HSC-Rocket: An interactive dialogue assistant to make agents composing service better through human feedback

Anonymous ACL submission

Abstract

Facing the current dynamic service environment, fast and efficient service composition has attracted great attention in recent years. Users prefer to express their personal requirements based on natural language, and their real-time feedback could better reflect the effect of service composition to a great extent. Consequently, this paper designs an interactive dialogue assistant, HSC-Rocket, to better provide service composition by considering human feedback. Firstly, we propose a human-computer interaction dynamic service composition algorithm based on reinforcement learning. The design of the reward mechanism considers the quality of service (QoS) and real-time feedback, which can more accurately meet the demands of users. Then, the functional requirements are analyzed through word embedding, to realize the dynamic composition of abstract and concrete services. Furthermore, we utilize the sample enhancement method to alleviate the issue of fewer sample data in the initial stage of user interaction, which improves the robustness of our system. Accordingly, we have implemented the HSC-Rocket prototype, which allows users to obtain multi-domain dialogue requirements. Extensive experiments on the RapidAPI dataset have demonstrated the superiority and effectiveness of the HSC-Rocket.

1 Introduction

In the current service-oriented environment, the types and quantity of services are growing massively. A single service may no longer meet the requirements of a complex business due to its functional limitations. Consequently, a variety of business requirements make it necessary to combine a single service to generate applications with rich functions. Based on the advantages of reusability and interoperability, service composition plays a vital role to combine multiple atomic services to deal with complex user requirements (Dustdar and Schreiner, 2005). Meanwhile, the Quality of Service (QoS) may also change dynamically over time due to the fluctuations in the heterogeneous service environment and user access mode. QoS mainly measures the nonfunctional attributes of services, including response time, availability, price, etc. As a result, it is imperative to design a service composition method that can adapt to the dynamic service environment (Sangsanit et al., 2018).

Intelligent service composition mainly analyzes QoS and then generates services that satisfy the users' requirements. As a typical machine learning technology for optimization in a dynamic environment, reinforcement learning can be well used in service composition. Wang et al. (2014) conducted a new model that integrates on-policy reinforcement learning and game theory, which keeps high efficiency when dealing with massive candidate services. However, it only considers local QoS constraints, while users may put forward the requirements of global QoS constraints, which results in inapplicable limitations. Subsequently, multi-agent method began to be applied to service composition, which decomposes the task into many sub-tasks and makes every agent focus on their sub-tasks (Wang et al., 2016). Nevertheless, this situation in the multi-agent environment is relatively complicated. And agents in such complex systems may impede one another with the increase of interaction.

The ultimate goal of service composition is to satisfy users, hence the direct functional requirements of users play a vital role in the result of service composition. In service composition, end-users will instinctively express their demands and feedback in natural language (Ito et al., 2020) (Campagna et al., 2019). Consequently, it is urgent to take the user's natural language-based feedback into account in service composition. With the proliferation of chat robots, dialogue systems (Li et al., 2018) (Xu et al., 2020) and assistants have attracted great attention in recent years (Siblini et al., 2021)

(Zhu et al., 2020). Some current popular methods based on the conversational system can effectively combine with the underlying services. However, these dialog systems are only concerned with abstract services but do not focus on the invocation of underlying concrete services and pay no attention to the QoS, which becomes impractical to some extent.

As far as we know, there are still fewer works to combine the dialogue mode with the service composition. By summarizing, we conclude the current issues and challenges. **Challenge 1**. In view of user's real-time feedback reflects the effect of service composition, therefore it is a challenge to consider users' demands and feedbacks to dynamically match with the underlying specific services. **Challenge 2**. The existing works only focus on the abstract service level but do not be indifferent to the underlying concrete service, which lacks of practicability. **Challenge 3**. There are only a few samples at the beginning of system interaction, which leads to slow convergence and increase the time complexity. **Challenge 4**. Most present conversational systems are only involved in specific fields, and lacks generality and scalability.

To deal with these challenges and issues, we develop HSC-Rocket, an intelligent service composition prototype based on user real-time interaction. Overall, this work makes the following contributions:

1. We propose a novel human-computer interactive dynamic service composition algorithm based on reinforcement learning. The reward mechanism is designed by considering the QoS and the real-time feedback, which can more accurately satisfy users' demands.

2. We analyze the functional requirements of users based on word embedding to complete the dynamic combination of abstract services and concrete services. Consequently, HSC-Rocket makes end-users access a wide collection of services from a single text-based user interface.

3. To address the issues of a few data samples in the initial stage of user interaction, this paper utilizes sample enhancement to alleviate it, which improves the robustness of the system to a certain extent.

4. HSC-Rocket derives its generality from Rockethouse, a service repository that contains interfaces in various fields. And we verify the effectiveness of the model through some actual scenarios.

2 Related work

We review the related works of dynamic service composition and the latest progress in the combination of dialogue system and service composition.

2.1 Dynamic service composition

Due to the complex network environment, QoS will change dynamically (Song et al., 2018)(Zheng et al., 2011). And it is hard to obtain the user's QoS preference, because they cannot determine their preference before the service is executed. Thus, most of the works are based on the assumption that users' preferences for QoS are known in advance. Yu et al. (2020) proposed a solution that can effectively model the uncertainty of services with fine-grained QoS attributes by training a DQN. Nevertheless, replacing the Q-table with two DQNs poses challenges to memory and time. For the constraint-satisfied service composition (CSSC) (Yuan et al., 2019) (Wang and Zhang, 2017), Ren et al. (2017) modeled the CSSC problem as a Markov decision process(CSSC-MDP), and designed a Q-learning algorithm. CSSC-MDP considers the uncertainty of QoS and service behavior. Unfortunately, it restricts the users' QoS requirements. In practice, users are not familiar with the QoS of service providers, such a scheme is no longer gets the desired result.

As a matter of fact, users' QoS are not easily available, hence recent works no longer restrict QoS. Alizadeh et al. (2020) proposed a vector-valued MDP approach for finding the optimal QoS-aware services composition, which applied for that the user's QoS preferences are unknown. But it limits the number of interactions with users and performs poorly with the number of user interactions becoming frequent. Also for unknown users' QoS issues, Zhao et al. (2017) applied a learning-to-rank algorithm, RankBoost, to automatically learn user preferences and the prioritization of preferences. Yet, due to the lack of historical data in the initial stage, this method behaves incapably. Meanwhile, it learns user preferences based on historical data and then combines services to meet user's needs, which often gets rigid results. In contrast, we take the timely feedback of users and current user preferences into account, so that the composite services are novel and real-time.

With the service environment gradually showing the high scalability and complexity, Moustafa and Ito (2018) adopt double Q-learning with a priority playback scheme for the dynamic and large-scale environment. Then, Wang et al. (2019) proposed a new scheme, which is suitable for partially observable environments. Nevertheless, the Recurrent Neural Network performs poorly in dealing with very complex state space. To reduce the computational complexity, Wang et al. (2020) and Hiratsuka et al. (2011) optimize the composition efficiency through skyline services. However, the heuristic algorithm in their paper relied heavily on the evaluation function which doesn't behave well when involving massive services. In our work, the proposed HSC-Rocket can solve this problem effectively by utilizing deep reinforcement learning.

2.2 Integration with Conversation system and service composition

With the rapid development of machine learning and natural language processing, expressing personal demands based on a conversation system has shown an upsurge(Chai et al., 2018) (Kirk and Laird, 2019). Recently, there have been some latest works to realize the dynamic composition of services through the human-computer interaction of dialogue systems. Romero et al. (2019) proposed the NLSC, mainly for service developers and end-users. NLSC firstly determines the abstract services based on users' requirements and then chooses the concrete services. Nonetheless, users' demands may change dynamically, and the final composite service given may no longer satisfy end-users. Instead, our HSC-Rocket receives the user's timely feedback and returns each concrete service step by step. Li et al. (2020) showed SUGILITE, an intelligent task automation agent that can learn tasks and relevant associated concepts (abstract service) from user's demonstrations. Unlike them, we not only stores tasks learned from users, but also recommends concrete services. In terms of functionality, HSC-Rocket is more in line with the complex requirements of users.

Furthermore, there are some other state-of-the-art pieces of literature. Liu et al. (2018) induced high-level 'workflows' based on each demonstration and proposed an exploration strategy then learns to recognize successful workflows and samples actions. However, this strategy only summarizes the workflow for the user's demonstration, that is, abstract services, but does not involve concrete services. Also, a virtual assistant, Almond(Campagna et al., 2017), was presented to make users specify trigger-action tasks in natural language and connect multiple services via open APIs, which provides satisfactory services to users. Li and Riva (2018) also designed Kite, a practical system for bootstrapping task-oriented bots, which automatically generates bot templates to meet developers' different task requirements. Different from the existing works, our HSC-Rocket not only meets the functional requirements of users, but also focuses on the nonfunctional QoS that dominates the underlying concrete service composition.

3 HSC-Rocket Assistant

3.1 Generality

The generality of HSC-Rocket lies in the service repository-Rockethouse, which captures and stores web services in all fields and industries. For instance, HSC-Rocket can access public services in financial, medical and other fields. The advantage of Rockethouse is that it stores services based on a knowledge graph, where the service semantics are stored as nodes in the graph, and the basic QoS attributes of the service (response time, delay, price, etc.) are stored in tags. At the same time, the correlation between two services can be established through edges. Based on Rockethouse, HSC-Rocket assistant shows generality, which is no longer limited to a specific field, but applicable to various industries.

3.2 Functions of User Interface

The user interface of HSC-Rocket is a human-computer interaction conversational system in which users could enter natural language requirements. To enhance the user's immersive interaction more friendly and then give timely feedback, we have designed 'like' and 'dislike' buttons in the HSC-Rocket interface to express users' satisfaction or dissatisfaction with the current service respectively. Thus, HSC rocket can optimize and adjust the underlying model according to timely feedback, which improves the robustness of the model on the one hand, and enables users to express their personal feelings directly and clearly on the other hand.

4 Preliminaries

4.1 Service Definition and Formalization

In this section, we will present some definitions and formal descriptions related to the HSC-Rocket model.

Definition 1. Service. A service could provide some functions, which can be formalized as a 3-tuple (ID, Function, QoS). Here ID is the unique identifier; Function describes the function of the service; QoS is the nonfunctional attribute provided by the service provider, which can be formalized as $(q_1, q_2, ..., q_n)$, where n represents the number of QoS attributes. In this paper, we focus on the three attributes of QoS (service_level, service latency, price).

Definition 2. Abstract Service. Abstract service describes the rules and logical relationships of business processes, and it describes the required business functions. For example, a sequentially executed business process can be denoted as $\{a_1, a_2, ..a_i, \dots a_m\}$, where a_i is the i_{th} abstract service.

Definition 3. Business Process. The workflow composed of abstract services determined by definition 2 is the business process $P = \{a_1, a_2, ..a_i, \dots a_m\}$.

Definition 4. Candidate Concrete Services. Each abstract service a_i corresponds to a plurality of candidate concrete services, which have the same function and different QoS attribute values. The candidate services of each abstract service a_i can be formalized as a set $a_i = \{c_{i1}, c_{i2}, \dots, c_{ik} | 1 \leq i \leq m\}$. Where k is the number of candidate concrete services corresponding to the a_i.

Definition 5. Composite Services. According to the abstract business logic, the candidate concrete services with different functions are composed together, which can meet the complex functional requirements. According to the definition of abstract and candidate concrete service, the composite service is as follows:

$$\begin{aligned} a_1 \times ... \times a_i \times ... \times a_m = \\ \{..., c_{1j}, ...\} \times ... \{..., c_{ir}, ...\} \\ ... \times \{..., c_{mp}, ...\} \end{aligned} \quad (1)$$

Where c_{1j}, c_{ir} and c_{mp} represent the candidate concrete services of the first, i, and m abstract services respectively.

4.2 Reinforcement Learning

4.2.1 Basic Concepts of RL

To achieve a certain goal in an unknown environment, the agent of RL will constantly explore the environment to obtain timely feedback, and then adjust its actions. The ultimate goal is to get the maximum return from the environment (Sutton and Barto, 2018). The input of RL is a Markov Decision Process (MDP) that is a 4-tuple (S, A, P, R), which is defined as follows:

- S is a finite set of all states;
- A is a finite set of actions, where $A(s)$ represents a set of actions that can be executed in state s;
- P is a probability distribution function. When action a is executed, the current state changes from s to s', and the transition probability is recorded as $P(s'|s, a)$;
- R is the immediate reward function. When the current status is s, the agent selects and executes an action a to obtain a timely reward $r = R(s, a)$ from the environment.

The output of reinforcement learning is an action selection strategy $\pi : S \rightarrow A$. When the agent selects action $A = \pi(S)$ in state S, the expected value of the total reward is the largest.

4.2.2 DDPG and SAC

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) is a reinforcement learning algorithm to solve continuous control problems, in which its output is an action directly. It has fast convergence speed and is more suitable for the scenario where the sample data is scarce in the early stage of service composition. Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is a reinforcement learning algorithm based on off policy, actor critic and maximum entropy, which mainly solves the issues of discrete action space and continuous action space. It's a great choice to utilize SAC in the complex service composition environment.

5 HSC-Rocket System

In Figure 1, the system architecture of the HSC-Rocket includes two modules (1) business abstract service layer; (2) concrete service composition layer.

Business Abstract Service Layer. Users express personal requirements based on natural language, and $AbstractAgent$ will recommend the business process P composed of abstract services $abSer$. Then users answer feedback: like or dislike. Assuming users are satisfied with some business processes, which means those processes can initially meet the demands of users. Consequently,

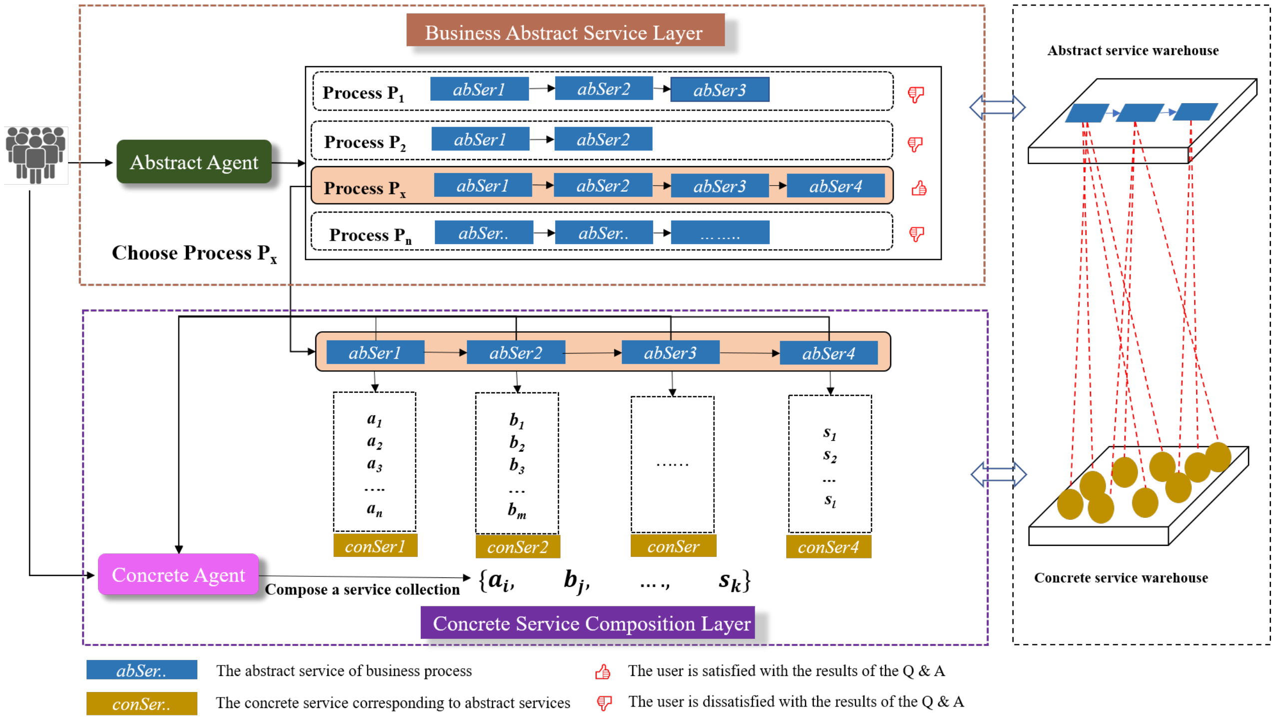


Figure 1: The System architecture of HSC-Rocket. abSer represents abstract services and conSer represents concrete services

We will make a further interactive mapping with the service composition layer for the deterministic process P_x.

Concrete Service Composition Layer. As a matter of fact, each abstract service in business process P corresponds to a series of concrete service collections that can provide the same functions, yet their QoS is different. Naturally, $ConcreteAgent$ dynamically composes concrete service collections based on the user's requirements and abstract services, which can satisfy the end-users more effectively.

The interaction process between the two layers is shown in Figure 2. Our service library, Rockethouse, stores the description semantic information and QoS attributes of the service. Here step 1 completes matching abstract services based on the user's demands, step 2 determines the business process by composition, and step 3 returns the business process to the end-users. The concrete service composition list set {*service1*,..., *service5*} is determined through the business processes *P1*,..., *P5*.

5.1 Business Abstract Service Model

As shown in Figure 3, we generate sentence vectors according to the user's demands through word compilation. *AbstractAgent* selects the best abstract service in the current state, returns it to the dialogue system, and then end-users give immediate rewards. Meanwhile, the *AbstractAgent* determines whether to continue to compose the next abstract service.

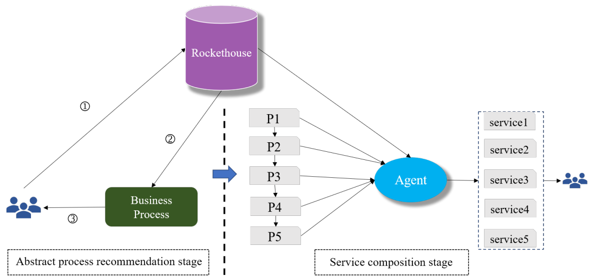


Figure 2: Flow chart of interaction between abstract process recommendation and service composition.

Environment and Action space. The environment is composed of the semantic description matrix of services, the sentence vector, and several currently composed service lists. The action of *AbstractAgent* is to compose and recommend abstract services.

Reward function. We divide rewards into immediate rewards and global rewards. Immediate reward obtains the connection relationship between two abstract services, and the reward in step i is defined as:

$$r_i = G(a_i, a^*) \tag{2}$$

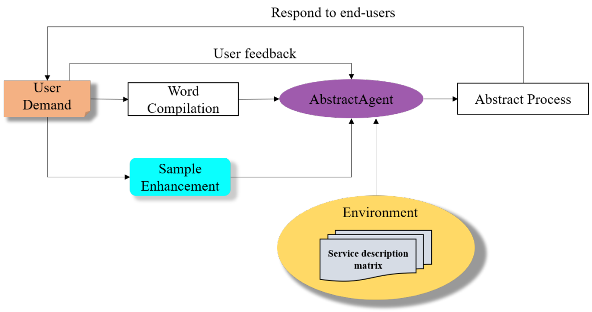


Figure 3: Business Abstract Service Model (Business-AbService) Based on Reinforcement learning.

Where a^* represents the abstract service required by users, a_i is the recommended abstract service. $G(x, y)$ is a Boolean expression. It reflects the real-time interaction between users and HSC-Rocket, When Formula (2) is equal to 1, it means the recommended services are consistent with user's expectations. And conversely, it means inconsistent when the value is -1. Correspondingly, the user will give 'like' or 'dislike' on the dialogue system interface.

Unlike immediate rewards, global rewards focus on obtaining the integrity of abstract services. It is an iterative process from back to front, so we define it as the following formula in step i:

$$R_i = G(\Phi(r_i), \Phi(r_{i+1}))R_{i+1}H_i \qquad (3)$$

Where $\Phi()$ is used to measure the sign (+, -) of r_i. R_{i+1} is the global reward of the latter service, and H_i represents the probability that the services are composed in step i. On the whole, we mainly determine the symbol of the global reward according to the immediate reward r_i and r_{i+1} of the two abstract services, and then weigh the global reward based on probability. And the final critical point is:

$$R_n = y \qquad (4)$$

Where y is the score given by the user after the abstract service composition is completed.

Sample enhancement. At the beginning of the conversation, there will be a small number of samples due to the limited questions and answers. In this case, the critical issue is to make the algorithm converge quickly based on a small number of samples. Therefore, we propose a soft sample enhancement method to alleviate this limitation.

During the interaction with users, we can get immediate rewards between services, so the problem can be summarized as to expand the sample data of 1:1 service to 1: n sample data. By comparing the functions of services, we regard them with the same functions as the positive samples and with different functions as the negative samples. The proposed soft enhancement can obtain the reward value according to the simple semantic function distance of the two services, which defines it as:

$$\tilde{r}_i = F[d(f_i, f_{a^*})] \cdot r_* + \frac{p_i}{t_i} \qquad (5)$$

Where $d(x, y)$ represents the semantic distance, f is the characteristic function, a_* means the service sample, and r_* represents the immediate reward. The p_i and t_i respectively represent QoS service_level and latency. Significantly, we can obtain more non-zero samples, which is conducive to improving the robustness of this system. Specifically, $F(\cdot)$ in this paper represents a Gaussian function, thus the reward is expressed as:

$$\tilde{r}_i = exp\left\{-[d(f_i, f_{a^*}) - b]^2/2c^2\right\} \cdot r_* + \frac{p_i}{t_i} \qquad (6)$$

5.2 Concrete Service Composition Model

After end-users determine the process in the abstract layer, we will further compose the candidate concrete services in the concrete layer. Since the sample enhancement module is the same as section 5.1, we will not repeat it in Figure 4.

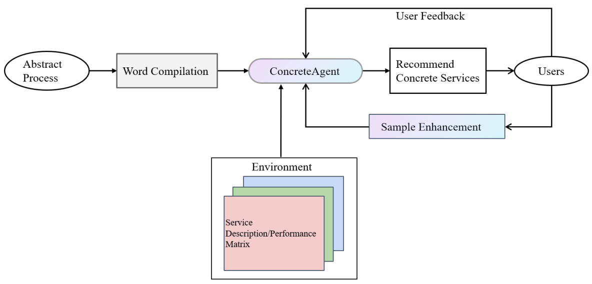


Figure 4: Concrete Service Composition Model (Composition-ConService) Based on Reinforcement learning.

Environment and Action space. The environment includes the abstract service determined in Section 5.1 and the specific service matrix mapped by some abstract services. The service matrix is composed of three QoS: service_level, latency, price. The action is to select the concrete service.

Reward function. Unlike the Bussiness-AbService, the Composition-ConService focuses on the composition of concrete services, hence there is only an immediate reward that is defined as follows:

$$r_i = r_{user} + \frac{p_i}{t_i} + \frac{1}{1 + price} \qquad (7)$$

Where r_{user} represents user feedback; p_i, t_i and $price$ respectively represent QoS service_level, latency and price, which describes the non-functionality of the service.

6 Implementation and Evaluation

6.1 Data Sets

In order to verify the HSC-Rocket, we store the `RapidAPI` datasets into the service warehouse Rockethouse. `RapidAPI`[1] is the largest API library in the world, including various types of services, such as data, sports, finance, travel, etc., which is shown in Figure 5. Rockethouse can meet the service requirements of different groups, which also conforms to the generality of HSC-Rocket.

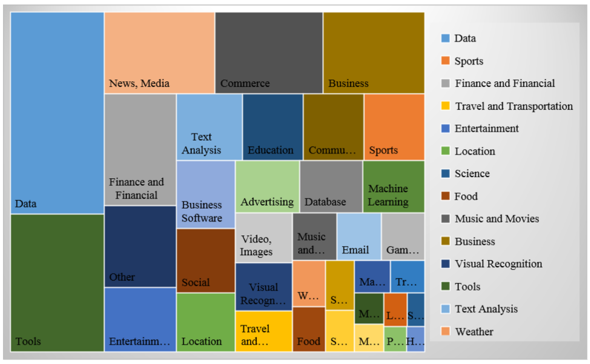


Figure 5: Distribution of RapidAPI Types.

Further, we store the semantic description and QoS attributes (latency, service level, price) in the Rockethouse based on the knowledge graph. The storage structure in Rockethouse is shown in Figure 6.

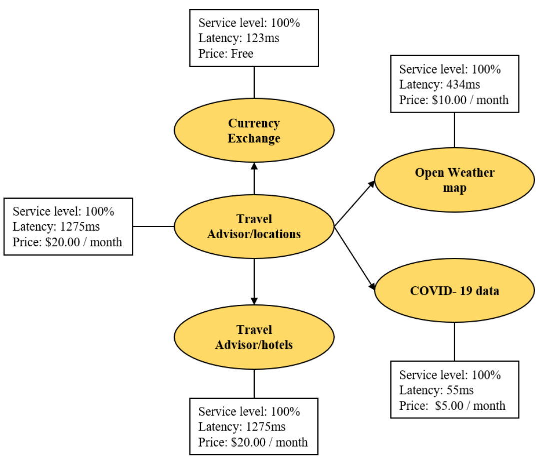


Figure 6: An example of the storage structure of services and their attribute QoS in Rockethouse.

[1] https://rapidapi.com/hub

6.2 Experimental setup and training

In the Abstract Service Layer, we propose the AbstractRL Algorithm based on a deterministic strategy DDPG. The parameters involved in this model and their specific values are shown in Table 1. And we utilize ConcreteRL Algorithm based on SAC in the concrete service composition layer. The parameters of actor and critic networks are randomly initialized, and the other parameters are shown in Table 2. The pseudo codes of the two algorithms are in the Appendix.

Parameter	Symbol/Value
Learning rate of Actor	α=0.001
Learning rate of Critic	α'=0.002
Discount factor	γ=0.8
Soft update coefficient	τ=0.01
Number of samples	m=64
Q-network update	C=100
Maximum iterations	T=500
Random noise function	N(Gaussian)
Enhanced samples	n=100
Reward in enhancement	F(Normal function)

Table 1: The detail names and values of parameters in Abstract Service Layer-AbstractRL Algorithm

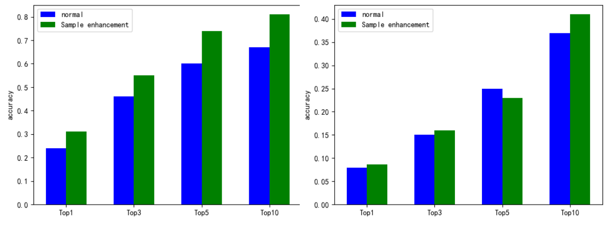
6.3 Effectiveness of HSC-Rocket

In this section, we mainly evaluate the effectiveness of HSC-Rocket. We use 213 user data for training, which comes from the feedback of students, epicure, and financiers. To test the performance of the model, we utilize the *TopN* index, which means that the first N services composed by HSC-Rocket can meet users' requirements. For instance, *Top3* means that the three services composed by the system can meet the needs of users. The data used for testing are mainly divided into two types, including 40 pieces respectively. The first category belongs to the sample coverage, and the experimental results

Parameter	Symbol/Value
Learning rate of Actor	a=0.001
Learning rate of Critic	β'=0.002
Discount factor	γ'=0.8
Exploration rate	ϵ'=0.01
Number of enhanced samples	n'=32
Maximum iterations	T=500

Table 2: The detail names and values of parameters in Concrete Service Layer-ConcreteRL Algorithm

are shown in Figure 7(a); The second type of data is not within the sample coverage, and Figure 7(b) shows the results.



(a) Within sample coverage (b) Out of sample coverage

Figure 7: Experimental results of two different types of test data

In Figure 7(a), '**normal**' means that the HSC-Rocket without sample enhancement technology, conversely, '**sample enhancement**' means that the sample enhancement technology is used in HSC-Rocket. It can be clearly shown that the accuracy of this system can be significantly improved by using sample enhancement when there is little sample data at the initial stage of user-system interaction. Simultaneously, as the number of composed services N increases, the accuracy will be improved accordingly.

It can be further concluded from Figure 7(b) that when our HSC-Rocket processes data that is not within the sample range, that is, when interacting with unfamiliar requirements, it can also well consider user requirements and present composed results. And as the user system interaction becomes more frequent, the understanding performance of HSC-Rocket will become stronger and stronger.

6.4 Scenario Cases

To better verify the practicability of HSC-Rocket, we provide the following real-life scenarios to explain the interaction process. As described in Figure 8, the user enters "*I want to get a film review*". Firstly, our assistant composes abstract service P ={ *Get the basic information and IMDB number of the movie -> Get movie details -> Get movie reviews -> Get emotional analysis of film reviews*} for users through interactive feedback with users. Then, for each abstract service in P, our system can compose the corresponding concrete services {*OTT details/Search*}, {*IMDB-Internet Movie Database/Film*}, {*movie.douban*}, {*Text Sentiment Analysis Method*} according to the user feedback button. Furthermore, the user inputs the movie name "*Wolf Warriors*", and HSC-Rocket will respond to the details of this movie. More detailed cases are in the Appendix.

Figure 8: Two screenshots of the our HSC-Rocket assistant user interface

7 Conclusion

In this paper, we propose a service composition algorithm based on human-computer interaction and design a dialogue assistant HSC-Rocket, which can better complete the interactive question and answer process combined with real-time feedback. In addition to meeting users' demands, our HSC-Rocket could complete the dynamic composition of abstract services and concrete services. Furthermore, we verify the effectiveness of the assistant through the case scenario, which has significant application value. In the future, we intend to consider more QoS and simultaneously focus on the execution of concrete services.

References

Pegah Alizadeh, Aomar Osmani, Mohamed Essaid Khanouche, Abdelghani Chibani, and Yacine Amirat. 2020. Reinforcement learning for interactive qos-aware services composition. *IEEE Systems Journal*, 15(1):1098–1108.

Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web*, pages 341–350.

Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S Lam. 2019. Genie: A generator of natural language semantic parsers for virtual assistant commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 394–410.

Joyce Y Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. 2018. Language to action: Towards interactive task learning with physical agents. In *IJCAI*, pages 2–9.

Schahram Dustdar and Wolfgang Schreiner. 2005. A survey on web services composition. *International journal of web and grid services*, 1(1):1–30.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Nobuaki Hiratsuka, Fuyuki Ishikawa, and Shinichi Honiden. 2011. Service selection with combinational use of functionally-equivalent services. In *2011 IEEE International Conference on Web Services*, pages 97–104. IEEE.

Nobuhiro Ito, Yuya Suzuki, and Akiko Aizawa. 2020. From natural language instructions to complex processes: issues in chaining trigger action rules. *arXiv preprint arXiv:2001.02462*.

James R Kirk and John E Laird. 2019. Learning hierarchical symbolic representations to support interactive task learning and knowledge transfer. In *IJCAI*, pages 6095–6102.

Toby Jia-Jun Li, Igor Labutov, Brad A Myers, Amos Azaria, Alexander I Rudnicky, and Tom M Mitchell. 2018. Teaching agents when they fail: end user development in goal-oriented conversational agents. In *Studies in Conversational UX Design*, pages 119–137. Springer.

Toby Jia-Jun Li, Tom Mitchell, and Brad Myers. 2020. Interactive task learning from gui-grounded natural language instructions and demonstrations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 215–223.

Toby Jia-Jun Li and Oriana Riva. 2018. Kite: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*.

Ahmed Moustafa and Takayuki Ito. 2018. A deep reinforcement learning approach for large-scale service composition. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 296–311. Springer.

Lifang Ren, Wenjian Wang, and Hang Xu. 2017. A reinforcement learning method for constraint-satisfied services composition. *IEEE Transactions on Services Computing*.

Oscar J Romero, Ankit Dangi, and Sushma A Akoju. 2019. Nlsc: Unrestricted natural language-based service composition through sentence embeddings. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 126–135. IEEE.

Krisada Sangsanit, Werasak Kurutach, and Suronapee Phoomvuthisarn. 2018. Rest web service composition: A survey of automation and techniques. In *2018 International Conference on Information Networking (ICOIN)*, pages 116–121. IEEE.

Wissam Siblini, Baris Sayil, and Yacine Kessaci. 2021. Towards a more robust evaluation for conversational question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1028–1034.

Yiguang Song, Li Hu, and Ming Yu. 2018. A novel qos-aware prediction approach for dynamic web services. *Plos one*, 13(8):e0202669.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Gang Wang and Zhen-Zhong Zhang. 2017. Qband: Indicating the implementation distribution of qos-based web service composition solutions. *Journal of Computers*, 28(1):149–166.

Hongbing Wang, Xin Chen, Qin Wu, Qi Yu, Zibin Zheng, and Athman Bouguettaya. 2014. Integrating on-policy reinforcement learning with multi-agent techniques for adaptive service composition. In *International Conference on Service-Oriented Computing*, pages 154–168. Springer.

Hongbing Wang, Mingzhu Gu, Qi Yu, Yong Tao, Jiajie Li, Huanhuan Fei, Jia Yan, Wei Zhao, and Tianjing Hong. 2019. Adaptive and large-scale service composition based on deep reinforcement learning. *Knowledge-Based Systems*, 180:75–90.

Hongbing Wang, Xingguo Hu, Qi Yu, Mingzhu Gu, Wei Zhao, Jia Yan, and Tianjing Hong. 2020. Integrating reinforcement learning and skyline computing for adaptive service composition. *Information Sciences*, 519:141–160.

Hongbing Wang, Xiaojun Wang, Xingguo Hu, Xingzhi Zhang, and Mingzhu Gu. 2016. A multi-agent reinforcement learning approach to dynamic service composition. *Information Sciences*, 363:96–119.

Silei Xu, Giovanni Campagna, Jian Li, and Monica S Lam. 2020. Schema2qa: High-quality and low-cost q&a agents for the structured web. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1685–1694.

Xuezhi Yu, Chunyang Ye, Bingzhuo Li, Hui Zhou, and Mengxing Huang. 2020. A deep q-learning network for dynamic constraint-satisfied service composition. *International Journal of Web Services Research (IJWSR)*, 17(4):55–75.

Yuan Yuan, Weishi Zhang, Xiuguo Zhang, and Huawei Zhai. 2019. Dynamic service selection based on adaptive global qos constraints decomposition. *Symmetry*, 11(3):403.

Yu Zhao, Shaohua Wang, Ying Zou, Joanna Ng, and Tinny Ng. 2017. Automatically learning user preferences for personalized service composition. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 776–783. IEEE.

Huiyuan Zheng, Jian Yang, Weiliang Zhao, and Athman Bouguettaya. 2011. Qos analysis for web service compositions based on probabilistic qos. In *International Conference on Service-Oriented Computing*, pages 47–61. Springer.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. Convlab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. *arXiv preprint arXiv:2002.04793*.

A Appendix

A.1 Scenario case

The financial and food scenario cases of HSC-Rocket user interface are shown in Figure 9 and 10 respectively.

A.2 Pseudo code

Algorithms 1 and 2 are AbstractRL and ConcreteRL algorithm respectively.

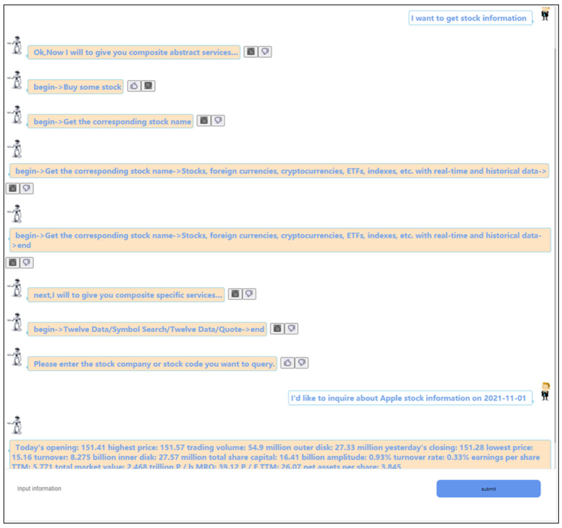


Figure 9: User cases in financial scenarios

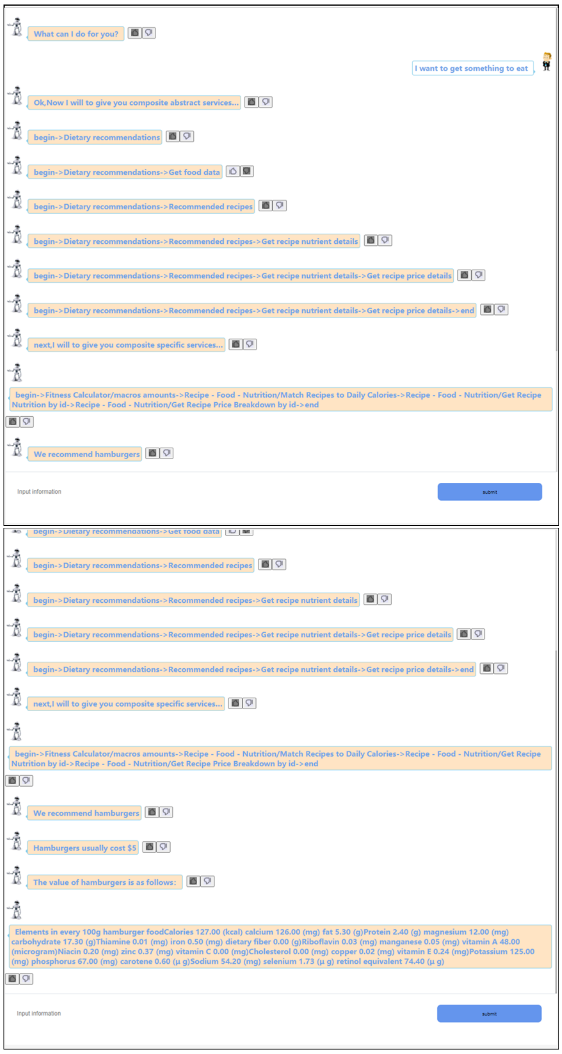


Figure 10: User cases in food scenes

Algorithm 1 AbstractRL algorithm

Input:
user demand;
Output:
Serve_list =[ ];
user_feature = use word2vec to get feature vector;
is_end=false;
while is_end is false **do**
S = Transform To State (s, serve_list, user_feature);
in Actor network get action: $A = \pi_\theta(\phi(s)) + N$;
Do the action to get Reward(R), next state(s') and isend;
R=GetReward Through UserInteraction(serve_action);
Use soft Sample enhancement to get more experiences;
Action_list,R_list,S_list=Soft SampleEnhancement(A,R,S);
For i=0 to Action.length;
put (ϕ(S),Action_list[i],R_list[i],ϕ(S'),isend) in experience replay D;
S=S';
M samples are sampled from the experience playback set D to get target value Q_{yj};
$\left\{\phi(S_j, A_j, R_j, \phi(S'_j)), isend_j\right\}$, j=1,2,...,m;
$$y_i = \begin{cases} R_j, & is\_end_j = true \\ R_j + \gamma Q'(\phi(S'_j), \pi_{\theta'}(\phi(S'_j)), w'), & is\_end_j = false \end{cases}$$
Using mean square loss function $\frac{1}{m}\sum_{j=1}^{m}(y_i - Q(\phi(S_j), A_j, w))^2$ to update the critic net work parameter ω;
Using $J(\theta) = -\frac{1}{m}\sum_{j=1}^{m} Q(S_j, A_j, \theta)$ to update all parameters of actor's network θ;
If T%C =1 update critic target network and actor target network parameters:
$w' \leftarrow \tau w + (1-\tau)w'$
$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
end while

Algorithm 2 ConcreteRL algorithm

Input:
one_abstract_services, s;
Output:
concrete_service_list;
user_feature = use word2vec to get feature vector;
S = Transform To State(s, serve_list, user_feature, concrete_service_list);
Continue = true;
while Continue **do**
Get action in Actor network and the next S' : serve_action, $S' = \varphi(S, serve\_action)$;
Compute reward R in actor network Through user interaction;
R=GetRewardThroughUserInteraction (serve_action);
if R in Positive **then**
Serve_list.add();
isend=true;
Continue =false;
else
isend=false;
Continue=true;
end if
In critic network use S and S' to get V(S), V(S');
Compute TD loss: $\delta = R + \gamma V(S_0)V(S)$;
Use the Mean square error loss function to update the critic network parameter ω : $\sum(R + \gamma V(S_0)V(S))^2$;
To update the parameter θ;
$\theta = \theta + \alpha\nabla_\theta log\pi_\theta(S, A)\delta$;
Use soft Sample enhancement to get more experiences;
Action_list,R_list=SoftSampleEnhancement (serve_action,R);
For i=0 to Action.length;
put $(\varphi(S), Action\_list[i], R\_list[i], \varphi(S'), isend)$ into experience replay D;
Use D to update the base decision-makers ϵ every time period m;
end while
Return serve_action