

Proactive Service Composition Based on Deep Reinforcement Learning

Xue Li

Anhui Automotive Vocational and Technical College, Hefei, Anhui, China

Abstract: With the development of cloud-network integration technology, how to compose cloud-network services with similar functions but inconsistent Quality of Service (QoS) to meet the diverse needs of users in different scenarios, such as space, air, land, and sea, has become an important research topic in the field of service computing. However, traditional service composition approaches ignore the importance of actively selecting the nearest server for service composition by perceiving user's mobility, which may lead in delayed delivery of results to users after service composition is completed. Moreover, these approaches ignore the geographic spatial characteristics in the cloud-network environment. When facing a massive number of cloud-network services, individual servers may struggle to meet users' QoS requirements, leading to inefficient proactive service composition. To address these issues, this paper proposes an innovative service composition approach, called Proactive Service Composition Based on Deep Reinforcement Learning (PSCDRL). In this method, a Long Short-Term Memory (LSTM) models are constructed using user's location data to predict user trajectory, enabling the early acquisition of user positions and proactive select the nearest servers based on the predicted locations. In the process of service composition, according to the selected server, the deep reinforcement learning (DRL) method is used to perform service composition under the cooperation of multiple servers. The experiment demonstrated the feasibility and effectiveness of the method.

Keywords: Service Composition; LSTM Neural Network; Deep Reinforcement Learning.

1. Introduction

With the in-depth development of cloud-network integration technology and the widespread use of service-oriented architecture, the number of various services has grown exponentially. At the same time, the homogenization phenomenon among services has become increasingly severe, leading to an increasing number of service providers offering services with similar functionalities but varying non-functional attributes. However, in this environment, service composition is one of the important elements for achieving efficient service delivery. Service composition technology can combine multiple heterogeneous services according to specific constraints based on user needs, forming a holistic solution [1,2], thereby meeting the complex user requirements in a specific environment. In such an environment, the collaborative service composition of edge servers holds immense potential to further enhance service quality, reduce latency, and decrease network load, among others [3]. Therefore, in the face of a vast array of services in a cloud-network environment, it is of great significance to utilize multiple edge servers to collaborate in selecting appropriate services for users and performing service composition [4].

At present, the most common method for Quality of Service (QoS) aware service composition [5] is to measure the quality of composite services through QoS[6-8], such as throughput, response time, reliability, and availability. Based on this type of method, researchers have proposed many composition methods, which include integer programming, graph programming, evolutionary algorithms, reinforcement learning and deep reinforcement learning. For example, Ardagna et al. [9] proposed a new modeling method to achieve service selection and composition, which formalizes the Web service selection problem as a mixed integer linear programming problem and can flexibly map abstract services

to concrete services. Beauche et al. [10] proposed a technique for integrating adaptive features in the process of service composition, which is achieved by combining semantic description and hierarchical planning. However, changes in services and environment need to respond to model updates and are not suitable for scenarios with a large number of services. Silva et al. [11] used genetic programming for service composition, studied three different genetic programming methods, and optimized them based on QoS to obtain the correct solution, but these methods are only applicable to small-scale service environments.

In addition to the aforementioned research, efforts have also been made to introduce reinforcement learning and deep reinforcement learning into service composition. Wang et al. [12] proposed a service composition method based on Markov decision processes. This method only utilizes reinforcement learning methods and cannot handle large-scale service composition problems. Ren et al. [13] modeled the service composition that satisfies constraints as a Markov decision process and designed the Q-Learning algorithm to solve the model. The above service composition methods demonstrate high algorithmic efficiency in handling small-scale service composition problems. However, there are still some issues when facing large-scale service composition problems. Wang et al. [14] proposed an adaptive deep reinforcement learning framework to ensure the adaptability and efficiency of large-scale service composition. This method integrates reinforcement learning and deep learning, which helps to achieve adaptive service composition and enhance expressive and generalization abilities. Moustafa et al. [15] proposed a method based on deep reinforcement learning to achieve adaptive service composition in large-scale service environments. On the basis of deep reinforcement learning, dual Q-learning techniques are used for enhancement, which solves the problem of overestimation bias in Q-learning. At the same time, a priority experience

replay scheme is also implemented, allowing the agent to sample from higher priority experiences and learn more from them. From the above related studies, it can be seen that deep reinforcement learning has achieved some results in large-scale service scenarios. However, in the context of cloud-network integration, traditional service composition methods do not consider the perceived mobility of users, and cannot proactively select the nearest server for service composition in advance. They ignore the geographic spatial characteristics of service composition and cannot solve the problems of delivery failure and long delay time in composite services. To solve the above problems, this paper proposes a Proactive Service Composition Based on Deep Reinforcement Learning (PSCDRL) method. The main contributions are summarized as follows:

(1) By fully considering the mobility characteristics of users in the cloud-network integration environment, predicting user locations and proactively selecting the nearest server for service composition in advance, we ensure that the composed services can be delivered directly and quickly to users, thereby effectively reducing service delivery latency and improving service response speed.

(2) Considering how latency calculations are affected by distance, we constructed a quadratic function model to accurately describe the complex relationship among distance, network congestion, and data transmission rates. Then, we utilize deep reinforcement learning to perform service composition in a multi-edge server environment, thereby delivering a composed service that meets user demands while minimizing latency.

(3) This paper constructs a simulated cloud-network integration environment using three real datasets for experiments. Compared with the optimal method in different service scale scenarios, the total revenue of the composed service achieved performance improvements of 4.07%, 8.63%, 18.29%, and 5.24%, respectively.

2. Preparatory Knowledge

2.1. LSTM Neural Network

As an improved variant of recurrent neural networks (RNNs), the core design goal of LSTM is to alleviate the inherent problem of traditional RNNs—namely, the difficulty in modeling long-range dependencies when processing long sequences due to gradient vanishing or gradient exploding—and it is widely applicable to tasks such as speech, text, video, and time series [16]. LSTM achieves selective retention and forgetting of information through a gating mechanism, specifically comprising three core components: the forget gate, the input gate, and the output gate. Firstly, the forget gate, based on the previous hidden state h_{t-1} and the current input x_t , generates a forget weight f_t through the sigmoid function, determining the proportion of information retained from the previous cell state C_{t-1} , that is, selectively discarding redundant or outdated memories:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (1)$$

Next, the input gate collaboratively generates new candidate information. This process is divided into two steps: the sigmoid layer outputs i_t , which is used to determine the cell state component to be updated; the tanh layer generates the candidate vector C_t , providing the specific numerical value of the new information:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (3)$$

Based on the aforementioned forget and input gating results, the old cell state C_{t-1} is updated to the current state C_t . The update method involves the element-wise product of the forget gate and the old state, plus the element-wise product of the input gate and the candidate state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (4)$$

Finally, the output gate controls the influence of the current cell state on the external output h_t . The output gate first calculates the output gating signal o_t using the sigmoid function, and then multiplies the cell state after scaling with tanh to obtain the hidden state at the current time step:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

Among them, all gating signals take values in $[0, 1]$, thereby ensuring the differentiable regulation of information flow.

2.2. Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) uses deep neural networks to handle high-dimensional state and action spaces, in order to learn complex policies and value functions. It has demonstrated excellent performance in decision-making tasks such as gaming and robot control [17]. Deep Q-Network (DQN) is an algorithm of DRL, proposed by DeepMind in 2013 [18]. The training of DQN relies on two techniques: experience replay and fixed target network. Experience replay stores interaction records in an experience pool, and after accumulating a certain number, randomly samples them for network training, thereby breaking data correlation, reducing uncertainty, and avoiding local optima. The fixed target network uses two neural networks with the same structure, one to estimate the current Q-value and the other to calculate the target Q-value. The target network is updated periodically to reduce the correlation between the target value and the estimated value and improve training stability. The calculation method of the Q-value function is as follows:

$$Q(s_t, a_t) = r + Q'(s_{t+1}, a_{t+1}) \quad (7)$$

Among them, the left side is the network output, and the right side is the target value. The purpose of introducing a target network is to fix the target value and minimize the mean square error between the model output and the target, in order to improve training efficiency and stability.

3. System Model

3.1. Problem Modeling

As shown in Fig 1 Assuming that user Tom attempts to request the nearest edge server S1 to make a service call while walking home near A point. Due to the complexity of user service requests, the services deployed in a single edge server cannot meet the service requirements and require access to other edge servers to invoke services for service composition, such as S2 and S3. During the service composition process, when the distance between the user and edge server S1 increases to a certain extent, that is, when the service requester moves out of the coverage area of the server receiving the request, the user cannot receive the composite service. Therefore, this paper will predict the user's future location L at a certain time when the user makes a request. Assuming that the edge server closest to the user is selected for service composition at location L, the service composition will be completed and delivered to the user at location L to receive the composite service for invocation.

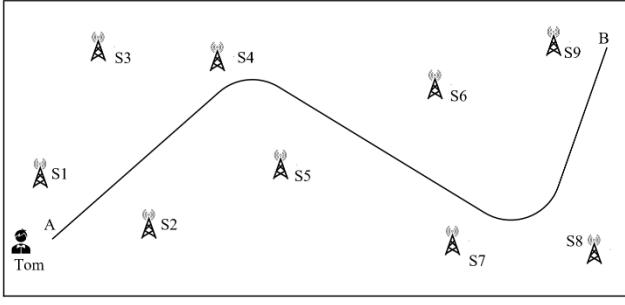


Fig 1. User mobile scenario

This paper begins by breaking down the composite task into several subtasks. Then, the service composition problem

is modeled as a Markov Decision Process (MDP) to provide a theoretical foundation for subsequent algorithmic solutions. This paper assumes that the service composition process of a sequential structure with a known number of subtasks needs to be solved. Then transform the problem into MDP, defined as a quintuple: $MDP = \langle S, A, P, R, G \rangle$. Each of these parameters represents, respectively, the set of states in the service composition process, the set of actions that can be taken when the state is $s \in S$, the state transition function, the reward function, and the total revenue.

3.2. Overall Framework

The overall framework of the proactive service composition method in advance is shown in Fig 2.

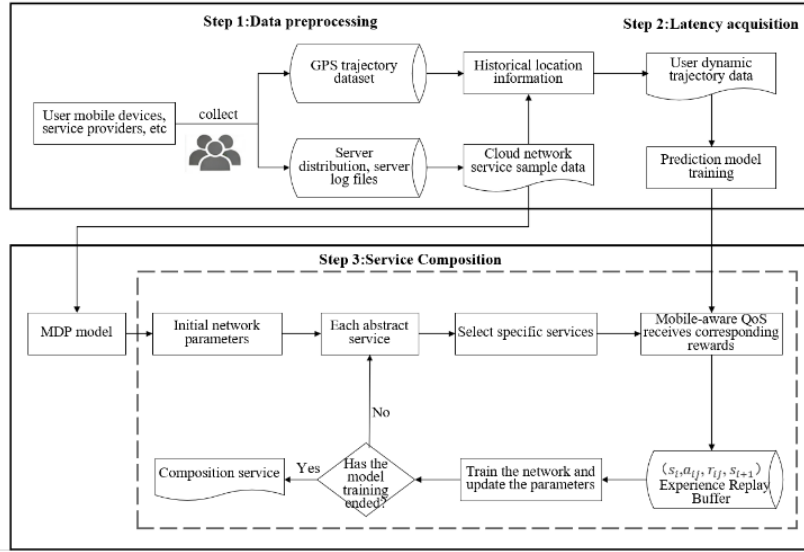


Fig 2. Methodology chart

(1) Data preprocessing. Firstly, due to the varying signal strengths at different locations within the coverage area of the server, the stronger the signal strength, the higher the data transmission rate [19-21]. Therefore, this paper designs a quadratic function to represent the relationship between server distance, network congestion, and data transmission rate. Different distances and levels of network congestion have different data transmission rates. The service composition process in cloud-network integration environment needs to consider the changes in server location and service invocation during user movement. The existing datasets do not meet such requirements, so this paper needs to build datasets for mobile users and servers in cloud-network integration environments. At the same time, it is necessary to filter out invalid data, such as sample data with excessively long service execution time, in order to make the experimental data more in line with the real situation.

(2) Latency acquisition. Simulate the user's movement path based on their historical location. This paper designs an LSTM model with self-attention mechanism, which can predict the user's position at a certain moment in the future, and obtain edge servers that may perform service composition at a certain moment in the future based on the predicted position. Then, based on the designed quadratic function, the data transmission rate with the server can be determined, and the latency of user service calls can be determined through this method.

(3) Service composition is implemented using deep reinforcement learning. When a user initiates a service

request, the system selects the nearest server as the local node to execute the composite task based on predictions of the user's future location. If the local server cannot fully meet the user's complex needs, it sends a collaboration request to neighboring servers to obtain the required services and complete the call; the composite results are then aggregated locally before being delivered to the user. Under this multi-edge server collaboration, the deep reinforcement learning model is trained using the inverse of latency as a reward signal, thereby selecting the service composition with the shortest total latency while meeting user needs.

4. Method Design

4.1. Latency Acquisition

Firstly, obtain the user's historical location to simulate their movement path. Then, predict the users' future locations through a LSTM model based on a self-attention mechanism, and calculate the distance between it and other servers to obtain the data transmission rate of the server, and the nearest server is selected for service composition. The model architecture is shown in Fig 3. It consists of two LSTM layers (each with 120 units), with a self-attention mechanism connected after the second layer. The self-attention layer applies weights to key positions in the trajectory to generate weighted representations. Higher-order features are then extracted via a ReLU fully connected layer, and finally, the output layer predicts the future position sequence.

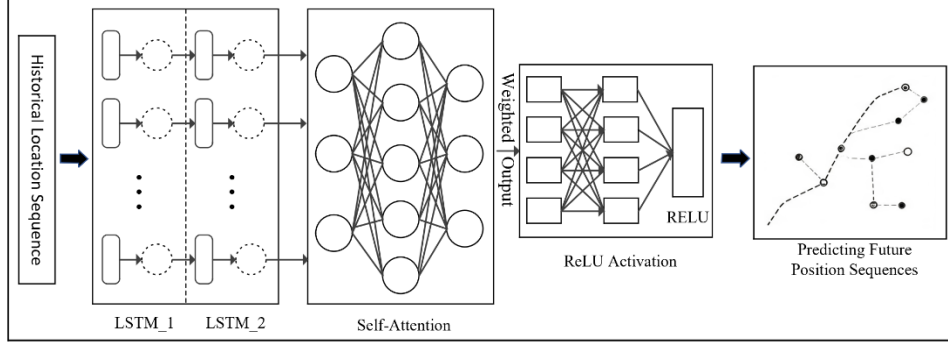


Fig 3. Architecture diagram of location sequence prediction model

In the aforementioned model, LSTM incorporates the self-attention mechanism, resulting in improved prediction performance, with the goal of parameter optimization being to minimize the deviation between the predicted position and the actual observed position, thereby establishing a mapping relationship from historical trajectory sequences to future movement trends. Based on the execution cycle of the service portfolio, this paper selects the most recent n historical time points as the input window; these continuous position data points are shown in formula (8):

$$Y(t+1) = f(\{Y(t-n+1), \dots, Y(t-1), Y(t)\}) \quad (8)$$

Within the server's coverage area, signal strength varies across different geographic locations, and signal strength is positively correlated with transmission rate. Based on this, this paper introduces a quadratic function relating the user's predicted location to the distance from the server to estimate the data transmission rate. Additionally, in a cloud-network integration environment, network congestion levels may vary when users initiate service calls from different locations; therefore, the congestion level is set as a constant parameter in this rate function, specifically as follows:

$$\text{rate} = a - b * \left(\frac{d}{2000}\right)^2 - m \quad (9)$$

This formula applies to both speed calculations between clients and servers and between server nodes. Following the parameter settings in Reference [20], on the client-server link, a and b are set to 100 MB/s and 20 MB/s, respectively; on the inter-server link, a and b are set to 3 MB/s and 2 MB/s, respectively. In the equation, m represents the degree of network congestion, and its value can be flexibly adjusted based on actual network conditions.

4.2. Service Composition Approach

For the constructed MDP model, a reward function suitable for service composition scenarios needs to be defined. If the invoked service is on the local server e , the latency calculation is as shown in equation (10), where U_{ei} and U_{eo} represent the input and output data volume required for the service, α_{ei} and α_{eo} are the transmission rates between the user and the local server e , and E_e is the service execution time. Based on these parameters, the corresponding delay can be calculated and used as part of the reward function to find the composition service with the shortest delay in the deep reinforcement learning algorithm.

$$t = \frac{U_{ei}}{\alpha_{ei}} + E_e + \frac{U_{eo}}{\alpha_{eo}} \quad (10)$$

If the invoked service is not on the local server e , the delay calculation is as shown in equation (11), where β_{en} and β_{ne} represent the average transmission rates from local server e and other servers n , while U_{en} and U_{ne} represent the amount

of data that service s needs to transmit.

$$t = \frac{U_{ei}}{\alpha_{ei}} + \frac{U_{en}}{\beta_{en}} + E_{ns} + \frac{U_{ne}}{\beta_{ne}} + \frac{U_{eo}}{\alpha_{eo}} \quad (11)$$

Since the smaller the latency of the service composition, the better. In order to find the composition service with the shortest latency, this paper can use the reciprocal of the latency as a reward and maximize the reward value through deep reinforcement learning algorithms. The calculation of the total reward is as follows:

$$R = \sum_{i=1}^m r_i \quad (12)$$

Among them, $r = \frac{1}{t}$, r_i represents the service reward corresponding to the i -th subtask, with a total of m subtasks. This reward function can optimize latency in the service composition process and provide the best composition of services that meet user needs.

Finally, in order to solve MDP, use the Deep Q-Network (DQN) model as the basis of deep reinforcement learning. Its Bellman optimal equation (the basis for iterative Q-value updates) is given by the following equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \quad (13)$$

In this equation, r represents the immediate reward, γ is the discount factor, and s' is the next state. Based on this, the Q-value is updated by calculating the variance, which is defined as follows:

$$L = E[(r + \gamma \max_{a'} Q(s', a') - Q(s, a))^2] \quad (14)$$

Next, the weights of the Q-network are updated using the gradient descent method, thereby enabling the gradual optimization of the DQN model. The gradient for updating its parameters is shown in the following equation:

$$\frac{\partial L(\theta)}{\partial \theta} = E[(r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \frac{\partial Q(s, a; \theta)}{\partial \theta}] \quad (15)$$

Through iterative training, we can obtain the optimal Q-value, thereby finding a set of composed services that meet user needs and have the shortest latency, bringing users a good experience.

5. Experimental Verification

5.1. Experimental Environment Setting

The experiments in this paper were conducted on an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz under the Ubuntu 18.04 operating system. In order to simulate service composition s of different scales, we set the number of abstract services to 10, 20, 30, and 40, with 50 executable candidate services under each abstract service. In this study, we used the Python programming language to design the proposed method and effectively implemented deep reinforcement learning algorithms using the TensorFlow deep learning framework. During the training process of the deep

reinforcement learning algorithm, we conducted 5000 iterations with an initial learning rate of 0.01 and a return discount factor of 0.9. The coverage area of each edge server is approximately 2000 meters in space, including land, air, and ocean domains.

5.2. Experimental Dataset

This paper uses three datasets to construct an edge environment based on the sky, earth, and sea scene to validate the proposed method.

Dataset 1 is the Shanghai Telecom dataset [22-24], which contains geographic location information, access user IDs, and access times for 3233 base stations. Dataset 2 is based on the GPS trajectory dataset, collected by 182 users over a period of more than 5 years in the Geolife project. This dataset contains 17621 GPS trajectories, each represented by a sequence of time stamped points, each point containing latitude, longitude, and altitude information. Dataset 3 is a partial server log file collected by a research team from Alibaba and Zhejiang University [25-27]. This dataset contains 500 services deployed on 10 servers, with an average of 50 services deployed on each server. Each service contains six pieces of information: the server it is on, the start time of service operation, the amount of input data, the amount of output data, the end time of service operation, and the server deployed for the next subtask.

Due to the need to consider changes in user location in cloud-network integration environments, the latency of calling services may also vary. Traditional datasets cannot meet the requirements, so this paper uses dataset fusion [28] to obtain a dataset that meets the requirements. Firstly, based on the geographical location of the base station, a unique base station number is assigned to each base station in the dataset. Then, based on the service start time, service input data volume, service output data volume, and service end time information contained in each service in dataset 2, they are associated according to the base station number, and the

corresponding service information is added to the base station location information to form a new dataset. However, there are a large number of services in the cloud-network environment. For large-scale service scenarios, the fused dataset only has 3190 service information, which cannot effectively verify the advantages of our method in large-scale scenarios. Therefore, we need to expand the dataset to a certain extent. Based on the distribution of various attribute values (such as each column) in the dataset, we use a global quality constraint to set random values to generate attribute values. Through this method, the dataset includes the geographic locations of 1000 edge servers, with 50 services on each server, each service including service execution time, service input data volume, and service output data volume.

5.3. Experimental Results and Analysis

In this paper, we use the total revenue of the composition service as an evaluation metric to assess the performance of service composition. During the hyperparameter tuning process for deep reinforcement learning, we first sequentially investigate the effects of the initial exploration rate ϵ , batch size b , and experience replay pool capacity p on the total revenue. The experiments are conducted in an environment consisting of 10 abstract services (each with 50 candidate services). To make the results more intuitive, the total reward obtained in each iteration is multiplied by 10 before being reported.

(1) First, we fixed the batch size at $b=128$ and the empirical pool capacity at $p=300$, while setting the exploration rate decay rate to 0.0008. During the iteration process, the exploration rate was gradually reduced to 0.01. The experimental results show that when ϵ is set to 0.3, 0.5, 0.7, and 0.9, respectively, the total revenue during the convergence phase is approximately 22 at low exploration rates, while it reaches 23 at high exploration rates 0.7 and 0.9. Considering that increasing exploration during the early training phase can enrich sample diversity, we set ϵ to 0.9.

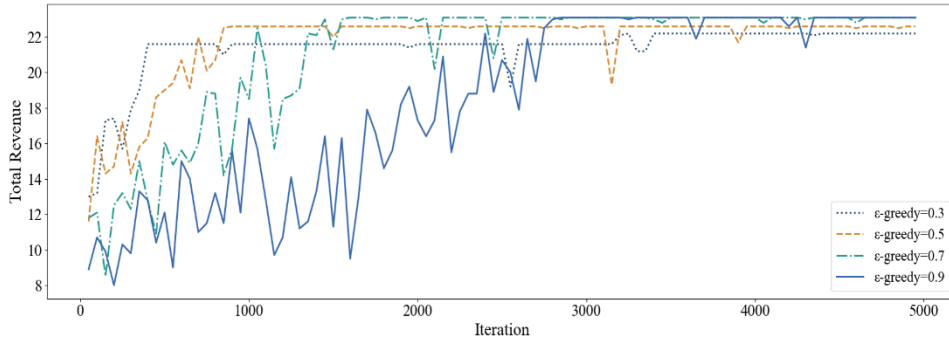


Fig 4. Exploration rate parameter experiment

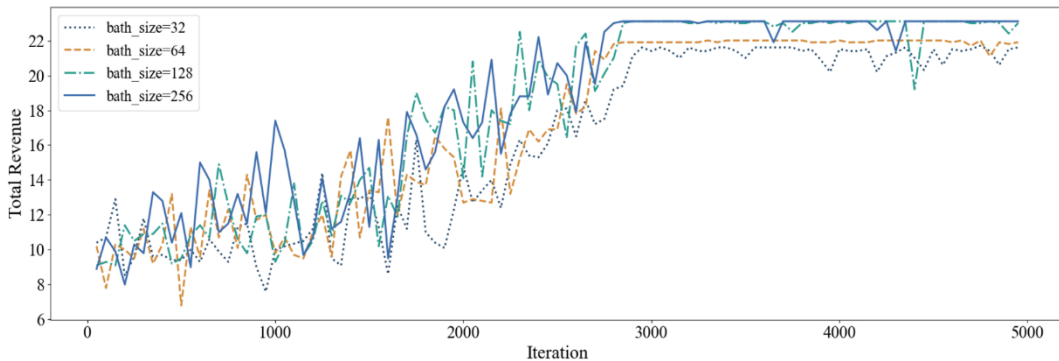


Fig 5. Batch size parameter experiment

(2) The batch size b refers to processing multiple samples in parallel to speed up model training. The figure below shows the total revenue for b values of 32, 64, 128, and 256. The exploration rate in the experiment was set to 0.9. The results show that convergence was unstable when b was set to 32. When b was set to 64, the model converged with total revenue of 21.9; when b was set to 128 and 256, the total revenue was

23, but convergence was faster when b was set to 256. Therefore, b was set to 256 for subsequent experiments.

(3) The capacity of the experience pool determines the amount of data fed into the network training each time and plays an important role in model convergence. The chart below shows the total revenue when the experience pool capacity is 300, 400, and 500, respectively.

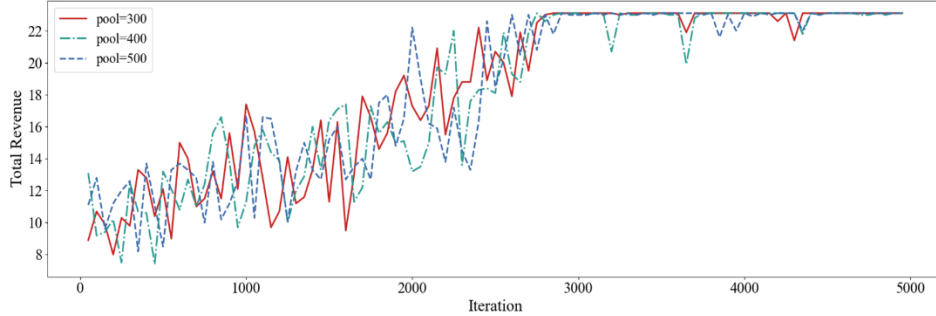


Fig 6. Experience pool parameter experiment

In the experiment, the exploration rate was 0.9 and b was 256. The experimental results showed that when the experience pool size was 400, the convergence results of the model were unstable. When the experience pool sizes were 300 and 500, the total revenue during convergence were both 23. However, in order to save resources, this paper believes that the experience pool size for subsequent experiments should be set to 300

(4) In this experiment, the effectiveness of active service

composition was also verified after the hyperparameters were determined. Fig 7 shows a comparison between direct service composition and predicting user location, with proactive service composition in advance. Based on the predicted user location, select the nearest server in advance for service composition, ensuring that the user moves to the vicinity of the server where the service composition is being performed within the completion time and issuance time of the service composition.

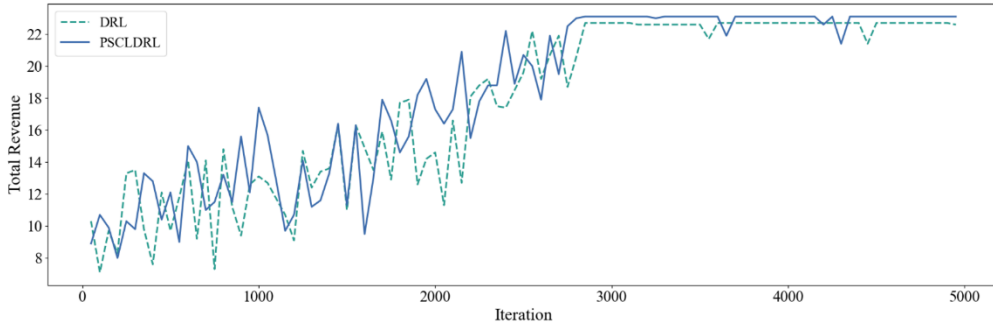


Fig 7. Effectiveness of proactive service composition s in advance

The greater the total revenue of a composition service, the smaller its latency. According to the experimental results above, it can be seen that as the number of iterations increases, when the number of iterations is 2850, the total revenue of active service composition reaches 23.1, and the total revenue of not composing services in advance is 22.7. This is because when the service composition is completed, the user has already moved out of the coverage area of the local server and cannot directly deliver the composed services from the local server to the user Causing an increase in routing latency.

(5) This experiment validates the overall performance of active service composition, as shown in Fig 8. By comparing the total latency benefits of the final composed services, the experiments compared the Particle Swarm Optimization (PSO) algorithm, the Reinforcement Learning (RL) algorithm, and the Deep Reinforcement Learning algorithm with random server selection (Random) across service scenarios of varying scales.

According to the experimental graph, it can be seen that the total revenue of active service composition is the highest, the total revenue of service composition based on the particle

swarm optimization algorithm is the lowest, followed by the total revenue of service composition based on reinforcement learning and deep reinforcement learning algorithm with randomly selected servers

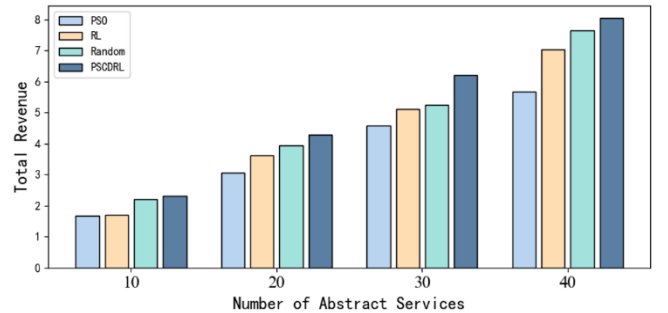


Fig 8. Comparison of total revenue of different algorithms

As the number of abstract services increases, the difference in total revenue between PSO and PSCDRL becomes larger and larger. Because as the number of abstract services continues to increase, the total service portfolio solution

grows exponentially, and PSO is not efficient in scenarios with multiple services.

6. Conclusion

This paper analyzes the relevant research on existing service composition methods, elaborates on the limitations of service composition in the cloud-network integration environment, and proposes an active service composition method based on LSTM model and deep reinforcement learning. Firstly, an LSTM model is constructed to predict the user's motion trajectory, thereby obtaining the user's location information in advance, and actively selecting the nearest server based on the predicted location. Then, a quadratic function was designed to calculate the data transmission rate between the user and the server, as well as between the servers. Finally, utilizing deep reinforcement learning algorithms for service composition in collaboration across multiple servers. The experimental results show that the service composition method proposed in this paper can deliver the service composition results to users in a timely manner, reduce the latency of composite services, and achieve better experimental results on public datasets than the current baseline method, making it possible to significantly improve the performance of service composition in cloud network environments.

For future work, the following tasks will be carried out: 1) Improve the method proposed in this article to adapt to multi-user service composition; 2) Design more accurate user trajectory prediction models; 3) Consider more indicators to measure the quality of composite services.

References

- [1] Billhardt, H., Hermoso, R., Ossowski, S., & Centeno, R. (2007). Trust-based service provider selection in open environments. *Proceedings of the 2007 ACM symposium on Applied computing*, 1375-1380. <https://doi.org/10.1145/1244002.1244298>.
- [2] Vakili, A., & Navimipour, N. J. (2017). Comprehensive and systematic review of the service composition mechanisms in the cloud environments. *Journal of Network and Computer Applications*, 81, 24-36. <https://doi.org/10.1016/j.jnca.2017.01.005>.
- [3] Rao, J., & Su, X. (2004). A survey of automated web service composition methods. *International workshop on semantic web services and web process composition*, 43-54. Springer Berlin Heidelberg.
- [4] Ismael, A. R., Moayad, A., Yehia, K., Yousif, A. R., & Yaser, J. (2018). A collaborative mobile edge computing and user solution for service composition in 5G systems. *Transactions on Emerging Telecommunications Technologies*, 29(11), e3446. <https://doi.org/10.1002/ett.3446>.
- [5] Strunk, A. (2010). QoS-aware service composition: A survey. *2010 Eighth IEEE European Conference on Web Services*, 67-74. <https://doi.org/10.1109/ECOWS.2010.18>.
- [6] Baresi, L., & Guinea, S. (2005). Towards dynamic monitoring of WS-BPEL processes. *Proceedings of the 3rd International Conference on Service-Oriented Computing*, 269-282. https://doi.org/10.1007/11596141_21.
- [7] Chan, K., Poernomo, I., Schmidt, H. W., & Jayaputera, J. (2005). A model-oriented framework for runtime monitoring of nonfunctional properties. *International Conference on the Quality of Software Architectures*, 38-52. https://doi.org/10.1007/11585597_4.
- [8] Sammapun, U., Lee, I., Sokolsky, O., & Regehr, J. (2007). Statistical runtime checking of probabilistic properties. *Proceedings of the 7th International Conference on Runtime Verification*, 164-175. https://doi.org/10.1007/978-3-540-73520-0_11.
- [9] Ardagna, D., & Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6), 369-384. <https://doi.org/10.1109/TSE.2007.1013>.
- [10] Beauche, S., & Poizat, P. (2008). Automated service composition with adaptive planning. *Service-Oriented Computing-ICSOC 2008: 6th International Conference*, 530-537. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-89648-3_41.
- [11] da Silva, A. S., Ma, H., & Zhang, M. (2016). Genetic programming for QoS-aware web service composition and selection. *Soft Computing*, 20(10), 3851-3867. <https://doi.org/10.1007/s00500-015-1754-0>.
- [12] Wang, H., Zhou, X., Zhou, X., Li, W., & Bouguettaya, A. (2010). Adaptive service composition based on reinforcement learning. *Service-Oriented Computing: 8th International Conference, ICSOC 2010*, 92-107. https://doi.org/10.1007/978-3-642-17313-2_8.
- [13] Ren, L., Wang, W., & Xu, H. (2017). A reinforcement learning method for constraint-satisfied services composition. *IEEE Transactions on Services Computing*, 13(5), 786-800. <https://doi.org/10.1109/TSC.2017.2778740>.
- [14] Wang, H., Gu, M., Yu, Q., Fei, H., Li, J., & Tao, Y. (2017). Large-scale and adaptive service composition using deep reinforcement learning. *Service-Oriented Computing: 15th International Conference, ICSOC 2017*, 383-391. https://doi.org/10.1007/978-3-319-69035-3_27.
- [15] Moustafa, A., & Ito, T. (2018). A deep reinforcement learning approach for large-scale service composition. *Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems*, 296-311. Springer-Verlag. https://doi.org/10.1007/978-3-030-01771-2_17.
- [16] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural Computation*, 31(7), 1235-1270. https://doi.org/10.1162/neco_a_01199.
- [17] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533. <https://doi.org/10.1038/nature14236>.
- [18] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with deep reinforcement learning [Preprint]. *arXiv*. <https://arxiv.org/abs/1312.5602>.
- [19] Xu, Z., Zou, G., Xia, X., Liu, Y., Gan, Y., Zhang, B., & He, Q. (2020). Distance-aware edge user allocation with QoE optimization. *2020 IEEE International Conference on Web Services (ICWS)*, 66-74. <https://doi.org/10.1109/ICWS.2020.00019>.
- [20] Wu, H., Deng, S., Li, W., Yin, J., & Zomaya, A. Y. (2019). Mobility-aware service selection in mobile edge computing systems. *2019 IEEE International Conference on Web Services (ICWS)*, 201-208. <https://doi.org/10.1109/ICWS.2019.00038>.
- [21] Deng, S., Xiang, Z., Taheri, J., Khoshkholghi, M. A., Yin, J., Zomaya, A. Y., & Dustdar, S. (2021). Optimal application deployment in resource constrained distributed edges. *IEEE Transactions on Mobile Computing*, 20(5), 1907-1923. <https://doi.org/10.1109/TMC.2020.2970698>.

- [22] Li, Y., Zhou, A., Ma, X., & Wang, S. (2022). Profit-aware edge server placement. *IEEE Internet of Things Journal*, 9(1), 55-67. <https://doi.org/10.1109/JIOT.2021.3082898>.
- [23] Guo, Y., Wang, S., Zhou, A., Xu, J., Yuan, J., & Hsu, C.-H. (2020). User allocation-aware edge cloud placement in mobile edge computing. *Software: Practice and Experience*, 50(5), 489-502. <https://doi.org/10.1002/spe.2785>.
- [24] Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., & Shen, X. (2021). Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. *IEEE Transactions on Mobile Computing*, 20(3), 939-951. <https://doi.org/10.1109/TMC.2020.2991513>.
- [25] Luo, Z., Li, Y., & Yin, J. (2015). A Framework for Transmission Cost Aware Service Selection. 2015 IEEE International Conference on Web Services, 503-510. <https://doi.org/10.1109/ICWS.2015.74>.
- [26] Luo, Z., Li, Y., & Yin, J. (2013). Location: A feature for service selection in the era of big data. 2013 IEEE 20th International Conference on Web Services, 515-522. <https://doi.org/10.1109/ICWS.2013.77>.
- [27] Chen, F., Dou, R., Li, M., & Wu, H. (2016). A flexible QoS-aware Web service composition method by multi-objective optimization in cloud manufacturing. *Computers & Industrial Engineering*, 99, 423-431. <https://doi.org/10.1016/j.cie.2015.12.018>.
- [28] Wang, S., Zhao, Y., Huang, L., Xu, J., & Yuan, J. (2019). QoS prediction for service recommendations in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127, 134-144. <https://doi.org/10.1016/j.jpdc.2019.01.009>.