Preference-Aware Fault-Tolerant Function Embedding in Energy-Harvesting Serverless Edge Computing

Kun Cao, *Member, IEEE*, Chaohong Tan, Yangguang Cui, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Serverless edge computing (SEC) that integrates serverless and edge computing paradigms has facilitated the deployment of intelligent Internet-of-Things (IoT) applications. In SEC systems, energy efficiency and serverless pricing are essential to maintain operational sustainability. Nevertheless, most existing energy-saving techniques focus only on stable energy scenarios and are therefore inapplicable to energy-harvesting SEC systems powered by intermittent renewable sources. On the other hand, serverless pricing policies generally neglect the personalized perceptions of user quality-of-experience (QoE) preferences, thereby resulting in holistic user QoE degradation from a system perspective. Moreover, these approaches cannot guarantee functional correctness of serverless applications due to the appearance of computation and communication errors in practical SEC systems. To tackle these challenges, we investigate the preference-aware fault-tolerant function embedding problem for enhancing the holistic user QoE in energy-harvesting SEC systems. We first design a personalized QoE preference predictor to characterize trade-offs between service completion time and resultant service fees of individual users. Subsequently, we develop a reinforcement learning method to decide static function embedding decisions at the offline phase. Considering the intermittency of renewable sources, we further provide an energy-adaptive function replica freezing strategy at the online phase. Evaluations demonstrate that our approach boosts the holistic user QoE by 32.2% over state-of-the-art algorithms.**

***Index Terms*—Serverless edge computing, energy-harvesting, fault-tolerance, function embedding, serverless pricing.**

Received 30 August 2025; revised 28 January 2026; accepted 23 February 2026. Date of publication 3 March 2026; date of current version 10 April 2026. The work of Kun Cao was supported in part by the National Natural Science Foundation of China under Grant 62102164, in part by the Open Research Fund of National Mobile Communications Research Laboratory (Southeast University) under Grant 2025D12, in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2024A1515010232, and in part by the Open Project Program of Guangxi Key Laboratory of Digital Infrastructure under Grant GXDIOP2024002. The work of Yangguang Cui was supported in part by the National Natural Science Foundation of China under Grant 62502298 and in part by the Natural Science Foundation of Shanghai under Grant 24ZR1421500. *(Corresponding authors: Chaohong Tan; Yangguang Cui.)*

Kun Cao is with the College of Cyber Security, Jinan University, Guangzhou 510632, China, and with the Guangxi Key Laboratory of Digital Infrastructure, Guangxi Zhuang Autonomous Region Information Center, Nanning 530000, China, and also with National Mobile Communication Research Laboratory, Southeast University, Nanjing 211189, China (e-mail: kuncao@jnu.edu.cn).

Chaohong Tan is with the Guangxi Key Laboratory of Digital Infrastructure, Guangxi Zhuang Autonomous Region Information Center, Nanning 530000, China (e-mail: tanch@gxi.gov.cn).

Yangguang Cui is with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China (e-mail: ygcui@shu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSC.2026.3670012

I. Introduction

OVER the past decade, edge computing [1], [2] has proliferated numerous latency-sensitive Internet-of-things (IoT) applications, such as smart healthcare, autonomous driving, and intelligent industrial automation. Recent advances in the edge computing community energetically advocate the integration of serverless paradigm (also known as function-as-a-service, FaaS) into edge infrastructures. This alliance thus gives rise to a new concept of serverless edge computing (SEC) [3]. It is envisioned that SEC will not only inherit the fast response advantages of edge computing, but also enjoy FaaS superiorities in seamless scalability, fine-grained resource management, and pay-per-use service billing.

One distinct characteristic of SEC systems is that an IoT application is generally organized as a service function chain (SFC) comprising a sequence of dependent functions, or as a directed acyclic graph (DAG) in which the nodes denote individual dependent functions [3]. Accordingly, a fundamental challenge in SEC systems is the *function embedding*, that is, determining the placement of serverless functions onto distributed edge servers. In this regard, several works [4], [5], [6], [7], [8], [9] have explored energy-efficient function embedding for SEC systems. However, most techniques [4], [5], [6], [7] are customized for SEC systems with stable energy sources. As a promising direction, the adoption of energy-harvesting technologies is attracting increasing attention in SEC studies [8], [9]. Nevertheless, the inherent uncertainty in harvested energy supply of edge servers incurs an increased risk of function interruption. Therefore, traditional function embedding strategies designed for stable energy supply environments are often ineffective or even infeasible in energy-harvesting SEC contexts.

The second characteristic of SEC systems lies in the diversity of user-centric quality-of-experience (QoE) preferences. In the service mode of SEC systems, individual users submit their requests to edge servers and then pay for service fees incurred upon service completion. Most of existing pricing models [10], [11], [12], [13], [14], [15] are commonly inspired by AWS Lambda and Lambda@Edge pricing schemes, where the total user cost is jointly determined by: (i) the number of function invocations, (ii) the overall memory size allocated for all functions, and (iii) the execution duration of the function. However, such native pricing models overlook the personalized perceptions of user QoE preferences in service completion time and resultant service fees. Due to the diverseness of serverless deployment purposes, the QoE of individual users could vary significantly even when provided with identical SEC resource configurations. Therefore,

personalized pricing is urgently needed to accommodate individual QoE preferences.

More importantly, the third characteristic that distinguishes SEC from conventional cloud platforms is its vulnerability to both communication and computation failures. Yet most of existing works [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] cannot guarantee dependable function accomplishment due to a lack of fault-tolerant mechanisms. In practice, SEC systems are suffering from two dominant error types [16], [17], [18]: (i) bit errors during inter-function communication, and (ii) soft errors during function execution on edge servers. Specifically, bit errors are primarily induced by channel noise, electromagnetic interference, or synchronization anomalies across communication links. Soft errors typically result from the transient faults due to ever-increasing processor integration density and shrinking transistor feature sizes in addition to harsh operating environments. Both bit errors and soft errors pose significant threats to function reliability that is defined as the probability of successful function completion. Unfortunately, few works have focused on the design of fault-tolerant function embedding solutions for SEC systems.

In summary, the majority of existing works have focused on the energy optimization and serverless pricing for conventional SEC systems with stable energy supply. However, the critical challenges of personalized pricing and function fault-tolerance remain largely unexplored, especially for energy-harvesting SEC systems characterized by intermittent power supply. On one hand, the interplay between user-centric QoE requirements and the stochastic nature of renewable energy is likely to result in a holistic user QoE degradation from a system perspective. On the other hand, the absence of fault-tolerant mechanisms could compromise the functional correctness of serverless applications in practical SEC deployments. In this paper, we conduct the first study of preference-aware fault-tolerant function embedding in energy-harvesting SEC systems. Our major contributions are as follows.

- We formulate the problem of serverless function embedding for holistic user QoE maximization subject to precedence, reliability, provider profitability, and energy constraints in energy-harvesting SEC systems.
- We incorporate a personalized QoE preference predictor into serverless pricing for accurately characterizing individual user trade-offs between service completion time and resultant service fees.
- We develop a hybrid function embedding approach composed of offline learning and online adaptation phases. At the offline phase, a reinforcement learning (RL) method is devised to conduct static function embedding. At the online phase, an energy-adaptive function replica freezing method is designed to accommodate the energy-harvesting intermittency.
- We perform extensive evaluations on a simulation platform to validate our approach. Experimental results confirm that our approach enhances the holistic user QoE by 32.2% compared with benchmarking algorithms.

The organization of this paper is as follows. Section II reviews related works. Section III describes system architecture and models. Section IV formulates our problem and outlines the proposed approach. Section V presents our personalized QoE preference predictor. Section VI and Section VII detail our function embedding policies. Finally, we exhibit evaluation results in Section VIII and conclude the paper in Section IX. For clarity, we summarize the main notations used in this paper in Table I.

TABLE I
DEFINITION OF MAIN NOTATIONS USED IN PAPER

Notation	Definition
$\mathcal{G} = (\mathcal{S}, \mathcal{L})$	An undirected connected graph for SEC systems
$\mathcal{S} = \{S_1, \cdots, S_M\}$	A collection of total M edge servers
$S_m \in \mathcal{S}$	The m-th edge server in server set $\mathcal{S}$
C_m	The computational capacity of edge server S_m
$\mathcal{L} = \{l_{m,n} \forall S_m, S_n\}$	A set of virtual communication links
$b_{m,n}$	The bandwidth for inter-server data transfer
$\mathcal{A} = \{\mathcal{A}_1, \cdots, \mathcal{A}_H\}$	A set of H serverless applications or users
$\mathcal{A}_h \in \mathcal{A}$	The hth serverless application in set $\mathcal{A}$
$\mathcal{M}_{H\times 9}$	The application attribute matrix of $\mathcal{A}$
$\Lambda_h \in [1, E]$	The index of deployment purpose for $\mathcal{A}_h$
$T^h_{\text{desire}}, R^h_{\text{goal}}$	The desirable finish time, reliability goal of $\mathcal{A}_h$
D_h, ϑ_h	The completion deadline, scheduling priority
$\mathcal{G}_h = (\mathcal{F}_h, \mathcal{E}_h)$	The DAG structure of application $\mathcal{A}_h$
$\mathcal{F}_h = \{f_{h,i} \mid i \in [1, O_h]\}$	A vertex set containing total O_h functions
$\mathcal{E}_h = \{e_{h,i,j}, d_{h,i,j} \mid i, j\}$	A set of directed edges with data stream sizes
$f_{h,i} : \{\mu_{h,i}, W_{h,i}\}$	The activity factor, instruction count of $f_{h,i}$
$\{f^b_{h,i} \mid b \in [1, b^{h,i}_{\text{replica}}]\}$	A set containing $b^{h,i}_{\text{replica}}$ replications of $f_{h,i}$
$\Phi(f^b_{h,i})$	The identifier of the server holding replica $f^b_{h,i}$
$R_{\text{com}}(\Phi(\Gamma^{h,i}_{\text{parent}}), \Phi(f^b_{h,i}))$	The communication reliability of replica $f^b_{h,i}$
$R_{\text{exe}}(\Phi(f^b_{h,i}), \Phi(\Gamma^{h,i}_{\text{parent}}))$	The execution reliability of replica $f^b_{h,i}$
$\Psi^m_{\text{supply}}, P^m_{\text{power}}$	The renewable energy supply, harvesting power
$\Psi^{\text{exe},m}_{\text{demand}}$	The computation energy demand of server S_m
$\Psi^{\text{out},m}_{\text{demand}}, \Psi^{\text{out},m}_{\text{demand}}$	The data receiving and delivering energy demands
$\Upsilon^{\max,h}_{\text{price}}$	The resultant service price for finishing $\mathcal{A}_h$
$\delta_h, \Upsilon^{\max,h}$	The price decaying factor, maximal service fee
$\alpha, \beta, \{\eta_h \mid 1 \le h \le H\}$	The common and unique preference parameters
Q, Q_h	The holistic user QoE, the QoE of $\mathcal{A}_h$
$\Psi_{\text{embed}}(f^b_{h,i}, S_m)$	The embedding energy metric for $f^b_{h,i}$ on S_m

II. RELATED WORKS

A. Energy Efficiency Optimization for SEC

Numerous approaches have focused on improving the energy efficiency of SEC systems. For example, Righetti et al. [4] leverages the mixed-integer-linear-programming to decide energy-efficient function embedding decisions. Shang et al. [5] design an online container deployment and data-flow routing algorithm. Golec et al. [6] exhibit energy-aware RL schemes for SEC resource management. Kim et al. [7] develop a stochastic game to alleviate the container cold-start occurrences under energy constraints. However, these methods are tailored for SEC systems with stable energy sources. As a result, the challenge of energy intermittency in renewable-powered SEC remains insufficiently addressed. To bridge this gap, Cao et al. [9] develop a stochastic function scheduling strategy under renewable energy supply. More recently, Aslanpour et al. [8] explore energy-minimized function-to-server embedding for energy-harvesting SEC systems.

B. Serverless Pricing in SEC

For serverless pricing, some works [10], [11], [12], [13], [14], [15] have been devoted to improving the revenue of service providers or reducing the monetary costs of terminal users. For example, Tutuncuoglu et al. [10] formulate a Stackelberg game for the co-optimization of serverless pricing and resource allocation. Hu et al. [11] study the problem of service request scheduling and price-aware container retention. Other works employ auction-based methods [12], Q-network variants [13], [14], and restricted Boltzmann machines [15] for serverless pricing. However, these pricing models in [10], [11], [12], [13], [14], [15]

are commonly inspired by AWS Lambda and Lambda@Edge pricing schemes that emphasize traditional resource-based metrics such as function invocation counts, memory allocation, and execution time. Thus, they cannot handle the personalized perceptions of user QoE preferences in service completion time and resultant service fees for different serverless deployment purposes.

C. Fault-Tolerance in SEC

Unlike energy efficiency and serverless pricing studies, the topic of designing fault-tolerant function embedding solutions for SEC systems is still an open research area. A pioneering work [16] quantifies the latency, throughput, and resource overheads of guaranteeing varied reliability goals. Sreekanti et al. [17] design a retry-based fault-tolerance method to ensure atomic function-updating visibility of serverless applications. Lately, Cao et al. [18] propose a decomposition-based function placement method to accomplish function-to-server mapping under bit errors and soft errors. However, a comprehensive solution that jointly considers user QoE preferences in serverless pricing, energy-harvesting fluctuations, and function reliability requirements has yet to be developed.

III. System Architecture and Models

A. System Architecture

We describe the architecture of SEC systems as an undirected connected graph $\mathcal{G} = (\mathcal{S}, \mathcal{L})$. On one hand, $\mathcal{S} = \{S_1, S_2, \cdots, S_M\}$ is a collection of heterogeneous edge servers that are geographically distributed at the same locations of selected base stations. In this regard, prior studies [19], [20] have comprehensively investigated the latency-optimal mapping between edge servers and base stations. On this basis, we assume that each edge server S_m is pre-associated with its optimally chosen base station. The heterogeneity among these edge servers is mainly manifested in their distinct computation capacities. We denote the m-th edge server in edge server set $\mathcal{S}$ by S_m $(1 \leq m \leq M)$ and symbolize its computational capacity by C_m. On the other hand, $\mathcal{L} = \{l_{m,n} | m, n \in [1, M]\}$ is a set of virtual communication links created to interconnect edge servers, where each link $l_{m,n}$ offers bandwidth $b_{m,n}$ for data stream transfer between edge servers S_m and S_n.

Further, we consider energy-harvesting SEC systems where individual edge servers are powered by renewable generations. As shown in Fig. 1, each edge server is constructed on three fundamental units: an energy harvesting unit, an energy storage unit, and an energy consumption unit. The energy harvesting unit gathers ambient renewable sources such as solar or wind energy. The energy storage unit, typically implemented with capacity-limited supercapacitors or rechargeable batteries, is used to mitigate the intermittency of harvested energy. That is, if there exists idle harvesting energy during one harvesting period, it will be automatically stored into the energy storage unit. Conversely, when renewable energy is insufficient, the energy storage module will discharge to maintain system operations until its electric quantity is exhausted. Accordingly, the energy consumption unit draws available power from either the harvesting unit or the storage unit, or both, to support function computation and communication.

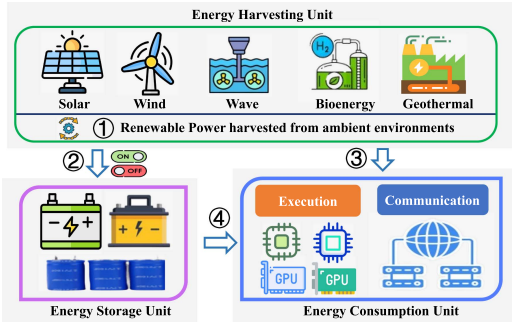

Fig. 1. Architecture of an energy-harvesting edge server. Power flow: ① harvested power, ② idle harvesting power, ③ consumed harvesting power, ④ consumed storage power.

B. Serverless Application Model

We assume that each user is uniquely bound with one serverless application, and let $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_H\}$ denote a set of total H serverless applications or distinct users. Each application or user $\mathcal{A}_h$ $(1 \leq h \leq H)$ is described by a tuple $\mathcal{A}_h = \{\Lambda_h, T^h_{\text{desire}}, D_h, R^h_{\text{goal}}, \vartheta_h, \mathcal{G}_h\}$. Specifically, $\Lambda_h \in [1, E]$ takes an integral number that specifies the application deployment purpose [3], [21]. For example, "$1''$ for video surveillance, "$2''$ for industrial data analysis, "$3''$ for machine learning, "$4''$ for smart healthcare, etc. In practice, the total number of serverless applications may scale to several thousands. For instance, Alibaba cluster traces contain 20365 DAG-structured serverless applications [22]. Whereas, since structure-dissimilar applications could have an identical deployment purpose, the number of application deployment purposes E is generally much less than H [3], [21]. T^h_{desire} is the desirable finish time. D_h is the completion deadline, R^h_{goal} is the reliability goal, and ϑ_h is the scheduling priority. $\mathcal{G}_h = (\mathcal{F}_h, \mathcal{E}_h)$ is utilized to represent the DAG structure of application $\mathcal{A}_h$. In such an organization, $\mathcal{F}_h = \{f_{h,i} | 1 \leq i \leq O_h\}$ denotes a vertex set containing O_h dependent functions sorted in topological order. $\mathcal{E}_h = \{e_{h,i,j}, d_{h,i,j} | f_{h,i}, f_{h,j} \in \mathcal{F}_h\}$ represents a set of directed edges capturing inter-function data-flow dependencies. If $f_{h,j}$ consumes the output produced by $f_{h,i}$, we call $f_{h,i}$ a direct predecessor of $f_{h,j}$, or call $f_{h,j}$ a direct successor of $f_{h,i}$. At this moment, a directed edge $e_{h,i,j}$ pointing from $f_{h,i}$ to $f_{h,j}$ and the data stream size $d_{h,i,j}$ are both added to edge set $\mathcal{E}_h$. For $f_{h,i}$, its features are summarized by a tuple $f_{h,i} : \{\mu_{h,i}, W_{h,i}\}$. $\mu_{h,i} \in (0, 1]$ refers to the activity factor. $W_{h,i}$ specifies the number of instruction cycles.

C. Fault-Tolerance Model

To achieve fault-tolerance against bit and soft errors, replication technique is adopted in this paper. Let $\Gamma_{h,i} = \{f^b_{h,i} | 1 \leq b \leq b^{h,i}_{\text{replica}}\}$ be a set containing total $b^{h,i}_{\text{replica}}$ replications of function $f_{h,i}$, and $\Phi(f^b_{h,i})$ indicate the identifier of the edge server allocated to replica $f^b_{h,i}$. To alleviate communication link congestion, we restrict each function $f^b_{h,i}$ to communicate only with a single copy of each predecessor function throughout this paper. Then, let $\Gamma^{h,i,b}_{\text{parent}} = \{f_{h,o_i}, f_{h,p_i}, f_{h,q_i}, \ldots, f_{h,z_i}\}$ represent a collection of all the direct predecessor replications that omit specific replication indexes for function $f^b_{h,i}$ for clarity. As investigated in [23], the probability that all direct predecessor

successfully deliver their outputs to function $f_{h,i}^b$ is inferred by

$$R_{\text{com}}(\Phi(\Gamma_{\text{parent}}^{h,i}), \Phi(f_{h,i}^b)) = \prod_{\kappa=o_i}^{z_i} \exp\left\{-\lambda\left(\Phi(f_{h,\kappa_i}), \Phi(f_{h,i}^b)\right) \times \frac{\gamma_{h,\kappa_i,i} \times d_{h,\kappa_i,i}}{b_{\Phi(f_{h,\kappa_i}),\Phi(f_{h,i}^b)}}\right\}, \quad \forall \Phi(f_{h,\kappa_i}) \neq \Phi(f_{h,i}^b), \quad (1)$$

where $\lambda(\Phi(f_{h,o_i})$ is the fault-arrival rate at link $l_{\Phi(f_{h,o_i}),\Phi(f_{h,i}^b)}$ on average, and $\gamma_{h,o_i,i}$ measures the vulnerability of $d_{h,o_i,i}$ to bit errors. When $\Phi(f_{h,o_i}) = \Phi(f_{h,i})$ holds, f_{h,o_i} and $f_{h,i}^b$ are dispatched to an identical edge server. In this special case, the probability $R_{\text{com}}(\Phi(f_{h,o_i}), \Phi(f_{h,i}^b))$ is deemed to be 1.

Besides, let λ_m denote the arrival rate of transient faults at edge server S_m on average. Following the fault model in [24], λ_m can be approximated by

$$\lambda_m = \chi_{m,1} \times \exp\{\chi_{m,2} \times C_m\}, \quad (2)$$

where $\chi_{m,1}$ and $\chi_{m,2}$ are constants. Since exponential distribution can model transient faults, the probability that function $f_{h,i}^b$ completes its execution on a designated server $S_{\Phi(f_{h,i}^b)}$ without experiencing a soft error is expressed as [24]

$$\begin{aligned} R_{\text{exe}}\left(\Phi(f_{h,i}^b), \Phi(\Gamma_{\text{parent}}^{h,i})\right) &= \exp\left\{-\lambda_{\Phi(f_{h,i}^b)} \times \frac{\zeta_{h,i} \times W_{h,i}}{C_{\Phi(f_{h,i}^b)}}\right\} \\ &\times R_{\text{exe}}(f_{h,o_i}, \Phi(f_{h,o_i})) \times R_{\text{exe}}(f_{h,p_i}, \Phi(f_{h,p_i})|f_{h,o_i}) \\ &\times R_{\text{exe}}(f_{h,q_i}, \Phi(f_{h,q_i})|f_{h,o_i}, f_{h,p_i}) \times \cdots \times R_{\text{exe}}\big(f_{h,z_i}, \\ &\Phi(f_{h,z_i})|f_{h,o_i}, f_{h,p_i}, f_{h,q_i}, \ldots, f_{h,y_i}\big). \end{aligned} \quad (3)$$

$\zeta_{h,i}$ quantifies the vulnerability of function $f_{h,i}$ to transient faults. The term $R_{\text{exe}}(f_{h,z_i}, \Phi(f_{h,z_i})|f_{h,o_i}, f_{h,p_i}, \ldots, f_{h,y_i})$ indicates the conditional probability of correct accomplishment of function f_{h,z_i} provided that all other direct predecessors prior to f_{h,z_i} have performed correctly.

Generally, the reliability of a function is defined as the probability of successful finish subject to bit and soft errors. Accordingly, the reliability of function $f_{h,i}$ is the probability that at least one of its replicas finishes correctly in the presence of both error sources. Thus, the reliability of function $f_{h,i}$ with replication set $\Gamma_{h,i} = \{f_{h,i}^b | 1 \leq b \leq b_{\text{replica}}^{h,i}\}$ is calculated as

$$\begin{aligned} R(\Phi(\Gamma_{h,i})) = 1 - \prod_{b=1}^{b_{\text{replica}}^{h,i}} &(1 - R_{\text{com}}\left(\Phi(\Gamma_{\text{parent}}^{h,i}), \Phi\left(f_{h,i}^b\right)\right) \times \\ &R_{\text{exe}}\left(\Phi\left(f_{h,i}^b\right), \Phi\left(\Gamma_{\text{parent}}^{h,i}\right)\right). \end{aligned} \quad (4)$$

D. Energy Harvesting Model

Let $P_{\text{power}}^m(t)$ denote the renewable power generated by the energy harvesting unit of edge server S_m at time t. Besides, let $\Psi_{\text{energy}}^{\text{buf},m}(t)$ represent the buffered energy in the storage unit at the same time instant. Accordingly, the total available energy over a given period $[t, t+T]$ can be modeled as

$$\Psi_{\text{supply}}^m(t,T) = \Psi_{\text{energy}}^{\text{buf},m}(t) + \int_t^{t+T} P_{\text{power}}^m(t)dt. \quad (5)$$

In practice, an accurate estimation of the renewable power $P_{\text{power}}^m(t)$ is inherently difficult due to environmental variability. In literatures, extensive attempts have been made to enhance the prediction accuracy of renewable power and one commonly used solar trace curve is given by [25]

$$P_{\text{power}}^m(t) = \left|\psi_m \times \text{Gau}(t) \times \cos\left(\frac{t}{219}\right) \times \cos\left(\frac{t}{314}\right)\right| \quad (6)$$

where $\psi_m \in (0,1]$ is a scaling coefficient that captures the variability in harvesting power output of individual edge servers. $\text{Gau}(t)$ is a Gaussian random process with mean zero and unit variance. In this paper, the study of improving the prediction accuracy of renewable power is not our focus, and we thus follow [9], [26], [27] to select the solar trace curve in [25] as our renewable power model. We should emphasize that our proposed technique in Section IV-B is not restricted to any specific renewable power model and can be readily replaced with other empirical energy traces in practical scenarios.

From the energy demand side, both the energy required for function computation and the energy consumed by inter-function data delivery should be considered. On the side of function execution on edge servers, the computation energy demand of edge server S_m is expressed as [8], [9]

$$\Psi_{\text{demand}}^{\text{exe},m} = \left(\sum_{h=1}^{H}\sum_{i=1}^{O_h}\sum_{b=1}^{b_{\text{replica}}^{h,i}} \xi_m \times \mu_{h,i} \times C_m^2 \times W_{h,i} \times \Delta_{\Phi(f_{h,i}^b)=m}\right) + P_{\text{power}}^{\text{con},m} \times T. \quad (7)$$

ξ_m is the effective switching capacitance and $P_{\text{power}}^{\text{con},m}$ is the static power dissipation. $\Delta_{\Phi(f_{h,i}^b)=m}$ is an indicator that equals 1 if replica $f_{h,i}^b$ is executed on edge server S_m, and 0 otherwise. On the side of inter-function data delivery, the receiving energy demand of edge server S_m is given by

$$\Psi_{\text{demand}}^{\text{in},m} = P_{\text{power}}^{\text{in},m} \times \sum_{h=1}^{H}\sum_{i=1}^{O_h}\sum_{b=1}^{b_{\text{replica}}^{h,i}} \left(\Delta_{\Phi(f_{h,i}^b)=m} \times \sum_{\kappa=o_i}^{z_i} \times \left(\frac{d_{h,\kappa,i}}{b_{\Phi(f_{h,\kappa}),\Phi(f_{h,i}^b)}} \times |\Delta_{\Phi(f_{h,\kappa})=m} - \Delta_{\Phi(f_{h,i}^b)=m}|\right)\right), \quad (8)$$

where $P_{\text{power}}^{\text{in},m}$ is the receiving power. Similarly, the delivering energy demand incurred by inter-function communication on edge server S_m is given by

$$\Psi_{\text{demand}}^{\text{out},m} = P_{\text{power}}^{\text{out},m} \times \sum_{h=1}^{H}\sum_{i=1}^{O_h}\sum_{b=1}^{b_{\text{replica}}^{h,i}} \left(\Delta_{\Phi(f_{h,i}^b)=m} \times \sum_{\varrho=\varpi_{i,b}}^{\varepsilon_{i,b}} \times \frac{d_{h,i,\varrho}}{b_{\Phi(f_{h,i}^m),\Phi(f_{h,\varrho})}} \times |\Delta_{\Phi(f_{h,i}^b)=m} - \Delta_{\Phi(f_{h,\varrho})=m}|\right), \quad (9)$$

where $\varpi_{i,b}$ and $\varepsilon_{i,b}$ are orderly the minimal and maximal successor identifiers of $f_{h,i}^b$. $P_{\text{power}}^{\text{out},m}$ is the data delivering power. By summing (7)–(9), the total energy demand for edge server S_m over a time period $[t, t+T]$ is thus derived by

$$\Psi_{\text{demand}}^m = \Psi_{\text{demand}}^{\text{exe},m} + \Psi_{\text{demand}}^{\text{in},m} + \Psi_{\text{demand}}^{\text{out},m}. \quad (10)$$

E. Pricing and QoE Model

Following the pay-per-use concept in serverless computing, we borrow a flexible pricing policy from [28] to balance the

completion time of serverless applications and their service fees. Specifically, let T^h_{finish} be the finish time of application $\mathcal{A}_h$, and $\Upsilon^h_{\text{price}}$ denote the resultant service price, then we have

$$\Upsilon^h_{\text{price}} = \begin{cases} \Upsilon^{\max,h}_{\text{price}}, & 0 \le T^h_{\text{finish}} \le T^h_{\text{desire}}, \\ \Upsilon^{\max,h}_{\text{price}} - \delta_h \times T^h_{\text{gap}}, & T^h_{\text{desire}} < T^h_{\text{finish}} \le D_h, \\ 0, & D_h < T^h_{\text{finish}}. \end{cases} \tag{11}$$

where $\Upsilon^{\max,h}_{\text{price}}$ is the maximal service fee. T^h_{gap} is calculated as the difference between T^h_{finish} and T^h_{desire}, and δ_h is the price decaying factor. To accommodate personalized user QoE preferences, we first introduce a latent variable ζ_h as

$$\zeta_h = \eta_h \times T^h_{\text{finish}} + (1-\eta_h) \times \Upsilon^h_{\text{price}}, \tag{12}$$

where $\eta_h \in [0,1]$ is the preference factor. Leveraging this latent variable, we then calculate the user QoE Q_h as

$$Q_h = \left(\alpha + \beta \times \left(\eta_h \times T^h_{\text{finish}} + (1-\eta_h) \times \Upsilon^h_{\text{price}}\right)\right)/\vartheta_h, \tag{13}$$

where α and β are preference parameters. ϑ_h is the scheduling priority that is determined in advance. Note that the preference parameters η_h, α, and β are treated as variables rather than fixed constants. We will lately detail the derivation of their values in Section V.

IV. Problem Formulation and Our Approach

A. Problem Description

In this paper, we focus on designing an optimal function embedding solution that optimizes the holistic user QoE in energy-harvesting SEC systems. Formally, given the target SEC system $\mathcal{G} = (\mathcal{S}, \mathcal{L})$ and application set $\mathcal{A} = \{\mathcal{A}_h | 1 \le h \le H\}$, our goal is to find the replica number $b^{h,i}_{\text{replica}}$ of function $f_{h,i}$, start time of function $f^b_{h,i}$, embedding location $\Phi(f^b_{h,i})$ of function $f^b_{h,i}$ for $h = 1, 2, \ldots, H$, $i = 1, 2, \ldots, O_h$, and $b = 1, 2, \ldots, b^{h,i}_{\text{replica}}$ that maximize the holistic user QoE. Our function embedding problem is formulated below.

$$\max \quad Q = \frac{1}{H}\sum_{h=1}^{H} Q_h \tag{14}$$

$$\text{s.t.} \quad \left(T^{h,\kappa_i,b}_{\text{finish}} + T^{h,\kappa_i,b}_{\text{transfer}}\right) \le T^{h,i,b}_{\text{begin}} \tag{15}$$

$$\sum_{h=1}^{H} \Upsilon^h_{\text{price}} \ge \Upsilon_{\text{goal}} \tag{16}$$

$$R(\Phi(\Gamma_{h,i})) \ge R^h_{\text{goal}} \tag{17}$$

$$\Psi^m_{\text{demand}}(t,T) \le \Psi^m_{\text{supply}}(t,T)$$

$$\forall h, i, m, b, f^b_{h,\kappa_i} \in \Gamma^{h,i,b}_{\text{parent}}. \tag{18}$$

Equation (15) ensures a proper function execution order according to the data-flow dependencies within each application. Here, $f^b_{h,\kappa_i} \in \Gamma^{h,i,b}_{\text{parent}}$ is the κ_i-th direct predecessor of function $f^b_{h,i}$. $T^{h,\kappa_i,b}_{\text{finish}}$ denotes the completion time of function f^b_{h,κ_i}. $T^{h,i,b}_{\text{begin}}$ represents the start time of function $f^b_{h,i}$. $T^{h,\kappa_i,b}_{\text{transfer}}$ indicates the communication delay between functions f^b_{h,κ_i} and $f^b_{h,i}$. (16) enforces that the total profit from user payments reaches a target revenue threshold. (17) is the reliability constraint on individual functions within each application. (18) restricts the energy consumption of each edge server within its energy budget during the scheduling horizon $[t, t+T]$.

B. Our Approach

As shown in Fig. 2, we propose a hybrid approach consisting of offline learning and online adaptation phases. At the offline phase, user QoE preferences are inferred by establishing QoE-preference mapping and preference-attribute mapping in Section V. This user QoE predictor quantitatively captures the personalized perceptions of service completion time and service costs across various application deployment purposes. Then, QoE preference profiles are leveraged by an RL-based function embedding scheme to derive static embedding decisions (see Section VI). At runtime, the intermittency of renewable energy sources may incur significant fluctuations in the available energy for SEC systems. To address this challenge, the online phase develops a lightweight function replica freezing scheme (see Section VII) that adaptively manages renewable energy variations. In our online scheme, we first conduct an energy-state analysis and design a novel embedding energy metric to guide dynamic function replica freezing. Accordingly, individual edge servers are classified into a high-energy or low-energy state based on their energy availability. For edge servers in the low-energy state, their deployed functions are then selectively frozen based on embedding energy metrics. For edge servers in the high-energy state, they adhere to static decisions while ensuring global data-flow consistency of serverless applications. By integrating online adaptation, our scheme can accommodate the renewable energy variations while incurring minimum system performance degradation.

V. Personalized QoE Preference Predictor

To address the diversity of user preferences, our QoE model in Section III-E incorporates unknown preference coefficients α, β, and η_h. This section presents a two-stage estimation scheme for inferring these preference parameters. Fig. 3 presents our personalized QoE preference predictor.

A. Establish QoE-Preference Mapping

1) Preference Estimation Across Application Deployment Purposes: To capture user preference diversity across various service scenarios, we customize a QoE questionnaire comprising representative time-fee levels that present feasible combinations of application finish time and resultant monetary costs. Each time-fee level is crafted to reflect realistic trade-offs that all users might encounter during function execution in SEC systems. Suppose that a total of H' IoT applications covering all service deployment purposes, i.e.,

$$\mathcal{A}_{\text{survey}} = \{\mathcal{A}_{h'} | 1 \le h' \le H' \wedge |\cup_{h'=1}^{H'} \Lambda_{h'}| = E\}, \tag{19}$$

is picked from the application pool $\mathcal{A} = \{\mathcal{A}_h | 1 \le h \le H\}$. Each selected application in $\mathcal{A}_{\text{survey}}$ serves as a deputy instance for its corresponding service deployment purpose and is used to evaluate all constructed time-fee levels on a quantitative scale. For application $\mathcal{A}_{h'}$ and level l $(1 \le l \le l_{\max})$, the offered finish time, the paid service fee, the latent utility, and the reported QoE score are recorded by $T^{h',l}_{\text{finish}}$, $\Upsilon^{h',l}_{\text{price}}$, $\zeta_{h',l}$, and $Q_{h',l}$, respectively. Then, the latent utility is expressed as

$$\zeta_{h',l} = \eta_{h'} \times T^{h',l}_{\text{finish}} + (1-\eta_{h'}) \times \Upsilon^{h',l}_{\text{price}}. \tag{20}$$

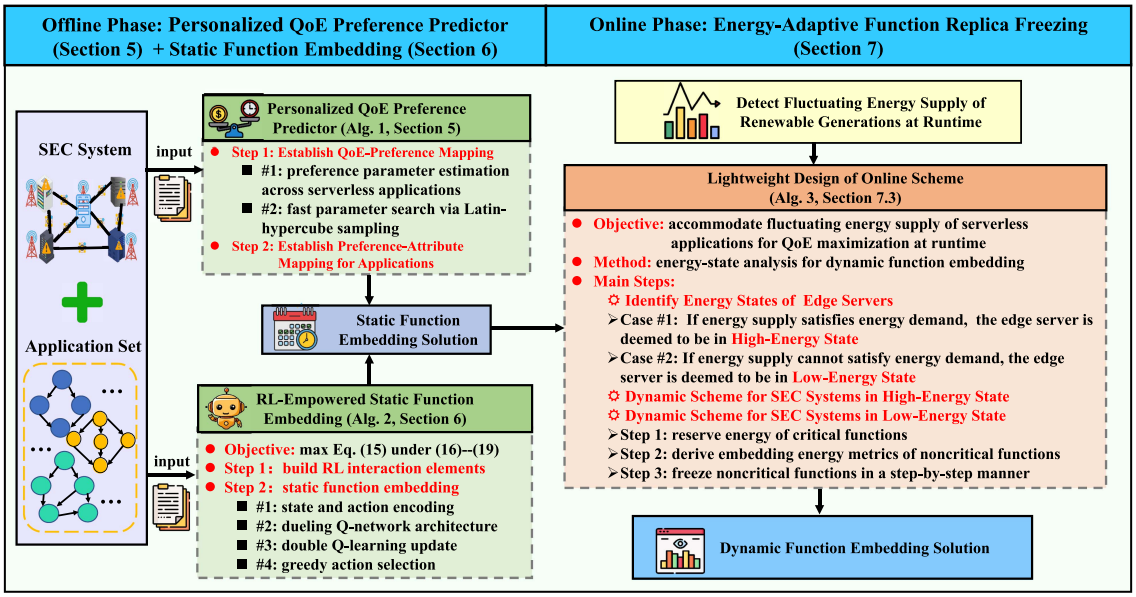

Fig. 2. Overview of our hybrid approach composed of offline learning and online adaptation phases.

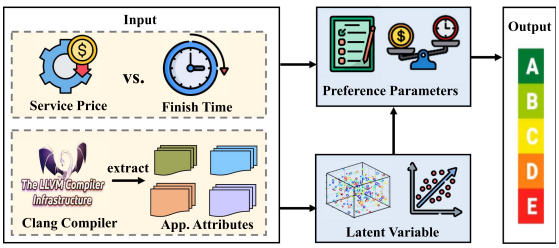

Fig. 3. Our personalized QoE preference predictor.

Intuitively, preference parameters α, β, and $[\eta_{h'}|1 \le h' \le H']$ could be estimated by minimizing the total squared error between the predicted and the observed QoE scores [29], i.e.,

$$\arg\min_{\alpha,\beta,\{\eta_{h'}\}} \sum_{h'=1}^{H'} \sum_{l=1}^{l_{\max}} \left(Q_{h',l} - \alpha - \beta \times \zeta_{h',l}\right)^2. \quad (21)$$

2) Fast Parameter Search Via Latin-Hypercube Sampling: In practice, directly solving (21) is computationally prohibitive because the search space expands rapidly with H' and $l_{\max}$. To overcome this challenge, we devise a fast parameter search scheme that leverages the powerful Latin-hypercube sampling (LHS) technique [30]. The procedure is as follows.

- *LHS Samples Generation:* We use the LHS technique to generate a candidate preference matrix $\mathcal{P}_{\text{sample}} = \{\eta_{h'}|1 \le h' \le H'\}$. The LHS is exploited here since it could yield a more representative set of preference weight samples compared with random sampling strategies.
- *Decomposition Fitting:* Given a fixed $\mathcal{P}_{\text{sample}}$, the preference coefficients α^* and β^* can be readily estimated by minimizing the sum of squared errors between the observed and predicted QoE scores, that is,

$$\arg\min_{\alpha,\beta} \sum_{h'=1}^{H'} \sum_{l=1}^{l_{\max}} \left(Q_{h',l} \times \vartheta_{h'} - \alpha - \beta \times \zeta_{h',l}\right)^2. \quad (22)$$

On the other hand, when holding α^* and β^*, an inferred preference weight matrix $\mathcal{P}^*_{\text{infer}}$ is in return obtained by solving another least-squares problem. In this step, we minimize the deviation between the normalized observed scores and the latent utility, as formulated by

$$\arg\min_{\eta_1,\ldots,\eta_{H'}} \sum_{h'=1}^{H'} \sum_{l=1}^{l_{\max}} \left(\frac{Q_{h',l} \times \vartheta_{h'} - \alpha^*}{\beta^*} - \zeta_{h',l}\right)^2. \quad (23)$$

- *Convergence Check:* The convergence of our search procedure is evaluated by the difference between the sampled preference matrix $\mathcal{P}_{\text{sample}}$ and the inferred preference matrix $\mathcal{P}^*_{\text{infer}}$. If their difference is less than a tolerance threshold ρ, the estimated parameters α^*, β^*, and $\mathcal{P}^*_{\text{infer}}$ at this iteration are thus accepted as the desirable solution. Otherwise, a new preference matrix $\mathcal{P}_{\text{sample}}$ is regenerated by invoking the LHS method again, and the iterative process continues until convergence criteria are satisfied.

B. Establish Preference-Attribute Mapping

We now seek to quantitatively establish a relationship between application-level serverless attributes and user-specific QoE preferences. In this regard, a prior study [31] identifies a set of program-level attributes that can describe behavioral distinctions across OpenCL applications. Building upon this foundation and considering the unique features of serverless workloads, we select nine attributes that are particularly suited for capturing serverless application diversity in SEC systems. Fig. 4 plots the relative importance of these nine attributes.

Specifically, we choose nine attributes comprise both general and serverless-specific elements. General attributes contain #1) the number of instruction cycles quantifying the total computation workload, #2) total volume of inter-function communication data within the application, 3) the number of execution blocks that indicates the function partitioning degree, #4) the number of control operations that measures the degree of branching, synchronization and conditional logic within an application, and #5) the number of mathematical operations that captures

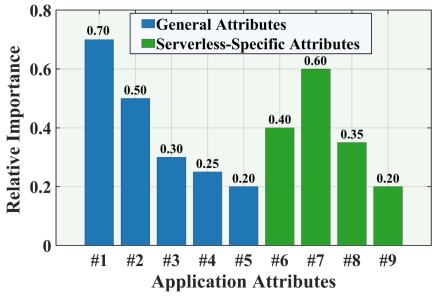

Fig. 4. The relative importance of application attributes.

arithmetic intensity. On the other hand, four serverless-specific attributes include #6) the depth of application DAG that determines the critical path length, #7) application deployment purposes impacting domain-specific latency and cost expectations for varied service scenarios, #8) ratio of DAG breadth to depth that characterizes a potential for parallel speedup, and #9) overall function count that reflects the inter-function orchestration complexity. For a single application, its nine attributes can be extracted through compiler-based profiling tools, such as the Clang suite [32].

Building on these insights, we develop a linear fitting approach to quantify the impact of serverless application attributes on user QoE preferences. In this approach, we first construct an application attribute matrix, denoted as $\mathcal{M}_{H'\times 9}$, where each row corresponds to a single application in $\mathcal{A}_{\text{survey}}$, and each column represents a weighted program-level attribute. Then, the relationship between application attributes and user QoE preferences is modeled as

$$\mathcal{X}^*_{9\times 1} = \arg\min_{\mathcal{X}_{9\times 1}} \|\mathcal{P}^*_{\text{infer}} - \mathcal{M}_{H'\times 9} \times \mathcal{X}_{9\times 1}\|^2, \quad (24)$$

where $\mathcal{X}_{9\times 1}$ is the mapping matrix. Because (24) is a standard linear least-squares problem, it can be efficiently solved using numerical optimization solvers. At this moment, preference parameters for the remaining applications can be inferred accordingly. That is, given the comprehensive attribute matrix $\mathcal{M}_{H\times 9}$ corresponding to all applications, the complete set $\mathcal{P}^*_{\text{wight}}$ with size $H \times 1$ is derived by

$$\mathcal{P}^*_{\text{weight}} = \mathcal{M}_{H\times 9} \times \mathcal{X}^*_{9\times 1}. \quad (25)$$

Putting together the procedures of building QoE-preference mapping in Section V-A and constructing preference-attribute mapping in Section V-B, Algorithm 1 presents our implementation details for user QoE preference estimation.

Algorithm 1: User-Centric QoE Preference Predictor.

Input: application set $\mathcal{A} = \{\mathcal{A}_h | 1 \le h \le H\}$.
select total H' IoT applications covering all service types in $\mathcal{A}$ to construct $\mathcal{A}_{\text{survey}} = \{\mathcal{A}_{h'} | 1 \le h' \le H' \wedge \cup_{h'=1}^{H'} \Lambda_{h'} = \Psi\} \subset \mathcal{A}$;
generate total $l_{\max}$ QoE-payment levels;
initialize convergence indicators: $\varsigma = [\varsigma_{h'} | 1 \le h' \le H'] \leftarrow 1$;
initialize preference weight matrix: $\mathcal{P}_{\text{sample}} = [\overline{\eta}_1, \overline{\eta}_2, \cdots, \overline{\eta}_{H'}] \leftarrow 1$;
while *outer stopping conditions are not met* **do**
  while $[\varsigma_{h'} | 1 \le h' \le H'] \neq 0$ **do**
    derive preference coefficient α^* and β^* using (22);
    for $h' = 1$ to H' **do**
      obtain preference weight $\eta^*_{h'}$ using (23);
      if $|\overline{\eta}_{h'} - \eta^*_{h'}| > \rho$ **then**
        break;
      else
        modify convergence indicator for $\mathcal{A}_{h'}$: $\varsigma_{h'} \leftarrow 0$;
  use LHS to re-generate $[\overline{\eta}_1, \overline{\eta}_2, \cdots, \overline{\eta}_{H'}]$;
obtain regression coefficient matrix $\mathcal{X}^*_{9\times 1}$ using (24);
for $h = 1$ to H *in parallel* **do**
  extract attribute vector $\mathcal{M}^h_{H\times 10}$ of application $\mathcal{A}_h$;
  infer preference weight $\eta^*_h \in \mathcal{P}^*_{\text{weight}}$ using (25);
return preference parameters: α^*, β^*, $\mathcal{P}^*_{\text{weight}} = \{\eta^*_1, \eta^*_2, \cdots, \eta^*_H\}$.

VI. RL-Empowered Static Function Embedding

A. Build RL Interaction Elements

We now describe our static function embedding problem as a Markov decision process (MDP). To this end, we first define the following essential elements of RL interaction.

1) State: To enable fine-grained scheduling decisions, the entire scheduling horizon $[t, t+T]$ is discretized into a series of consecutive time slots, denoted as $\mathbb{T} = \{t_u | 1 \le u \le U\}$. At each time slot t_u, the system state is represented by

$$\boldsymbol{s}_u = \left(\mathcal{A}^u_{\text{ready}}, \Psi^u_{\text{available}}, \mathcal{W}^u_{\text{load}}\right). \quad (26)$$

Specifically, $\mathcal{A}^u_{\text{ready}} = \{\mathcal{A}_w | 1 \le w \le A_u\}$ includes a set of serverless applications ready for scheduling at the current time slot t_u. The vector $\Psi_{\text{available}} = [\Psi^{m,u}_{\text{available}} | 1 \le m \le M]$ captures the remaining energy of each edge server at the beginning of time slot t_u, i.e.,

$$\Psi^{m,u}_{\text{available}} = \Psi^m_{\text{supply}}(t, T) - \sum_{v=1}^{u-1} \Psi^m_{\text{demand}}(t, t_v). \quad (27)$$

Additionally, a workload vector $\mathcal{W}^u_{\text{load}} = [W^{m,u}_{\text{load}} | 1 \le m \le M]$ is introduced to capture the accumulated computation workload on each edge server up to time slot t_u. For a single edge server S_m, its cumulative workload is inferred by

$$W^{m,u}_{\text{load}} = \sum_{v=1}^{u} \sum_{w=1}^{A_v} \sum_{i=1}^{O_w} \sum_{b=1}^{b^{w,i}_{\text{replica}}} \left(\Delta_{\Phi(f^b_{w,i})=m} \times W_{w,i}\right). \quad (28)$$

2) Action: At each time slot t_u, the RL agent determines a sequence of function embedding actions for pending applications in the ready queue. Each function embedding action specifies the destination edge server $\Phi(f^b_{w,i})$ for the b-th replica of function $f_{w,i}$, the total number of replicas $b^{w,i}_{\text{replica}}$, and the starting time $T^{w,i,b}_{\text{begin}}$ for each replica within t_u. Accordingly, the full action set at time slot t_u is given by

$$\boldsymbol{a}_u = \left\{ \left(\Phi(f^b_{w,i}), b^{w,i}_{\text{replica}}, T^{w,i,b}_{\text{begin}}\right) \mid \forall w, i, b \right\} \quad (29)$$

where $\mathcal{A}_w \in \mathcal{A}^u_{\text{ready}}$, $i \in [1, O_w]$, and $b \in [1, b^{w,i}_{\text{replica}}]$. The action trajectory $\boldsymbol{a}_u$ thus encapsulates all embedding decisions for the current set of ready applications at time slot t_u. After an action $a(\Phi(f^b_{w,i}), b^{w,i}_{\text{replica}}, T^{w,i,b}_{\text{begin}})$ is executed, the ready application set $\mathcal{A}^u_{\text{ready}}$ is thus modified by removing the scheduled functions. Meanwhile, the energy vector $\Psi^{m,u}_{\text{available}}$ for each edge server S_m is decremented by the energy consumed for executing the assigned functions and data transfers during t_u. Similarly, the workload vector $\mathcal{W}^u_{\text{load}}$ is updated to include the computational workload incurred by the scheduled function replicas on the selected servers. As a result, at the onset of the next time slot t_{u+1}, the system state is updated to $\boldsymbol{s}_{u+1} = (\mathcal{A}^{u+1}_{\text{ready}}, \Psi^{u+1}_{\text{available}}, \mathcal{W}^{u+1}_{\text{load}})$.

3) Reward: After executing all embedding actions at time slot t_u, the RL agent should receive an immediate reward r_u. Our

reward function is defined to jointly capture the user QoE and operational constraints, i.e.,

$$r_u = \sum_{w=1}^{A_u} \left(Q_w - \Delta_{\left[\Upsilon^w_{\text{price}} < \Upsilon_{\text{goal}}/H\right]} - \sum_{i=1}^{O_w} \Delta_{[R(\Phi(\Gamma_{w,i})) < R^w_{\text{goal}})]} \right) - \sum_{m=1}^{M} \Delta_{[\Psi^{m,u}_{\text{available}} < 0]} - \sum_{m=1}^{M} |W^{m,u}_{\text{load}}/C_m - \left(\sum_{m=1}^{M} W^{m,u}_{\text{load}} \right) / \sum_{m=1}^{M} C_m| \Bigg). \quad (30)$$

Here, Q_w denotes the QoE for application $\mathcal{A}_w$ at time slot t_u. The three indicator functions impose penalties for violations of the revenue, reliability, and energy constraints, respectively. The final term penalizes the reward function in proportion to the degree of workload imbalance among edge servers.

B. Design of RainbowDQN-Based Static Function Embedding

We leverage the Rainbow deep Q-Network (RainbowDQN) [33] technique to solve our MDP problem. RainbowDQN is employed here because it integrates several advanced RL mechanisms. For example, the dueling architecture could independently capture the state value and the advantage of each action in a large discrete-continuous embedding decision space. Besides, the adoption of multi-step return estimation facilitates the promising propagation of delayed rewards, which is particularly beneficial for capturing long-term QoE impacts. In addition, prioritized replay is able to sample beneficial transitions to improve the entire training efficiency.

1) Dueling Q-Network Architecture with Noisy Layers: RainbowDQN adopts a double Q-learning framework that decouples action selection and action evaluation by using an evaluation network with parameters $\boldsymbol{\theta}_1$ and a target network with parameters $\boldsymbol{\theta}_2$. In the evaluation network, a value stream estimates the state-value function $V(\boldsymbol{s}_u; \boldsymbol{\theta}_1)$ for quantifying an expected return of being in state $\boldsymbol{s}_u$. An advantage stream derives the superiority $A(\boldsymbol{s}_u, \boldsymbol{a}_u; \boldsymbol{\theta}_1)$ of each action in a given state. The Q-value for a state-action pair is then aggregated as

$$Q(\boldsymbol{s}_u, \boldsymbol{a}_u; \boldsymbol{\theta}_1) = V(\boldsymbol{s}_u; \boldsymbol{\theta}_1) + A(\boldsymbol{s}_u, \boldsymbol{a}_u; \boldsymbol{\theta}_1) - \frac{1}{|\boldsymbol{A}_u|} \sum_{\boldsymbol{a}_u \in \boldsymbol{A}_u} A(\boldsymbol{s}_u, \boldsymbol{a}_u; \boldsymbol{\theta}_1), \quad (31)$$

where $\boldsymbol{A}_u$ denotes all the possible actions at time slot t_u and $|\boldsymbol{A}_u|$ is the size of this action set. Furthermore, RainbowDQN enhances both value and advantage streams with learnable noisy layers instead of using traditional greedy strategies. The noise parameters will be co-optimized with the evaluation network weights, thus resulting in state-dependent and more efficient exploration throughout network training.

2) Double Q-Learning Update With Multi-Step Returns: At each learning iteration for time slot t_u, a mini-batch of historical transitions is sampled from a prioritized experience replay buffer $\mathcal{D}_{\text{PERB}}$. Transitions are drawn with probability proportional to their temporal-difference error magnitude for revisiting informative experiences. Each sampled transition is indexed by c and denoted as $(\boldsymbol{s}_c, \boldsymbol{a}_c, r_c, \boldsymbol{s}_{c+1})$. Rather than using an immediate reward in Q-value targets, our scheme aggregates the discounted rewards over G successive steps for long-term embedding action benefits. In this context, the target Q-value for transition c is given by

$$Q^c_{\text{target}} = \sum_{g=0}^{G-1} \gamma^g \times r_{c+g} + \gamma^G \times Q\left(\boldsymbol{s}_{c+G}, \boldsymbol{a}^*_{c+G}; \boldsymbol{\theta}_2\right), \quad (32)$$

where G is the multi-step return length and γ the discount factor. The optimal action $\boldsymbol{a}^*_{c+G}$ at state $\boldsymbol{s}_{c+G}$ is identified from the evaluation network as

$$\boldsymbol{a}^*_{c+G} = \arg\max_{\boldsymbol{a}_{c+G} \in \boldsymbol{A}_{c+G}} Q\left(\boldsymbol{s}_{c+G}, \boldsymbol{a}_{c+G}; \boldsymbol{\theta}_1\right). \quad (33)$$

Subsequently, a loss function for the evaluation network can be defined as the weighted mean squared error (MSE) between the target Q-values and the Q-values predicted by the evaluation network over the sampled mini-batch

$$\mathcal{Z}(\boldsymbol{\theta}_1) = \mathbb{E}_{\mathcal{D}_{\text{PERB}}} \left[y_c \times \left(Q^c_{\text{target}} - Q\left(\boldsymbol{s}_c, \boldsymbol{a}_c; \boldsymbol{\theta}_1\right) \right)^2 \right], \quad (34)$$

where y_c is the importance-sampling weight. Generally, stochastic gradient descent is employed to minimize $\mathcal{Z}(\boldsymbol{\theta}_1)$ such that the evaluation network parameters $\boldsymbol{\theta}_1$ are renewed accordingly. Meanwhile, the target network parameters $\boldsymbol{\theta}_2$ are updated via soft synchronization

$$\boldsymbol{\theta}_2 \leftarrow \tau \times \boldsymbol{\theta}_1 + (1 - \tau) \times \boldsymbol{\theta}_2, \quad (35)$$

where $\tau \in (0, 1]$ is the soft-update rate.

3) Action Selection with Noisy Networks: As aforementioned, RainbowDQN enhances the exploration process by inducing noise stochasticity in the Q-network architecture. In particular, the weights and biases of the noisy layers within the evaluation network are adaptively perturbed as follows.

$$\boldsymbol{w} = y_1 + \sigma_1 \odot \epsilon_1, \quad \boldsymbol{b} = y_2 + \sigma_2 \odot \epsilon_2. \quad (36)$$

$\boldsymbol{w}$ and $\boldsymbol{b}$ denote the weight and bias vectors of the noisy linear layers, respectively. y_1 and y_2 are the mean parameters. σ_1 and σ_2 are the standard deviations. ϵ_1 and ϵ_2 are zero-mean noise variables. The element-wise multiplication $\odot$ allows independent modulation of stochasticity for each parameter. During action selection, the Q-value estimation becomes stochastic due to these noisy parameters. The policy for selecting an action at time slot t_u is thus defined as

$$\boldsymbol{a}_u = \arg\max_{\boldsymbol{a}_u \in \boldsymbol{A}_u} Q\left(\boldsymbol{s}_u, \boldsymbol{a}_u; \boldsymbol{\theta}_1\right). \quad (37)$$

Algorithm 2 presents our static function embedding scheme. Initially, Algorithm 1 is called to estimate user QoE preference parameters (line 1). Then, lines 2-17 conduct the training process of evaluation and target networks. After completing all training episodes, the optimized static embedding decisions are derived from the evaluation network outputs in line 18.

VII. Energy-Adaptive Function Replica Freezing

A. Energy-State Analysis for Dynamic Function Embedding

In practice, the intermittency of renewable energy sources often incurs fluctuations in the SEC available energy at runtime [34]. To facilitate fine-grained analyses, we categorize the online operational state of each edge server with respect to energy supply into two states. Specifically, an edge server is considered to be in a *high-energy state* if its available energy at the online stage is sufficient to accomplish all functions arranged by the static function embedding decisions. Otherwise, it is designated as being in a *low-energy state*.

1) Edge Servers in the High-Energy State: For edge servers in the high-energy state, their available energy is sufficient to finish all statically embedded functions within the scheduling horizon. In this situation, these edge servers are likely to follow the static

Algorithm 2: Static Function Embedding Scheme.

Input: SEC system $\mathcal{G} = (\mathcal{S}, \mathcal{L})$, application set $\mathcal{A}$.
call Algorithm 1 to estimate user preference parameters;
initialize evaluation network parameters with noisy layers: $\boldsymbol{\theta}_1$;
initialize target network parameters: $\boldsymbol{\theta}_2 \leftarrow \boldsymbol{\theta}_1$;
initialize a prioritized experience replay buffer $\mathcal{D}_{\text{PERB}}$;
for each training episode **do**
  construct an initial state $\boldsymbol{s}_1 = (\mathcal{A}^1_{\text{ready}}, \Psi^1_{\text{available}}, \mathcal{W}^1_{\text{load}})$;
  for each time slot $u = 1, 2, ..., U$ **do**
    encode state $\boldsymbol{s}_u$ as input for the evaluation network;
    select action $\boldsymbol{a}_u$ using noisy exploration in (37);
    execute action $\boldsymbol{a}_u$ and observe reward r_u and next state $\boldsymbol{s}_{u+1}$;
    store transition $(\boldsymbol{s}_u, \boldsymbol{a}_u, r_u, \boldsymbol{s}_{u+1})$ into $\mathcal{D}_{\text{PERB}}$;
    sample a mini-batch of prioritized transitions from $\mathcal{D}_{\text{PERB}}$;
    for each sampled transition $(\boldsymbol{s}_c, \boldsymbol{a}_c, r_c, \boldsymbol{s}_{c+G})$ **do**
      infer multi-step optimal action $\boldsymbol{a}^*_{c+G}$ via (33);
      derive multi-step Q-value Q^c_{target} via (32);
    update evaluation network parameters $\boldsymbol{\theta}_1$ in (34) via stochastic gradient descent;
    renew target network parameters $\boldsymbol{\theta}_2$ via soft-update rule (35);
return static embedding decisions $(\Phi(f^b_{h,i}), b^{h,i}_{\text{replica}}, T^{h,i,b}_{\text{begin}} | h \in [1, H], i \in [1, O_h], b \in [1, b^{h,i}_{\text{replica}}])$ from the evaluation network.

Algorithm 3: Energy-Adaptive Function Replica Freezing.

Input: online energy supply $\{\Psi^m_{\text{supply}} | m \in [1, M]\}$, statically embedded function sets $\{\Gamma_m | m \in [1, M]\}$ generated by Algorithm 2.
for *each edge server* $S_m \in \mathcal{S}$ **do**
  if $\Psi^m_{\text{demand}} < \Psi^m_{\text{supply}}$ **then**
    retain current embedded functions Γ_m on S_m; `// Edge servers in the high-energy state;`
  else
    `// Edge servers in the low-energy state;`
    identify all noncritical functions in Γ_m to construct $\Gamma^*_m = \{f_{m,u} | 1 \le u \le U_m\}$;
    for *each function* $f_{m,u} \in \Gamma^*_m$ **do**
      derive embedding energy metric $\Psi^{m,u}_{\text{embed}}$ using (39);
    sort all functions in Γ^*_m in descending order of embedding energy metric;
    reset function counter: $u \leftarrow 1$;
    while $\Psi^m_{\text{demand}} > \Psi^m_{\text{supply}}$ **do**
      discard $f_{m,u}$: $\Gamma_m \leftarrow \Gamma_m - \{f_{m,u}\}$;
      $\Psi^m_{\text{demand}} \leftarrow \Psi^m_{\text{demand}} - \Psi_{\text{demand}}(f_{m,u}, S_m)$;
      discard all downstream dependent functions of $f_{m,u}$ from $\{\Gamma_m | m \in [1, M]\}$ and update energy demand of corresponding edge servers;
      renew function counter: $u \leftarrow u + 1$;

embedding decisions. Nevertheless, the data-flow consistency within each application DAG across all distributed edge servers must still be enforced. That is, if a function has been discarded on an edge server due to local energy limitations, then all of its downstream dependent functions must also be removed from other edge servers.

2) Edge Servers in the Low-Energy State: For edge servers in the low-energy state, their available energy is insufficient to accomplish all embedded functions at runtime. To guide function retention effectively, we below present an energy profile analysis of dynamic function embedding on individual edge servers. Specifically, according to (7)–(9), the energy consumption of a single function $f^b_{h,i}$ when embedded on edge server S_m is readily derived by

$$\Psi_{\text{demand}}(f^b_{h,i}, S_m) = P^{\text{in},m}_{\text{power}} \times \sum_{\kappa=o_i}^{z_i} \left(\frac{d_{h,\kappa,i}}{b_{\Phi(f_{h,\kappa}),\Phi(f^b_{h,i})}} \times |\Delta_{\Phi(f_{h,\kappa})=m} - \Delta_{\Phi(f^b_{h,i})=m}| \right) + \mu_{h,i} \times C^2_m \times W_{h,i} + P^{\text{out},m}_{\text{power}} \times \sum_{\varrho=\varpi_{i,b}}^{\phi_{i,b}} \frac{d_{h,i,\varrho} \times |\Delta_{\Phi(f^b_{h,i})=m} - \Delta_{\Phi(f_{h,\varrho})=m}|}{b_{\Phi(f^m_{h,i}),\Phi(f_{h,\varrho})}}. \quad (38)$$

We then introduce an embedding energy metric defined as the energy consumption per instruction cycle of a function, i.e.,

$$\Psi_{\text{embed}}(f^b_{h,i}, S_m) = \frac{\Psi_{\text{demand}}(f^b_{h,i}, S_m)}{W_{h,i}}. \quad (39)$$

Essentially, this embedding energy metric quantifies the energy efficiency of executing each function on individual edge servers. On this basis, we design a greedy function replica freezing mechanism. Specifically, it gives top retain-priority to preserve the critical functions that are located on the critical-path of each application DAG or only have a single replica (i.e., the function itself). After reserving energy for all critical functions, edge servers will examine the remaining noncritical functions. These noncritical functions are sorted in descending order according to their embedding energy metric. The edge servers then discard less energy-efficient functions in a step-by-step manner until the reduced energy demand of retained functions matches the available energy budget. Similarly, when a function is discarded due to energy limitations, all of its downstream dependent functions within the application DAG, must also be eliminated regardless of their embedding locations on individual edge servers.

We should emphasize that the dynamic function embedding may require real-time energy supply measurements to support online adaptation. In practical SEC systems, each edge server can be equipped with an energy monitoring module that records energy-harvesting power by using built-in sensors or microcontroller-based measurement circuits. These measurements are typically sampled at some fixed intervals, such as every few seconds or minutes. Following prior works [8], [9], [34], we neglect the overhead of continuous energy-harvesting power monitoring, as it is much smaller compared to the costs of function execution and communication. As part of future work, we will quantify the impact of different measurement intervals and monitoring strategies on system performance.

B. Error Check for Low-Energy and High-Energy States

As discussed earlier, our SEC system is subject to both bit errors and soft errors. To detect such hybrid errors, an acceptance-rejection test [24], [35] is conducted after the completion of each function. Only those functions whose outputs satisfy the acceptance criteria are allowed to commit their results. Otherwise, any function failing the test is directly discarded, and all its downstream dependent functions in the application DAG are recursively removed to prohibit the cascading propagation of erroneous outputs. Since the time cost of acceptance-rejection test has been validated to be negligible compared with function execution time [24], [35], we hereby assume that such error detection does not prolong the overall function execution duration.

Fig. 5. Spatial deployment distribution of a portion of the 3223 base stations in Shanghai Telecom [20]. Each red-colored marker in this map indicates an exact deployment location (i.e., longitude and latitude) of a base station.

TABLE II
HARDWARE SPECIFICATIONS OF COMMERCIAL EDGE SERVERS IN OUR SEC PLATFORM

Manufacturer	HUAWEI [36], [37]		ASUSTeK Computer Inc. [38]				Dell Technologies Inc. [39]	
Product	AtlasPro-3000	TaiShan 2280E	ESC-NB8	ESC-A8A	RS720A-E13	RS720-E12	XR11	XR7620
CPU Type	Kunpeng 320-9210	Kunpeng 920-5520	Intel Xeon Gold 6548Y	AMD EPYC 9135	AMD EPYC 9334	Intel Xeon Gold 5512U	Intel Xeon Platinum 8351N	Intel Xeon Platinum 8580
#Cores	24	32	16	16	32	28	36	60
Frequency Per Core	2.6 GHz	2.6 GHz	3.5 GHz	4.3 GHz	2.7 GHz	2.1 GHz	2.4 GHz	2.0 GHz
Total Capacity	62.4 GHz	83.2 GHz	56.0 GHz	68.8 GHz	86.4 GHz	58.8 GHz	86.4 GHz	120.0 GHz
#Servers	40	60	70	80	120	130	80	120

C. Algorithm of Dynamic Function Replica Freezing

Our dynamic function replica freezing procedure is presented in Algorithm 3. For easy presentation, let $\Gamma_m = \{f_{m,1}, f_{m,2}, \ldots, f_{m,u}, \ldots, f_{m,U_m}\}$ denote a collection of functions statically embedded on edge server S_m, where the explicit application and replica indices are omitted. The embedding energy metric associated with function $f_{m,u}$ is denoted by $\Psi_{\text{embed}}^{m,u}$. Initially, each edge server S_m is assessed by comparing its energy demand with current energy supply (lines 1-2). If the available energy of edge server S_m is sufficient, it remains in the high-energy state, and all statically embedded functions in Γ_m are temporarily retained (line 3). In contrast, a greedy function freezing mechanism is activated (lines 4-14). The algorithm first constructs a set of noncritical functions Γ_m^* (line 5). Subsequently, the embedding energy metric $\Psi_{\text{embed}}^{m,u}$ for each noncritical function $f_{m,u} \in \Gamma_m^*$ is inferred (lines 6-7). Next, noncritical functions are sorted in descending order according to their embedding energy metrics (line 8). Starting from the function with the highest embedding energy metric, functions are sequentially discarded until the energy demand matches the available supply (lines 9-12). Notably, when a function is discarded due to energy limitations, the algorithm recursively discards all downstream dependent functions embedded on other edge servers (line 13).

VIII. NUMERICAL RESULTS

A. SEC Platform

We build an SEC simulation platform upon the real-world base station dataset from Shanghai Telecom [20]. Fig. 5 illustrates the spatial deployment distribution of a portion of the total 3223 base stations. Besides, our SEC platform incorporates a variety of commercial edge servers, including HUAWEI AtlasPro [36] and TaiShan [37], ASUS ESC NB8-E11, ESC A8A-E12 U, RS720A-E13-RS24 U and RS720-E12-RS8G [38], as well as Dell PowerEdge XR11 and XR7620 [39]. The hardware specifications of these edge servers are summarized in Table II. The inter-server communication bandwidth is randomly assigned values within $[20, 200] \times 10^2$ MB/s. Fault rates of edge servers and communication links are sampled from $[2 \times 10^{-9}, 8 \times 10^{-6}]$ and $[3 \times 10^{-8}, 9 \times 10^{-5}]$, respectively.

B. Serverless Applications

For serverless applications, we select 2000 DAG-structured tasks from the Alibaba 8-day open cluster traces in realistic production environments [22]. To ensure a broad coverage of practical scenarios, these applications are evenly distributed across 10 distinct deployment purposes. For individual functions within a single application, the number of their instruction cycles and the amount of their output data are scaled into $[2 \times 10^8, 7 \times 10^{13}]$ and $[10, 200] \times 10^2$ MB, respectively. Application deadlines are assigned within $[20, 500]$ seconds, while target completion time is 0.5 to 0.8 of their corresponding deadlines. The maximal service fees and price decaying factors of individual applications are set to $[5, 100]$ dollars and $[1, 10]$, respectively. In addition, every application receives a priority between 1 (lowest) and 10 (highest).

C. Comparative Algorithms

1) Basic Idea of Comparative Algorithms: In the comparative investigations, we evaluate our hybrid approach against the following state-of-the-art baseline algorithms.

- CoRE [10] is developed on the utility density of serverless applications to enhance the revenue of service providers and energy savings of wireless devices.
- AUCT [12] is an auction-based serverless pricing scheme to balance the monetary costs of serverless users and the profit of serverless providers in SEC systems.
- MOSEC [14] is a double deep Q-network (DDQN)-based function offloading algorithm that aims at jointly

optimizing the three objectives of application finish time, energy consumption, and user monetary costs.
- ORel [18] is a decomposition-based function placement method to accomplish function-to-server mapping under bit errors and soft errors in SEC systems.
- faasHouse [8] is an energy-aware scheduling scheme that leverages the house allocation theory to decide function placements in energy-harvesting SEC systems.
- Retry [17] is an online function embedding scheme that uses random function replica freezing method and retry-based fault-tolerance mechanisms to ensure atomic visibility of function updates.

2) Parameter Settings of Comparative Algorithms: For peer methods [8], [10], [12], [14], [17], [18], we adopt their default parameter configurations provided in original studies. For our RainbowDQN method, the evaluation and target networks both have three hidden layers, and each hidden layer has 512 units. Referring to [14], [33], the target network is updated via soft synchronization with a rate of 0.3. The length of multi-step returns is 5, and the discount factor is 0.99. We employ the Adam optimizer at a learning rate of 6.25×10^{-5} and set a mini-batch size of 64. All comparative algorithms are evaluated on a Dell P5820x machine configured with an Intel i9 10900X processor and two GeForce RTX3080 GPUs.

D. Evaluation on QoE Preference Prediction

In this set of experiments, a total of 1000 applications are sampled from the application pool to train our QoE preference predictor. Specifically, 900 applications are served as the training set while the remaining 100 applications are divided between validation and test cases. To capture user preferences, each application is configured with five completion time options: 80%, 85%, 90%, 95%, and 100% of its baseline value. Meanwhile, the service fees charging for that application are set to 140%, 130%, 120%, 110%, and 100% of a baseline service fee, respectively. By pairing these options, a total of 25 distinct time-fee combinations are formed for every application. Then, we evenly assign 1000 applications to 30 invited experts, so that each expert is bounded with about 350 applications. The 30 participants include students, university faculty, software developers, scientific researchers and financial analysts, and are currently aged between 20 and 45 years. Their educational backgrounds are distributed as 20% bachelor degrees, 40% master degrees, and 40% doctoral degrees. All participants need to rate their QoE for time-fee combinations on a scale from 1 (lowest) to 10 (highest).

Fig. 6 shows the confusion matrix of prediction results. The diagonal values represent successful estimations that range from 77.2% to 90.3%. Whereas the off-diagonal elements indicate underestimation or overestimation results. These results confirm that the superiority of our predictor in capturing user QoE preferences across diverse application scenarios.

E. Evaluation on Static Algorithms

1) Hyperparameter Study on RainbowDQN: We first generate baseline configurations of application deadlines, reliability goals, and revenue targets as specified in Section VIII-B. To investigate the effect of key hyperparameters on RainbowDQN, we conduct ablation experiments on the soft-update rate for the target network and the length of multi-step returns. As

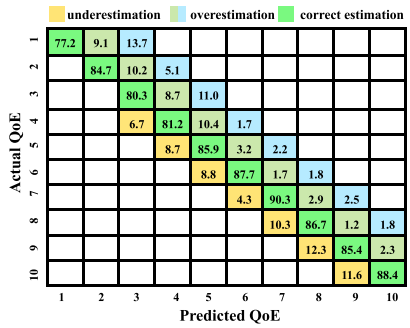

Fig. 6. Accuracy of QoE preference predictor.

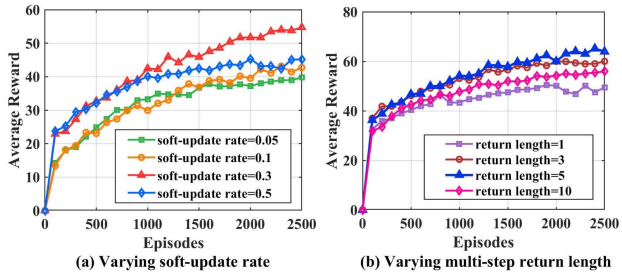

Fig. 7. Comparison of average reward for different hyperparameters of our RainbowDQN-based scheme.

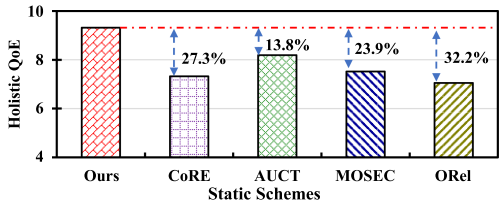

Fig. 8. Holistic QoE under baseline configurations.

illustrated in Fig. 7, setting the soft-update rate to 0.3 achieves consistently higher average rewards compared to other values. Similarly, a multi-step return length of 5 leads to the best learning performance among the tested options. Thus, we adopt these values as default settings in all experiments.

2) Baseline Comparison Results: Under baseline configurations, Fig. 8 demonstrates the corresponding holistic QoE achieved by our RainbowDQN-based offline method and four static peer schemes CoRE [10], AUCT [12], MOSEC [14] as well as ORel [18]. As shown in Fig. 8, our method attains the highest holistic QoE values among all competitors. The holistic QoE improvements over the four representative peer schemes are 27.3%, 13.8%, 23.9%, and 32.2%, respectively. These striking gains are attributed to the joint use of personalized QoE preference estimation and RainbowDQN-based function embedding mechanisms.

Table III further lists the solution feasibility of five comparative algorithms. The metric *RevGoal* denotes the ratio of an aggregated provider revenue collected from all users to the predefined revenue target. *TimeRatio* measures the fraction of applications meeting deadline constraints, and *RelRatio* captures the fraction of applications satisfying reliability requirements. A combined metric *TimeRelRatio* reveals the proportion of

TABLE III
SOLUTION FEASIBILITY UNDER BASELINE CONFIGURATIONS

	RevGoal	TimeRatio	RelRatio	TimeRelRatio
CoRE	100%	86.3%	70.7%	67.9%
AUCT	100%	79.8%	77.8%	74.8%
MOSEC	100%	88.5%	84.2%	83.3%
ORel	70.5%	92.7%	90.5%	87.7%
Ours	100%	94.4%	100%	94.4%

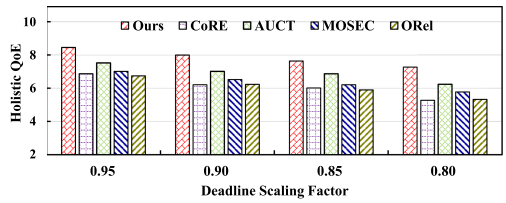

Fig. 9. Holistic QoE under varying application deadlines.

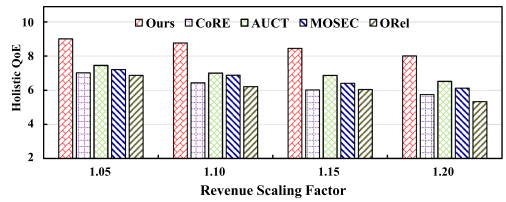

Fig. 10. Holistic QoE under varying provider revenue targets.

applications that simultaneously meet both deadline and reliability constraints. Note that all algorithms enforce inter-function precedence and energy constraints, so these metrics are omitted. Overall, only our approach is capable of meeting both revenue targets and reliability requirements.

3) Impact of Scaling Application Deadlines: We then investigate the impact of scaling application deadlines on holistic QoE degradation. To this end, we introduce a set of deadline scaling factors $\phi_{\text{time}} \in \{0.95, 0.90, 0.85, 0.80\}$, where the deadline for each application is proportionally reduced to 0.95, 0.90, 0.85, and 0.80 times the baseline value. Throughout this evaluation, both reliability goals and revenue configurations are maintained at default settings. We observe from Fig. 9 that all methods present an expected QoE decrease as deadlines tighten due to a reduced feasible space for function placement. However, our method degrades more slowly as the deadline budget shrinks and maintains consistent advantages across the entire range of deadline scaling factors.

4) Impact of Scaling Revenue Targets: To evaluate the sensitivity of holistic QoE to varying revenue targets, we introduce a revenue scaling factor $\phi_{\text{rev}} \in \{1.05, 1.10, 1.15, 1.20\}$, which proportionally increases the provider revenue requirement to 1.05, 1.10, 1.15, and 1.20 times the baseline value. Fig. 10 shows that the holistic QoE decreases as provider revenue targets become stricter for all comparative algorithms. This is because a higher revenue target enforces the resultant function embedding decisions to favor serverless applications that can tolerate higher user payments. Accordingly, the fee-sensitive applications are more likely to experience a degradation in QoE due to longer completion time. On the other hand, we observe that our method remains the best across all revenue scaling factors. For instance, when ϕ_{rev} is set to 1.20, our method surpasses CoRE [10], AUCT [12], MOSEC [14] and ORel [18] by 39.3%, 22.9%, 30.8%, and 50.2%, respectively.

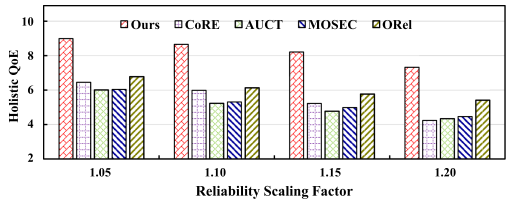

Fig. 11. Holistic QoE under varying reliability goals.

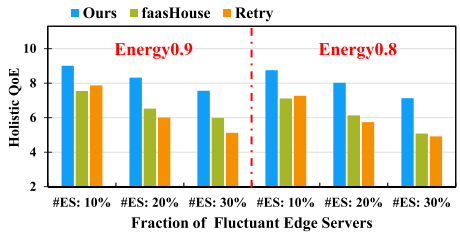

Fig. 12. Holistic QoE achieved by dynamic algorithms.

5) Impact of Scaling Reliability Goals: To assess the sensitivity of holistic QoE to reliability requirements, we select a group of reliability scaling factors $\phi_{\text{rel}} \in \{1.05, 1.10, 1.15, 1.20\}$. Fig. 11 shows that a stricter reliability target results in a sharper QoE decline than either tighter application deadlines or higher provider revenue targets. Across all reliability scaling factors, our method consistently achieves the highest QoE values and exhibits a smaller relative drop of 18.6%, compared with 34.3% for CoRE [10], 27.8% for AUCT [12], 26.0% for MOSEC [14], and 20.2% for ORel [18]. At the strictest reliability scaling factor of 1.20, our method still exceeds CoRE [10], AUCT [12], MOSEC [14] and ORel [18] by 72.6%, 68.7%, 63.8%, and 35.3%, respectively.

6) Solution Feasibility Under Scaled Conditions: Table IV compares the feasibility of five algorithms. For each scaling factor (i.e., $\phi_{\text{time}} = 0.9$), the four columns in this table orderly record the *RevGoal*, *TimeRatio*, *RelRatio*, and *TimeRelRatio* values. We see that our method is the only one that maintains *RevGoal* $= 100\%$ and *RelRatio* $= 100\%$ for all scaling factors. The infeasibility of CoRE [10], AUCT [12], and MOSEC [14] mainly stems from their complete disregard of soft and bit errors during function embedding. By contrast, ORel [18] attempts to mitigate faults by preferentially mapping fault-sensitive applications to the edge servers with lower error arrival rates. However, the absence of function redundancy or other fault-masking techniques prevents ORel [18] from satisfying stricter reliability requirements.

F. Evaluation on Dynamic Algorithms

1) Holistic QoE Under Fluctuating Energy Supply: To simulate online fluctuating energy scenarios, we scale the available energy on a subset of edge servers by an energy scaling factor $\phi_{\text{energy}} \in \{0.9, 0.8\}$ and vary the affected share of edge servers by $\{10\%, 20\%, 30\%\}$. Fig. 12 plots the holistic QoE achieved by our dynamic scheme, faasHouse [8], and Retry [17]. We observe that our dynamic scheme improves the holistic QoE by 27.9% and 34.2% on average compared with faasHouse [8] and Retry [17], respectively.

TABLE IV
SOLUTION FEASIBILITY UNDER SCALED CONDITIONS OF APPLICATION DEADLINES, REVENUE TARGETS, AND RELIABILITY GOALS

ϕ_{time}	Scaling Application Deadlines: $\phi_{\text{time}} \in \{0.95, 0.90, 0.85, 0.80\}$, {*RevGoal*, *TimeRatio*, *RelRatio*, *TimeRelRatio*}															
	$\phi_{\text{time}} = 0.95$, (%)				$\phi_{\text{time}} = 0.90$, (%)				$\phi_{\text{time}} = 0.85$, (%)				$\phi_{\text{time}} = 0.80$, (%)			
CoRE	100	80.4	64.2	60.2	100	71.8	53.4	51.8	100	58.0	43.9	46.2	100	51.5	37.8	38.5
AUCT	100	74.8	68.3	66.5	100	64.4	58.0	58.3	100	51.7	50.6	51.2	100	45.1	45.3	43.2
MOSEC	100	80.7	76.4	70.1	100	69.5	62.0	60.3	100	59.7	53.3	53.6	100	50.6	45.4	45.3
ORel	65.5	88.2	84.7	82.4	60.1	74.3	72.3	71.8	53.1	61.2	64.3	58.6	50.4	51.1	55.0	48.9
Ours	100	90.3	100	90.3	100	85.4	100	85.4	100	80.2	100	80.2	100	74.2	100	74.2
ϕ_{rev}	Scaling Revenue Targets: $\phi_{\text{rev}} \in \{1.05, 1.10, 1.15, 1.20\}$, {*RevGoal*, *TimeRatio*, *RelRatio*, *TimeRelRatio*}															
	$\phi_{\text{time}} = 1.05$, (%)				$\phi_{\text{time}} = 1.10$, (%)				$\phi_{\text{time}} = 1.15$, (%)				$\phi_{\text{time}} = 1.20$, (%)			
CoRE	100	81.3	64.2	61.2	100	73.1	52.5	51.8	100	63.7	43.8	45.4	100	55.1	36.0	37.3
AUCT	100	70.2	65.6	63.4	100	62.7	57.2	55.7	100	54.9	47.6	45.5	100	44.6	41.8	40.7
MOSEC	100	77.8	75.4	72.2	100	67.1	62.8	57.9	100	59.5	51.9	49.1	100	51.7	43.7	40.2
ORel	65.5	81.7	82.1	79.4	58.2	67.8	70.7	65.0	50.7	57.7	61.7	56.8	45.3	48.5	52.3	45.9
Ours	100	85.8	100	85.8	100	82.1	100	82.1	100	77.3	100	77.3	100	70.2	100	70.2
ϕ_{rel}	Scaling Reliability Goals: $\phi_{\text{rel}} \in \{1.05, 1.10, 1.15, 1.20\}$, {*RevGoal*, *TimeRatio*, *RelRatio*, *TimeRelRatio*}															
	$\phi_{\text{time}} = 1.05$, (%)				$\phi_{\text{time}} = 1.10$, (%)				$\phi_{\text{time}} = 1.15$, (%)				$\phi_{\text{time}} = 1.20$, (%)			
CoRE	100	77.4	60.3	54.7	100	65.7	51.2	48.0	100	56.2	41.6	40.9	100	48.9	37.2	34.4
AUCT	100	68.4	61.7	53.2	100	59.0	53.5	44.8	100	51.0	44.8	38.9	100	44.7	39.6	32.2
MOSEC	100	72.9	70.4	65.3	100	64.2	63.0	54.4	100	54.4	55.4	46.0	100	46.3	46.2	39.6
ORel	60.1	80.3	80.2	77.2	60.1	71.3	65.1	67.4	53.1	58.3	55.6	60.6	47.7	49.2	45.7	52.8
Ours	100	83.2	100	83.2	100	80.5	100	80.5	100	74.7	100	74.7	100	70.4	100	70.4

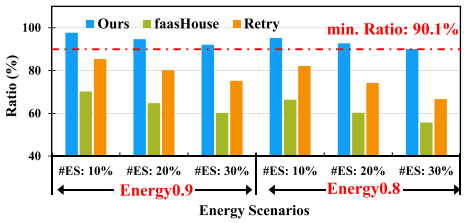

Fig. 13. The ratio of applications that meet reliability goals.

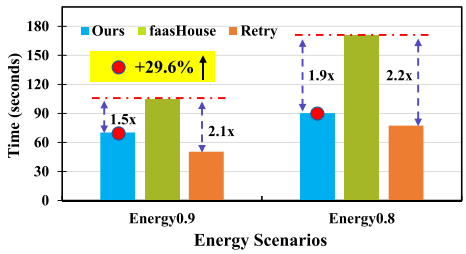

Fig. 14. Measured time overheads of dynamic algorithms.

2) Ratio of Application Reliability Satisfaction: We evaluate the ratio of applications that satisfy reliability requirements under provider revenue constraints in two distinct energy supply scenarios. As shown in Fig. 13, the *RelRatio* values of our scheme lie between 92.1% and 97.7% when $\phi_{\text{energy}} = 0.9$. For a harsher energy scenario of $\phi_{\text{energy}} = 0.8$, our scheme still achieves the minimum *RelRatio* value of 90.1%. On the other hand, faasHouse [8] and Retry [17] yield markedly lower *RelRatio* values across the two energy scenarios.

3) Measured Time Overheads: Fig. 14 illustrates the measured time required by dynamic methods to generate online function embedding solutions. Our approach consumes 70.3 seconds at $\phi_{\text{energy}} = 0.9$ and 90.4 seconds at $\phi_{\text{energy}} = 0.8$. In comparison, faasHouse [8] requires 104.5 and 170.8 seconds, while Retry [17] records 50.5 and 77.4 seconds under the same conditions. The results indicate that our scheme achieves substantially faster scheduling than faasHouse [8] and maintains a comparable time overhead to Retry [17] across both scenarios.

IX. CONCLUSION

In this paper, we present a hybrid function embedding approach for SEC systems powered by renewable energy. Our proposed approach jointly considers user QoE preferences, energy supply fluctuations, provider revenue targets, and reliability constraints. Extensive experiments under both static and dynamic conditions demonstrate that the proposed method outperforms representative baselines.

REFERENCES

[1] B. Zolfaghari, S. Abrishami, A. Rasoolzadegan, and B. Javadi, "Dynamic function placement and request scheduling of serverless workflows in edge environment," *IEEE Trans. Serv. Comput.*, vol. 18, no. 5, pp. 2781–2793, Sep./Oct. 2025.

[2] J. Mai, K. Cao, and T. Wei, "Personalized federated learning with state-adaptive IoT device scheduling in mobile-edge computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 44, no. 11, pp. 4086–4099, Nov. 2025.

[3] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless computing: State-of-the-art, challenges and opportunities," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1522–1539, Mar./Apr. 2023.

[4] F. Righetti, B. Cornacchia, G. R. Russo, N. Tonellotto, V. Cardellini, and C. Vallati, "Energy-efficient function invocation scheduling for edge FaaS platforms," in *Proc. IEEE Int. Conf. Smart Comput.*, 2025, pp. 18–25.

[5] X. Shang, Y. Mao, Y. Liu, Y. Huang, Z. Liu, and Y. Yang, "Online container scheduling for data-intensive applications in serverless edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2023, pp. 1–10.

[6] M. Golec et al., "ATOM: AI-powered sustainable resource management for serverless edge computing environments," *IEEE Trans. Sustain. Comput.*, vol. 9, no. 6, pp. 817–829, Nov./Dec. 2024.

[7] Y. Kim, B. Kim, T. Song, and H. Ko, "Neighbor-aware shared container instance warming framework for serverless edge computing," *Future Gener. Comput. Syst.*, vol. 174, pp. 1–12, 2026.

[8] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and M. B. Chhetri, "Faashouse: Sustainable serverless edge computing through energy-aware resource scheduling," *IEEE Trans. Serv. Comput.*, vol. 17, no. 4, pp. 1533–1547, Jul./Aug. 2024.

[9] K. Cao and J. Weng, "REPFS: Reliability-ensured personalized function scheduling in sustainable serverless edge computing," *IEEE Trans. Sustain. Comput.*, vol. 9, no. 3, pp. 494–511, May/Jun. 2024.

[10] F. Tütüncüoğlu and G. Dán, "Joint resource management and pricing for task offloading in serverless edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7438–7452, Jun. 2024.

[11] S. Hu, Z. Qu, B. Tang, B. Ye, G. Li, and W. Shi, "Joint service request scheduling and container retention in serverless edge computing for vehicle-infrastructure collaboration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 6508–6521, Jun. 2024.

[12] F. Liu and Y. Niu, "Demystifying the cost of serverless computing: Towards a win-win deal," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 1, pp. 59–72, Jan. 2024.
[13] R. Xie, D. Gu, Q. Tang, T. Huang, and F. R. Yu, "Workflow scheduling in serverless edge computing for the industrial Internet of Things: A learning approach," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 8242–8252, Jul. 2023.
[14] Y. Yang et al., "Multi-objective deep reinforcement learning for function offloading in serverless edge computing," *IEEE Trans. Serv. Comput.*, vol. 18, no. 1, pp. 288–301, Jan./Feb. 2025.
[15] A. Lakhan et al., "Restricted Boltzmann machine assisted secure serverless edge system for Internet of medical things," *IEEE J. Biomed. Health Informat.*, vol. 27, no. 2, pp. 673–683, Feb. 2023.
[16] S. G. Kulkarni, G. Liu, K. K. Ramakrishnan, and T. Wood, "Living on the edge: Serverless computing and the cost of failure resiliency," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw.*, 2019, pp. 1–6.
[17] V. Sreekanti et al., "A fault-tolerance shim for serverless computing," in *Proc. 15th Eur. Conf. Comput. Syst.*, 2020, pp. 1–15.
[18] K. Cao, M. Chen, S. Karnouskos, and S. Hu, "Reliability-aware personalized deployment of approximate computation IoT applications in serverless mobile edge computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 44, no. 2, pp. 430–443, Feb. 2025.
[19] K. Cao, L. Li, Y. Cui, T. Wei, and S. Hu, "Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing," *IEEE Trans. Ind. Informat.*, vol. 17, no. 1, pp. 494–503, Jan. 2021.
[20] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C. Hsu, "Edge server placement in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 160–168, 2019.
[21] C.-S Yang, R. Pedarsani, and A. S. Avestimehr, "Communication-aware scheduling of serial tasks for dispersed computing," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1330–1343, Aug. 2019.
[22] Alibaba Group, 2023, "Alibaba cluster trace program," Accessed: Sep. 23, 2025. [Online]. Available: https://github.com/alibaba/clusterdata
[23] L. Li et al., "Game theoretic feedback control for reliability enhancement of EtherCAT-based networked systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 9, pp. 1599–1610, Sep. 2019.
[24] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 813–825, Mar. 2017.
[25] C. Moser, L. Thiele, D. Brunelli, and L. Benini, "Adaptive power management for environmentally powered systems," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 478–491, Apr. 2010.
[26] K. Cao, J. Zhou, G. Xu, T. Wei, and S. Hu, "Exploring renewable-adaptive computation offloading for hierarchical QoS optimization in fog computing," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2095–2108, Oct. 2020.
[27] R.V. Prasad, V. S. Rao, C. Sarkar, and I. Niemegeers, "ReNEW: A practical module for reliable routing in networks of energy-harvesting wireless sensors," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 3, pp. 1558–1569, Sep. 2021.
[28] R. Yang, K. Cao, P. Cong, J. Zhou, M. Chen, and T. Wei, "Personality-aware VNF deployment for profit maximization," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl.*, 2019, pp. 380–387.
[29] K. Yan, X. Zhang, J. Tan, and X. Fu, "Redefining QoS and customizing the power management policy to satisfy individual mobile users," in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.
[30] K. S. Sidhu and R. Khazaka, "Nested latin hypercube-based sampling for efficient uncertainty quantification using sensitivity-assisted least squares SVM," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 15, no. 1, pp. 75–84, Jan. 2025.
[31] Y. Wen, Z. Wang, and M. F. P. O'Boyle, "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms," in *Proc. IEEE Int. Conf. High Perform. Comput.*, 2014, pp. 1–10.
[32] "The LLVM compiler infrastructure," 2025, Accessed: Aug. 10, 2025. [Online]. Available: https://llvm.org/
[33] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–8.
[34] C. -S. Yang, C. -S. Huang-Fu, and I. -K. Fu, "Carbon-neutralized task scheduling for green computing networks," in *Proc. IEEE Glob. Commun. Conf.*, 2022, pp. 4824–4829.
[35] K. Cao et al., "Affinity-driven modeling and scheduling for makespan optimization in heterogeneous multiprocessor systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 7, pp. 1189–1202, Jul. 2019.
[36] Huawei Technologies Company, Ltd., "Atlas 500 pro-3000 white paper," 2025, Accessed: Aug. 10, 2025. [Online]. Available: https://e.huawei.com/cn/material/server/a53a85e3869746d1a6dfb397d39c135e
[37] Huawei TaiShan, "Taishan 2280e edge servers," 2025, Accessed: Aug. 13, 2025. [Online]. Available: https://e.huawei.com/cn/products/computing/kunpeng/taishan/taishan-2280e
[38] ASUSTeK Computer Inc., "Asustek edge servers," 2025, Accessed: Aug. 13, 2025. [Online]. Available: https://servers.asus.com.cn/products/servers/edge-servers
[39] Dell Technologies Inc., "Dell EMC poweredge edge servers," 2025, Accessed: Aug. 13, 2025. [Online]. Available: https://www.dell.com/zh-cn/shop/storage-servers-and-networking-for-business/sf/poweredge-edge-servers

Kun Cao (Member, IEEE) received the PhD degree in computer science and technology from East China Normal University, Shanghai, China, in 2020. He is currently an associate professor with the College of Cyber Security, Jinan University, Guangzhou, China. His research interests include edge computing and cyber-physical systems security. He was the recipient of Young Professional Award from the IEEE Technical Committee on Secure and Dependable Measurement in 2022.

Chaohong Tan received the BS degree in computer software and theory from the Huazhong University of Technology, in 1986. He is currently the deputy director of the Guangxi Key Laboratory of Digital Infrastructure. His research interests include software engineering and system integration.

Yangguang Cui (Member, IEEE) received the PhD degree in computer science from East China Normal University, Shanghai, China, in 2023. He is currently an assistant professor with the School of Computer Engineering and Science, Shanghai University, Shanghai. His research interests include federated learning and Internet of Things. He has been serving as an associate editor for *Journal of Circuits, Systems, and Computers*.

Keqin Li (Fellow, IEEE) is currently a SUNY distinguished professor of computer science with the State University of New York. His research interests include parallel computing and high-performance computing, distributed computing, heterogeneous computing systems, cloud computing, CPU-GPU hybrid and cooperative computing, multicore computing, mobile computing, service computing, and cyber-physical systems. He is an AAAS fellow, AAIA fellow, and ACIS founding fellow.