ASC4AI: A Novel Automatic Service Composition Framework for AI Systems

Zhiguo Li, Qin Li*
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
Shanghai, China

Abstract—**As cloud computing becomes more prevalent in various domains, such as e-commerce, healthcare, education, etc., there is a growing demand for cloud-based applications that can perform complex tasks by integrating multiple cloud services. Despite QoS-aware automatic service composition has been extensively studied, conventional approaches face limitations in evaluating intelligent services and ensuring correctness and reliability for AI systems. To address these challenges, we propose five metrics for differentiating between intelligent and non-intelligent services and evaluating service composition solutions based on user-defined metrics constraints. We also explore ways to improve system correctness and reliability, and adapt these approaches to fit with AI systems. Building on these insights, we propose ASC4AI, a novel automatic service composition framework designed specifically for AI systems, which can automatically generate service composition solutions that meet user-defined functional and metrics constraints. Furthermore, we implement ASC4AI as a user-friendly tool that minimizes technical complexity for developers.**

Index Terms—**Artificial intelligent systems, Automatic service composition, Service metrics, Service composition patterns**

I. Introduction

As cloud native technology advances, the use of cloud services for performing complex tasks has become increasingly popular. Additionally, to promote the diffusion and adoption of AI technologies form cloud, cloud service providers have started to offer out-of-the-box reasoning services [1]. However, service-oriented AI systems consist of a multitude of logical independent intelligent and non-intelligent services, which makes existing service composition approaches and evaluation metrics unsuitable for AI systems. Hence, it is necessary to develop a new automatic service composition framework specifically tailored to AI systems.

To our knowledge, no study has yielded a systematic solution to the problem. For example, some works propose to automatic service composition based on the service description model, but this approach not be suitable for AI systems due to lack evaluation metrics and satisfy user constraints [2]–[4]. And some other works propose to evaluate the reliability of the component-based systems [5]–[7], but user constraints extend beyond just reliability metrics.

*Corresponding author: Qin Li (qli@sei.ecnu.edu.cn)
DOI reference number: 10.18293/SEKE2023-109

Therefore, an intuitive idea we want to explore is, can we automatically provide service composition solutions to satisfy specific metrics constraints for AI systems? Realizing this intuition based on existing research raises two challenges: (1) Establishing distinct metrics for AI systems can help differentiate between intelligent and non-intelligent services. (2) Automatically compositing services that can meet user workflow's functional and metrics constraints while enhancing correctness or reliability.

Regarding the first challenge, we begin by defining what is intelligent task or service in AI systems. Next, we propose five key metrics extended quality of service: correctness, reliability, throughput, response time, and GPU memory usage. Of these metrics, correctness stands out as particularly important since it provides a comprehensive measure of AI systems accuracy and can effectively differentiate between intelligent and non-intelligent services based on functionality.

To address the second challenge, we compare the semantic information of services' input and output to determine compatibility and establish edges between matching services with associated metrics information for automatic composition. In terms of correctness and reliability, we analyze commonly used service composition patterns. Our findings suggest that fault-tolerant structures can enhance system correctness and reliability. These structures can be customized and applied to critical service nodes to effectively improve AI systems.

Based on the above explorations, we have designed a framework for automatic service composition called ASC4AI. This framework is capable of automatically generating service composition solutions that meet user-defined metrics and functional constraints, while improving the correctness or reliability of the solution. To make it more accessible, we have also developed a user-friendly tool that simplifies the technical complexity for developers or users.

The main contributions of this paper are as follows:

- We propose five metrics for AI systems: correctness, reliability, throughput, response time and GPU memory usage that can be used to differentiate between intelligent and non-intelligent services.

- We design a framework for AI systems that can automatically generate trustworthy and reliable service composition solutions and implement the framework as a user-friendly tool. The result demonstrates that our approach can be applied to AI systems.

The rest of this paper is organized as follows. In Section II, we review the related work. In Section III, we introduce the architecture of ASC4AI. In Section IV, we present the process of the proposed approach and the evaluation method. In Section V, we present the implementation of the framework and a case study. Finally, we conclude this paper in Section VI.

II. Related Work

Approaches for ASC problem can be categorized into AI planning or graph-based. For instance, [8] proposes an integrated approach for automated semantic web service composition using AI planning techniques. The authors of [2] and subsequent work [3] address the automatic service composition and QoS-aware service selection by utilizing an inverted index table and a counting mechanism. However, their works focus solely on generating optimal QoS-aware compositions without taking into account user QoS requirements constraints and intelligent services.

To guarantee QoS requirements constraints of users, the QoS-aware ASC problem has attracted the attention of a lot of researchers from different fields. For example, [9] introduce a top k query mechanism to satisfy user constraints as much as possible. The work in [10] presents a novel approach based on a Harmony Search algorithm under respect global QoS constraints. In [11] the authors propose a new SOC-based approach to ensure application development which ensures the discovery, selection, and composition of the most appropriate Web services to meet the developer requirements. Unfortunately, these approaches do not consider the reliability and correctness of the AI systems.

In many cases, the effects of service faults on the business are disastrous. The fault-tolerant structure can improve the reliability and real-time capabilities of the systems [12] and the work in [13] detailed introduction of those structures. Although these works can provide guidance on how to improve systems reliability, further research is still needed on how to combine automated service composition techniques to automatically select the appropriate structure.

Overall, these approaches enable a wider search space and flexible service composition under QoS constraints, but they have limitations. Firstly, the composition patterns and QoS calculation methods are not always clear. Secondly, they do not take into account the specific requirements of AI systems.

III. ASC4AI Architecture

The architecture of ASC4AI is illustrated in Fig. 1. The browser, representing the presentation layer, facilitates user visual interaction. The ASC4AI, serving as the logic layer, generates service composition solutions that meet the metrics constraints of the user's workflow requirements. Finally, the database acts as the data layer and is responsible for persistently storing the data used.

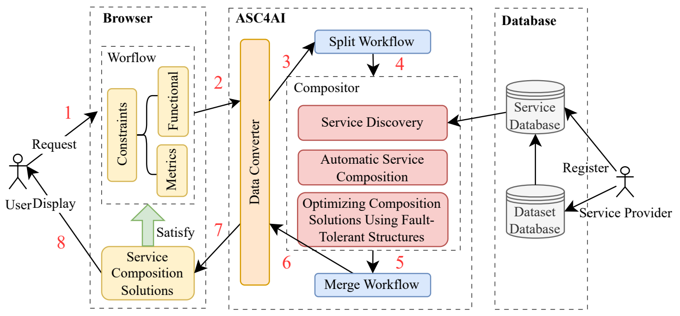


Fig. 1: ASC4AI architecture

The remainder of this section we describe the ASC4AI main process from user requirement to automatic service composition solution generation, the compositor will be introduced in next Section.

Users provide their workflow requirements in a flowchart format, consisting of intelligent and non-intelligent task nodes. Each node requires the description of input and output, functional requirements, and metrics constraints. For intelligent nodes, users must also specify the dataset used and evaluation method. In order to better understand the workflow requirements, we introduce the following concepts.

1) Intelligent Tasks or Services: Intelligent tasks or services are the ones that use artificial intelligence technology. They have the ability to analyze and process complex data beyond human capabilities in specific domains. However, they require extensive data training and GPU resources to function efficiently. For example, image recognition is an intelligent service, whereas adding a watermark to an image is a non-intelligent service.

2) Interaction Patterns in Workflow: The interaction between task nodes based four basic composite structure, *Sequential*, *Parallel* and *Conditional* from [14], shown in Fig. 2. Fox example, the workflow start from task node T_1 with output $[c, d, e]$, then the T_2 and T_3 can be executed in *Parallel* with input $[c, d, e]$, output $[f, g]$. Finally, the output $[h]$ is can be executed in *Conditional* with conditions c_1 and c_2 to get final output $[i]$.

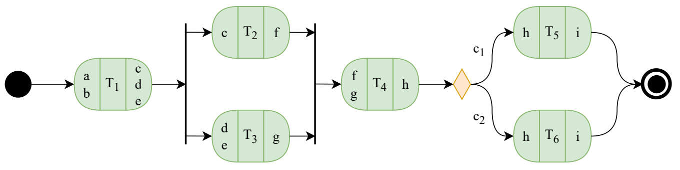


Fig. 2: Workflow interaction patterns

3) Metrics: To evaluate and distinguish between intelligent and non-intelligent services in AI systems, this paper proposes five metrics are as follows.

- Correctness(*corr*) refers to the ability of a service or component to produce accurate results. This metric is represented as a vector $corr = (e_1, e_2, \dots, e_n)$, where e_i represents the evaluation result obtained using various evaluation techniques such as confusion matrix, F1-score, ROC curve, etc. For simplifies, we use the weighted summation to calculate the correctness, which is defined as follows:

$$corr = \sum_{i=1}^{n} w_i e_i' \tag{1}$$

  where w_i is the weight of the i-th evaluation technique, and e_i' is the normalized evaluation result of the i-th evaluation technique. For non-intelligent services, the correctness is always 1.
- Reliability(*rel*) is the ability of a service or component to perform the required functions within a specified time interval and under specified conditions.
- Throughput(*tp*) refers to the ability of the service to handle multiple requests.
- Response Time(*rt*) refers to the average get response time for a single request sent to the service.
- GPU Memory(*gm*) reflects the minimum required GPU memory for a service to perform optimally and often represents the service's usage cost, with higher GPU Memory indicating higher cost.

Since the architecture cannot directly accept a flowchart, a data converter is used to translate the user's workflow requirements into some format that can be processed by the architecture. This converter is also responsible for translating the results generated by ASC4AI back into a format that can be easily parsed by the flowchart.

The split and merge workflow is a design approach that simplifies the execution of large-scale workflows. It achieves this by dividing them into smaller sub-workflows or task nodes, processing each one separately, and then merging their results to form a single completed workflow.

After obtaining a set of service composition solutions, the user can select the most appropriate solution based on the metrics of the service composition. The user can also modify the workflow requirements and re-run the ASC4AI to obtain a new set of service composition solutions.

IV. Service Compositor

Our aim is to automatically generate trustworthy and reliable service composition solutions while taking into account the constraints of user workflow requirements. The remainder of this section we describe the service composition process and evaluation method.

A. Automatic Service Composition

The automatic service composition process for one task node is shown in Fig. 3. The process includes three steps: generate a service composition graph, find a set of service sequences maybe DAGs that satisfy the constraints, and optimize reliability by replacing the key service nodes in fault-tolerant structures.

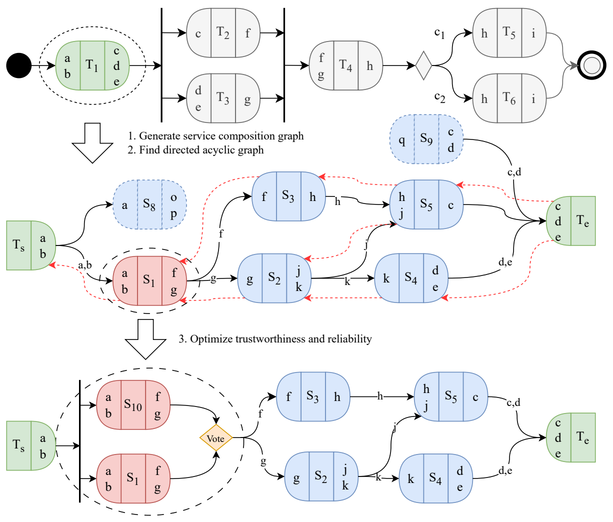

1. Generate service composition graph
2. Find directed acyclic graph
3. Optimize trustworthiness and reliability
Vote

Fig. 3: Automatic service composition process

1) Generate Service Composition Graph: Generating service composition diagrams requires the use of service semantic information, and by constructing a concept tree structure, the semantic similarity between concepts can be calculated. Suppose the output of service A is O_A and the input of service B is I_B. The formula for calculating the similarity between O_A and I_B is as follows:

$$Sim(O_A, I_B) = 1 - \sqrt{\frac{1}{2}a \cdot Dis(O_A, I_B)} \tag{2}$$

where a is $\frac{Dep(O_A)}{Dep(I_B)+Dep(O_A)}$, $Dep(X)$ is the depth of concept X in the concept tree. And $Dis(O_A, I_B)$ is the distance between O_A and I_B in the concept tree. The distance between two concepts is defined as the length of the shortest path between them in the concept tree.

Once the similarity between two services up to a value θ (which can be provided by the user) then the link between the two services is considered to exist. After that, a service composition directed graph is constructed starting from the task node input to the output.

2) Find DAGs: After constructing the service composition directed graph, the automatic service composition problem is transformed into a directed graph path-finding problem. This paper uses the algorithm from [2] to find all the paths that satisfy the metrics constraints. But our approach differs from the original method in how we calculate these new metrics for AI systems.

In the service composition graph, it can be found the the *Sequential* and *Parallel* are deterministic, i.e., for the *Sequential*, the output of the previous service is the input of the next service; for the *Parallel*, the output of each service is the input of the next service. So the metrics in the process of finding the optimal path are calculated in the table I.

However, the *Loop* and *Conditional* patterns in workflow interaction(see Fig. 2) involve uncertainty. For *Loop*, the number of loops is uncertain, and for *Conditional*, the probability of each condition is uncertain. So for these two patterns, we can calculate the corresponding metrics based on the component [15], [16] analysis, and a workflow is a component(task node) software with transfer probabilities(can be specified by user). We also have some assumptions: (1) The correctness and reliability of a task node is known (determined by calculating the service composition later). (2) The control transfer between task nodes is Markovian in nature, i.e., future behavior is independent of past behavior. (3) The failure of one task node does not affect any other task node.

In this way, the calculation method for each metric can be obtained shown in Table I. In the table, $P_{i,j}$ represents the probability of task node T_i transitioning to task node T_j, while k denotes the maximum loop value collected by users.

TABLE I: Metrics calculation method for *Sequential*, *Parallel*, and *Conditional*

Metrics	Sequential	Parallel	Conditional
q^{corr}	$\prod_{i=1}^{n} q_i^{corr}$	$\prod_{i=1}^{k} q_i^{corr}$	$P_{i,j}q_j^{corr} + P_{i,k}q_k^{corr}$
q^{rel}	$\prod_{i=1}^{n} q_i^{rel}$	$\prod_{i=1}^{k} q_i^{rel}$	$P_{i,j}q_j^{rel} + P_{i,k}q_k^{rel}$
q^{tp}	$min\{q_1^{tp}, \dots, q_n^{tp}\}$	$min\{q_1^{tp}, \dots, q_k^{tp}\}$	$P_{i,j}q_j^{tp} + P_{i,k}q_k^{tp}$
q^{rt}	$\sum_{i=1}^{n} q_i^{rt}$	$max\{q_1^{rt}, \dots, q_k^{rt}\}$	$P_{i,j}q_j^{rt} + P_{i,k}q_k^{rt}$
q^{gm}	$max\{q_1^{gm}, \dots, q_n^{gm}\}$	$\sum_{i=1}^{k} q_i^{gm}$	$P_{i,j}q_j^{gm} + P_{i,k}q_k^{gm}$

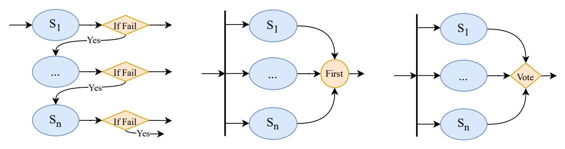
3) Optimize Correctness and Reliability: AI systems used in production require accuracy and reliable capabilities. However, a limitation of the previously mentioned algorithms is that if any service within the service composition solution fails or produces an error output, it can cause a system-wide failure. Since we do not have prior knowledge of a service's internal implementation, we can only improve these capabilities of AI systems by using functionally equivalent components for fault-tolerant processing.

To more efficiently determine which service nodes should use a fault-tolerant structure, i.e. which service nodes have the greatest impact on the system, we use a method proposed in [17] as follows:

$$F_i = \alpha_1 In_F(i) + \alpha_2 In_P(i) + (1-\varphi) In_S(i) \tag{3}$$

where $In_F(i)$, $In_P(i)$, and $In_S(i)$ represent failure influence, fault propagation influence and self-influence of service i, respectively. The user can provide the weights of the three influences, which are represented by α_1, α_2, and $\varphi = \alpha_1 + \alpha_2$.

In this way, three commonly fault-tolerant structures can be used to improve correctness or reliability capabilities are shown in Fig. 4. The following introduces the three fault-tolerant structures in detail.



(a) RB (b) PS (c) NVP

Fig. 4: Three commonly used fault-tolerant structures

- Recovery block mechanism(RB): When the main service fails, the backup services will be executed in sequence, and the recovery block mechanism will only fail when all the services have failed.
- Parallel strategy(PS): Multiple independent services are called in parallel, and the final result is determined by the return result of the first service. This strategy will only fail if all services fail to execute.
- N-version programming(NVP): Multiple independent services are called in parallel, and the final result is determined by a voting result. Typically, when M services are executing normally, N-version programming will not fail, where M is equal to $[n/2] + 1$, the number of services executing in parallel is n.

It should be noted that when a fault-tolerant structure is used, we assume that all services are executed and the final selection of results follows the uniform distribution. The calculation method is shown in Table II.

TABLE II: Metrics calculation method for fault-tolerant structures

Metrics	Recovery Block	Parallel Strategy	N-version Programming
q^{corr}	$\frac{1}{n}\sum_{i=1}^{n} q_i^{corr}$	$\frac{1}{n}\sum_{i=1}^{n} q_i^{corr}$	$1 - \prod_{i=1}^{n}(1 - q_i^{corr})$
q^{rel}	$1 - \prod_{i=1}^{n}(1 - q_i^{rel})$	$1 - \prod_{i=1}^{n}(1 - q_i^{rel})$	R'^{*}
q^{tp}	$min\{q_1^{tp}, \dots, q_n^{tp}\}$	$min\{q_1^{tp}, \dots, q_n^{tp}\}$	$min\{q_1^{tp}, \dots, q_k^{tp}\}$
q^{rt}	$\sum_{i=1}^{n} q_i^{rt}$	$max\{q_1^{rt}, \dots, q_n^{rt}\}$	$max\{q_1^{rt}, \dots, q_n^{rt}\}$
q^{gm}	$max\{q_1^{gm}, \dots, q_n^{gm}\}$	$\sum_{i=1}^{n} q_i^{gm}$	$\sum_{i=1}^{n} q_i^{gm}$

* Assuming a three-version structure, with each service's reliability being q_i^{rel}, and requiring $M = 2$ services to be functioning normally, the reliability of the structure is : $R'^{*} = q_1^{rel}q_2^{rel}q_3^{rel} + q_1^{rel}q_2^{rel}(1-q_3^{rel}) + q_1^{rel}q_3^{rel}(1-q_2^{rel}) + q_2^{rel}q_3^{rel}(1-q_1^{rel})$.

While fault-tolerant structures may decrease certain metrics such as correctness, throughput, and response time, they are generally effective at identifying a range of solutions that satisfy user constraints under normal conditions. If the user's constraints are not met, the approach can still offer several relatively better composition solutions for the user to choose from or modify.

B. Evaluation

After the metrics of the service composition solution are calculated, the evaluation is performed, which based on the user's constraints and the metrics of the service composition solution. These metrics are classified into positive and negative attributes. Positive attributes are characterized by higher values being better, such as correctness, reliability and throughput. Negative attributes are characterized by smaller values being better, such as average response time and GPU graphics memory.

Since the metrics of a service are not always within the same range, differences in such ranges will impact the final evaluation results. To address this issue, we normalize the metrics of the service to a range of $[0, 1]$ before evaluation. The resulting normalized for both positive and negative attributes can be defined as follows:

$$For\ positive,\ \ q_i' = \begin{cases} \frac{q_i - q_i^{min}}{q_i^{max} - q_i^{min}}, & q_i^{min} \neq q_i^{max} \\ 1, & q_i^{min} = q_i^{max} \end{cases} \tag{4}$$

$$For\ negative,\ \ q_i' = \begin{cases} \frac{q_i^{max} - q_i}{q_i^{max} - q_i^{min}}, & q_i^{min} \neq q_i^{max} \\ 1, & q_i^{min} = q_i^{max} \end{cases} \tag{5}$$

where q_i' is the normalized metric of the service, q_i is the one metric of the service, q_i^{min} and q_i^{max} are the minimum and maximum values of the metric, respectively.

However, users have varying preferences for different metrics. For example, some may prioritize higher throughput and lower response time. The user's preferences can be represented by a weight vector $w = (w_1, w_2, \cdots, w_n)$, where w_i is the weight of the ith metric. The weight vector is normalized to the range of $[0, 1]$, and the sum of the weight vector is 1. The score of the service is then calculated as follows:

$$Score_{metrics} = \sum_{i=1}^{n} w_i q_i' \tag{6}$$

By doing this, we can obtain the score of the candidate service composition solution($Score_{composition}$) and the constraint score of the task node($Score_{constraints}$). The constraint score of the workflow is calculated based on the user's workflow requirements, which are represented by a constraint vector $c = (c_1, c_2, \cdots, c_n)$, where c_i is the constraint of the ith metric.

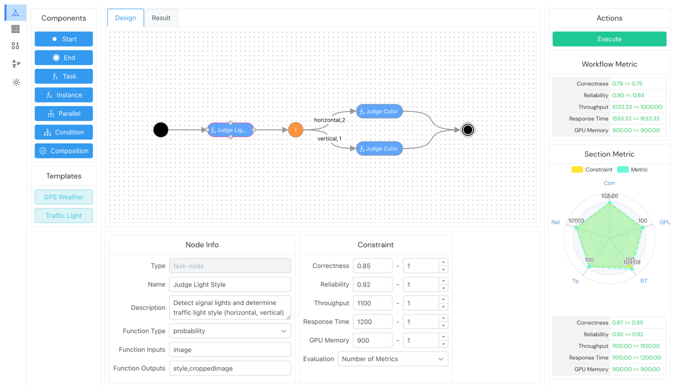
Once we have obtained the metrics score and constraints score, we can get the overall score($Score = Score_{composition} - Score_{constraints}$) of the service composition solution by subtracting the two values. The overall score should be positive, otherwise, it means that the candidate service composition solution does not meet the user's workflow requirements.

If the weighted sum method fails to meet all requirements, a full evaluation method is available. Unfortunately, it's unlikely that all metric constraints can be met in real systems. This method involves comparing the number of metrics that a service composition solution with the given constraints and determining its domination number. The resulting value is then subtracted from 5 (the total number of metrics in this paper) to align with the weighted sum method.

V. Case Study

This section demonstrates the application of the proposed approach using a traffic light recognition example from Baidu's open-source Apollo platform in the context of advanced autonomous driving systems that contain both intelligent and non-intelligent services driven by machine learning.

The workflow example includes three tasks, as illustrated in Fig. 5. The first task detects signal lights, determines the traffic light style using an input image, and outputs the traffic light style and the cropped image. Then, a conditional node is employed to execute different tasks based on the traffic light style and probability. The remaining tasks are intelligent, but differ in the datasets used for training and styles of traffic lights applied.



Node Info

Type	task-node
Name	Judge Light Style
Description	Detect signal lights and determine traffic light style (horizontal, vertical)
Function Type	probability
Function Inputs	image
Function Outputs	style,croppedImage

Constraint

Correctness	0.85	1
Reliability	0.92	1
Throughput	1100	1
Response Time	1200	1
GPU Memory	900	1
Evaluation	Number of Metrics	

Workflow Metric

Correctness	0.79 >= 0.75
Reliability	0.90 >= 0.84
Throughput	1033.33 >= 1000.00
Response Time	1593.33 <= 1633.33
GPU Memory	900.00 <= 900.00

Correctness	0.87 >= 0.85
Reliability	0.93 >= 0.92
Throughput	1100.00 >= 1100.00
Response Time	1100.00 <= 1200.00
GPU Memory	900.00 <= 900.00

Fig. 5: Workflow design

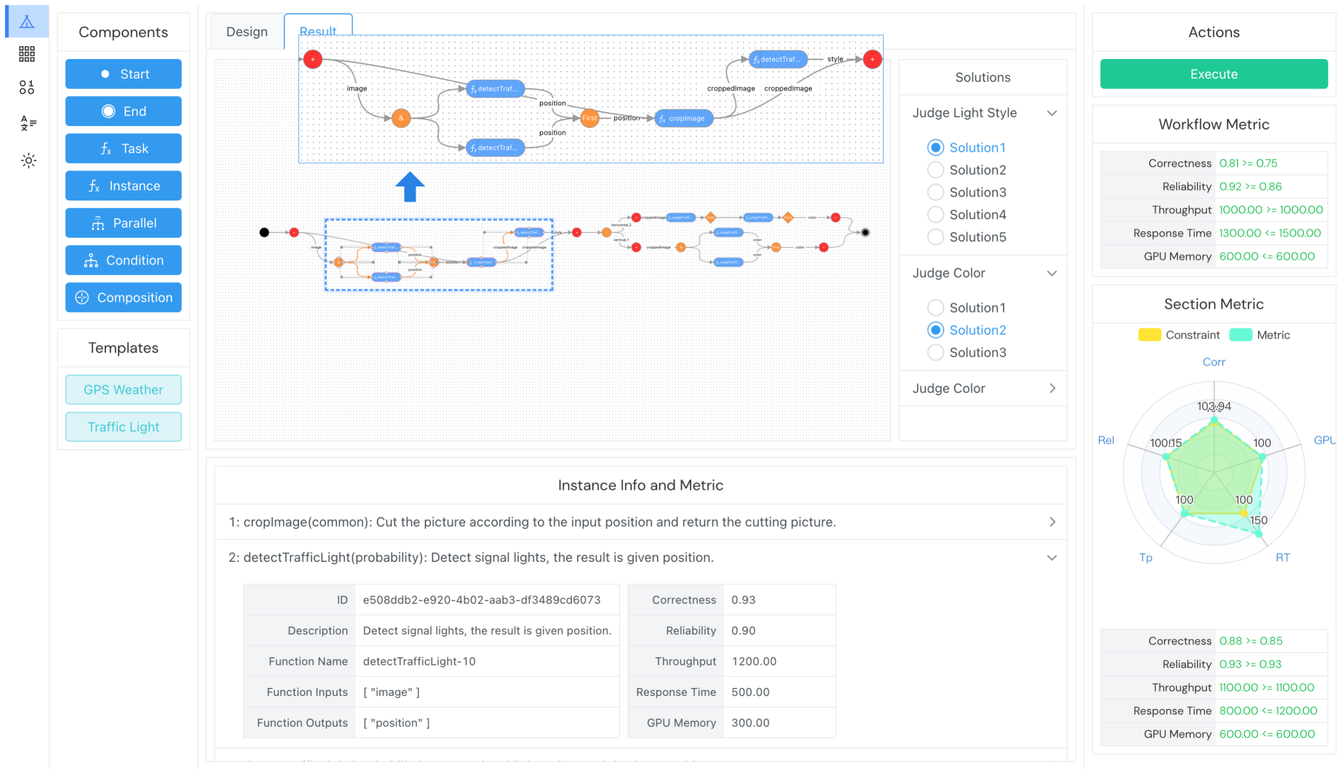
The generate service composition solutions shown in Fig. 6. With the aid of expert evaluation, our tool can generate service composition solutions that meet user demand constraints.

VI. Conclusion

In this paper, we present ASC4AI, a framework for automated service composition, which is tailored specifically for AI systems. The novelty of our framework is that we propose five metrics executed quality of service that distinguish intelligent services from non-intelligent ones. We also utilize fault-tolerant structures to improve the correctness and reliability capabilities of AI systems. Furthermore, we have developed a user-friendly tool based on the ASC4AI that has been successfully applied to intelligent manufacturing systems. Our ongoing research efforts aim to improve the algorithm efficiency and add more service composition patterns to the framework.

Acknowledgment

This work is supported by the National Key Research and Development Program (2020AAA0107800) and the Reliability Technology Lab of Huawei Technology Co., Ltd.



Solutions

Judge Light Style
[✓ Solution1] Solution2 Solution3 Solution4 Solution5

Judge Color
Solution1 [✓ Solution2] Solution3

Judge Color

Workflow Metric

Correctness	0.81 >= 0.75
Reliability	0.92 >= 0.86
Throughput	1000.00 >= 1000.00
Response Time	1300.00 <= 1500.00
GPU Memory	600.00 <= 600.00

Section Metric

🟨 Constraint 🟩 Metric

Correctness	0.88 >= 0.85
Reliability	0.93 >= 0.93
Throughput	1100.00 >= 1100.00
Response Time	800.00 <= 1200.00
GPU Memory	600.00 <= 600.00

Instance Info and Metric

1: cropImage(common): Cut the picture according to the input position and return the cutting picture.

2: detectTrafficLight(probability): Detect signal lights, the result is given position.

ID	e508ddb2-e920-4b02-aab3-df3489cd6073	Correctness	0.93
Description	Detect signal lights, the result is given position.	Reliability	0.90
Function Name	detectTrafficLight-10	Throughput	1200.00
Function Inputs	["image"]	Response Time	500.00
Function Outputs	["position"]	GPU Memory	300.00

Fig. 6: Service composition solution

References

[1] S. Lins, K. D. Pandl, H. Teigeler, S. Thiebes, C. Bayer, and A. Sunyaev, "Artificial intelligence as a service," *Business & Information Systems Engineering*, vol. 63, no. 4, pp. 441–456, 2021.

[2] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu, "Qsynth: A tool for qos-aware automatic service composition," in *2010 IEEE International Conference on Web Services*. IEEE, 2010, pp. 42–49.

[3] W. Jiang, S. Hu, D. Lee, S. Gong, and Z. Liu, "Continuous query for qos-aware automatic service composition," in *2012 IEEE 19th International Conference on Web Services*. IEEE, 2012, pp. 50–57.

[4] S. Chattopadhyay and A. Banerjee, "Qos-aware automatic web service composition with multiple objectives," *ACM Transactions on the Web (TWEB)*, vol. 14, no. 3, pp. 1–38, 2020.

[5] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 241–246, 1979.

[6] S. S. Gokhale, W. E. Wong, K. S. Trivedi, and J. Horgan, "An analytical approach to architecture-based software reliability prediction," in *Proceedings. IEEE International Computer Performance and Dependability Symposium. IPDS'98 (Cat. No. 98TB100248)*. IEEE, 1998, pp. 13–22.

[7] R. C. Cheung, "A user-oriented software reliability model," *IEEE transactions on Software Engineering*, no. 2, pp. 118–125, 1980.

[8] O. Hatzi, D. Vrakas, M. Nikolaidou, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas, "An integrated approach to automated semantic web service composition through planning," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 319–332, 2011.

[9] W. Jiang, S. Hu, and Z. Liu, "Top k query for qos-aware automatic service composition," *IEEE Transactions on Services Computing*, vol. 7, no. 4, pp. 681–695, 2013.

[10] A. Bekkouche, S. M. Benslimane, M. Huchard, C. Tibermacine, F. Hadjila, and M. Merzoug, "Qos-aware optimal and automated semantic web service composition with user's constraints," *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 183–201, 2017. [Online]. Available: https://doi.org/10.1007/s11761-017-0205-1

[11] M. Driss, S. Ben Atitallah, A. Albalawi, and W. Boulila, "Req-wscomposer: a novel platform for requirements-driven composition of semantic web services," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–17, 2022.

[12] T.-T. Pham and X. Défago, "Reliability prediction for component-based systems: Incorporating error propagation analysis and different execution models," in *2012 12th International Conference on Quality Software*, 2012, pp. 106–115.

[13] Z. Zheng, T. C. Zhou, M. R. Lyu, and I. King, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2011.

[14] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation in web service compositions," in *2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*. IEEE, 2005, pp. 181–185.

[15] M. D. McIlroy, J. Buxton, P. Naur, and B. Randell, "Mass-produced software components," in *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*, 1968, pp. 88–98.

[16] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[17] Y. Chen, X. Yan, and A. A. Khan, "A novel reliability assessment method based on the effects of components," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 69–76.