Edge-Computing-Assisted Intelligent Processing of AI-Generated Image Content

Suzhen Wang
Hebei University of Economics and Business
Yongchen Deng
Hebei University of Economics and Business
Lisha Hu
Hebei University of Economics and Business
Ning Cao
ning.cao2008@hotmail.com

Wuxi Vocational College of Science and Technology

Research Article

Keywords: AIGIC, edge computing, serverless computing, WebAssembly, E-PPO2

Posted Date: September 25th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3366364/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Real-Time Image Processing on February 25th, 2024. See the published version at https://doi.org/10.1007/s11554-023-01400-w.

Edge-Computing-Assisted Intelligent Processing of AI-Generated Image Content

Suzhen Wang[1]·Yongchen Deng[1]·Lisha Hu[1]·Ning Cao[2, 3]

Abstract Artificial intelligence-generated image content (AIGIC) is produced through the extraction of features and patterns from a vast image dataset, requiring substantial computational resources for training. This study aims to enhance image processing and response time on terminal devices by utilizing edge computing technology to offload specific training tasks to edge nodes. Additionally, task offloading and resource allocation strategies are developed to effectively generate image content on terminal devices. Edge computing aims to execute computing tasks in close proximity to data sources; however, the computing resources of edge devices are limited. Therefore, the development of suitable resource allocation strategies for resource-constrained environments is crucial in edge computing research. Serverless computing, which heavily relies on container technology for program hosting, is recognized as one of the most suitable architectures for edge computing. WebAssembly (WASM) is a binary instruction format that operates on a stack and enables the execution of computing tasks on both client and server sides. Its advantages encompass reducing cold start time, enhancing efficiency, and improving portability, thereby addressing challenges encountered by container technology in Serverless deployments. This paper commences with an introduction and analysis of the research status of Serverless and WASM, subsequently delving into the investigation of task offloading and resource allocation in edge computing within the Serverless architecture supported by WASM. To facilitate collaboration among edge nodes, an enhanced deep reinforcement learning algorithm, called entropy-based Proximal Policy Optimization (E-PPO2), is employed. This algorithm allows edge devices to share a global reward and continuously update parameters, leading to an optimized response strategy and maximizing the utilization of edge device resources.

Keywords: AIGIC; edge computing; serverless computing; WebAssembly; E-PPO2

✉ **Correspondence**
Ning Cao
ning.cao2008@hotmail.com

[1] School of Information Technology, Hebei University of Economics and Business, Shijiazhuang, Hebei, China

[2] School of Integrated Circuits, Wuxi Vocational College of Science and Technology, Jiangsu, China

[3] School of Information Engineering, Shandong Vocational and Technical University of International Studies, Shandong, China

Funding information

Natural Science Foundation of Hebei Province, Grant/Award Number: F2021207005

1. Introduction

Due to the continuous advancements in deep learning and image processing technology, the capabilities of AIGIC have significantly improved, enabling its widespread application across various domains such as computer games, virtual reality, design, and artistic creation. However, the training process of AIGIC usually requires substantial computing

resources and storage capacity, limiting its efficient utilization on resource-constrained terminal devices. This study utilizes edge computing technology to resolve this challenge. The integration of AIGIC with edge computing technology can improve system efficiency, minimize network transmission overhead, reduce latency, bolster privacy protection, and improve reliability. Edge computing represents a distributed computing model that utilizes the computational capabilities of terminal devices and edge servers for data processing and decision-making in close proximity to data sources. By doing so, it intends to improve system performance, minimize latency, and strengthen data privacy and security. The fundamental idea of edge computing is to process computing tasks on computing resources close to data sources [1]. The rational task offloading to edge nodes and efficient allocation of limited computing resources to each computing task remain pivotal challenges in edge computing research.

Serverless was originally proposed for cloud computing environments, aiming to simplify the development and deployment process and improve efficiency. Serverless architecture includes Function as a Service (FaaS) and Backend as a Service (BaaS) [2], and its core idea is to execute functions on demand. The automatic scaling feature of the serverless architecture is well-suited for edge computing scenarios, where edge nodes typically experience changing load and traffic demands. The serverless architecture can automatically scale up or down the size of functions according to actual needs to meet the requirements of horizontal scaling. This automatic scaling function can effectively handle sudden high load situations and ensure system scalability and other performance metrics. The serverless architecture on the edge node can offer functions such as simplified deployment, elastic scaling, event driven and cost-effective without requiring expertise in underlying infrastructure details, making it an efficient and scalable application development and deployment model in the edge computing environment. However, there are still many challenges and issues that need to be addressed in Serverless [3,4], such as the increase in program running time caused by container cold start [5].

WebAssembly (WASM) is a low-level bytecode format designed for stack based virtual machines [6], designed to provide efficient cross platform execution environments. WASM is capable of running in web browsers and is also compatible with server-side and embedded devices. By packaging the program as a WASM module and utilizing the Serverless platform, faster cold start times can be realized without relying on Docker. This method can boost the responsiveness and scalability of the application while decreasing resource consumption and costs. Therefore, this paper investigates the serverless task offloading and resource allocation supported by WASM in edge computing environment. And in order to enhance the collaboration capability between edge nodes, E-PPO2 is employed to enable each edge node agent to share global rewards and develop final task offloading and resource allocation strategies.

2. Related works

Most major cloud service providers, such as Microsoft, Amazon, Google, and IBM, offer Serverless computing. With the exponential proliferation of IoT devices, an immense volume of data is being generated by numerous sensors, smart devices, and terminal devices. Many applications necessitate real-time or near real-time processing and decision-making capabilities due to their

reliance on this data, such as smart cities, autonomous driving, and industrial automation. However, deploying Serverless to the cloud center alone proves inadequate in effectively managing the surge of tasks originating from these terminals. Consequently, the integration of serverless computing with edge computing has emerged as a prominent development trend. Recognizing this need, certain cloud service providers have extended their offerings to include edge capabilities, exemplified by AWS IoT Greengrass [8] and Azure IoT Edge [9]. Recent study [10] have illustrated the successful utilization of Serverless in resource-constrained environments, particularly in edge computing scenarios. Consequently, a plethora of research endeavors have commenced to explore the application of Serverless in edge computing scenarios. The performance of Serverless in edge computing has been examined by previous studies [11-13]. In a comparative evaluation of various open-source serverless architectures, it was determined that Kubeless outperformed other architectures [14]. The utilization of container technology within the Serverless architecture was investigated in the context of video analytics applications at the edge [15]. While container technology effectively mitigates memory and CPU overhead, the cold start of containers introduces delays to the application. To assess the maturity of serverless computing methods in cloud edge environments, new standards were introduced [16]. Experimental results revealed that the current product exhibits immature features in data management, serverless programming models, and AI support.

An introductory document on WASM, which includes performance benchmarks, was published by a previous study [17]. Comparisons were made between the startup time of WASM and containers, revealing that WASM workloads exhibit a startup time about ten times faster with a lower memory footprint [18]. In another study, the cold start time and total execution time of WASM and containers were compared in 13 tests, highlighting the significant advantages of WASM [19]. Furthermore, the feasibility of utilizing edge WASM in serverless environments was demonstrated [20]. This study applied WASM to edge computing and conducted a performance comparison against native execution. The overall runtime of WASM exhibited a 2-5 times slower performance compared to local execution due to the introduction of an additional intermediate layer during actual execution [21, 22]. A hybrid serverless platform, proposed by a recent study [23], incorporates both WASM and Docker while extending Kubernetes (K8S) for container management. Experimental results have demonstrated significant advantages of this platform for serverless applications requiring frequent execution and elasticity. WASM technology introduces numerous benefits to the serverless environment, which are not achievable in the current operating environment [24]. Another study [25] explored the features and usage scenarios of edge serverless computing utilizing WASM. It also examined the current status, trends of adoption, and discussed the developmental direction of utilizing containers as edge serverless computing platforms within the WASM execution environment. Furthermore, in a novel approach suggested by researchers [26], an edge computing scenario without servers utilizing WASM was proposed. Experimental results revealed a 99.5% reduction in cold start delay and a boost of 4.2 times in function execution throughput.

The study by [27] examined the resource scheduling problem in serverless platforms and categorized them based on scheduling objectives: resource utilization, response time

delay, and multi-objective optimization. A comprehensive review by [28] discussed the current research status of key technologies in resource management for serverless platforms. In order to tackle the resource management bottlenecks arising from imbalanced call frequency and resource allocation in function applications, [29] presented two resource management strategies. [13] developed a real-time data analysis system utilizing serverless platform and cloud edge architecture. The system distinguished between multi-level real-time data analysis workflows based on time sensitivity. However, it did not address resource scheduling issues. In a comprehensive review of 164 papers, [30] analyzed the topics of performance optimization, programming framework, application migration, and multi-cloud development in the context of serverless computing. The analysis highlighted resource scheduling as a prominent research trend in the field. To tackle the challenge of heterogeneous computing resources in serverless computing, [31] employed deep reinforcement learning algorithms and conducted training and evaluation in a cloud center. The experimental results revealed a 24% increase in application response time and a 34% increase in resource utilization cost compared to the baseline technology. In conclusion, in the context of an edge computing environment and under the serverless architecture supported by WASM, further investigation is required to optimize the strategies for task offloading and resource allocation.

3. Edge Serverless computing

This section presents the multi-layer architecture of edge serverless computing, along with the utilization of WASM to support edge serverless computing. This includes an enhanced version of Kubernetes (K8S) for deploying and managing WASM modules, the selection of benchmark functions for testing, and the adoption of a Serverless platform.

3.1 Edge serverless computing multi-layer architecture

Edge serverless computing refers to the utilization of the Serverless computing platform on edge devices to carry out computational tasks. In the context of edge computing, Serverless computing plays a vital role in providing an efficient task processing platform between edge devices and clients. Edge serverless computing is characterized by a multi-layer architecture, illustrated in Figure 1. The initial layer comprises the edge device network, where diverse edge devices are deployed in proximity to users and capable of intercommunication. The orchestration and management of the edge device network are facilitated by edge controllers. The serverless layer facilitates the deployment, execution, and collaborative scheduling of serverless functions, establish direct connections with the user layer to receive and execute diverse edge applications.

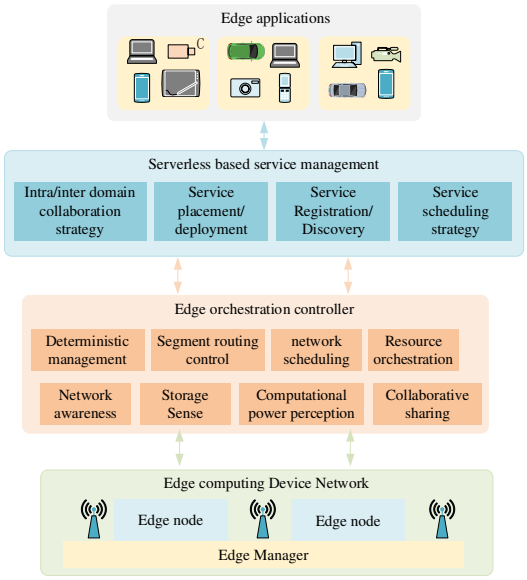


Fig.1 Edge serverless computing multi-layer architecture

3.2 Serverless Computing with WASM support

To facilitate the deployment and orchestration of containers, K8S is employed as the chosen tool in this article. In order to enable the deployment of WASM modules in K8S clusters and simulate the instantiation process of workload based on containers, a dedicated K8S WASM operator [23] is utilized. The K8S operator serves as a custom controller that extends the functionality of K8S. Leveraging the API and resource model of K8S, the operator manages and operates applications or services through automated programming. Operators typically comprise three components: Custom Resource Definition (CRD), controller, and automation operations. The CRD allows developers to define new resource types for describing and managing the configuration and status of specific applications or services. Thus, operators are capable of extending the resource model of K8S via CRD. In this article, a new CRD called WASM-CRD is introduced to describe WASM workloads. The newly defined CRD seamlessly integrates with existing K8S tools and can be readily incorporated into K8S-based platforms. Figure 2 illustrates the serverless deployment process facilitated by WASM, depicting the following steps in the deployment process:

1. Create a new WASM-CRD in the cluster;

2. Generate events from computational tasks to trigger the coordination function of the K8S operator. The operator verifies the parameters passed in WASM-CRD, and if successful, instantiates the built-in resources. And each WASM-CRD has at least the following resource support: deployment, specifying OCI images, customizing K8S RuntimeClass, and exposed ports;

3. Create portal services and deploy resources;

4. After creating a deployment, the pod instantiation will be triggered, and the pod will be scheduled by the K8S scheduler on the selected node. Bind an HTTP listener on a randomly selected port, which is shared with the K8S operator through a Redis instance;

5. The local Kubelet running on the node is based on the associated RuntimeClass, and calls the modified version of Spin Ship and deploys the WASM module;

6. Store the random port of the Spin instance in Redis;

7. WASM module reporting status;

8. Container reports the current status to K8S;

9. The module Pod representing WASM has entered a ready state;

10. Once the module Pod representing WASM enters the ready state, the K8S operator will query the Redis instance;

11. Obtain the value of the random port and correct the Service and Ingress objects.

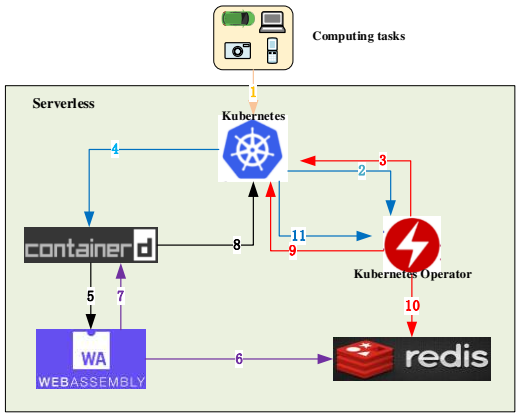


Fig.2 serverless deployment process supported by WASM

3.3 Selection of benchmark functions for testing and the serverless platform

This study deploys the Serverless platform in the edge computing environment to address the challenges of task offloading and resource allocation. To validate the effectiveness of the chosen platform, a benchmark testing function is employed. The Serverless benchmark function serves as a standardized tool for evaluating the performance of various Serverless platforms, measuring key indicators such as cold start latency, execution time, and

resource utilization. The benchmark function can be tailored and expanded as per specific requirements to assess the performance and capabilities of different Serverless platforms. To obtain more precise performance measurements, factors such as request concurrency, request frequency, and data volume should be taken into consideration during performance testing.

To assess the performance of the serverless architecture supported by the proposed WASM technology and make a comparison with existing container-oriented serverless platforms, this study employs an expanded set of serverless benchmark testing functions [32]. The set comprises 13 test functions (refer to Table 1), all of which are compatible with the WASM execution environment of the shared code library. The execution process of each test function involves the following steps: firstly, compiling the function code into a WASM module, then bundling it with the necessary dependencies, and finally deploying it onto the serverless platform. Subsequently, the function can be tested by triggering its execution. Function triggering can be accomplished through various means, such as invoking the function's API endpoint, sending specific events, or utilizing the triggers provided by the platform. Upon triggering a function, the platform initiates the loading of the corresponding WASM module and commences the execution of its code. The WASM-based serverless platform employs the Wasmtime Runtime (Wasmtime) as the execution engine for the WASM module. Wasmtime, an open-source WASM runtime, enables the execution of WASM modules within a server environment. When a function is triggered, Wasmtime loads the associated WASM module and executes its code within a virtual machine. During the execution of the function, Wasmtime parses the input parameters and carries out the function's code. The function's code is exported through the functions defined in the WASM module and interacts with the platform's provided API. Subsequently, upon completion of the function's execution, Wasmtime delivers the result to the caller. Throughout the execution process, various performance metrics, such as execution time and memory usage, are continuously monitored and recorded.

Table 1 Severless test benchmark contains test function information

Name	Text number	P.Lang[1]	Description	Type[2]
audio-sine-wave	(1)	Rust	Render a 440Hz sine wave and store it as a mono .wav file	M
fuzzysearch	(2)	Rust	Search for the occurrence of a phrase in a text file	R
n-body	(3)	Rust	Model the orbits of Jovian planets	M
prime-numbers	(4)	Rust	Search for prime numbers among the first n numbers	M
whatlang	(5)	Rust	Determine the natural language of a given string	R
zip-compression	(6)	Rust	Compress multiple files in a single zip archive	R
aes	(7)	Go	Encrypt a given text using AES symmetrical encryption multiple times	R
checksum	(8)	Go	Calculate the MD5, SHA256, and SHA512 checksums of a given file	R
diskio	(9)	Go	Write and then read back a file containing 100000 lines of text	M
float-operation	(10)	Go	Calculate sin, cos and square root of an arbitrary number	M
imageprocessing	(11)	Go	Rotate an image and change its color palette	R
linear-equations	(12)	Go	Solve a system of linear equations	M
matmul	(13)	Go	Square matrix multiplication	M

[1]Source programming language of the given benchmarking function

[2]M - Microbenchmark; R - Real-word workload

OpenFaaS is specifically designed for K8S and can be deployed to effectively utilize various functions of K8S, such as automatic scaling, service discovery, and load balancing. The OpenFaaS platform supports the use of WASM technology to write and deploy functions. Compared with other serverless platforms, OpenFaaS is more suitable for edge computing application scenarios. Therefore, this paper uses OpenFaaS based on WASM technology as the serverless platform to study task unloading and resource allocation strategies in edge computing environment.

4 Research on Task Unloading and Resource Allocation Strategies

4.1Network model construction

Each edge system consists of a wireless base station and *K* edge servers, where the edge servers are represented as:$r = \{U_1, U_2, \dots, U_k\}$, each edge server U_i has *N* independent computing tasks, and the task set of U_i is represented as $\Gamma_i = \{T_{i,1}, T_{i,2}, \dots, T_{iN}\}$. The edge computing server provides the device with high-intensity computing services. The terminal device offloads computing tasks to the edge server, and the server allocates resources to each user to reduce the delay and energy consumption of the user device. In this model, the cache of unprocessed tasks stored by the edge server is large enough, and both the user device and the edge server are equipped with a single core CPU. All tasks are processed in the order of the input CPU.

4.2 Problem Description

The system consists of *K* edge servers, and all task sets of all edge servers are represented as $G = \{T_{i,j} | 1 \le i \le K, 1 \le j \le N\}$. There is a total of *K*N* (denoted as *m*) independent computing tasks that need to be processed, where all tasks can be uninstalled and scheduled. Each task $T_{i,j}$ is described by two parameters: $D_{i,j}$ and $C_{i,j}$,which $D_{i,j}$ represents the amount of task data that needs to be processed and $C_{i,j}$ represents the number of CPU cycles required to process each unit of data for that task. In this article, the main focus is on coarse-grained task offloading and resource allocation. Tasks are not further divided and each device is only equipped with one antenna, which can only transmit one task at a time; All terminal devices share the transmission bandwidth equally, so the rate at which the computing tasks $T_{i,j}$ of the terminal devices U_i are transmitted to the edge server is shown in formula (1):

$$R = (B/K) log_2[1 + S/N] \quad (1)$$

Where *B* is the channel bandwidth, in hertz, *S* represents the average power of the signal transmitted within the channel, and *N* is the internal Gaussian noise power of the channel. Therefore, the time for each task $T_{i,j}$ to be transmitted from the terminal device to the edge server is shown in formula (2).

$$t_{i,j}^{trans} = D_{i,j}/R \quad (2)$$

The total frequency of the server is *F*, the frequency assigned to the edge server U_i is $f_{i,ser}$, and the CPU frequency for tasks to be executed locally is $f_{i,user}$ .Therefore, the execution time of the task $T_{i,j}$ on the edge server U_i and locally is shown in formulas (3) and (4).

$$t_{i,j}^{edge} = C_{i,j}/f_{i,ser} \quad (3)$$

$$t_{i,j}^{local} = C_{i,j}/f_{i,user} \quad (4)$$

The energy consumption for task $T_{i,j}$ to be executed locally and unloaded to edge servers are shown in formulas (5), (6). The transmission energy consumption is shown in

formulas (7).

$$e_{i,j,local} = \delta_i f_{i,user}^2 C_{i,j} \quad (5)$$

$$e_{i,j,edge} = \delta_i f_{i,ser}^2 C_{i,j} \quad (6)$$

$$e_{i,j} = t_{i,j}^{trans} p \quad (7)$$

Among them, δ_i is the effective capacitance coefficient of the edge server, and p is the transmission power of the terminal device unloading tasks to the edge server. Therefore, the total energy consumption for processing tasks is:

$$e_i = \sum_{j=1}^{N} [\ (e_{i,j} + e_{i,j,edge})\ x_{i,j} + e_{i,j,local}$$

$$(1 - x_{i,j})\ ] \quad (8)$$

Where $x_{i,j}$ is the task offloading coefficient, $x_{i,j} \in \{0,1\}$. When the value of $x_{i,j}$ is 0, it indicates that the task is executed locally, and when the value is 1, it indicates that the task is executed on edge server *j*. The total consumption of the system is the weighted sum of the completion time of each user's last task and the energy consumption of the edge server:

$$E_i = max\{t_{i,j,l}^q, t_{i,j,s}^q\} + e_i \quad (9)$$

Among them, $t_{i,j,l}^q$ represents the time required for the task queue to be executed locally, and $t_{i,j,s}^q$ represents the time required for the task queue to be executed on the edge server. The mathematical model described in this article is a constrained joint optimization problem that optimizes offloading decisions, offloading scheduling, and server resource allocation under limited server resources. Therefore, the optimization objective of the model can be expressed as:

$$P1:\ min \sum_{i=1}^{K} E_i \quad (10a)$$

$$\text{s.t.} \sum_{i=1}^{K} f_{i,ser} \le F \quad (10b)$$

$$f_{i,ser} \le F,\ \forall i \in r \quad (10c)$$

$$x_{i,j} \in \{0,1\},\ \forall T_{i,j} \in G \quad (10d)$$

The variables optimized in this article include task offloading decisions and edge server CPU frequency allocation. The optimization objective (10a) is the weighted sum of system delay and energy consumption. The constraint (10b) is that the total frequency assigned to the server cannot exceed the maximum frequency. The constraint condition (10c) indicates that the frequency allocated to each edge server cannot exceed the maximum frequency. The constraint condition (10d) indicates that the execution location of the task is only on terminal devices and edge servers. The optimization objective (10a) can be solved by finding the optimal decision and calculating the resource allocation for unloading. However, the unloading decision vector X is a feasible set of binary variables, and the objective function is a non-convex problem. As the number of tasks increases, the difficulty of solving the objective function increases exponentially, making it an NP hard problem. This article uses an improved deep reinforcement learning method (E-PPO2) for solving.

4.3 algorithm optimization

4.3.1 MDP optimization problem

In this section, the defined total system consumption is modeled as a Markov Decision Process (MDP), which can be represented by tuples *(S, A, P, R)*, where *S* is the object in the state space representing the next state. *A* is the action space, representing the object of the next action. *P* is the probability set of transitioning from the current state to another state. *R* is the reward function corresponding to the action in a specific state. The goal of MDP is to find a policy and select the optimal action decision

rule in each state to maximize cumulative rewards and obtain the optimal set of policies.

A. State

The state space of MDP can be represented as: $S = \{s_t = \{\left(x_m(t), F(t), E_{budget}(t)\right)\}\}$. Where s_t represents the current state of all tasks in time slot *t*, $s_t = \{s_1, \dots, s_m\}$. $x_m(t)$ represents the computational characteristics of all tasks in time slot *t*. $F(t) = \{f_1, \dots, f_k\}$ indicates the CPU computing power corresponding to the edge server in time slot *t*. $E_{budget}(t) = \{E_i\}$ indicates the total consumption of tasks executed locally or uninstalled to edge servers during time slot *t*.

B. Action

The actions selected for each task in its current state can be defined as: $A = \{a_t = \{\chi_m(t), \xi_m(t)\}\}$, where $\chi_m(t) \in \{0,1\}$ represents the execution location of the task, $\chi_m(t) = 0$ represents the task being executed locally, and $\chi_m(t) = 1$ represents the task being unloaded to the edge server for execution. $\xi_m(t) \in \{1,2, \dots, k\}$ indicates the final execution location for uninstalling to the edge server, such as when the task is uninstalled to the *n* edge server, $\xi_m(t) = n$.

C. State-to-state transition

The transition probability between states is expressed as: $P = \{P_{at}(s, s_{t+1}) | s, s_{t+1} \in S, a_t \in A, p \in P\}$. Its meaning is that the probability of taking action a_t in state s_t and entering the state s_{t+1} is $P_{at}(s, s_{t+1})$.

D. Reward function

In each state transition process, the decision-maker receives a reward or punishment. In deep reinforcement learning algorithms, rewards can be represented by states and actions: $S \times A \rightarrow R$. Therefore, in this system, the reward function is defined as: $R = \{r_t(s_t, a_t)\}$, which $r_t(s_t, a_t)$ can also be represented as:

$$r_t(s_t, a_t) = minimize \sum_{m=1}^{M} \chi_m E_m^e + (1 - \chi_m) E_m^l \quad (11)$$

By introducing a state action function $Q(s_t, a_t) = E_\pi[r_t | S = s_t, A = a_t]\}$ in MDP, the system consumption is minimized by selecting actions a_t in the current state s_t, resulting in an overall task offloading and resource allocation strategy π. The update function of $Q(s_t, a_t)$ is shown in formula (12).

$$Q_\pi^*(s_t + a_t) = \mathrm{r}(s_t + a_t) + \sum_{s_{t+1} \in S} P_{at}(s_t, s_{t+1}) \sum \pi(a_{t+1} | s_{t+1}) (Q_\pi (s_{t+1}, a_{t+1})) \quad (12)$$

Among them, $P_{at}(s_t, s_{t+1})$ is defined as the probability of state s_t entering the next state s_{t+1}, and task offloading strategy π_m^* is obtained to maximize long-term discount rewards. γ is the discount parameter of reward importance and π_m^* is expressed as:

$$\pi_m^* = argmax_{\pi m} E[\sum_{t \in T} \gamma^{t-1} R(s_t, a_t) | \pi_m] \quad (13)$$

4.3.2 E-PPO2 algorithm

In this section, the PPO2 algorithm based on maximum entropy is used to solve the MDP optimization problem. Entropy is used to measure the uncertainty or information content of random variables. For discrete random variable *X*, entropy *H (X)* can be calculated using formula (14). Entropy has wide applications in fields such as information theory, statistics, and machine learning [33]. In this article, the exploration ability of task offloading strategy is improved by using maximum entropy optimization reward function, which increases the entropy determined by the action distribution on the basis of the original reward. The modified reward function is shown in formula (15).

$$H(X) = -\sum P(X) * \log 2(P(x)) \quad (14)$$

$$R_{t+1}^e = R_{t+1} + eH\left(\pi(\cdot | s_t)\right), t = 0,1,2, \dots \quad (15)$$

e is a random parameter that controls the

optimal strategy and *e>0*. $H(\pi(\cdot|s_t)) = -log\pi(\cdot|s_t)$ represents the entropy of strategy π in the state s_t. The long-term reward G_t^e based on entropy, value function $V_\pi^e(s_t)$, and state action $Q_\pi^e(s_t, a_t)$ function are shown in formulas (16), (17) and (18).

$$G_t^e = \sum_{t=0}^{+\infty} \gamma^t R_{t+1}^e, t = 0,1,.... \quad (16)$$

$$V_\pi^e(s_t) = E_\pi[G_t^e|s_t], s_t \in S \quad (17)$$

$$Q_\pi^e(s_t, a_t) = E_\pi[G_t^e|s_t, a_t], s_t \in S, a_t \in A \quad (18)$$

The modified state value function V_θ（s_t） and strategy function π_θ（$s_t|a_t$） can be obtained from formulas (16) to (18), as shown in formulas (19) and (20).

$$V_\theta(s_t) = E_{A\sim\pi}[Q_\pi^e(s_t, A)] + eH(\pi(\cdot|S)) = E_{A\sim\pi}[Q_\pi^e(s_t, A)] - elog\pi(A|s_t), s_t \in S \quad (19)$$

$$\pi_\theta(s_t|a_t) = argmax\, E_{(s_t,a_t)\sim p(\pi)}[\sum_{t=0}^{+\infty} \gamma^t (r_t(s_t, a_t) - elog\pi(\cdot|s_t))] \quad (20)$$

The PPO2 algorithm is a new type of Policy Gradient algorithm (PG), whose main principle is to use gradient enhancement to update strategy π to maximize expected returns. In the PG algorithm, the updated objective function of network parameters θ is as follows:

$$L^{PG}(\theta) = E_t[lg\pi_\theta(a_t|s_t) \times A_t] \quad (21)$$

In reinforcement learning, the process of policy updating may lead to drastic fluctuations in policy performance, and if the policy update amplitude is too large, it may lead to overfitting of current empirical data, thereby reducing the generalization ability of the policy in unknown environments. Therefore, this article introduces comparative parameters between the new strategy and the old strategy in the pruning objective function to maintain the stability of policy updates. The ratio of the action probability under the current strategy to the action probability under the previous strategy is denoted as $p_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta old}(a_t|s_t)$. To avoid policy mutations during the parameter update process, the objective function is constrained. The PPO algorithm limits the training stability to a small range through policy updates. The PPO2 algorithm can use two constraints: limiting KL divergence and truncation function. This article uses a truncation function to optimize the PPO2 objective function, as shown in formula (22).

$$L_{\pi_\theta}^{CLIP}(\pi\theta_{old}) = E_{r\sim\pi_\theta}[\sum_{t=0}^{T}[\min(p_t(\pi_\theta, \pi_{\theta old})A_t^{\pi_{\theta old}}, clip(p_t(\pi_\theta, \pi_{\theta old}), 1-\epsilon, 1+\epsilon)A_t^{\pi_{\theta old}})]] \quad (22)$$

ϵ is a relatively small value, and the experimental results were good when $\epsilon = 0.2$ in this article. The *clip* operation in formula (12) limits the amplitude of policy updates to avoid excessive deviation from a single policy update. The structure of the E-PPO2 algorithm is shown in Figure 3.

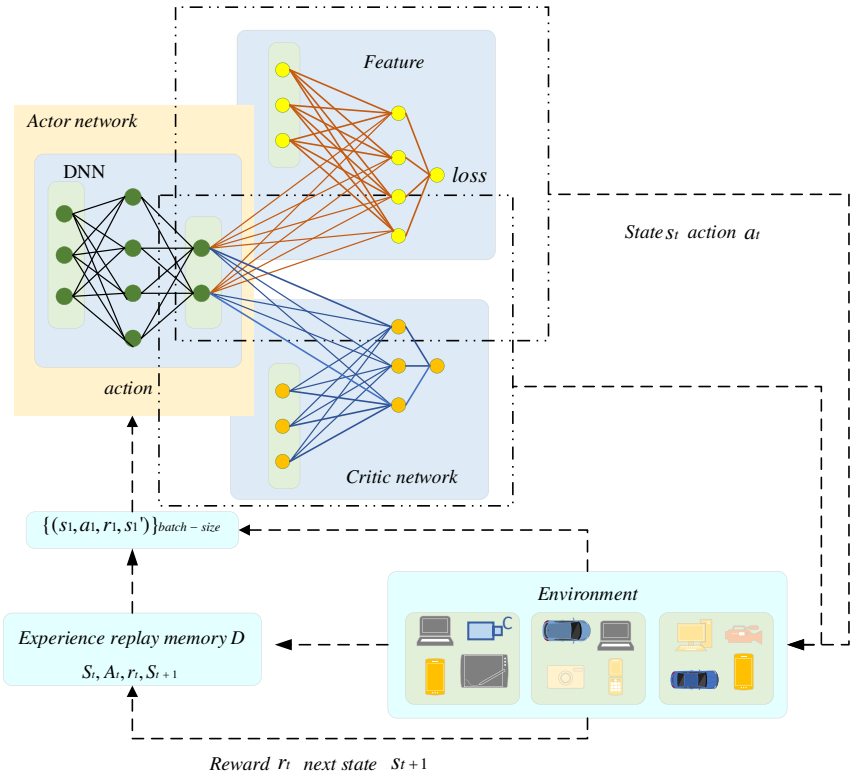
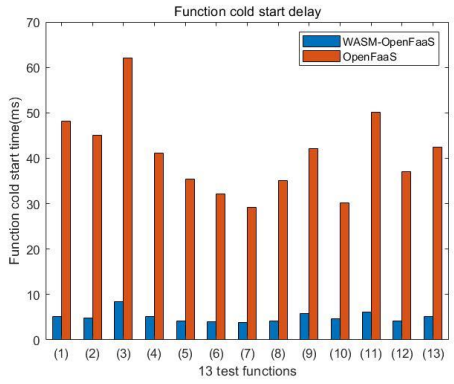


Fig.3 E-PPO2 algorithm architecture

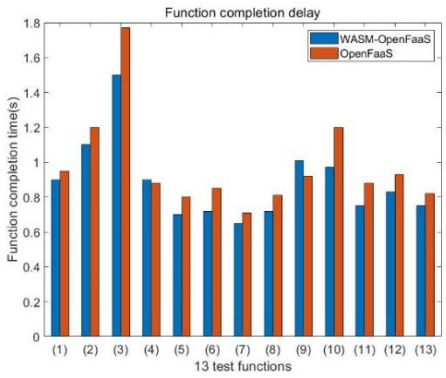
5 Experimental results

The experiment in this article is divided into two parts. In the first part, in order to verify the effectiveness of the proposed serverless platform based on SWAM technology, 13 benchmark test functions were used to compare the performance of different platforms. The platform performance was evaluated from two indicators: the cold start delay of the function and the overall execution delay of the function. In the second part, in order to verify the effectiveness of the proposed task unloading and resource allocation strategies in the edge computing scenario, the proposed E-PPO2 strategy is compared with the existing three task unloading methods.

In this paper, serverless computing is introduced into the edge computing environment. In order to reduce the container cold start delay, WASM is used to replace the container technology in the serverless platform. In order to verify the effectiveness of using WASM technology on the Severless platform, this article uses 13 benchmark testing functions to perform performance tests on the OpenFaaS platform based on WASM technology and the unmodified OpenFaaS platform. Performance evaluation is conducted from two indicators: cold start time and function completion time. The experimental results are shown in Figure 4. From Figure 4 (a), it can be seen that the OpenFaaS serverless platform based on WASM can effectively solve the problem of cold start latency for functions, with a reduction of approximately 81% in cold start latency. Figure 4 (b) shows the completion time of the test function on different Serverless platforms, and the results show that the execution speed of the test function on the improved platform has increased by approximately 5%.



(a)



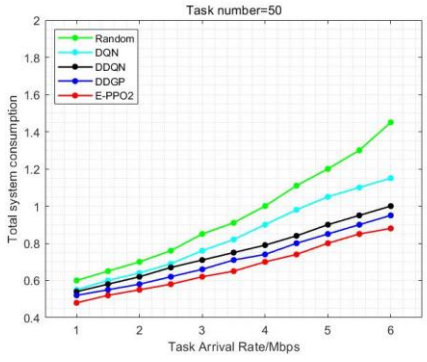
(b)

Fig.4 Cold start time and completion time of 13 test functions on different Serverless platforms: **(a)** Cold start time of test functions; **(b)** Completion time of test functions

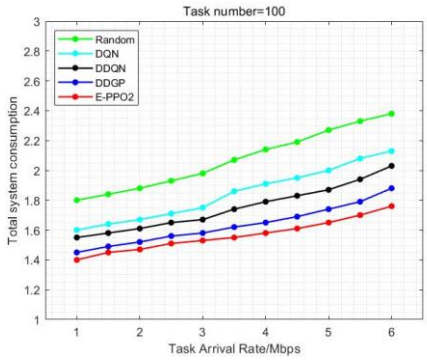
To verify the effectiveness of the task offloading and resource allocation strategies proposed in this article, the E-PPO2 algorithm was compared with random algorithms (Random), DDPG (Deep Deterministic Policy Gradient) [34], DQN (Deep Q-network), and DDQN (Double DQN) [35]. First, considering the edge computing scenario, we set the number of tasks to 5~100 and a total of 10 edge nodes. The calculation task transmission bandwidth is 6MHz, the task transmission power is 25dBm, the calculation power of the edge node CPU is 50GHz, the calculation power of the terminal device CPU is 10GHz, and the discount parameter is 0.5. The E-PPO2 algorithm parameters are shown in Table 2.

Table .2 E-PPO2 algorithm parameters

Parameters	Value
Learning rate	3e-4
Batch_size	64
N_Steps	2048
Clip_range	0.2
Gramma	0.99
Gae-lambda	0.95



(a)

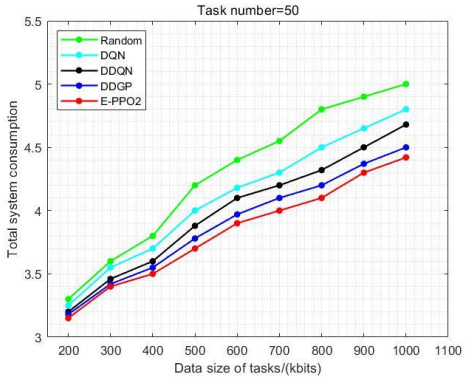


(b)

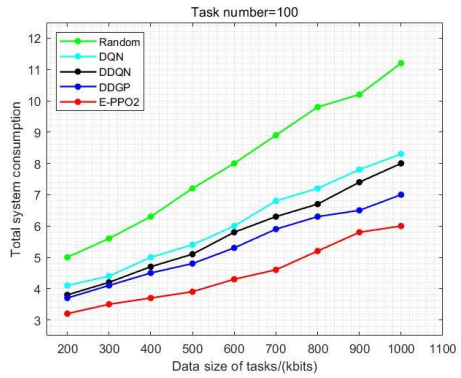
Fig.5 For the total system consumption of different strategies under different task arrival rates, the number of tasks in (a) is set to 50, and the number of tasks in (b) is set to 100.

The total system consumption of the five methods shows a certain growth trend with the increase of task arrival rate, but overall, the DQN, DDQN, DDGP, and E-PPO2 algorithms outperform the random algorithm. This is because the deep reinforcement learning algorithm introduces an experience playback process for obtaining intelligent computing offloading decisions. The system consumption of the E-PPO2 algorithm is superior to the total system consumption of the DDGP

algorithm. This is because the unloading decisions of E-PPO2 and DDGP are generated through learning and training data. DDPG adopts a deterministic strategy, while E-PPO2 adopts a random training strategy, and introduces entropy in the training to improve exploration ability.



(a)



(b)

Fig.6 Total system consumption under different task sizes: (a) 50 task quantities (b) 100 task quantities

Figure 6 shows the system consumption of different strategies under different task sizes. The number of tasks in Figures 6 (a) and (b) is set to 50 and 100, respectively. As the task data volume increases, the system consumption gradually increases. The random algorithm has the fastest increase in system consumption, while the growth of the E-PPO2 algorithm is significantly lower than other strategies. This is mainly because the E-PPO2 algorithm can effectively solve the unknown load state at the edge server. When the task data volume increases to 900kbits, the E-PPO2 strategy reduces system consumption by approximately 8% compared to the DDGP offloading strategy.

6 Conclusion

This paper investigates the synergy between Artificial Intelligence Generated Image Content (AIGIC) and edge computing, enabling efficient image content generation on terminal devices. In contrast to traditional generation models, leveraging edge computing technology allows for on-device model reasoning and calculation, thereby reducing data transmission, latency, network bandwidth pressure, and enhancing the user experience. The combined implementation of artificial intelligence image generation and edge computing technology offers low computational resource requirements. By offloading certain model computing tasks to the edge server for processing, the burden on the cloud server is alleviated, leading to reduced energy consumption. Additionally, it enhances real-time image generation, mitigates operational expenses, and proves conducive to resource-constrained devices or applications. This paper integrates edge computing technology with serverless technology, leveraging the characteristics of the serverless architecture, such as resource management, automatic scalability, rapid deployment, and distributed data. These features render it well-suited for edge computing scenarios. Addressing the challenge of cold start latency in Serverless containers, this article proposes an effective solution utilizing WebAssembly (WASM) technology. However, it should be noted that WASM technology cannot entirely replace container technology due to their distinct application scenarios and respective advantages. Consequently, to extend the applicability of the Serverless platform across diverse scenarios, the research direction is

currently focused on a hybrid Serverless platform that combines WASM and Docker technology.

To address the task offloading and resource allocation challenge in edge computing scenarios, this paper aims to minimize the combined weights of delay and energy consumption. A model for task offloading and resource allocation is constructed, and an optimization algorithm based on maximum entropy, referred to as the near-end strategy optimization algorithm, is proposed. The effectiveness of the E-PPO2 algorithm is validated through experimental evaluation. Moving forward, we will incorporate the mobility factor of end users and devise dynamic strategies for task offloading and resource allocation.

References:

[1] Shi Weisong, Zhang Xingzhou, Yifan Wang, and Zhang Qingyang, Edge computing: State-of-the-Art and future directions, *Journal of Computer Research and Development*, pp. 69-89, 2019.

[2] M.S. Aslanpour, A.N. Toosi, C. Cicconetti, B. Javadi, and P. Sbarski, Serverless Edge Computing: Vision and Challenges, in Proc. 2021 Australasian Computer Science Week Multiconference, 2021.

[3] Joseph M. Hellerstein, Jose Faleiro Joseph E, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and C. Wu, Serverless Computing: One Step Forward, Two Steps Back, *arXiv preprint*, 2018.

[4] V. Kjorveziroski, C. Bernad Canto, P. Juan Roig, K. Gilly, A. Mishev, V. Trajkovikj, and S. Filiposka, IoT serverless computing at the edge: Open issues and research direction, *Transactions on Networks and Communications*, 2021.

[5] P. Mendki, Evaluating webassembly enabled serverless approach for edge computing, in Proc. 2020 IEEE Cloud Summit. IEEE, 2020.

[6] , Webassembly, in Book Webassembly, vol. 2023, *Series* Webassembly, 2020.

[7] J. Wen, Y. Liu, Z. Chen, J. Chen, and Y. Ma, Characterizing commodity serverless computing platforms, *Journal of Software: Evolution and Process*, 2021.

[8] Aws iot greengrass - amazon web services, in Book Aws iot greengrass - amazon web services, vol. 2023, *Series* Aws iot greengrass - amazon web services, 2020.

[9] Iot edge | microsoft azure, in Book Iot edge | microsoft azure, vol. 2023, *Series* Iot edge | microsoft azure, 2020

[10] V. Kjorveziroski and S. Filiposka, Kubernetes distributions for the edge: serverless performance evaluation, *The Journal of Supercomputing*, pp. 13728-13755, 2022.

[11] Baresi, Luciano and D.F. Mendonça, Towards a serverless platform for edge computing, in Proc. 2019 IEEE International Conference on Fog Computing (ICFC), 2019.

[12] P. Mendki, Blockchain enabled IoT edge computing, in Proc. Proceedings of the 2019 international conference on blockchain technology, 2019.

[13] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, and Bojana Koteska, A serverless real-time data analytics platform for edge computing, *IEEE INTERNET COMPUT*, pp. 64-71, 2017.

[14] S.K. Mohanty, G. Premsankar and M. di Francesco, An Evaluation of Open Source Serverless Computing Frameworks, *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018.

[15] P. Mendki, Docker container based analytics at IoT edge Video analytics usecase, in Proc. 2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU), 2018.

[16] P. Raith, S. Nastic and S. Dustdar, Serverless Edge Computing—Where We Are and What Lies Ahead, *IEEE Internet Computing*,

2023.
[17] A. Haas, A. Rossberg, D.L. Schuff, B.L. Titzer, M. Holman, and E.A. Dan Gohman, Bringing the web up to speed with WebAssembly, in Proc. Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017.
[18] J. Long, H. Tai, S. Hsieh, and M.J. Yuan, A Lightweight Design for Serverless Function as a Service, *IEEE SOFTWARE*, pp. 75-80, 2020.
[19] V. Kjorveziroski and S. Filiposka, WebAssembly as an Enabler for Next Generation Serverless Computing, *J GRID COMPUT*, 2023.
[20] A. Hall and U. Ramachandran, An execution model for serverless functions at the edge, in Proc. Proceedings of the International Conference on Internet of Things Design and Implementation. 2019, 2019, pp. 225-236.
[21] D. Hockley and C. Williamson, Benchmarking Runtime Scripting Performance in Wasmer, in Proc. Companion of the 2022 ACM/SPEC International Conference on Performance Engineering, 2022.
[22] Abhinav Jangda, Bobby Powers Emery D. Berger and A. Guha, Not so fast: Analyzing the performance of {WebAssembly} vs. native code, in Proc. 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019, 2019, pp. 107-120.
[23] V. Kjorveziroski and S. Filiposka, WebAssembly Orchestration in the Context of Serverless Computing, *J NETW SYST MANAG*, 2023.
[24] J. Ménétrey, M. Pasin, P. Felber, and V. Schiavoni, WebAssembly as a Common Layer for the Cloud-edge Continuum, in Proc. Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, 2022.
[25] Xin Wang, Kai Zhao and Qin Bin, Overview of WebAssembly Application Research for Edge serverless Computing, *Computer Engineering and Applications*, 2023.
[26] P. Gackstatter, P.A. Frangoudis and S. Dustdar, Pushing serverless to the edge with webassembly runtimes, in Proc. 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2022.
[27] Zehua Ma, Bo Liu, Weiwei Lin, and jiawei li, Overview of Resource Scheduling on Server Free Platforms, *computer science*, 2021.
[28] Baiai Yang, Shan Zhao and F. Liu, Overview of serverless computing technology research, *Computer Engineering & Science*, pp. 611-619, 2022.
[29] Mohammad Shahrad, R.F.Í. Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and R. Bianchini, Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider, in Proc. 2020 USENIX annual technical conference (USENIX ATC 20), 2020, pp. 205-218.
[30] J. Wen, Z. Chen, X. Jin, and X. Liu, Rise of the planet of serverless computing: A systematic review, *ACM T SOFTW ENG METH*, 2023.
[31] A. Mampage, S. Karunasekera and R. Buyya, Deep reinforcement learning for application scheduling in resource-constrained, multi-tenant serverless computing environments, *Future Generation Computer Systems*, 2023.
[32] J. Kim and K. Lee, Functionbench: A suite of workloads for serverless cloud function service, in Proc. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), 2019.
[33] G. Ciuperca, V. Girardin and L. Lhote, Computation and estimation of generalized entropy rates for denumerable Markov chains, *IEEE T INFORM THEORY*, 2011.
[34] Y. Ren, X. Yu, X. Chen, S. Guo, and Q. Xue-Song, Vehicular network edge intelligent management: A deep deterministic policy gradient approach for service offloading decision, in Proc. 2020 International Wireless Communications and Mobile Computing (IWCMC), 2020.

[35] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning, *IEEE INTERNET THINGS*, 2018.