
<th>QoS</th>
<th>Sequential</th>
<th>Parallel</th>
<th>Conditional</th>
<th>Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>$\sum_{i=1}^{n} cost_i$</td>
<td>$\sum_{i=1}^{n} cost_i$</td>
<td>$\sum_{i=1}^{n} prob_i \times cost_i$</td>
<td>$k \times cost_i$</td>
</tr>
<tr>
<td>Reliability</td>
<td>$\prod_{i=1}^{n} rel_i$</td>
<td>$\min_{i=1}^{n} (rel_i)$</td>
<td>$\sum_{i=1}^{n} prob_i \times rel_i$</td>
<td>$(rel_i)^k$</td>
</tr>
<tr>
<td>Response time</td>
<td>$\sum_{i=1}^{n} rt_i$</td>
<td>$\max_{i=1}^{n} (rt_i)$</td>
<td>$\sum_{i=1}^{n} prob_i \times rt_i$</td>
<td>$k \times rt_i$</td>
</tr>
<tr>
<td>Energy</td>
<td>$\sum_{i=1}^{n} EC_i$</td>
<td>$\sum_{i=1}^{n} EC_i$</td>
<td>$\sum_{i=1}^{n} prob_i \times EC_i$</td>
<td>$k \times EC_i$</td>
</tr>
</tbody>
</table>

To provide a more in-depth understanding of the service composition scenario in a multi-cloud environment, we present a medical example. The service composition in this example includes three tasks, doctor selection, appointment scheduling, and payment; hence, $Task = \{T_1, T_2, T_3\}$. In this scenario, the user can consult with a doctor who will offer specific medical advice based on the user’s current condition and symptoms. In this specific example, there exist three clouds, $CD = \{cd_1, cd_2, cd_3\}$, with each cloud having three service classes. Additionally, each service class comprises an indefinite number of services, as presented in Table 2.

### Table 2. Example of services in multi-cloud.

<table>
<thead>
<tr>
<th>Cloud</th>
<th>Cloud Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cd_1$</td>
<td>$cs_{1,1}^{1,1}, cs_{1,2}^{1,1}, cs_{1,3}^{1,1}$, $cs_{1,1}^{1,2}, cs_{1,2}^{1,2}$, $cs_{1,3}^{1,2}, cs_{1,1}^{1,4}, cs_{1,2}^{1,4}, cs_{1,3}^{1,4}$</td>
</tr>
<tr>
<td>$cd_2$</td>
<td>$cs_{2,1}^{2,1}, cs_{2,2}^{2,1}, cs_{2,3}^{2,1}$, $cs_{2,1}^{2,2}, cs_{2,2}^{2,2}, cs_{2,3}^{2,2}$, $cs_{2,1}^{2,3}, cs_{2,2}^{2,3}, cs_{2,3}^{2,3}$</td>
</tr>
<tr>
<td>$cd_3$</td>
<td>$cs_{3,1}^{3,1}, cs_{3,2}^{3,1}, cs_{3,3}^{3,1}, cs_{3,4}^{3,1}$, $cs_{3,1}^{3,2}, cs_{3,2}^{3,2}$, $cs_{3,3}^{3,2}, cs_{3,3}^{3,3}$</td>
</tr>
</tbody>
</table>

### 3.2. Energy Consumption Model

In a multi-cloud environment, the energy consumption model is a crucial concept because the energy consumption of cloud services directly affects the overall operating costs and environmental issues, such as energy conservation and cloud platform emission reduction. We believe that the energy consumption of cloud services in a multi-cloud service composition is composed of two parts, the energy consumption generated by the execution of the cloud service itself and the energy consumption generated by file
transmission between cloud services [30,49]. For any cloud service, its execution energy consumption can be calculated based on two parameters, execution time and energy consumption efficiency. The file transmission energy consumption between cloud services can be calculated based on the time required for the file transmission and the power required for transmitting files.

Cloud services consume energy during execution. For example, some programs may frequently call hardware devices, such as network cards and disks, which result in additional energy consumption and burden. For the energy consumption of the \(i\)-th cloud service execution in a cloud service composition, we can define it as follows:

\[
EC_{\text{exec}}^i = t_{\text{exec}}^i \times EF_{\text{exec}}^i
\]  

(1)

where \(EC_{\text{exec}}^i\) represents the energy consumption generated by the execution of the \(i\)-th cloud service in a cloud service composition; \(t_{\text{exec}}^i\) represents the execution time of the \(i\)-th cloud service, and \(EF_{\text{exec}}^i\) represents the efficiency of energy consumption during the execution of the \(i\)-th cloud service. We know that efficiency multiplied by time equals workload, so the execution time of a cloud service multiplied by its energy consumption efficiency equals the energy consumption generated during the execution of the cloud service.

The problem of cloud service composition in a multi-cloud environment presents unique challenges, including an increase in data transfer and computation between cloud services compared to a single-cloud environment. In a multi-cloud environment, the additional expenses incurred by data transfer energy consumption can be attributed to the following three reasons:

1. Different cloud services may be stored in distinct physical locations, necessitating data transfer operations when processing data;
2. In a multi-cloud environment, each cloud service may process a considerable amount of data, resulting in a significant amount of data transfer and network communication. As the time and energy required for data transfer increase proportionally with the amount of data transmitted, the energy consumed during data transfer must be accounted for during data processing;
3. Different cloud services may collaborate in processing a specific task, requiring data exchange and transmission and resulting in additional energy consumption.

It is, therefore, essential to consider both data transfer and computation between different cloud services in different clouds in a multi-cloud environment. To account for the energy consumption of file transfer between cloud services in a multi-cloud environment, we define it as follows:

\[
EC_{\text{trans}} = P_{\text{trans}} \times t_{\text{trans}}
\]  

(2)

where \(EC_{\text{trans}}\) is the energy consumption generated by file transfer between cloud services; \(P_{\text{trans}}\) is the power consumed during file transfer between cloud services, and \(t_{\text{trans}}\) is the time required for file transfer between cloud services. We use a concrete example to illustrate: assuming we are currently at the \(i\)-th cloud service, the energy consumption generated by file transfer between the \(i\)-th cloud service and the previous service in the composition, i.e., the \((i-1)\)-th cloud service, is as follows:

\[
EC_{\text{trans}}^{i-1,i} = P_{\text{trans}}^{i-1,i} \times t_{\text{trans}}^{i-1,i}
\]  

(3)

It should be noted that if two cloud services are in the same cloud, their data transfer is relatively easy, and the time required for transfer is minimal. Therefore, when data exchange occurs within the same cloud, the time consumed for transfer can be considered as approximately 0. That is, if the \((i-1)\)-th cloud service and the \(i\)-th cloud service in the cloud service composition are located in the same cloud, then \(t_{\text{trans}}^{i-1,i} \approx 0\). In summary, we can define the energy consumption of any cloud service in a multi-cloud environment as follows:

\[
EC_i^{\text{total}} = EC_{\text{exec}}^i + EC_{\text{trans}}^{i-1,i}
\]  

(4)
that is,

\[ EC_{i}^{\text{total}} = t_{i}^{\text{exec}} \times EF_{i}^{\text{exec}} + P_{i-1,i}^{\text{trans}} \times t_{i-1,i}^{\text{trans}} \]  \hspace{1cm} (5)

Therefore, the total energy consumption of cloud service composition in a multi-cloud environment is as follows:

\[ EC(MCECSC) = \sum_{i} EC_{i}^{\text{total}} \]  \hspace{1cm} (6)

This definition can help us quantify the energy consumption problem of cloud service composition in a multi-cloud environment, thus enhancing the accuracy and reliability of decision-making when making service composition decisions.

3.3. QoS-Aware Service Composition

The QoS-aware service composition problem aims to select and combine suitable cloud services to meet the user’s QoS expectations. In this context, the QoS value of the composite service is aggregated from the QoS values of individual services. We consider a sequential structure and calculate the QoS value of a cloud service composition using the following equation:

\[ QoS(MCECSC) = \sum_{i=1}^{n} \sum_{j=1}^{r} w_{i,j} \times q_{i,j} \]  \hspace{1cm} (7)

where MCECSC represents the cloud service composition plan, and QoS(MCECSC) is the aggregated QoS value of the cloud service composition. We use \( q_{i,j} \) to represent the \( j \)-th QoS attribute of the cloud service selected for the \( i \)-th subtask; \( r \) represents the number of QoS attributes, and \( n \) represents the number of subtasks.

To select QoS attributes, we consider cost, response time, and reliability, which are commonly used by most researchers in this field. However, these attributes have different ranges, making effective comparison challenging. Specifically, the necessity of normalization lies in the fact that normalizing various indicators can reduce the differences between different QoS indicators, making them easier to be comprehensively evaluated and compared. This helps achieve effective and efficient QoS estimation, improves system reliability, stability, and maintainability, and further enhances user satisfaction. Therefore, normalization has a high practical value \[39\]. Hence, normalization is necessary to map the range of QoS to the \([0, 1]\) interval, enabling more effective comparison. Additionally, some QoS attributes are positively correlated (such as reliability), while others are negatively correlated (such as cost and response time). Therefore, we use Equation (8) for positive correlation QoS normalization and Equation (9) for negative correlation QoS normalization.

\[ V_{\text{normpositive}} = \begin{cases} \frac{v - \min}{\max - \min}, & \text{max} \neq \min \\ 1, & \text{max} = \min \end{cases} \]  \hspace{1cm} (8)

\[ V_{\text{normnegative}} = \begin{cases} \frac{\max - v}{\max - \min}, & \text{max} \neq \min \\ 1, & \text{max} = \min \end{cases} \]  \hspace{1cm} (9)

Our goal is to maximize the QoS value of the cloud service composition and minimize the energy consumption of the cloud service composition to ensure high-quality and low-energy service composition. This is a multi-objective optimization problem, which we approach with the following objective formula:

\[ Obj = \{ \max QoS(MCECSC), \min EC(MCECSC) \} \]  \hspace{1cm} (10)

4. SkyDRL for Multi-Cloud Service Composition Problem

This paper proposes the SkyDRL method, which combines the branch and bound skyline algorithm with a deep reinforcement learning algorithm. This approach addresses the cloud service composition problem in a multi-cloud environment by reducing energy consumption while finding a cloud service composition with better QoS performance. More-
The core concept of DQN involves using deep neural networks and nonlinear transformations to gradually extract abstract features from input data [51]. The algorithm employs reinforcement learning to maximize cumulative rewards, ultimately facilitating optimal decision-making.

4.1. Preliminary

4.1.1. Reinforcement Learning

Reinforcement learning is a major branch of machine learning and serves as the core idea behind the proposed method presented in this paper. Five critical elements constitute reinforcement learning: agent; environment; state; action; and reward. The agent is responsible for making decisions and interacting with the environment by performing actions in a particular state. The environment represents the external context in which the agent operates, including states and transitions between them. The state describes the condition of the agent at a given moment within the environment. Action refers to the decision made by the agent in the current state, while the reward is feedback information that the environment provides based on the actions taken by the agent. Rewards can be positive or negative, with positive rewards serving as encouragements to the agent, while negative ones can be considered as punishments.

The primary purpose of reinforcement learning is to enable the agent to learn by itself in the environment, continuously adjusting its strategy based on the feedback and eventually determining the optimal approach to maximize cumulative rewards [50]. The agent selects an action based on a specific policy and executes it, with the environment giving feedback based on the agent’s action. The agent then moves to a new state, adjusts its strategy based on the newly acquired information, and performs a new action. This process continues until the optimal strategy that maximizes cumulative rewards is found. Figure 3 presents the basic framework for reinforcement learning. Assuming that the current agent is in a state \(s_t\), it executes a particular action \(a_t\), and the environment provides feedback \(r_t\), based on the action while simultaneously triggering a transition from state \(s_t\) to \(s_{t+1}\).

![Figure 3. Reinforcement learning.](image)

4.1.2. Deep Reinforcement Learning

Q-learning is a classic value-based algorithm used in reinforcement learning. Its primary objective is to estimate the value of an action via a Q-table. Whenever the estimated value differs from the actual value, the Q-table gets updated based on the difference between the two values. With repeated updates, the Q-table becomes capable of accurately estimating the true value. However, utilizing a Q-table to store Q-values presents difficulties when dealing with large-scale and continuous data. To overcome the limitations of a Q-table, a Q-function is utilized as a replacement.

Considering the excellent ability of neural networks in modeling complex functions, a neural network can be used to fit this Q-function. The integration of neural networks and Q-learning is known as Deep Q Network (DQN), and it is a DRL-based algorithm. The core concept of DQN involves using deep neural networks and nonlinear transformations to gradually extract abstract features from input data [51]. The algorithm employs reinforcement learning to maximize cumulative rewards, ultimately facilitating optimal decision-making [50]. In DQN, the input is the state, and the output is the value estimation.
for each action. DQN has two crucial enhancements. One is to use the experience replay mechanism to break the correlation between samples, and the other is to use two neural networks with the same structure, Q-network and Q-target. The Q-network estimates the predicted Q-value, while the Q-target calculates the target Q-value. During each step, the agent-environment interaction generates a sample that is stored in the replay buffer. During training, the Q-learning update formula is utilized, and a batch of samples is uniformly extracted from the replay buffer to update the parameters of the Q-network using stochastic gradient descent. The update formula is shown below:

\[ Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \]  \tag{11}

In the update formula, \( Q(s, a) \) is the Q-value when taking action \( a \) under state \( s \). \( \alpha \) denotes the learning rate; \( r \) represents the reward value, and \( \gamma \) represents the discount factor. A higher \( \gamma \) value suggests that the model prioritizes long-term benefits, whereas a lower \( \gamma \) value places more emphasis on short-term gains. The variable \( s' \) stands for the next state; \( \max Q(s', a') \) signifies the maximum Q-value under state \( s' \), and \( \text{TargetQ} = r + \gamma \max Q(s', a') \) represents the target Q-value. We use \( \theta \) to denote the parameters of the Q-network and \( \theta' \) to represent those of the Q-target. During training, the parameters \( \theta \) get updated every time, while \( \theta' \) remains unchanged. To ensure consistency between \( \theta \) and \( \theta' \), we assign the value of \( \theta \) to \( \theta' \) only once every \( C \) steps. Specifically, when a loop of state-action-reward is completed, we call it the end of a time step. After every \( C \) time steps are completed, the parameters are updated.

4.1.3. Skyline Algorithm

The skyline algorithm is a well-known approach for tackling multi-objective optimization problems. Its main objective is to select a set of possible optimal solutions based on the dominance relationship. In the context of cloud service composition problems, the skyline algorithm utilizes the dominance relationship to choose services from a candidate set that is not dominated by other cloud services. All these non-dominated cloud services form the skyline set.

Assuming that there are two cloud services, \( CS_1 \) and \( CS_2 \), with \( Q \) QoS attributes. If \( \forall i \in [1, Q] : q_i(CS_1) \geq q_i(CS_2) \) and \( \exists k \in [1, N] : q_k(CS_1) > q_k(CS_2) \), then we can say that the cloud service \( CS_1 \) dominates \( CS_2 \), denoted as \( CS_1 \succ CS_2 \).

\( q_i(CS_1) \) represents the value of the \( i \)-th QoS attribute of \( CS_1 \). To better illustrate the meaning of the dominance relationship, we use the example shown in Figure 4. As shown in Figure 4, we describe each service using response time and cost attributes, and thus, the services are represented as points in space. From the figure, we can observe that cloud service A is not dominated by any other cloud service, so cloud service A belongs to the skyline set. Similarly, it can be seen that cloud services I and J both belong to the skyline set. On the other hand, for example, cloud service C is dominated by cloud service I, so it does not belong to the skyline set. In the same vein, cloud services B, D, E, F, G, H, and K do not belong to the skyline set.

Utilizing the skyline algorithm to process the candidate cloud service set can significantly improve computing efficiency when compared to performing calculations on the original candidate cloud service set. Deep reinforcement learning is a learning approach that relies on training with a large amount of data. By reducing the search space with skyline, the model prioritizes higher-quality cloud services, which enhances the training process, simplifies optimization, and reduces computational costs. Currently, several variants of skyline algorithms are available, such as the bitmap method, nearest neighbor method, and branch and bound method. Among them, the branch and bound skyline algorithm is an incremental approach that leverages R-trees to index data points based on nearest neighbor search [52]. In this study, we adopt the branch and bound skyline algorithm to optimize the search space and reduce the computing time required for deep reinforcement learning, thereby improving the training speed. A detailed description of this algorithm is presented in Algorithm 1.
4.2. State and Action

We model the cloud service composition problem as a Markov Decision Process (MDP), where a cloud service composition based on MDP is defined as a 5-tuple \( \langle State, s_0, \gamma, A, R \rangle \) [41]:

- **State** is a set of finite states;
- \( s_0 \) is the initial state of the agent, and the execution process of the cloud service composition starts from this state;
- \( s_f \) is a set of terminal states. The execution of the cloud service composition will terminate when a state in \( s_f \) is reached;
- \( A(s) \) is the set of executable actions in state \( s \). In the cloud service composition problem, the action set is a set of candidate cloud service sets. At each state, there is a set of candidate cloud service sets from which we can select and execute cloud services;
- \( R \) is the reward function. Assuming the current state is \( s_i \), a service is selected and called, and the state transitions to \( s_{i+1} \). Afterward, we will receive a reward for executing this action from the environment.

For each state, we can define it as \( s_i = \langle t, cs, cld \rangle \), where \( t \) represents the state at the time \( t \); \( cs \) represents the selected cloud services in the current state, and \( cld \) indicates which cloud environment the selected cloud service belongs to. In this problem, the

---

**Algorithm 1.** Branch and Bound skyline algorithm

1. Initialize \( SK = \emptyset \) //skyline list
2. Insert all cloud services of the root R into the heap
3. for heap not empty do
4. Remove top cloud services \( cs \)
5. if \( cs \) is dominated by some point in \( SK \) then
6. Discard \( cs \)
7. else //\( cs \) is not dominated
8. if \( cs \) is an intermediate entry then
9. for each child \( cs_i \) of \( cs \) do
10. if \( cs_i \) is not dominated by some point in \( SK \) then
11. Insert \( cs_i \) into the heap
12. else //\( cs \) is a datapoint
13. Insert \( cs_i \) into the SK
14. end if
15. end for
16. end if
17. end if
18. end for

---

Figure 4. Example of skyline services.
The quality of service composition can partly reflect user satisfaction. Therefore, selecting QoS attributes to model the reward function is appropriate. Additionally, users have different preferences for QoS, with some placing more weight on response time than cost and vice versa. Hence, we use \( w \) to represent user preferences for specific QoS attributes. Cost, response time, and reliability are commonly considered QoS attributes, and we consider these factors in this paper. Furthermore, energy consumption affects service composition, and hence, it needs to be incorporated into the reward function.

\[
R_i = R_i^{qos} + R_i^{ec} 
\]  
\[
R_i^{qos} = w_c q_{i,c} + w_{rt} q_{i,rt} + w_{rl} q_{i,rl} 
\]  
\[
R_i^{ec} = w_{ec} \times \frac{1}{EC_i^{total}} 
\]
$R_i$ denotes the reward value of the $i$-th cloud service in a given service composition. Since the goal is to maximize the QoS value and minimize the energy consumption value of the cloud service composition, it is a multi-objective optimization problem. Therefore, we divide the reward value into two parts: the first part, $R_{i,\text{qos}}$, represents the reward obtained from the QoS attributes of the cloud service, while the second part, $R_{i,\text{ec}}$, represents the reward obtained from the energy consumption of the cloud service. The sum of these two parts represents the total reward value of the cloud service. Additionally, $q_{i,r}$ denotes the response time attribute value; $q_{i,r}$ denotes the response time attribute value, and $q_{i,r}$ denotes the reliability attribute value. Furthermore, $EC_{\text{total}}$ is defined in Section 3.2 as the total energy consumption of the cloud service. To consider user preferences for different QoS attributes, we adopt weight values for each attribute and denote them as $w_c$ for cost, $w_r$ for response time, $w_r$ for reliability, and $w_{ec}$ for energy consumption. Moreover, we have

$$w_c + w_r + w_r + w_{ec} = 1$$ (15)

We model the reward function based on the QoS and energy consumption of cloud services, which can guide the decision-making of the agent, prioritizing cloud services with higher QoS values and lower energy consumption.

4.4. SkyDRL

Figure 6 depicts the overall architecture of our proposed SkyDRL method, which leverages the branch and bound skyline algorithm and deep reinforcement learning techniques. The DRL algorithm is based on the deep Q-network (DQN) algorithm and integrates three enhancements: double DQN (DDQN), Dueling Network, and Prioritized Experience Replay.

In the neural network architecture, we modify the original DQN network to a Dueling Network, which divides the single-path output of $Q(s, a)$ in the original DQN into two separate outputs. The first output is the state’s value $V(s)$, which represents the state’s intrinsic value, while the second output is the advantage $A(s, a)$ of each action in the state relative to others. These outputs are then combined to obtain the total Q value, i.e.,

$$Q(s, a) = V(s) + A(s, a)$$ (16)
By separating the state value and action advantage, the network structure enhances training speed and stability, making it more effective than the traditional DQN network.

When calculating the loss function, due to the overestimation problem of DQN, the estimation error increases as the number of actions increases, which leads to local optima. Therefore, we change the calculation method of the Q-target value to that of DDQN to address this issue.

\[
\text{Target}Q = r + \gamma Q(s', \text{argmax}_{a} Q(s', a; \theta)); \theta')
\]  

(17)

Differing from the calculation method of the Q-target value in Equation (11), Equation (17) first calculates all Q values by inputting the following state \( s' \) to the Q-network, selects the action with the highest Q value, and then inputs this action and the next state \( s' \) to the Q-target network to calculate the Q value. This is an easy-to-understand approach but has a significant effect on solving the overestimation problem.

Regarding the experience replay method, we adopt the Prioritized Experience Replay instead of randomly and uniformly selecting data from the replay buffer, as in traditional DQN methods. This is because the traditional approach overlooks the varying importance levels of samples for the agents. If we select samples based on their importance, the training speed can be greatly enhanced. The Prioritized Experience Replay selects samples with priority based on their importance, leading to more efficient learning. Typically, to evaluate the value of the experience, TD-Error is used, as our goal is to minimize it. If TD-Error is large, it indicates that the current Q function is far from the target Q function and more updates are required. Therefore, we utilize TD-Error to measure the value of experience. The calculation formula for TD-Error is

\[
\delta = r + \gamma Q(s', \text{argmax}_{a} Q(s', a; \theta)); \theta') - Q(s, a)
\]  

(18)

As indicated by Equation (18), the actual calculation method of TD-Error is the same as that of the loss function. In order to prevent overfitting of the network, we use a probability-based approach to sample experiences and set the priority value of each experience to

\[
P(i) = \frac{p_i^a}{\sum_k p_k^a}
\]  

(19)

where \( p_i \) is the priority of the \( i \)-th sample; \( a \) is used to adjust the priority level, and the calculation method of \( p_i \) is as follows:

\[
p_i = |\delta + \epsilon|
\]  

(20)

\( \epsilon \) is a very small value to ensure that the probability of extracting experiences with TD-Error equaling zero is not zero. To facilitate the storage and sampling of Prioritized Experience, a SumTree structure is used to store the priority of samples. SumTree is a binary tree structure. The priority we need to store is in the leaf nodes of the tree. The parent node of the leaf node stores the sum of the priority of its child nodes. After updating the parameters of the neural network, the priority of the samples needs to be recalculated.

Before starting the training, the user’s needs are decomposed into several subtasks, and a workflow is constructed accordingly. Next, a candidate service set is formed by retrieving all cloud services that meet the requirements of the workflow from the cloud service pool. When retrieving services, we need to consider services in multi-cloud environments in order to provide more comprehensive and high-quality choices. This step will establish a candidate set of cloud services for each subtask. The cloud services in the set may come from multiple clouds. To expedite deep reinforcement learning training and reduce its optimization difficulty and computational cost, we use a branch and bound skyline algorithm to pre-query the candidate service set. This helps narrow down the search space for the problem, enabling the model to focus on high-quality cloud services.
During training, the state is passed into the neural network to calculate the Q values of all executable actions (i.e., Q values of all candidate cloud services). Similarly, these candidate cloud services may come from multiple different clouds. The ε-greedy strategy is used to select an action based on the calculated Q values. This strategy selects the action with maximum Q value with probability ε and selects a random action with probability 1-ε. This creates a balance between the exploration and utilization rates and prevents the model from getting stuck in local optima. The transition information shown in Figure 6 is stored in the replay buffer, and since the participating cloud services may not come from the same cloud and may come from different clouds, the stored sample information will include the information of the clouds to which the cloud services belong. This process is repeated until the buffer is full.

Once the replay buffer is full, the neural network parameters are updated. A batch of samples is extracted from the buffer via the Prioritized Experience Replay, and the loss function mentioned earlier is used to update the network parameters. During updates, the Q-target parameters remain unchanged, and only in every C steps do the Q-network’s parameters get assigned to Q-target.

Finally, the priority of the samples needs to be recalculated after updating the neural network parameters. By carrying out the aforementioned steps iteratively, i.e., passing the state into the neural network, selecting an action based on the policy, storing samples in the replay buffer, calculating the loss function and updating the network parameters, and updating the priority tree, the neural network’s parameters are updated continuously. Eventually, the system achieves convergence, and the optimal cloud service composition solution is determined. Algorithm 2 provides a detailed implementation of the SkyDRL method.

---

**Algorithm 2. Skyline-Enhanced Deep Reinforcement Learning Approach**

**Input:** discounted factor γ, learning rate α, minibatch size b, initial exploration and final exploration ε, sampling weight β

1. Initialize replay memory D with capacity N and initial priority p₁ = 1
2. Initialize action-value function Q with Q-network parameters θ
3. Initialize target action-value function Q’ with target-network parameters θ’
4. Initialize SK = ∅ / skyline list
5. Insert all cloud services of the root R into the heap
6. for each episode do
   7. for each t do
      8. Acquire skyline services SK using the Branch and Bound skyline algorithm
      9. With probability ε select a random action aᵢ in SK
     10. Otherwise select aᵢ = argmaxₐₗₐₜₜᵢₖ(Q(sᵢ, aᵢ; θ) in SK
     11. Execute action aᵢ, observe reward rₜ and next state sᵢ₊₁, sᵢ = sᵢ₊₁
     12. Store transition (sᵢ, aᵢ, rᵢ, sᵢ₊₁) in D with maximal priority pᵢ = maxᵢ < tᵢ [pᵢ]
     13. Sample mini-batch b of transition (sᵢ, aᵢ, rᵢ, sᵢ₊₁), for each sample
     14. Compute importance-sampling weight wᵢ = (N × P(j))⁻β / maxᵢ(wᵢ)
     15. if episode terminates at step j + 1 then
        16. set yⱼ = rⱼ
     17. else
        18. yⱼ = rⱼ + γQ(sᵢ₊₁, argmaxₐₗₐₜₜᵢₖ(Q(sᵢ₊₁, a; θ), θ’)
     19. end if
     20. Update Q-network parameters θ with a loss function of \( \frac{1}{N} \sum_{j=1}^{b} w_j (y_j - Q(s_j, a_j; θ))^2 \)
     21. Compute TD-error δⱼ = yⱼ - Q(sⱼ, aⱼ; θ)
     22. Update transition priority Pⱼ ← |δⱼ|
     23. Every C steps reset θ’ ← θ
   7. end for
6. end for
5. Experiments

This section presents a series of experiments conducted to primarily validate the proposed algorithm in the following aspects:

1. The effectiveness of the algorithm in reducing energy consumption in the cloud service composition problem under multi-cloud environments;
2. The algorithm’s ability to solve QoS-aware problems effectively. Although energy consumption is an important consideration, optimizing the QoS of cloud service composition remains paramount. To achieve this, we aim to minimize the energy consumption of service composition while optimizing the QoS. For instance, if selecting Service B over Service A reduces energy consumption while still maintaining a relatively optimal QoS, we prefer the former, even if Service A has a slightly better response time. Such trade-offs are essential in ensuring efficient service composition;
3. The adaptability of the proposed algorithm to dynamic multi-cloud environments. Cloud services’ QoS changes frequently due to various reasons, such as service providers adjusting strategies or services becoming suddenly unavailable. Therefore, we test the proposed algorithm’s effectiveness in dealing with such changes dynamically;
4. In a dynamic multi-cloud environment, generating new compositions quickly is crucial. Thus, we verify how long it takes to generate a cloud service composition using the proposed algorithm;
5. Scalability is a vital indicator of method generalization. We investigate the impact of varying scales of cloud service composition problems on method efficiency.

5.1. Experiment Setting

In this study, we utilized the QWS dataset, consisting of 2507 real services, each with 9 QoS attributes. However, this dataset lacked essential cost attributes and energy consumption data required for our research objectives. To overcome this limitation, we augmented the dataset with additional data from multiple sources, ensuring it met our study’s needs. Considering the multi-cloud context of our research, we extended the dataset to include up to 20 clouds, with each service assigned a 10% probability of belonging to a specific cloud. We designed multiple experimental scenarios based on prior research [25,39,43], including various subtask numbers (20, 40, 60, 80), and candidate cloud service numbers (100, 200, 300, 400, 500, 600). Our experiments were conducted based on the requirements of validating different indicators. To assess SkyDRL’s effectiveness in addressing cloud service composition problems in a multi-cloud environment, we compared it against other methods, including PD-DQN [43], Q-learning with skyline [39], SFGA [23], and MAACS [25], all of which were machine learning-based methods and heuristic algorithms. All experiments were conducted under the same dataset, and we set the recommended parameters for these methods based on the prior research, ensuring a fair comparison. For the SkyDRL method, we set the capacity of the buffer to 10,000 and extracted 32 samples each time. The number of Q-target network update steps is set to 500. We set the learning rate to 0.01; the initial value of $\epsilon$ used in the $\epsilon$-greedy policy is 0.9, which reduces linearly to 0.1 during training. The discount factor $\gamma$ is set to 0.9.

5.2. Energy Consumption

Multi-cloud environments typically involve a large amount of data transmission and computation, often resulting in increased energy consumption during service composition. The objective of this paper is to propose an approach that can effectively reduce the energy consumption of cloud service composition. Since the energy consumption of cloud services is closely related to the operating costs of cloud platforms and environmental protection issues, such as energy conservation and emission reduction, reducing the energy consumption of cloud service composition proves extremely important.

We selected the energy consumption of cloud service composition as our comparative metric for this experiment. We designed four scenarios for this purpose. Each scenario
Appl. Sci. 2023, 13, x FOR PEER REVIEW 23 of 30

consists of 60 subtasks, and for each subtask within the four scenarios, there are 100, 200, 300, and 400 candidate cloud services, respectively. Figure 7 displays the total energy consumption results for each group. Our SkyDRL method achieved an average energy consumption of 1687.50, while PD-DQN consumed 2402.25, Q-learning with skyline consumed 2570.5, SFGA consumed 1829.5, and MAACS consumed 2338.5.

Figure 7. Energy consumption for PD-DQN, Q-learning with skyline, SFGA, MAACS, and SkyDRL, with 60 subtasks but different numbers of candidate services for each subtask.

Compared to the other methods, our SkyDRL method proved superior in terms of reducing energy consumption during the service composition. Specifically, we achieved an average reduction of 29.75% in energy consumption compared to PD-DQN, 34.35% compared to Q-learning with skyline, 7.76% compared to SFGA, and 27.83% compared to MAACS. Notably, our SkyDRL method was not affected by the number of candidate services, demonstrating its effectiveness and robustness in practical, real-world applications.

In conclusion, our experimental results demonstrate that the proposed SkyDRL method can effectively reduce the energy consumption of cloud service composition. This has significant implications for optimizing cloud platforms, reducing operating costs, and promoting environmental protection through energy conservation and emission reduction.

5.3. Effectiveness and Efficiency

In this section, we aim to evaluate the efficiency and effectiveness of the proposed method in optimizing QoS values of service composition. We conducted two types of experiments. The first type consisted of 60 subtasks divided into four groups with 100, 150, 200, and 250 candidate cloud services for each subtask, respectively, while the second type had four groups with 20, 40, 60, and 80 subtasks and a fixed number of 200 candidate services for each subtask. Experimental results are presented in Figures 8 and 9.

As shown in Figure 8, the SkyDRL method achieved an average QoS value of 39.1, which is slightly lower than PD-DQN and SFGA. This can be attributed to the fact that SkyDRL aims to balance both the QoS optimization and energy consumption of service composition, resulting in a trade-off between the two metrics. However, SkyDRL still performed well in optimizing the overall QoS of service composition, indicating its effectiveness in solving service composition problems. Similarly, Figure 9 shows that the SkyDRL method obtained an average QoS value of 30.795, while PD-DQN, Q-learning with skyline, SFGA, and MAACS achieved averages of 31.86, 25.21, 31.42, and 30.76, respectively. These results demonstrate that SkyDRL is effective in optimizing QoS values of service composition.
composition for varying problem scales. Although SkyDRL’s QoS performance is not the best among the evaluated methods, it is important to note that SkyDRL also reduces energy consumption in addition to QoS optimization. Therefore, a certain decrease in QoS values for service composition could be a reasonable trade-off. Overall, the experimental results confirm that SkyDRL is an effective method for solving service composition problems with good performance in optimizing QoS and reducing energy consumption.

![QoS for PD-DQN, Q-learning with skyline, SFGA, MAACS, and SkyDRL](image)

**Figure 8.** QoS for PD-DQN, Q-learning with skyline, SFGA, MAACS, and SkyDRL, with 60 subtasks but different numbers of candidate services for each subtask.

![QoS for PD-DQN, Q-learning with skyline, SFGA, MAACS, and SkyDRL](image)

**Figure 9.** QoS for PD-DQN, Q-learning with skyline, SFGA, MAACS, and SkyDRL, with 200 candidate services but different numbers of subtasks.

### 5.4. Adaptability

The multi-cloud environment is a complex and dynamic system where many factors can influence the outcome of service composition. For instance, the QoS attributes of services may change due to a variety of reasons, such as seasonal factors or internal issues specific to the service provider. Additionally, certain services may become temporarily unavailable, further complicating service composition. In such scenarios, it is crucial to have a method that can identify the optimal solution within a given timeframe without requiring manual parameter adjustments or human intervention. This ability is known as adaptability.
To evaluate the adaptability of our proposed approach, we conducted an experiment using 60 subtasks and 200 candidate services for each subtask. To simulate the dynamic changes in the cloud environment, we followed the methodology outlined in the references [39,43] by randomly disabling a certain proportion of cloud services. Figure 10 shows the experimental results obtained. After the algorithm reached a stable state, we randomly disabled a certain percentage of cloud services in four different scenarios: no disabled services and 1%, 5%, and 10% of randomly disabled services. This approach forced SkyDRL to readjust and learn, finding a new service composition. As seen from Figure 10, the higher the percentage of disabled cloud services, the greater the fluctuations in the algorithm and the longer the adjustment time. Moreover, when a high percentage of services was disabled, such as 10%, the QoS value of the newly found service composition decreased. This was due to the loss of high-QoS services as a result of disabling some services, which affected the final QoS value of the selected service composition. Conversely, when a small percentage of services was disabled, such as 1%, the impact on the algorithm’s performance was minimal and had little influence on the final outcome. Nevertheless, SkyDRL demonstrated its adaptability by readjusting its strategy and finding a new service composition, regardless of the percentage of disabled services.

![Figure 10](image)

**Figure 10.** Adaptability of SkyDRL in solving MCECSC when certain percentages of services are made unavailable.

Figure 11 displays the QoS values of the newly found service composition after disabling different percentages of cloud services. Unlike heuristic algorithms designed for specific problems that require modifications and parameter adjustments when the environment changes, DRL can learn the optimal policy through training and update weights adaptively based on the environment. As shown in Figure 11, as the percentage of disabled services increased, the QoS values of heuristic algorithms dropped significantly due to environmental changes. In contrast, algorithms based on reinforcement learning were less affected by environmental changes, with the QoS values decreasing only slightly.
Figure 11. Performance of different methods in solving MCECSC when certain percentages of services are made unavailable.

5.5. Execution Time

In the context of cloud service composition, the ability to generate cloud service composition quickly is critical. To evaluate the efficiency of our proposed approach, we fixed the number of candidate services for each subtask to 200, which corresponds to scenarios with four different subtasks, 20, 40, 60, and 80.

Figure 12 presents the execution time of several algorithms. The results show that SkyDRL outperforms other algorithms in terms of execution time, indicating that it takes less time to generate cloud service composition. This is due to the utilization of a pre-trained neural network model and the use of skyline to reduce the search space, which allows SkyDRL to swiftly generate new service composition solutions. In fact, using a pre-trained deep reinforcement learning model for offline decision-making is one of the effective measures to reduce execution time. Additionally, while the required time increases with the size of the problem, the method’s execution time consistently remains good.

Figure 12. Average execution time.
5.6. Scalability

Scalability is a crucial aspect of service composition methods, and we verify the generalization ability of our proposed approach by testing it on different experimental scenarios. To achieve this, we refer to [39,43] and create two types of scenarios. The first scenario involves fixing the number of subtasks to 30, and each subtask contains 300, 400, 500, and 600 candidate cloud services, respectively. The second type fixes the number of candidate services to 500, corresponding to 50, 60, 70, and 80 subtasks. We evaluate the scalability of SkyDRL by observing the execution efficiency of the algorithm for problems of different scales. As demonstrated in Table 3, SkyDRL can always find optimal cloud service compositions, regardless of the scale of the problem. This indicates that the proposed approach has good scalability.

Table 3. Scalability test.

<table>
<thead>
<tr>
<th>Number of Subtasks</th>
<th>Number of Candidate Services</th>
<th>QoS Value</th>
<th>Number of Subtasks</th>
<th>Number of Candidate Services</th>
<th>QoS Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>300</td>
<td>20.63</td>
<td>50</td>
<td>500</td>
<td>29.13</td>
</tr>
<tr>
<td>30</td>
<td>400</td>
<td>19.79</td>
<td>60</td>
<td>500</td>
<td>36.10</td>
</tr>
<tr>
<td>30</td>
<td>500</td>
<td>21.38</td>
<td>70</td>
<td>500</td>
<td>42.27</td>
</tr>
<tr>
<td>30</td>
<td>600</td>
<td>21.51</td>
<td>80</td>
<td>500</td>
<td>49.69</td>
</tr>
</tbody>
</table>

The experimental analysis above provides compelling evidence that SkyDRL is a powerful tool for achieving energy consumption and QoS balance in service composition. Although SkyDRL may not deliver optimal results for optimizing QoS effectiveness since it accounts for energy consumption, its overall performance remains strong. When facing a dynamic environment, SkyDRL showed good adaptability, which proves that SkyDRL can adaptively adjust its learning strategy to find the optimal service composition. Moreover, SkyDRL delivers excellent outcomes for problems of different scales, underscoring its scalability.

6. Conclusions

With the increasing popularity of cloud computing, there has been an influx of cloud service providers offering similar cloud services but differing in QoS attributes. Moreover, the number of clouds has increased, resulting in scenarios where the required services for a request are spread across multiple clouds, making cross-cloud scheduling more common. In such multi-cloud environments, cloud service composition faces complex situations, such as additional energy consumption due to data transmission and increased variability from multiple clouds. Therefore, this paper proposes the skyline-enhanced deep reinforcement learning approach to minimize the energy consumption of cloud service composition in a multi-cloud environment while optimizing QoS. We define an energy consumption model for cloud service composition in a multi-cloud environment and incorporate it into the reward function to guide the agent’s choice of low-energy services. To reduce the algorithm execution time and simplify optimization difficulty, we integrate the branch and bound skyline algorithm to effectively narrow the search space. We improve the basic DQN algorithm by incorporating double DQN to address overestimation problems, utilizing Dueling Network and Prioritized Experience Replay to accelerate training speed and enhance stability. We conduct a series of experiments to validate the effectiveness of our proposed method in reducing energy consumption while performing well in terms of execution time, adaptability, computation time, and scalability in a multi-cloud environment. The experimental results demonstrate that SkyDRL effectively reduces energy consumption and finds relatively optimal cloud service composition solutions in multi-cloud environments.
Although our method has achieved good results, we are also aware that there are still some potential limitations. For example, since the QoS of services is usually dynamically changing, in the future, we plan to use QoS prediction techniques to further improve the flexibility and practicality of the method. In addition, we also hope to introduce multi-agent technology to improve learning efficiency through communication between multiple agents in order to further improve the performance of our method. We also intend to address the uncertainty factors of cost, response time, and reliability in service composition. To model these factors, we plan to employ the method of intuitionistic fuzzy sets.

Author Contributions: Writing—original draft, W.M.; Writing—review & editing, H.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Scientific Research Plan Projects of the Education Department of Jiangxi Province of China, grant No. GJJ160554, the Talent Plan Project of Fuzhou City of Jiangxi Province of China, grant No. 2021ED008, and the Opening Project of Jiangxi Key Laboratory of Cybersecurity Intelligent Perception, grant No. JKLCP202202.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data used in this study can be downloaded from https://qwsdata.github.io.

Conflicts of Interest: The authors declare no conflict of interest.

References

8. Wang, H.; Gu, M.; Yu, Q.; Tao, Y.; Li, J.; Fei, H.; Yan, J.; Zhao, W.; Hong, T. Adaptive and large-scale service composition based on deep reinforcement learning. Knowl. -Based Syst. 2019, 180, 75–90. [CrossRef]
27. Tarawneh, H.; Alhaidid, I.; Khwaldeh, S.; Afaneh, S. An intelligent cloud service composition optimization using spider monkey and multi-stage forward search algorithms. Symmetry 2022, 14, 82. [CrossRef]
35. Bouzary, H.; Chen, F.F.; Shahin, M. Natural language processing for comprehensive service composition in cloud manufacturing systems. Procedia Manuf. 2021, 55, 343–349. [CrossRef]


**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.