AttentionFunc: Balancing FaaS Compute across Edge-Cloud Continuum with Reinforcement Learning

Kexin Li*
Laboratory of Intelligent Collaborative Computing School of Computer Science and Engineering, University of Electronic Science and Technology of China
Chengdu, China

Stefan Nastic
Nastic@dsg.tuwien.ac.at
Distributed Systems Group TU Wien
Vienna, Austria

ABSTRACT

Serverless computing is emerging as a promising paradigm to manage compute in Edge-Cloud continuum. However, distributing and balancing the computational load (serverless functions) across the continuum remains a significant challenge. In this paper, we introduce AttentionFunc – a novel framework for decentralized and efficient function offloading and computation balancing in the Edge-Cloud continuum. The AttentionFunc framework strives to introduce a fully decentralized decision-making model that accounts for the multi-objective nature of serverless workflows, the limitations of shared resources in the Edge-Cloud environment, and the dynamic behaviors such as resource contentions or cooperations among serverless functions. In addition, AttentionFunc incorporates an innovative multi-agent offloading model based on the Markov Decision Process (MDP), designed to minimize functions' execution time and costs. The application of MDP allows the framework to efficiently address these issues using deep reinforcement learning approaches, with an aim to significantly improve function completion latency. Furthermore, AttentionFunc pioneers an attention-based optimization mechanism for multi-agent deep reinforcement learning. This mechanism permits DRL agents to reach a consensus with minimal coordination information, leading to substantial reductions in communication and computation overhead. We evaluate AttentionFunc and compare it against select relevant state-of-the-art approaches. Our experiments and simulations show that AttentionFunc outperforms state-of-the-art approaches in terms of 1) the completion latency (up to 44.2% reduction), 2) the function success rate (up to 43.3% increase). Additionally, we provide the results of many experiments with different MEC scenarios to highlight the components of our approach that influence the results. We conclude that our approach reduces the low-latency challenge faced by most offloading models and improves the successful completion rate of the function.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Computing methodologies → Multi-agent reinforcement learning**; **Modeling and simulation**.

KEYWORDS

Multi-agent reinforcement learning, serverless functions offloading, Edge-Cloud computing continuum, FaaS

1 INTRODUCTION

As computation-intensive Edge-Cloud applications grow in complexity and scale, conventional resource provisioning and management become less efficient and more challenging. Serverless computing emerges as a potential remedy by offering scalable and cost-efficient solutions for executing Edge-Cloud applications. It introduces numerous benefits, like automatic scaling, scale-to-zero, reduced infrastructure management overhead, and innate reliability and availability [11], positioning serverless computing as a highly promising paradigm for the Edge-Cloud continuum [12]. However, Edge-Cloud serverless applications, unlike their purely cloud-based counterparts, function within a heterogeneous computing continuum/spectrum [14]. In this environment, not all devices possess the necessary computational power to perform intricate and resource-intensive tasks, often on behalf of IoT or mobile devices. Computation offloading addresses the limitations of individual Edge devices by distributing and balancing the computational burden across the Edge-Cloud continuum. This enhances application performance, prolongs device battery life, and reduces latency for users. Still, determining what and when to offload is a complex issue, given several factors that need to be considered, such as network conditions, the computational capacity of the device, the requirements of the application, and the cost of infrastructure (cloud) resources.

Serverless computation offloading: State of the art & limitations.

The research community has begun looking into finding solutions to better support serverless computing at the edge. Some new frameworks that were more suitable for edge environments have recently been proposed that leverage lightweight, functional sandbox mechanisms to achieve isolation guarantees for task parallelism, such as Faasm [16] and Sledge [8]. However, these solutions either work within a single edge node (e.g., [8]) or scale to multiple nodes without considering geographical distribution (e.g., [16]). On the other hand, a relatively large number of research efforts also work on proposing solutions to place and manage serverless functions and applications in the edge environment and consider their integration with serverless cloud services. They study optimal function placement, intending to minimize the completion time of serverless applications by making a trade-off between processing time and communication overhead [6, 10].

The above approach relies on centralized decision components, and serverless edge task scheduling is more challenging due to the heterogeneous and resource-constrained nature of edge resources. Other works (e.g., [15]) study architectures and algorithms for function placement and load distribution in decentralized Edge-Cloud

systems, which aims to address this problem by allocating admission, scheduling, and provisioning decisions[15][18]. However, relying on existing cloud-oriented frameworks for actual function execution may create some practical problems when running at the edge.

Based on the above discussion, the existing work on serverless edge computing mainly focuses on the design of static models and needs to consider better the dynamic network environment changes and resource-constrained situations during task scheduling. However, it is crucial to fully evaluate the dynamic serverless edge network environment in resource-constrained scenarios to make scheduling computing tasks more adaptable to realistic scenarios. This work fills this gap by considering proposed distributed task scheduling schemes in dynamic serverless Edge-Cloud computing networks.

Finding an optimal offloading policy in the Edge-Cloud continuum is challenging due to the complex and dynamic system state. The traditional methods are mainly based on heuristic algorithms for one-shot optimization, which leads to performance degradation in long-term operations. Therefore, we provide the optimal decision of the function as a Markov Decision Process problem.

Contributions. In this paper, we introduce AttentionFunc – a novel approach for distributed and decentralized function offloading in the Edge-Cloud continuum, based on a new *attention multi-agent deep reinforcement learning (AM-DRL)* method that we developed. By exploiting AM-DRL, AttentionFunc dynamically determines where to execute serverless functions (e.g., an Edge node or a in the cloud) to optimize the function execution time (completion latency) and the cost of its execution. The main contributions of this paper include:

- *A fully decentralized decision-making model for function offloading* that enables modeling/representing globally optimal offloading strategies in the Edge-Cloud continuum.
- *A novel multi-agent offloading model*, which represents the function offloading based on Markov Decision Process (MDP) with a goal to minimize the functions' execution time (completion latency) and their costs. Moreover, formulating the problem as MDP allows AttentionFunc to efficiently solve it with deep reinforcement learning approaches, yielding an improvement in the function's completion latency of 16.5%-44.2% compared to the state-of-the-art approaches.
- *An attention-based optimization mechanism* for multi-agent deep reinforcement learning, which allows the DRL agents to reach a consensus by sharing only the minimal coordination information among themselves, leading to significant reductions in communication and computation overhead (27.8% reduction), while improving the function success rate by 43.3% compared to the state-of-the-art approaches.

Outline. The rest of this paper is organized as follows. The AttentionFunc architecture and usage overview are reviewed in Section II. The AttentionFunc model is presented in Section III. The details of the AM-DRL function offloading algorithm are described in Section IV. The evaluation is presented in Section V. The related work is presented in Section VI. Finally, Section VII summarizes this paper and provides insights into possible future work.

2 ATTENTIONFUNC ARCHITECTURE & USAGE OVERVIEW

The main objective of the AttentionFunc framework is to facilitate decentralized and efficient serverless function offloading and computation balancing in the Edge-Cloud continuum. It aims to dynamically ascertain the best location to execute serverless functions, (e.g., on an Edge node or in the cloud), with a focus on optimizing both the function execution time (completion latency), network and computation overhead of the decision-making, and the associated execution costs.

Figure 1 gives a high-level architecture and usage overview of AttentionFunc framework. Each Edge node maintains a queue of incoming requests for computation. This *Edge Queue* is used by the user or an IoT/mobile device to invoke a serverless function, which performs some computation on their behalf. Each Edge node can receive the user request and needs to decide how and where to execute a corresponding function. To this end, it coordinates within its local *Communication Group* and if necessary triggers the *Function Offloading* process. Conversely, to integrate with a cloud provider, such as AWS, AttentionFunc expects typical APIs for invoking serverless functions. Note that AttentionFunc does not make any assumptions about the type or implementation of the serverless function, but they need to be configured and ready for execution. This means that AttentionFunc relies on the cloud platform to provision, schedule and execute the function. However, AttentionFunc framework makes sure that the results of the function execution are eventually delivered to the calling Edge node.

2.1 Functions Offloading & Decentralized Decision-Making

Figure 1 (bottom-left) also gives a high-level overview of the main steps of function offloading. As shown in the figure, an Edge node (EN) requests a specific number of stateless functions at the time t for the function x_m^t. Due to the limited computing power of devices

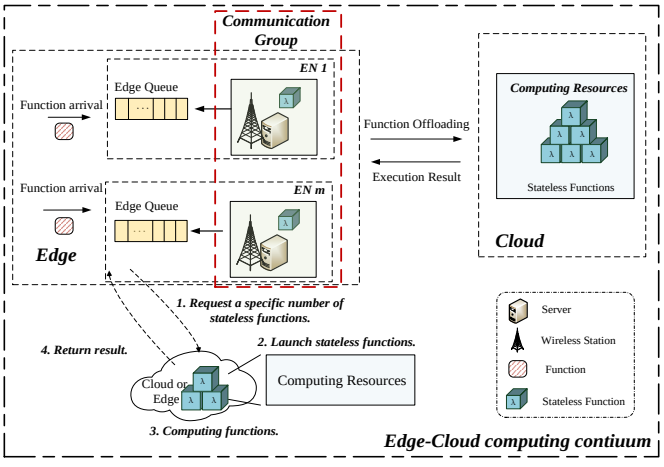

Figure 1: The overview of the function offloading in Edge-Cloud continuum.

of ENs, it is impossible to process the entire function in a limited completion delay. Many applications require real-time feedback, i.e., VR/AR, and demand the device to provide sufficient computing power. However, the computing power of current devices is insufficient. Therefore, we consider serverless architectures where the cloud can initiate several stateless functions to handle subtasks offloaded from ENs. The AttentionFunc decision-making algorithm is deployed on each EN. First, the agent deployed on the node obtains the entire network data, including the network status, functional requirements, and the cost of performing the function. Based on this data, the algorithm finds the optimal offloading decision, given the task at hand. To adapt to the dynamicity of the environment, all EN offloading decisions interact in the communication group and eventually find the globally optimal offloading decision by integrating all information.

According to the offloading decision, the function x_m^t can be processed in the Cloud or at the EN (step 1 in Figure 1). After receiving the request, the Cloud or EN launches several stateless functions for the function according to the computation service requirements (step 2 in Figure 1). Then the stateless functions conduct the function (step 3 in Figure 1), and we can calculate the function completion latency and cost according to the function's requirement. After finishing the function, the Cloud returns the results of the function to the device (step 4 in Figure 1).

2.2 Communication Groups & Attention Overview

The *Communication Group* is a logical overlay on top of Edge-Cloud infrastructure that facilitates information sharing between ENs. To ensure that all offloading decisions are globally optimal, the *Communication Group* is introduced after each EN has made its offloading decision based on locally observable information. During the information interaction process, dynamic conditions such as serverless functions arrival rate, computing power, network, and other factors affect user states, offloading services are not necessary for everyone. Therefore, we introduce an *attention mechanism* to identify which ENs' information has positive significance for decision-making and keep these ENs in the *Communication Group*, to reduce the waste of communication resources and improve decision-making efficiency.

3 ATTENTIONFUNC MODEL

3.1 Infrastructure model

A serverless edge-cloud continuum architecture consists of a number of ENs and clouds. Without loss of generality, a function is generated on each EN at time t. A control centre is deployed on each EN, where the ENs can communicate with each other and then make a function offload decision based on the current state. the ENs are connected via a network to a cloud data centre with computing power and the function can be offloaded to the cloud data centre for execution. Moreover, we define a set of time slots $T = \{1, 2, ...t\}$ to indicate the time epochs.

Function: We defined x_m^t as the function generated at EN at time t. $x_m^t = \{z_m, k_m, T_m\}$, where z_m is the required CPU cycles, k_m is the total CPU requirements, and T_m is the maximum function completion time acceptable to function x_m^t, respectively. For simplicity, the functions in this system are minimum function units for offloading.

EN: This system includes a set of $\mathcal{M} = \{1, 2, ...m\}$ of EN with the computing capability f_m which depends on how many stateless functions are deployed on the EN. We assume that only one function x_m^t is generated on the EN at each time t. The system can make offloading decisions for the function x_m^t based on the current state after communication. Some functions are sent to the edge queue and wait to execute. Therefore, we defined $\bar{w}_m$ as the average waiting at the edge queue of EN m.

Cloud: In contrast, others are offloaded to the cloud for execution over a fibre optic network. The cloud provides extensive computing resources, while the EN has low communication latency and high bandwidth. We defined f_c as the computing capability of the cloud. The Cloud consists of one or more Cloud servers, each running multiple instances of stateless functions for processing functions. Therefore, there are no waiting time in this case.

3.2 System model

3.2.1 Latency of uploading. We consider that EN m generates latency sensitize computational function x_m^t. The function x_m^t needs to upload the relevant information to the EN before it can be executed and this part of the transmission delay needs to be taken into account in the completion delay of the function. We define r^{up} as the transfer rate of the upload, z_m as the required CPU cycles that need to be uploaded, and r^{up} as the average upload rate from devices to the EN, which can be calculated by the following formula: $\mathcal{D}_m^l(t) = \frac{z_m}{r^{up}}$.

Moreover, for the time to start the cloud or edge container for function execution d_m^{start}. The time depends on the container memory, but not the input required CPU cycles and varies for a cold start or warm start. Therefore, we define $D_m^{start}(t)$ as the start latency for function x_m^t which can be computed as: $\mathcal{D}_m^{start}(t) = \mathcal{D}_m^l(t) + d_m^{start}$.

3.2.2 Latency of functions executed at EN. For the EN execution approach, functions are briefly stored in a queue and processed in First In First Out (FIFO) order. The functions' execution latency is $d_m^t = \frac{k_m}{f_m}$, where k_m is the total CPU requirement of the function x_m^t, and f_m is the service rate of the Stateless functions of the EN m. In addition, we define $\bar{w}_m = w_m / f_m$ as the average wait latency in the queue of EN m during time episode T, where w_m is the total CPU requirement of the function queue at EN m. Therefore, the process latency EN can be expressed as: $\mathcal{D}_m^l(t) = d_m^t + \bar{w}_m$.

3.2.3 Latency of functions executed at Cloud. In the cloud approach, the device uploads the input data to a cloud service. This upload triggers the execution of the stateless function that performs the data processing function.

For the offload execution case, the functions' execution latency is $\mathcal{D}_m^{ex}(t) = k_m / f_c$, where f_c is the service rate of the Cloud.For the transmission, the data transmission rate $r_m^{tr,t}$ from the EN in the wireless group based on the transmission power consumption p_n^{tr}, it's calculated as follows:

$$r_m^{tr,t} = b^{tr} log_2(1 + \frac{p_m^{tr} \cdot h_m^{tr}}{\sigma^2 + I^{tr}}) \quad (1)$$

where b^{tr} is the group bandwidth, h_m^{tr} is the group gain, σ^2 is the noise variance, and I^{tr} is the Signal Interference plus Noise Ratio (SINR). Therefore, function x_m^t's completion latency is $\mathcal{D}_m^{tr}(t) = w_m / r_m^{tr,t}$, where w_m is the required CPU cycles of function x_m^t.

For the execution result return, the communication latency D_m^{re} can calculate as $D_m^{re}(t) = c_m / r_n^{re,t}$, where c_m means the required CPU cycles of the execution result. The communication rate from the Cloud and EN m at time t can be calculated as $r_m^{re,t} = \omega^{re} log_2(1+ p_m^{re} \cdot h_m^{re} / (\sigma^2 + I^{re}))$, where $\omega^{re}, p_m^{re}, h_m^{re}$, and I^{re} are the group bandwidth, transmission power consumption, group gain, the Signal Interference plus Noise Ratio (SINR) during the result return, respectively. In addition, the network is dynamic, we consider an idle latency $D_m^{idle}(t)$ during the function x_m^t execution caused by the unstable network at time t.

To sum up, the function completion latency D_m for the function x_m^t can be expressed as

$$D_m = \sum_{t=1}^{T} \mathcal{D}_m^{ex}(t) + \mathcal{D}_{x_m^t}^{tr}(t) + D_m^{re}(t) + D_m^{idle}(t) + \mathcal{D}_m^{start}(t) \quad (2)$$

3.2.4 Cost of functions executed at Cloud. Because the stateless functions can help to operate the function, a shorter function completion time can be obtained. However, the commercial serverless architecture (i.e., AWS and Google) generally utilizes serverless architectures, where prices are determined based on the amount of computing resource requirement. In addition to being more cost-effective for certain types of services, this setup simplifies the deployment process and provides better scalability [4]. That is if the unit cost to use one computation resource is ρ and there are k_m computation resources are requested by the function x_m^t, the computing cost is given by $\rho \cdot k_m$.

In conclusion, there is a trade-off P_m between the function completion time and the computing cost: $P_m = \omega_1 \rho \cdot k_m + \omega_2 D_m$, where $0 < \omega_1 < 1$ and $0 < \omega_2 < 1$ are set as the weights of probability. To balance this trade-off, the EN decides whether to request stateless functions or not.

3.3 AttentionFunc Function Offloading Model

Next, we define the function offloading model and express it as Markov Decision Process problem. In the Markov Decision Process problem, the agent conducts successively a specific action to minimize the cost under the defined constraints. That is, we consider each EN to ack as an agent, which decides whether to offload functions or not at time t in order **to minimize the computing cost and the function completion latency**.

At the beginning of each time slot, the agent observes environment state $\mathcal{S}$. Then the agent chooses an action $\mathcal{A}$ for the function when there is a newly arrived function to be processed. The observed state and the selected action will result in a reward $\mathcal{R}$ for the function x_m^t. Next, the three elements of the MDP problem will describe as follow.

State space The state space $\mathcal{S}$ is defined as $\mathcal{S} = \{x_m^t, w_m, \mu_m, r_m^{re,t}, r_m^{tr,t}, f_m, f_c\}$, where $x_m^t = \{T_m, k_m, z_m\}$.

Action space Each Agent (EN) can make the offload decision for each function. Thus, we can define the action space $\mathcal{A} = \{a_t \in \{0,1\}, t \in T\}$. $a_t = 1$ means that execute the function x_m^t at EN, and vice versa.

Reward space To minimize the computing cost, the cost function $\mathcal{R}(\mathcal{S}, \mathcal{A})$ is defined. The computing cost is proportional to the computation resource requirement and the completion latency of function x_m^t. Hence, the reward can be defined as

$$\mathcal{R}(\mathcal{S}, \mathcal{A}) = a_t \cdot \mathcal{D}_m^l(t) + |1 - a_t| \cdot P_m \quad (3)$$

3.4 Optimization formulation

The average computing cost C_m for the function x_m^t can be defined as

$$C_m = \lim_{t \to +\infty} \frac{1}{t} \sum_{0}^{t} \mathbb{E}[\mathcal{R}(s_t, a_t)] \quad (4)$$

where $\mathbb{E}[\cdot]$ denotes the expectation of a random variable, s_t and a_t denote the state and action at time t, respectively. Therefore, the optimal objective of this problem can be express as

$$\text{argmin}_{\pi^*(s_t, a_t)} C_m \quad (5a)$$

$$\textbf{s.t.} C1 : D_m \leq T_m, \forall m \in \mathcal{M} \quad (5b)$$

$$C2 : k_m \leq m, \forall m \in \mathcal{M}, \forall t \in T \quad (5c)$$

$$C3 : a_t \in \{0,1\}, a_t \in \mathcal{A} \quad (5d)$$

where $\pi^*(s_t, a_t)$ is the optimal stochastic policy that implies the probabilities of choosing a specific action at each state.

4 AM-DRL FUNCTION OFFLOADING ALGORITHM

Each EN has an attention mechanism, communication group, and actor-critic network. First, we consider the partially observable distributed environments where each EN agent m (the agent of initiator m) receives local state $\mathbf{s}_m^t$ at each time slot $t, t \in \mathcal{T}$, the ActorNet network takes local states as input and extracts a hidden layer as *thought* which encodes local states and action intentions, represented as $\mathcal{H}_m^t = \mu 1$. The attention mechanism decides whether an EN must collaborate in each episode $\mathcal{T}$ based on *thought*. If needed, it initiates a communication group and chooses collaborators.

The attention mechanism determines and maintains the communication group within episode $\mathcal{T}$. The communication group communicates with other agents in initiator m's communication group, inputs their ideas (local states and action intentions), and outputs the integrated thoughts to lead the agent's next action intention. By exchanging local observational knowledge and encoding action intent in a dynamic group, agents may associate global environmental forecasts, assist other agents, and collaborate to make action intention judgments.

Lastly, Deep Neural Networks generate offloading actions (DNN). It generates estimated $Q(\mathbf{s}_t, \mathbf{a}_t)$ from the historical transition. The agent may observe the state $\mathbf{s}_t$ from the environment, pick an action $\mathbf{a}t$ according to the policy $\pi(\mathbf{s}_t, \mathbf{a}_t)$, and produce an instantaneous reward $\mathbf{r}_t$.

4.1 Attention mechanism

Our attention mechanism cannot observe global information, but it can encode observable areas and action intentions to assess whether

Algorithm 1: Attention Multi-agent Deep Reinforcement Learning offloading method

Input: θ^{μ_1}, θ^{μ_2}, θ^{μ}, θ^{c}, θ^{Q}, θ^{g} and θ^{p} for attention mechanism, γ for the target network, α and β for PER method.

Output: Function offloading decision

Initialize the queue $\mathcal{U}$ and the replay buffer $\mathcal{D}$
for $episode = 1, \cdots, episode_{max}$ **do**
  Initialize system environment and obtain state s_t
  Choose action $a_t \sim \mu_\theta(a_t|s_t)$ for each Agent
  Get thought $\mathcal{H}_m^t = \mu_1(\mathbf{s}_m^t; \theta^{\mu_1})$for each initiator m
  Each agent m decides whether to initiate communication based on $\mathcal{H}_m^t$ every $\mathcal{T}$ time slot
  Integrate the thought in communication group i
  for $m = 1, \cdots, M$ **do**
    Obtain reward $\mathbf{r}_t^m$ based on $\mathbf{ai}_m^t$, and get new observation $\mathbf{s}_{t+1}$
    Get action for $\bar{\mathbf{ai}}_m^t = \mu_2(\mathcal{H}_m^t; \theta^{\mu_2})$ in initiator m's communication group C
    Store $(\mathcal{H}_m^t, \Delta Q_m^t)$ in $\mathcal{U}$
  end
  Compute $\Delta\hat{Q}$ by (6)
  Update the attention parameter θ^p by (7)
  Store transition $(a_t, s_t, r_t, s_{t+1}, C)$ in $\mathcal{D}$;
  Set $y = r + \gamma Q^{\mu'}(s_{t+1}, a_{t+1})|_{a'_j=\mu_j(s_j)}$
  for *each gradient step* **do**
    Update Actor, Critic and target network by using (8)-(13)
  end
end

agents need to offload services. In particular, the attention mechanism is an RNN network that takes the hidden state of the previous step and the agent's local observations and actions as input, and the output can be seen as a classifier to evaluate whether it is incorporated into subsequent collaboration. Because collaboration policies take time to work, there is no need to take care at every time step to determine whether users participate in collaboration. During the experiment, we updated the communication group at a specific time. Algorithm 1 shows the technique pseudo-code.

4.2 Communication group

As the user's function is dynamic, maintaining active contributors in the communication group reduces computation dimension and changes system design. Hence, a communication group integrates all agents' input to define the communication group. LSTM unit θ^g is the communication group. Each agent partially observes the environment, and the communication group can let agents share knowledge for better function offloading decisions.

A user who joins numerous initiators' communication groups acts as a bridge for information sharing. Assume initiators a and b successively pick user k. The communication group integrates their thoughts: $\{\tilde{\mathcal{H}}_a^t, ...\tilde{\mathcal{H}}_{k'}^t\} = g(\mathcal{H}_a^t, ...\mathcal{H}_k^t)$. User k connects with b's

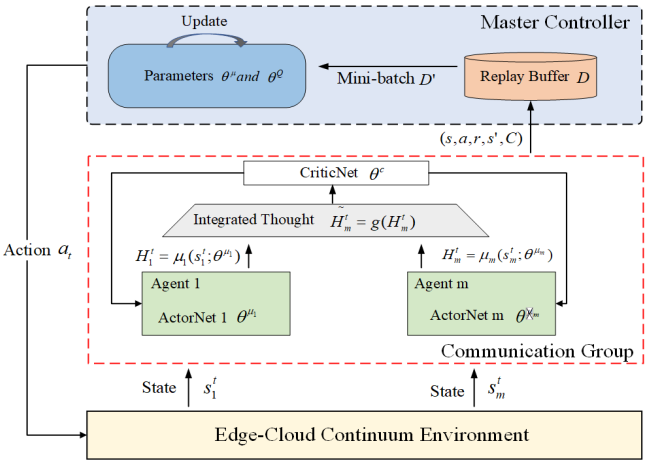

Figure 2: AM-DRL framework for the function offloading.

group: $\{\tilde{\mathcal{H}}_b^t, ...\tilde{\mathcal{H}}_{k''}^t\} = g(\mathcal{H}_b^t, ...\mathcal{H}_{k'}^t)$. User k helps groups cooperate by spreading the notion. θ^p parameters the attention mechanism for each initiator m and communication group C. One critic network θ^c with an attention mechanism for each initiator m is trained. The integrated notion is $\tilde{\mathcal{H}}_m^t$, which is merged and supplied to the next attention mechanism policy network. The attention mechanism policy network produces the action intention $\mathbf{ai}_m^t = \mu 2(\mathcal{H}_m^t, \tilde{\mathcal{H}}_m^t; \theta^{\mu_2})$. We average the Q-value difference between independent acts intention $\bar{\mathbf{ai}}_i$ and cooperative actions:

$$\Delta Q_m^t = \frac{1}{|C|}\left(\sum_{i\in C} Q(\mathbf{s}_i, \mathbf{ai}_i|\theta^c) - \sum_{i\in C} Q(\mathbf{s}_i, \bar{\mathbf{ai}}_i|\theta^c)\right) \tag{6}$$

We store $(\Delta Q_m^t, \mathcal{H}_m^t)$ into a queue $\mathcal{U}$, and min-max normalize ΔQ. The binary classification tag is $\Delta\hat{Q} \in [0, 1]$. Update θ^p in the loss function:

$$\mathcal{L}(\theta^p) = -\Delta\hat{Q}_m^t log(p(\mathcal{H}_m|\theta^p)) - (1-\Delta\hat{Q}_m^t) log(1-p(\mathcal{H}_m|\theta^p) \tag{7}$$

4.3 The training process

For actor-critic network strategy enhancement, we use the Actor network $\mu(\theta)$ to produce the next state action $\mu(\mathbf{a}_{t+1}|\theta^\mu)$ instead of picking the action with the biggest Q-value in the action space. The greedy technique cannot discover the global maximum Q-value in such a multi-dimensional action space since it must find it in every step. We also use the Critic network $Q(\theta)$ to critique the policy based on the predicted Q-value $Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q) \approx Q(\mathbf{s}_t, \mathbf{a}_t)$ via a manner similar to supervised learning:

$$\nabla_\theta J(\mu) = E_{\mathbf{s}_t, \mathbf{a}_t\sim\mathcal{D}}[\nabla_\theta \mu(\mathbf{a}_t|\mathbf{s}_t) \nabla Q^\mu(\mathbf{s}_t, \mathbf{a}_t)|_{\mathbf{a}=\mu(\mathbf{s}_t)}] \tag{8}$$

where $\mathcal{D}$ is the experience replay memory containing $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, C\}$, and stores the transitions of all agents. The action value function can be updated as follow:

$$L(\theta) = E_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}, C}[(Q^\mu(\mathbf{s}_t, \mathbf{a}_t) - y)]^2 \tag{9}$$

where $y = r + \gamma Q^{\mu'}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|_{\mathbf{a}_{t+1}=\mu(\mathbf{s}_t)}$. The Critic network's evaluation of the Actor's forward transfer action $\mathbf{a}_t$ can update the Actor network $\mu(\theta)$. Hence, the Actor network is updated using

the policy gradient iterative formula:

$$\nabla_{\theta^\mu}\mu = \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t|\theta^Q)_{\mathbf{s}_t=\mathbf{s}_t, \mathbf{a}_t=\mu(\mathbf{s}_t)} \tag{10}$$

We present two target networks $\theta^{\mu'}$ and $\theta^{Q'}$ for Actor and Critic networks to restrict target value change pace and increase learning stability. Because the Critic network $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ calculates the goal value $\mathbf{r}_t + \gamma$, $Q(\mathbf{s}_{t+1}, \mu(\mathbf{s}_{t+1})|\theta^\mu))|\theta^Q)$, Q-value update shocks. The target network changes the weight by slowly following the online network, $\theta' \leftarrow \tau + (1-\tau)\theta', \tau \ll 1$.

The Temporal-Difference Learning (TD-error) ψ, which approximates Agent's ability to learn from current experience, is stored as Q-value. It may be stated as follows:

$$\psi = \mathbf{r}_t + \gamma[\max_{\mathbf{a}_{t+1}} Q^{\mu'}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q^{\mu}(\mathbf{s}_t, \mathbf{a}_t)]|_{\mathbf{a}_{t+1}=\mu(\mathbf{s}_t)} \tag{11}$$

Finally, apply the following update to the Actor and Critic networks, where α^Q and α^μ are the network learning rates:

$$\theta^Q \leftarrow \theta^Q + \alpha^Q \cdot \nabla_{\theta^\mu} L^Q(\theta^Q), \theta^\mu \leftarrow \theta^\mu + \alpha^\mu \cdot \theta^\mu \tag{12}$$

The target Critic network and target Actor network are updated according to the following way to step-by-step track the online Critic network and Actor network:

$$\theta^{Q\prime} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \theta^{\mu\prime} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{13}$$

where $\tau \ll 1$ is the temperature parameter.

5 EVALUATION

This section first describes the simulation settings. Then, we compare it with other existing function offloading algorithms from multiple dimensions.

5.1 Experiments Setup

A comparison of several methods is conducted using Pytorch 1.3 framework. Let's assume an Edge-Cloud continuum function offloading scenario with 150 ENs. We assume that ENs are randomly distributed over a $350m \times 350m$ physical region throughout the experiments. According to [17], the computation capabilities of each EN f_m are different, evenly distributed between 0.5 and 3.5 GHz, and the computation capabilities of cloud f_c range between 31.5 and 51.5 GHz.

In our experiments, we compare AttentionFunc's AM-DRL with the following four state-of-the-art function offloading methods.

- Random: Each function is randomly assigned to the EN queue and cloud.
- Greedy: Each function is greedily offloaded to EN and cloud based on the estimated time.
- MADDPG [9]: A function offloading based on multi-agent deep deterministic policy gradient is commonly used in recent research for offloading.
- ES-DRL [19]: A distribution function offloading method based on experience-shard deep reinforcement learning.

5.2 Performance analysis of AttentionFunc approach

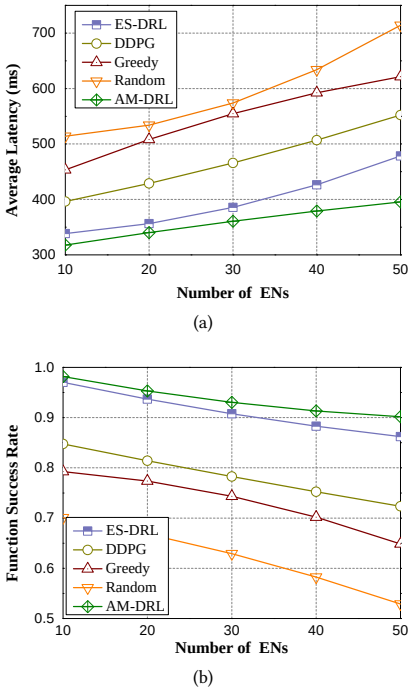

Figure 3: The number of ENs with respect to: (a) Average Latency; (b) Function Success Rate.

5.2.1 Performance analysis based on Different Numbers of ENs. Fig. 3(a) illustrates the performance of average latency for Random, Greedy, DDPG, ES-DRL, and the proposed algorithm, AM-DRL, with different numbers of ENs. We take the average value after all experiments are executed more than 10 times. As the amount of ENs grows, the average function completion latency of each algorithm increases. This is because as the number of ENs increases, generated functions within the Edgo-Cloud continuum system increase accordingly.

Moreover, we look at how the number of ENs affects the function success rate of these computation offloading algorithms. In Fig. 3(b), as the amount of ENs grows, the normalized function success rate of each algorithm decreases. The function success rate of Random and Greedy tends to decrease linearly as the number of ENs increases. Specifically, the function success rate of DDPG, ES-DRL, and our algorithm shows a slow downward trend. It can be draw that our proposed method based on computation offloading scheme is near-optimal and reduce the function success rate of the whole system by 12.7% and 0.9% compared with DDPG and ES-DRL, respectively.

5.2.2 Performance analysis based on Different Deadline of functions. Fig. 4(a) shows the function completion latency for different deadline of different strategies. For the purpose of comparing the effects of different deadline, the deadlines were set from 1.0 to 2.2 seconds. As the deadline increases, the function completion latency increases for the five offloading strategies. It is noteworthy that our proposed offloading scheme is optimal no matter what deadline is

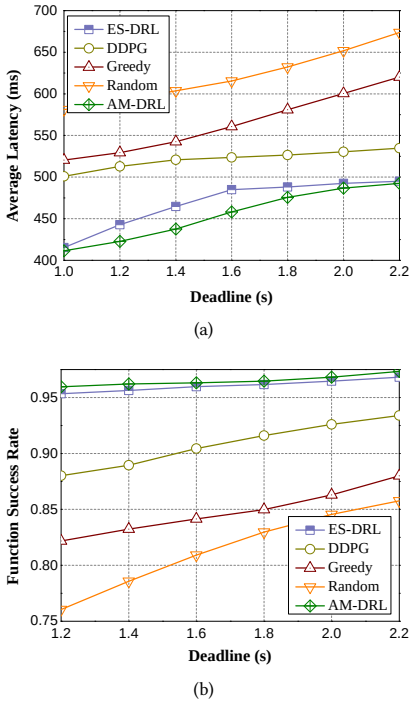

Figure 4: The deadline with respect to: (a) Average Latency; (b) Function Success Rate.

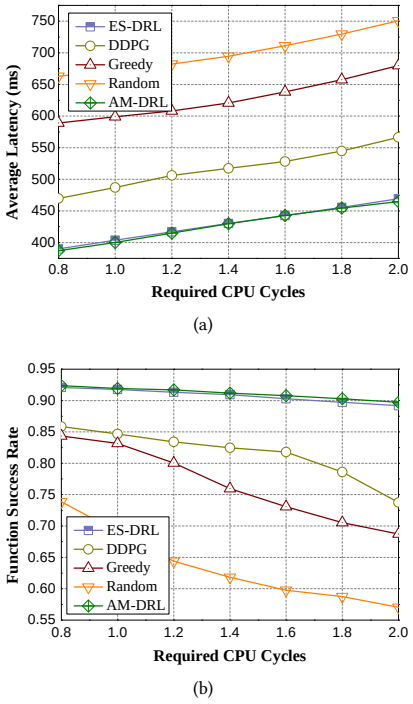

Figure 5: The required CPU cycles with respect to: (a) Average Latency; (b) Function Success Rate.

set. This is because as the deadline increases, more functions are generated in the system.

In addition, Fig. 4(b) shows the functions' success rate with the increase of functions' deadline, the success rate of Random and Greedy increases sharply due to the increase of functions' deadline, while the success rate of DDPG, ES-DRL and AM-DRL is relatively stable. This is because with the decrease of the deadline, local and edge gradually lack the ability to deal with functions, and the proportion of failed functions increases. It can be seen from Fig. 4(b), the proposed algorithm keeps a good profit when the functions' deadline is small, which shows that the success rate of the function is not affected by the deadline. Therefore, it can be inferred that the proposed algorithm can deal with computing functions with latency-sensitive, which is more suitable for practical use.

5.2.3 Computational overhead and trade-offs. Fig. 5(a) illustrates the average latency impact of five algorithms with different required CPU cycles. When the required CPU cycles increase, the increase in average latency can be verified. This is because the requirements for computing power are not strict for light loads, while heavy loads require relatively large computing power.

In addition, Fig. 5(b) shows that as the CPU cycle requirements for the function increases, the success rate of all five methods decreases. This is because as the CPU cycle required for the function increases, the resources available for local and edge allocation gradually decrease, and the completion time of the function exceeds the deadline, resulting in a gradual decrease in the success rate. From the figure, it can also be seen that when the CPU cycle required by the function is large, the proposed algorithm maintains a good success rate, indicating that the success rate of the function is less affected by the CPU cycle required by the function.

6 RELATED WORK

As latency-sensitive applications motivate the migration of computation functions from the cloud to edge locations, interest in transferring FaaS solutions to the edge is also increasing. One possible option is reusing open source FaaS options, such as Apache OpenWhisk [1], Knative [2] or Kubeless [3]. Other works focus more on improving some aspects of FaaS to adapt existing solutions to edge-specific issues, such as [13] which builds an extension over OpenWhisk. Here authors identify problematic areas in serverless edge scenarios based on diverse use cases. They touch on issues related to function offloading based on type (i.e., latency-sensitiveness, computation or data-intensiveness), collaboration among edge nodes, data or state locality, and handling.

Some studies have achieved performance optimisation through function offloading in Edge-Cloud continuum scenarios, for example, an algorithm for dynamically offloading serverless functions is proposed in [7], a work that focuses on efficient partitioning of serverless applications and a combination of functions to improve performance. A performance optimisation framework is proposed in [5] for serverless applications on edge cloud platforms, using regression techniques to make predictions and dynamically decide whether to process serverless functions on the edge or on the cloud. However, the drawbacks of this supervised learning approach are

the lack of large amounts of labelled data and long decision times. Therefore, some research introduces an unsupervised learning-based DRL algorithm to develop function offloading policies in the Edge-Cloud continuum scenario without labelled data and shorter decision times. For instance, Yao *et. al* [19] proposed a function offloading algorithm DRLFO with a deep reinforcement learning algorithm based on the actor-critic framework.

Existing research has considered fully-centralized control of resource provisioning and allocation. However, such solutions require significant communication and coordination overhead in a distributed Edge-Cloud continuum scenario [18]. Instead, we consider practical approaches that require minimum coordination. Our starting point is the efforts related to distributed function dispatching. For instance, Gabriele *et. al* [15] presents for the first time a distributed architecture Serverledge to support collaborative scheduling of edge nodes with each other, providing the basis for a distributed and collaborative computational offloading approach.

The above research has two drawbacks. One is that some studies do not consider competition or cooperation in offloading multiple functions in the Edge-Cloud continuum environment. Secondly, these efforts do not make full use of the distributed characteristics of edge computing and ignore the communication delay caused by agent information interaction. In contrast, we study the function offloading problem in the Edge-Cloud continuum environment, in which finer-grained functions are used as the basic unit of function offloading. A common feature of the above related work is performance optimization to reduce latency and cost. We propose a distributed function offloading method based on attention multi-agent reinforcement learning, which allows EN to interact with the most valuable information and helps to generate a real-time and globally optimal function offloading strategy.

7 CONCLUSION

In this paper, we study the function offloading problem in the multiple EN dynamic Edge-Cloud continuum environment. Our objective is to minimize the average system latency and cost. To achieve the optimal problem, we propose an attention multi-agent deep reinforcement learning algorithm, which allows ENs to transmit valuable information among their communication group thus obtaining the global optimal offloading strategy. The proposed method reduces communication overhead in a dynamic edge cloud environment, promotes collaborative decision-making, and improves the effectiveness and rationality of collaboration. Numerical results show that the proposed AM-DRL method is effective in offloading function edge computation and is superior to the baseline algorithm.

As future work, we plan to extend AttentionFunc to also support decentralized and efficient function offloading and computation balancing for serverless applications, based on function composition and function chaining. We also intend to extend our approach to account for the mobility challenges that are typically encountered when performing computation balancing in real-world Edge-Cloud scenarios.

REFERENCES

[1] [n. d.]. The Apache Software Foundation, Jun. 2016, [online] Available: https://openwhisk.apache.org.

[2] [n. d.]. Knative, Jun. 2020, [online] Available: https://knative.dev.

[3] [n. d.]. Kubeless, Jun. 2020, [online] Available: https://kubeless.io/.

[4] Yoni Birman, Shaked Hindi, Gilad Katz, and Asaf Shabtai. 2020. Cost-Effective Malware Detection as a Service Over Serverless Cloud Using Deep Reinforcement Learning. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 420–429. https://doi.org/10.1109/CCGrid49817.2020.00-51

[5] Anirban Das, Shigeru Imai, Stacy Patterson, and Mike P. Wittie. 2020. Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 41–50. https://doi.org/10.1109/CCGrid49817.2020.00-89

[6] Shuiguang Deng, Hailiang Zhao, Zhengzhe Xiang, Cheng Zhang, Rong Jiang, Ying Li, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. 2022. Dependent Function Embedding for Distributed Serverless Edge Computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 10 (2022), 2346–2357. https://doi.org/10.1109/TPDS.2021.3137380

[7] Tarek Elgamal, Atul Sandur, Klara Nahrstedt, and Gul Agha. 2018. Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 300–312. https://doi.org/10.1109/SEC.2018.00029

[8] Phani Kishore Gadepalli, Sean McBride, Gregor Peach, Ludmila Cherkasova, and Gabriel Parmer. 2020. Sledge: A Serverless-First, Light-Weight Wasm Runtime for the Edge. In *Proceedings of the 21st International Middleware Conference* (Delft, Netherlands) *(Middleware '20)*. Association for Computing Machinery, New York, NY, USA, 265–279. https://doi.org/10.1145/3423211.3425680

[9] Wenjing Hou, Hong Wen, Huanhuan Song, Wenxin Lei, and Wei Zhang. 2021. Multiagent Deep Reinforcement Learning for Task Offloading and Resource Allocation in Cybertwin-Based Networks. *IEEE Internet of Things Journal* 8, 22 (2021), 16256–16268. https://doi.org/10.1109/JIOT.2021.3095677

[10] Yuepeng Li, Deze Zeng, Lin Gu, Kun Wang, and Song Guo. 2022. On the Joint Optimization of Function Assignment and Communication Scheduling toward Performance Efficient Serverless Edge Computing. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–9. https://doi.org/10.1109/IWQoS54832.2022.9812887

[11] Stefan Nastic and Schahram Dustdar. 2018. Towards Deviceless Edge Computing: Challenges, Design Aspects, and Models for Serverless Paradigm at the Edge. In *The Essence of Software Engineering*. Springer, Cham, 121–136. https://doi.org/10.1007/978-3-319-73897-0{_}8

[12] Stefan Nastic, Schahram Dustdar, Raith Philipp, Furutanpey Alireza, and Thomas Pusztai. 2022. A Serverless Computing Fabric for Edge & Cloud. In *4th IEEE International Conference on Cognitive Machine Intelligence (CogMi)*. https://doi.org/10.1109/CogMI56440.2022.00011

[13] István Pelle, János Czentye, János Dóka, András Kern, Balázs P. Gerő, and Balázs Sonkoly. 2021. Operating Latency Sensitive Applications on Public Serverless Edge Cloud Platforms. *IEEE Internet of Things Journal* 8, 10 (2021), 7954–7972. https://doi.org/10.1109/JIOT.2020.3042428

[14] Philipp Raith, Stefan Nastic, and Schahram Dustdar. 2023. Serverless Edge Computing—Where We Are and What Lies Ahead. *IEEE Internet Computing* 27, 3 (2023), 50–64. https://doi.org/10.1109/MIC.2023.3260939

[15] Gabriele Russo Russo, Tiziana Mannucci, Valeria Cardellini, and Francesco Lo Presti. 2023. Serverledge: Decentralized Function-as-a-Service for the Edge-Cloud Continuum. In *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 131–140. https://doi.org/10.1109/PERCOM56429.2023.10099372

[16] Simon Shillaker and Peter R. Pietzuch. 2020. Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. *CoRR* abs/2002.09344 (2020). arXiv:2002.09344 https://arxiv.org/abs/2002.09344

[17] Jie Tang, Hengbin Tang, Nan Zhao, Kanapathippillai Cumanan, Shunqing Zhang, and Yongjin Zhou. 2019. A Reinforcement Learning Approach for D2D-Assisted Cache-Enabled HetNets. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9014027

[18] Qinqin Tang, Renchao Xie, Fei Richard Yu, Tianjiao Chen, Ran Zhang, Tao Huang, and Yunjie Liu. 2022. Distributed Task Scheduling in Serverless Edge Computing Networks for the Internet of Things: A Learning Approach. *IEEE Internet of Things Journal* 9, 20 (2022), 19634–19648. https://doi.org/10.1109/JIOT.2022.3167417

[19] Xuyi Yao, Ningjiang Chen, and Xuemei Yuan. 2022. Performance optimization of serverless edge computing function offloading based on deep reinforcement learning. *Future Generation Computer Systems* 139 (09 2022), 74–86. https://doi.org/10.1016/j.future.2022.09.009