Scilight

Journal of Machine Learning and Information Security
https://www.sciltp.com/journals/jmlis

Article

CERL: Evolutionary Reinforcement Learning for Partitioned Collaborative Inference on On-Device Models

Lin Tan [1], Songtao Guo [1,*], Pengzhan Zhou [1] and Zhufang Kuang [2]

[1] College of Computer Science, Chongqing University, Chongqing 400044, China;
[2] College of Computer Science and Mathematics, Central South University of Forestry and Technology, Changsha 410004, China
* Correspondence: guosongtao@cqu.edu.cn

How To Cite: Tan, L.; Guo, S.; Zhou, P.; et al. CERL: Evolutionary Reinforcement Learning for Partitioned Collaborative Inference on On-Device Models. *Journal of Machine Learning and Information Security* **2025**, *1*(1), 5.

Received: 24 August 2025
Revised: 21 October 2025
Accepted: 24 October 2025
Published: 29 October 2025

Abstract: With the proliferation of intelligent mobile applications, the ability to deploy and operate Deep Neural Networks (DNNs) on mobile Edge Devices (EDs) has become fundamentally important. The primary challenge, however, stems from the constrained computational power of EDs, which often leads to excessive energy use and degraded inference accuracy. To mitigate these issues, we introduce a Distributed Collaborative Inference (DCI) system designed to lower on-device inference costs. Our system achieves this by distributing the inference workload across a network of multiple EDs and Mobile Edge Computing (MEC) servers. To dynamically model the complex relationships within the system and make optimal decisions, we develop a Evolutionary Reinforcement Learning algorithm based on the Cross-Entropy Method (CEM). This algorithm uniquely employs negative temporal difference (TD) error as a fitness metric to identify and select elite individuals from the population. These elite solutions generate high-quality samples that accelerate the learning process, enabling the system to determine optimal decisions for distributed collaborative inference and resource management within complex, dynamic search spaces. Our extensive simulation results confirm that this approach markedly surpasses existing methods and benchmark standards, yielding a 57.5% increase in the successful completion rate of inference tasks and a 65.7% reduction in total system costs.

Keywords: Mobile Edge Computing; distributed collaborative inference; resource allocation

1. Introduction

The evolution of intelligent mobile applications, such as real-time voice assistants, augmented reality (AR), and object detection, is fundamentally driven by Deep Neural Networks (DNNs) [1]. While these applications have historically relied on the vast computational power of cloud computing [2], a significant trend has emerged toward deploying models directly on edge devices (EDs) to reduce network latency and mitigate data communication concerns [3]. This on-device approach processes tasks closer to the source of data generation.

However, this shift presents a critical bottleneck: EDs possess limited computational capacity compared to centralized cloud servers. Executing complex or large DNN models directly on these devices often results in prohibitive energy consumption and compromised inference quality [4]. To address this, DNN partitioning has emerged as a strategic solution. By splitting a DNN into smaller segments, the inference workload can be offloaded and distributed, alleviating the burden on a single device [5]. This approach builds upon the foundations of Mobile Edge Computing (MEC), a paradigm widely studied for task offloading and resource management at the network edge [6,7]. Early research in DNN partitioning focused on simple binary splits, where parts of a model are computed locally on the ED while the rest are offloaded to a cloud or MEC server [8–11].

While these initial approaches demonstrated the potential of distributed inference, they often overlook a crucial opportunity: the vast, underutilized computational power within the edge network itself. Many EDs in an environment may be engaged in light tasks or remain completely idle for extended periods [12]. This untapped resource pool presents

a promising avenue for creating a more efficient, decentralized inference paradigm. Existing methods, which are typically restricted to simple device-to-server collaboration models, fail to capitalize on this potential, leading to resource wastage and processing bottlenecks on the primary devices and servers [13–15].

This paper proposes a more advanced multi-node Distributed collaborative inference framework that leverages these idle resources to distribute DNN workloads dynamically. Consider a smart transportation application where multiple smart cameras monitor urban traffic [16]. Individually, these cameras may lack the power for complex analysis. By collaborating with other nearby EDs—such as idle smart vehicles in a parking lot or traffic light controllers—they can collectively analyze vehicle congestion and predict traffic trends rapidly, without relying on high-latency public networks to reach a distant server. Beyond intelligent transportation, our framework could be highly beneficial in scenarios such as Industrial IoT (IIoT) for real-time quality control in smart factories and remote healthcare monitoring, where data from various wearable sensors is collaboratively processed for timely health alerts.

While this collaborative paradigm offers significant flexibility, it also introduces fundamental challenges. First, distributing a DNN across numerous heterogeneous nodes—the source ED, multiple collaborative EDs, and the MEC server—dramatically expands the search space for optimal partitioning decisions. This creates a complex combinatorial problem due to strict layer dependencies and potential communication overhead [17–19]. Second, coordinating this collaboration requires sophisticated dynamic resource orchestration. The system must constantly balance performance gains against resource costs for each node, adapting to fluctuating network conditions and resource availability to maintain stable and fair collaboration [20]. Third, effectively implementing such a system requires a technique that can accurately represent the dynamic, intricate relationships between nodes, inference tasks, and resources and efficiently navigate the vast search space to find optimal solutions.

To overcome these challenges, we introduce a novel Evolutionary Reinforcement Learning based method. Our approach harnesses an adaptive evolutionary reinforcement learning strategy, which uses the Cross-Entropy Method (CEM), to efficiently explore the solution space and optimize co-inference decisions and resource allocation in highly dynamic and complex environments.

The primary contributions of this paper can be summarized as follows:

- We design and implement a multi-node Distributed Collaborative Inference (DCI) system that enables partitioned model inference across a distributed network of computing nodes. This system is designed to dynamically manage the complex, evolving relationships between nodes, tasks, and resources, allowing for real-time adaptation to changes in network topology and user mobility.
- We propose a novel CEM-based Evolutionary Reinforcement Learning (CERL) algorithm to navigate the vast and dynamic search space inherent in our DCI system. This method utilizes negative temporal difference (TD) error as a fitness metric to identify elite solutions within a population of networks. By allowing these elite individuals to guide the learning process, our algorithm generates high-quality training samples, leading to more efficient and effective optimization of partitioning decisions and resource allocation.
- We conduct extensive simulations that demonstrate the superiority of our proposed method. The results show that our approach achieves a 57.5% improvement in the inference task completion rate and a 65.7% reduction in overall system cost when compared to existing methods (e.g., D^2QN, DDQN) and other benchmark schemes.

The remainder of the paper is organized as follows. We present the system model and the problem formulated in Section 2. Section 3 outlines the settings for reinforcement learning. Section 4 describes the details of our proposed optimization framework. The experimental results are analyzed in Section 5. Finally, Section 6 concludes the paper.

2. System Model and Problem Formulation

In this section, we present the system model, analyze the energy consumption and time required for the co-inference task, and then formulate the optimization problem. The notations used throughout this paper are defined in Table 1.

2.1. System Model

2.1.1. System Overview

As illustrated in Figure 1, our proposed system facilitates collaborative DNN inference by distributing the computational workload across a network of heterogeneous computing nodes. When an ED (the source node) initiates an inference task, instead of processing the entire DNN model locally, the system partitions the model's layers for execution on multiple nodes. For example, the source ED might process the initial layers, offload

intermediate layers to nearby idle EDs for computation, and have the final layers executed on the MEC server. This distributed execution pattern significantly alleviates the computational burden on any single device. Moreover, it provides essential computational support for EDs that may lack a direct connection to the MEC server by enabling collaboration with neighboring devices. The final inference result is obtained after all layer segments are processed sequentially through this collaborative chain.

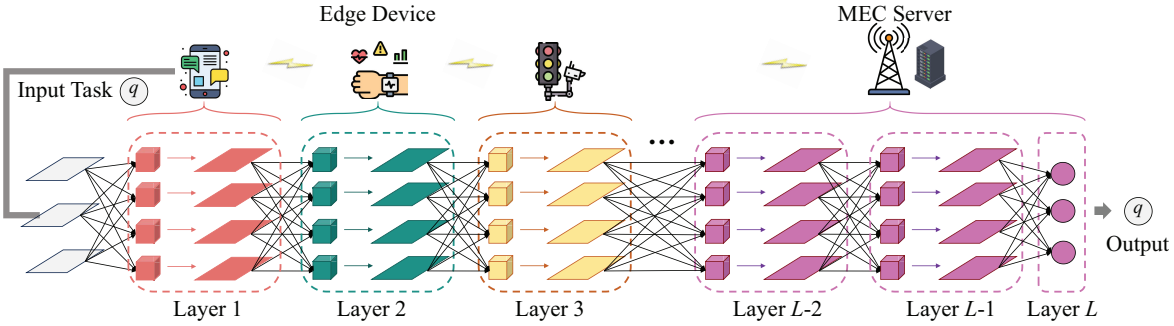

Figure 1. The architecture of the proposed distributed collaborative inference system.

Table 1. Summary of Important Notations.

Symbol	Description
Sets and Indices	
$\mathcal{N}$	Set of all computing nodes (MEC server indexed as 0).
$\mathcal{L}$	Set of DNN layers, $\lvert\mathcal{L}\rvert = L$.
$\mathcal{Q}$	Set of inference tasks, $\lvert\mathcal{Q}\rvert = Q$.
$\mathcal{T}$	Set of time slots, $\lvert\mathcal{T}\rvert = T$.
$\mathcal{S}$	Set of available subcarriers, $\lvert\mathcal{S}\rvert = S$.
$n, m \in \mathcal{N}$	Indices for computing nodes.
$q, p \in \mathcal{Q}$	Indices for inference tasks.
$t \in \mathcal{T}$	Index for a time slot.
System Parameters	
C_q^i	Computational workload of layer i for task q.
O_q^i	Output data size of layer i for task q.
$R_n^{\max}$	Maximum computational capacity of node n.
B	Total communication bandwidth.
p	Allocated transmission power.
$g[\cdot]$	Channel gain of the allocated subcarrier.
δ^2	Gaussian noise power.
k_0	Computational efficiency constant.
ω_e, ω_d	Weight coefficients for energy and delay cost.
Decision Variables	
d_q^t	Node assigned to process task q at time t.
$u[d_q^t]$	Layer block (e.g., $[i:j]$) assigned to node d_q^t.
$r\left[u[d_q^t]\right]$	Computational resources allocated to the assigned layer block.
$w[d_q^t, d_q^{t+1}]$	Subcarrier allocated for transmission between nodes.
Performance Metrics	
T_q^{Comp}	Computation delay for task q.
T_q^{Trans}	Transmission delay for task q.
E_q^{Comp}	Computational energy consumption for task q.
E_q^{Trans}	Transmission energy consumption for task q.
Reinforcement Learning (CERL)	
$\mathcal{S}$	System state, represented as a graph.
$a(t)$	Action taken by the agent at time t.
$\mathcal{R}(t)$	Reward received at time t.
$\mathbf{X}, \mathbf{A}$	GCN node feature matrix and adjacency matrix.
Q_θ^{GCN}	Prediction Q-network with parameters θ.
$Q_{\theta'}^{\text{GCN}}$	Target Q-network with parameters θ'.
μ, Σ	Mean vector and covariance matrix for CEM.
$f(\cdot)$	Fitness function (negative TD error).
γ	Discount factor.
$\mathcal{D}$	Experience replay buffer.

2.1.2. System Components and Notation

The system is composed of one MEC server and N EDs. We define the set of all computing nodes as $\mathcal{N} = \{0, 1, 2, ..., n, ..., N\}$, where the index 0 represents the MEC server. We consider an on-device DNN model with L layers, denoted by $\mathcal{L} = \{1, 2, \ldots, l, \ldots, L\}$, which is distributed for distributed execution [9]. The set of all inference tasks is $\mathcal{Q} = \{1, 2, \ldots, q, \ldots, Q\}$, and their arrival is modeled as a Poisson process. Each task must be completed within a maximum delay tolerance of T, which is divided into discrete time slots $\mathcal{T} = \{1, 2, \ldots, t, \ldots, T\}$. In any time slot t, if inference task q is still in progress, it is assigned to a computing node n, which we denote as $d_q^t = n$. The node assigned for the subsequent computation step is denoted by $d_q^{t+1} = m$. If $n = m$, the task continues processing on the same node. The specific block of layers from i to j assigned to node n for task q is determined by the decision variable $u[d_q^t] = [i : j]$, where $i, j \in \mathcal{L}$. Each layer is characterized by its computational workload and the size of its output data. The output of layer i serves as the input for layer $(i + 1)$. For task q, the output data size and computational workload of layer i are denoted as O_q^i and C_q^i, respectively. Therefore, the total computational workload for the layer block assigned to node n is $C\big[u[d_q^t]\big] = \sum_{x=i}^{j} C_q^x$, and the resulting output data size is determined by the final layer in the block, i.e., $O\big[u[d_q^t]\big] = O_q^j$. We denote the computational resources allocated by node n to compute this block of layers for task q as $r\big[u[d_q^t]\big]$. For communication resource management, we adopt Orthogonal Frequency Division Multiple Access (OFDMA) to mitigate inter-device channel interference. The total bandwidth B is divided into S available subcarriers, denoted by the set $\mathcal{S} = \{1, 2, \ldots, s, \ldots S\}$. When the computation for task q moves from node d_q^t to a different collaborative node d_q^{t+1} (i.e., $d_q^t \neq d_q^{t+1}$), a specific subcarrier s is allocated for the transmission of intermediate data. This allocation is denoted as $w[d_q^t, d_q^{t+1}] = s$. In summary, the optimization variables for the system consist of the distributed co-inference decisions (d, u), the computational resource allocation r, and the communication resource allocation w.

Then, we present the co-inference delay model and the co-inference energy consumption model to comprehensively model the performance of the proposed system.

2.1.3. Co-Inference Delay Model

The co-inference task delay model consists of two parts. We first introduce the computation delay.

To facilitate calculations, we abstract the CPU, GPU, and NPU as a unified computational resource. The kernel scheduling for tasks across diverse computational architectures is highly complex, and we will investigate the scheduling problem for heterogeneous architectures in future work [21]. Therefore, the computation delay of task q in slot t is defined as

$$T_q^{\text{Comp}} = \frac{C\big[u[d_q^t]\big]}{r\big[u[d_q^t]\big]}. \tag{1}$$

Secondly, the communication delay is given as follows.

The transmission rate from n to collaborative node m in slot t is given by

$$R^{Trans}[d_q^t, d_q^{t+1}] = \frac{B}{S} \cdot \log_2(1 + \frac{g\big[w[d_q^t, d_q^{t+1}]\big]p}{\delta^2}), \tag{2}$$

where $g\big[w[d_q^t, d_q^{t+1}]\big]$ represents the channel gain of the allocated subcarrier s, p denotes the allocated transmission power, and δ^2 is the Gaussian noise power.

Furthermore, the transmission delay of task q in slot t is given by

$$T_q^{Trans} = \frac{O\big[u[d_q^t]\big]}{R^{Trans}[d_q^t, d_q^{t+1}]}. \tag{3}$$

2.1.4. Co-Inference Energy Consumption Model

We first present the computational energy consumption. The energy consumption of task q in slot t is given by [9]

$$E_q^{Comp} = k_0 \cdot C\big[u[d_q^t]\big] \cdot r\big[u[d_q^t]\big]^2, \tag{4}$$

where $k_0 > 0$ is a computational efficiency constant.

Secondly, the transmission energy consumption of task q in slot t is given by

$$E_q^{Trans} = p \cdot \frac{O[u[d_q^t]]}{R^{Trans}[d_q^t, d_q^{t+1}]}. \tag{5}$$

2.2. Problem Formulation

We introduce four $Q \times T$ matrices $\{(\mathbf{D}, \mathbf{U}), \mathbf{R}, \mathbf{W}\}$ to represent the distributed co-inference decision, as well as the allocation of computational resources and subcarriers across all tasks and slots, respectively. The elements in the q -th row and the t -th column of these matrices correspond to the optimization variables for the task q during the slot t. The system cost is defined as the weighted sum of energy consumption and delay, and the goal is to minimize the system cost for accomplishing all inference tasks, which can be formulated as:

$$\mathbf{P1}: \min_{\{(\mathbf{D},\mathbf{U}),\mathbf{R},\mathbf{W}\}} \sum_{q=1}^{Q} \left[\omega_e \sum_{t=1}^{T} (E_q^{Comp} + E_q^{Trans}) + \omega_d \sum_{t=1}^{T} (T_q^{Comp} + T_q^{Trans})\right] \tag{6}$$

Subject to

$$w[d_q^t, d_q^{t+1}] \neq w[d_p^t, d_p^{t+1}], \forall q, p \in \mathcal{Q}, q \neq p, t \in \mathcal{T}, \tag{7}$$

$$\sum_{q \in \mathcal{Q}} r\left[u[d_q^t]\right] \leq R_n^{\max}, \forall n \in \mathcal{N}, t \in \mathcal{T}, \tag{8}$$

$$\sum_{t=1}^{T} u[d_q^t] = L, \forall q \in \mathcal{Q}, \tag{9}$$

$$\sum_{t=1}^{T} (T_q^{\text{Comp}} + T_q^{\text{Trans}}) \leq T, \forall q \in \mathcal{Q}, \tag{10}$$

where constraint (7) enforces that each subcarrier can be allocated to only one inference task for transmission in each slot; (8) ensures that the total allocated computational resources at each node do not exceed its capacity; (9) ensures that all layers of each task are processed; (10) imposes a time constraint for each task, ensuring that computation and transmission complete within the specified maximum tolerance time. ω_e and ω_d are the weight coefficients that represent the relative importance of energy consumption and latency in the system. By adjusting these values, the optimization can focus more on minimizing energy consumption or reducing delay, depending on the specific requirements of the system or application.

The proposed problem presents several fundamental challenges that make it particularly difficult to solve. First, the optimization variables are highly coupled, as the distributed co-inference decisions directly affect both computational and communication resource allocation, rendering it a mixed integer nonlinear programming (MINLP) problem, which is NP-hard. Second, the problem is inherently time-sequential, requiring real-time decisions for task partitioning and resource allocation, where each decision influences subsequent system states. This sequential nature is further complicated by the dynamic characteristics of edge environments: inference tasks arrive with varying characteristics, computational resources fluctuate over time, and network conditions change continuously. Moreover, these dynamics require the system to balance immediate performance with future task requirements, as inappropriate resource allocation or task distribution can create bottlenecks that affect system performance across multiple time slots.

To address these challenges, traditional optimization methods prove insufficient as they primarily focus on single time slot scenarios and struggle with sequential decision making over extended periods. RL, in contrast, is particularly well suited for our problem due to several key characteristics. First, the sequential nature of decisions aligns with RL's capability to learn from continuous interaction with the environment. Second, the uncertainty in task arrivals and resource availability can be effectively handled through RL's exploration mechanisms. Third, RL's reward framework naturally accommodates our objective of minimizing system cost while maintaining task completion requirements. This alignment enables our system to make effective decisions at each time slot, continuously adapting to changes in task arrivals and resource availability to minimize latency and energy consumption.

Therefore, we propose to solve this problem using reinforcement learning, specifically developing a novel CEM-based Evolutionary Reinforcement Learning method. In the following sections, we first present how to model our problem in the RL framework and then detail our proposed methodology that combines GCN's representation

power with evolutionary strategies to effectively solve the formulated problem.

3. Reinforcement Learning Setting

To leverage RL for our optimization problem, we first represent the target problem as a Markov Decision Process (MDP). The initial step involves designing a comprehensive reinforcement learning (RL) environment that consists of three most critical components: state, action, and reward. In the subsequent sections, we will dive into these components and elaborate on them.

3.1. Graph Convolutional Network for System State

The complexity of our system lies in the intricate relationships between co-inference tasks, resources, and computing nodes. Traditional state representations using vectors or matrices fail to capture these complex interdependencies. Therefore, we adopt a graph-based approach to construct the system state representation. Specifically, we describe the system state as a graph $\mathcal{S} = (\mathcal{V}_\alpha, \mathcal{V}_\beta, \mathcal{E}_\alpha, \mathcal{E}_\beta, \mathcal{E}_{\alpha-\beta})$. The components are defined as follows:

- $\mathcal{V}$ (Node Features): There are two sets of nodes α and β, and $\mathcal{V}_\alpha$ $\mathcal{V}_\beta$ representing the computational node layers and co-inference task layers, respectively. Specifically, $\mathcal{V}_\alpha = \{v_0, v_1, \ldots, v_N\}$ represents the computational nodes, where each node is associated with a feature vector that includes computational capacity and available resources, etc. $\mathcal{V}_\beta = \{l_1^q, l_2^q, \ldots, l_L^q\}$ represents the model layers of all co-inference tasks, where each layer is associated with a feature vector that includes computational workload, output data size, execution status, etc. We denote the feature matrix $\mathbf{X}$ as a $|\mathcal{V}| \times d$ matrix, where each row represents the feature vector of a node, with d being the number of input features per node.
- $\mathcal{E}$ (Edge Features): The set of edges consists of three parts, each describing the relationships between the graph nodes. $\mathcal{E}\alpha$ represents the edges within graph layer α between computational nodes, corresponding to the communication link status, where no edge refers to a communication link that is not connected or is faulty; $\mathcal{E}_\beta$ represents the edges within graph layer β, which define the dependencies between co-inference layers, i.e., the order of execution of the inference layers; and $\mathcal{E}_{\alpha-\beta}$ represents the edges between α and β, connecting computational nodes and co-inference task layers, indicating the assignment relationships of the task layer partition. We denote the adjacency matrix $\mathbf{A}$ as representing the graph structure in matrix form, with dimensions $|\mathcal{V}_\alpha| \times |\mathcal{V}_\beta|$, capturing the relationships between the computational node layers and the computational task layers.

The features of the graph evolve over time, reflecting dynamic task arrivals, resource availability, and communication conditions. This graph representation facilitates modeling and solving the co-inference problem by leveraging the topological and relational information embedded in the system. By leveraging a graph-based framework, we can efficiently represent large-scale systems with numerous tasks and devices, enabling effective optimization of co-inference decisions.

Moreover, to fully utilize the relational information in the graph, we employ a Graph Convolutional Network (GCN) as a feature extractor. The GCN processes the system state graph to generate embeddings for each node and edge, encoding their local and global properties. A GCN layer processes this input by taking the tuple $\mathbf{X}, \mathbf{A}$ as input and producing a node-level output $\mathbf{H}$ of dimension $|\mathcal{V}| \times f$ (where f denotes the number of output features per node), thereby approximating a nonlinear function $f(\mathbf{X}, \mathbf{A})$. The layer-wise propagation rule of a GCN is defined as follows:

$$\mathbf{H}^{(l+1)} = f(\mathbf{H}^{(l)}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\psi^{(l)}\right), \tag{11}$$

where $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\mathbf{A}$ with $\hat{\mathbf{D}}ii = \sum_j \hat{\mathbf{A}}ij$, $\psi^{(l)}$ represents the weight matrix of the i-th layer, $\sigma(\cdot)$ is the activation function, and $\mathbf{H}^{(l)}$ is the node feature representation at the l-th layer, with the initial input as $\mathbf{H}^{(0)} = \mathbf{X}$, the node feature matrix. With $\mathbf{X}$ as the input feature and $\mathbf{A}$ as the adjacency matrix, the GCN model generates intermediate latent features while embedding the underlying network topology to assist the RL agent in decision making.

3.2. Action Space

Each slot t, agent performs action $a(t)$ under the constraints of the target problem, which consists of three parts: distributed co-inference decision, computational resource allocation, and subcarrier allocation.

The distributed co-inference decision consists of two vectors: $\mathbf{D}(t) = [d_1^t, ..., d_q^t, ..., d_Q^t]$ indicating which computing node is assigned to each task and $\mathbf{U}(t) = [u[d_1^t], ..., u[d_q^t], ..., u[d_Q^t]]$ determining which layers are processed for each task.

The vector $\mathbf{R}(t) = [r[u[d_1^t]], ..., r[u[d_q^t]], ..., r[u[d_Q^t]]]$ represents the allocation of computational resources,

where $r[u[d_q^t]]$ denotes the computational resources allocated to task q in its assigned node. The allocation of computational resources is discretized into percentages ranging from 1% to 100% with an integer interval 1%, resulting in 100 possible discrete values for each task.

The vector $\mathbf{W}(t) = [w[d_1^t, d_1^{t+1}], ..., w[d_q^t, d_q^{t+1}], ..., w[d_Q^t, d_Q^{t+1}]]$ represents the allocation of subcarriers, where $w[d_q^t, d_q^{t+1}]$ indicates the subcarrier allocated for the transmission of task q between its current and next computing nodes.

Consequently, by combining these vectors, the action of the system in time slot t is defined as:

$$a(t) = \begin{bmatrix} \mathbf{D}(t), \mathbf{U}(t) \\ \mathbf{R}(t) \\ \mathbf{W}(t) \end{bmatrix}. \tag{12}$$

The dimension of the action space is $[(N+1) \times L \times 100 \times S]$. Since our action space consists of discrete values, it is well suited for value-based RL. Value-based RL is a method in which decisions are made by evaluating the value of each state or state-action pair. The core idea is to learn the expected return (value) for each state, which then guides the agent in selecting the optimal action policy.

3.3. System Reward

The reward function, denoted as $\mathcal{R}$, is designed to evaluate the efficiency of the action $a(t)$ selected in state $s(t)$. After executing the selected action, the total system cost is derived based on Equation (6). In RL, the goal is to maximize the cumulative reward. Therefore, the reward function is defined as the negative value of the system cost, with the aim of minimizing the overall cost of the system.

Moreover, considering that the impact of task failure can significantly outweigh the effects of energy consumption or delay, a penalty term $\mathcal{P}(t)$ is introduced to account for the severity of task failure. Specifically, $\mathcal{P}(t)$ is a constant, relatively large compared to the system cost, that amplifies the impact of task failure, ensuring that task failure is prioritized in the optimization process.

Thus, the reward function is formulated as:

$$\mathcal{R}(t) = -\omega_e \left(E_q^{\text{Comp}}(t) + E_q^{\text{Trans}}(t)\right) - \omega_d \left(T_q^{\text{Comp}}(t) + T_q^{\text{Trans}}(t)\right) - \mathcal{P}(t), \tag{13}$$

This design not only considers the trade-off between energy consumption and latency but also incorporates a penalty for task failure, enhancing robustness and accelerating the convergence to a reliable strategy during RL training. The weights and penalty values can be adjusted according to the specific characteristics of the task to adapt to different scenarios.

However, directly applying standard RL methods to our problem is challenging because of its discrete nature and complex optimization space. Therefore, in the next section, we propose an evolutionary RL framework that integrates evolutionary strategies with value-based RL.

4. Methodology

In this section, we introduce the CERL optimization framework in detail. Existing studies have shown that the integration of evolutionary algorithms with RL for policy search can enhance the performance of RL. However, these studies are mainly focused on combining EAs with policy-based RL methods, which are typically designed to address optimization problems involving continuous variables [22]. In contrast, since most of our optimization variables are discrete, existing evolutionary RL methods face challenges in adaptation. Therefore, this paper introduces an evolutionary mechanism based on CEM in value-based RL (Deep Q-Network, DQN) to obtain discrete optimization solutions. We selected DQN due to its fast training speed, thereby enabling the efficient acquisition of discrete optimization solutions under the evolutionary mechanism.

Specifically, CERL maintains a population of value functions rather than policies, consisting of GCN-based Q_{GCN} networks and their corresponding target networks. We adopt the negative TD error as a new fitness metric inspired by [23], which quantifies the discrepancy between the estimated values of the Q_{GCN} networks and their target values. CERL allows elites in the population to interact with the environment, providing high-quality samples for optimization of RL, while the RL value functions participate in the evolution of the population in each generation. The structure of the proposed CERL is shown in Figure 2. Specifically, CERL consists of four main stages: CEM Optimization, Elite Interaction, GCN-Based DQN Optimization, and RL Injection. These stages will be introduced in the following subsections.

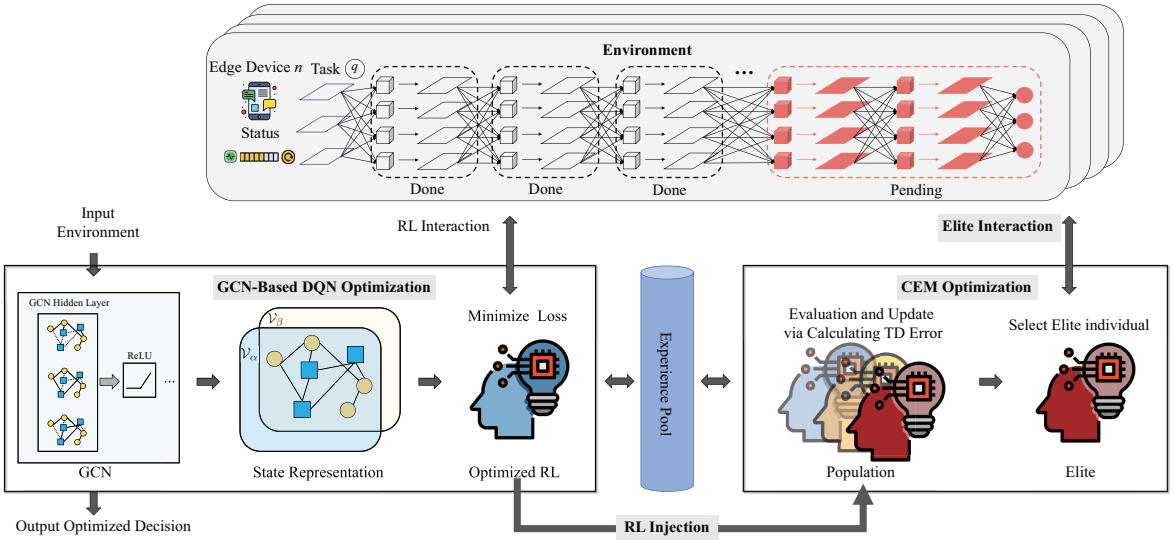

Figure 2. The structure of the proposed CERL.

4.1. Cross-Entropy Method Optimization

CEM [24] is an evolutionary strategy inspired by the distribution estimation algorithm [25]. In CEM optimization, the population is represented using a Gaussian distribution parameterized by a mean vector μ and a covariance matrix Σ. This distribution evolves iteratively, refining μ and Σ toward the optimal solution by focusing on the best performing individuals. The details are elaborated as follows:

4.1.1. Population Representation and Sampling

In CERL, CEM evolves a population of Q^{GCN}-networks and their corresponding target networks $\mathbb{P} = \{Q^{\text{GCN}}_{\theta_i}, Q^{\text{GCN}}_{\theta'_i}\}^k_{i=1}$. In each generation, individuals in the population are sampled as $\{Q^{\text{GCN}}_{\theta_i}, Q^{\text{GCN}}_{\theta'_i}\} \sim \mathcal{N}(\mu, \Sigma)$. To ensure the influence of previous learning, the optimized Q-network Q^{GCN}_{θ} and its corresponding target network $Q^{\text{GCN}}_{\theta'}$ of agent are periodically injected into the population, which will be introduced in the fourth stage.

4.1.2. Fitness Evaluation

To assess the performance of the individuals, the fitness values are calculated for all individuals in the population. For CERL, fitness is evaluated using the negative TD error. To measure the discrepancy between predicted Q-values $Q^{\text{GCN}}_{\theta_{\text{RL}}}$ and target Q-values $Q^{\text{GCN}}_{\theta'_{\text{RL}}}$, the equation is given by

$$f(\theta_i, \theta'_i) = -\mathbb{E}\left[\left(r + \gamma \max_{a'} Q^{\text{GCN}}_{\theta'_i}(s', a') - Q^{\text{GCN}}_{\theta_i}(s, a)\right)^2\right]. \tag{14}$$

where (s, a, r, s') denotes the state, action, reward and next state, respectively, and γ is the discount factor. A smaller TD error corresponds to a more accurate value function approximation. To efficiently calculate fitness, a batch of size J is sampled from the experience pool $\mathcal{D}$. This sampling strategy ensures a diverse coverage of state action transitions, enabling a reliable fitness assessment for all individuals in the population. As $f(\theta_i, \theta'_i)$ increases, the accuracy of the Q-value approximation improves.

4.1.3. Elite Selection

The top-performing individuals, referred to as the elites, are identified according to their fitness values. Specifically, the individuals with the highest fitness values, $\{z_1, z_2, \ldots, z_{k/2}\}$, are selected as the elite subset. The number of elite individuals can be adjusted as needed. These elites serve as the basis for refining the population distribution in the next generation.

4.1.4. Distribution Update

The distribution parameters μ and Σ are updated using elite individuals to guide the search process to regions of higher fitness. The updates are computed as follows:

$$\begin{aligned} \mu_{new} &= \sum_{i=1}^{|\mathbb{P}|/2} \lambda_i z_i; \\ \Sigma_{new} &= \sum_{i=1}^{|\mathbb{P}|/2} \lambda_i \left(z_i - \mu_{old}\right)\left(z_i - \mu_{old}\right)^T + \epsilon\mathcal{I}, \end{aligned} \tag{15}$$

where λ_i represents the weight assigned to each elite individual, and $\epsilon\mathcal{I}$ is a small positive term added to ensure numerical stability.

As the distribution evolves over generations, μ and Σ adapt to focus on regions of high fitness, allowing the population to converge to optimal solutions. By iteratively refining the population using fitness metrics based on TD error, the framework ensures that Q-value function approximations become increasingly accurate.

4.2. Elite Interaction

In previous Evolutionary Reinforcement Learning works, all individuals in the population were required to interact with the environment to obtain fitness [26]. A key mechanism to improve RL performance in these works was the inclusion of samples generated through Uniform Interaction into the shared replay buffer for RL optimization. This mechanism alleviates the problem of poor exploration in RL, preventing the waste of evaluation samples. However, it also introduces a new issue: Only high-fit individuals generate useful samples for RL optimization, while low-fit samples may be ineffective and potentially lead the RL agent to suboptimal solutions [27].

Based on the novel fitness metric used in our approach, individuals in the population no longer need to interact directly with the environment to obtain fitness. Instead, their fitness is calculated from the samples stored in the replay buffer, allowing for active selection of which individuals interact with the environment to generate samples.

Specifically, we first compute the fitness of all individuals in the population, $\{f(\theta_i, \theta_i')\}_{i=1}^{k}$, using (14). Then, the top N individuals with the highest fitness are selected to interact with the environment. Intuitively, Elite Interaction ensures that the best performers contribute high-quality samples to the replay buffer $\mathcal{D}$, while the poorly performing individuals are prevented from wasting interaction resources, thus avoiding the potential negative impact of low-quality samples on the RL optimization process. The mechanism improves the efficiency of the optimization process, ensuring that the agent only learns from the most valuable experiences while avoiding suboptimal training data.

4.3. GCN-Based DQN Optimization

The GCN-Based DQN Optimization process is critical to refining the RL agent policy using experiences stored in the replay buffer $\mathcal{D}$. This process consists of two main components: interacting with the environment to gather new experiences and performing gradient-based optimization of the Q-network. As described in Section 3.1, to enhance the modeling of relational dependencies in the state of the system, our approach adopts a DQN based on GCN to approximate the Q function.

Specifically, GCN-Based DQN includes a replay buffer $\mathcal{D}$ and two neural networks: a prediction network Q_θ^{GCN} with parameters θ and a target network $Q_{\theta'}^{\text{GCN}}$ with parameters θ'. The replay buffer $\mathcal{D}$ stores observed experiences as tuples $(s(t), a(t), r(t), s(t))$. Based on (11), by aggregating node embeddings, the prediction network computes the Q-values for a given state $s(t)$ and action $a(t)$ expressed as:

$$Q_\theta^{\text{GCN}}(s(t), a(t)) = \mathbf{H}(s(t), a(t)) \cdot \psi, \tag{16}$$

where $\mathbf{H}(s(t), a(t))$ is the node feature representation after processing through the all GCN layers.

The target network computes the target Q-values $Q_{\theta'}^{\text{GCN}}$ for the next state $s(t+1)$, which is given by:

$$Q_{\theta'}^{\text{GCN}}(s(t+1), a(t+1)) = r(t) + \gamma \max_{a(t+1)} Q'_{\theta'}(s(t+1), a(t+1)), \tag{17}$$

where $Q'_{\theta'}(s(t+1), a(t+1))$ approximates the Q-value for action $a(t+1)$ in the future state $s(t+1)$.

The loss function is defined as the mean squared error (MSE) between Q_{θ}^{GCN} and $Q_{\theta'}^{\text{GCN}}$, which is given by

$$\mathcal{L}(\theta) = \mathbb{E}\left[Q_{\theta'}^{\text{GCN}}(s(t+1), a(t+1)) - Q_{\theta}^{\text{GCN}}(s(t), a(t))\right]^2. \tag{18}$$

To minimize the difference between two networks, gradient descent is used to update the parameters θ of the evaluation network:

$$\Delta\theta = \alpha\nabla_{\theta}\mathcal{L}(\theta) = \alpha\mathbb{E}\left[\left(Q_{\theta'}^{\text{GCN}}(s(t+1), a(t+1)) - Q_{\theta}^{\text{GCN}}(s(t), a(t))\right) \cdot \nabla_{\theta} Q_{\theta}^{\text{GCN}}(s(t), a(t))\right]. \tag{19}$$

where α is the learning rate. In particular, the target network parameters are periodically updated by replacing $\theta' \leftarrow \theta$ to stabilize training.

4.4. RL Injection

The RL Injection step integrates the refined Q-network of the RL optimization process into the evolutionary population, ensuring that the progress made by the RL agent directly informs the evolutionary optimization. This fosters synergy between RL and evolutionary algorithms. After optimization of RL, the updated Q network $Q_{\theta_{\text{RL}}}^{\text{GCN}}$ and its corresponding target network $Q_{\theta'_{\text{RL}}}^{\text{GCN}}$ are injected into the population $\mathbb{P}$ each generation. If the RL individual achieves a higher fitness value $f(\theta_{\text{RL}}, \theta'_{\text{RL}})$ and is selected as an elite, it will guide the population and promote its evolution. Otherwise, both $Q_{\theta_{\text{RL}}}^{\text{GCN}}$ and $Q_{\theta'_{\text{RL}}}^{\text{GCN}}$ are injected into the population $\mathbb{P}$ will be eliminated. Moreover, the target network $Q_{\theta'_i}^{\text{GCN}}$ is updated every generation H by performing a hard update using its corresponding Q network $Q_{\theta_i}^{\text{GCN}}$.

This injection allows the population to benefit from the enhanced Q-value estimates of the RL agent, introducing high-quality approximations into the evolutionary process. By incorporating RL-optimized networks, the population converges more efficiently towards optimal solutions.

In summary, the details of the CERL algorithm are presented in Algorithm 1. First, the relevant parameters are initialized (Lines 1–2). During each iteration, the algorithm performs four key stages: (1) CEM Optimization (Lines 3–6), where a new population $\mathbb{P}$ is sampled from a Gaussian distribution parameterized by μ and Σ, and fitness is evaluated to update the distribution; The top individuals with the highest fitness are identified as elites (Line 5), and the distribution parameters μ and Σ are updated using the highest performing half of the population (Line 6). (2) Elite Interaction (Line 7), where the elite individuals interact with the environment and store experiences in the replay buffer $\mathcal{D}$; (3) GCN-Based DQN Optimization (Lines 8–14), where the RL agent interacts with the environment, and updates the Q-network by minimizing the loss function; and (4) RL Injection (Lines 15–17), where the optimized Q-network and target network are periodically injected into the population to stabilize training, and the target network is periodically updated. Through these iterative steps, CERL effectively combines evolutionary optimization and GCN-based RL techniques for robust Q-network training.

Algorithm 1: CERL

Initialize : Replay buffer $\mathcal{D}$, GCN Q-network $Q_{\theta_{\text{RL}}}^{\text{GCN}}$, target network $Q_{\theta'_{\text{RL}}}^{\text{GCN}}$, population $P = \{Q_{\theta_i}, Q_{\theta'_i}\}_{i=1}^{k}$ with size k, covariance matrix $\Sigma = \sigma_{\text{init}} I$

1 **while** *not reached maximum training steps* **do**
2 ① **CEM Optimization: Population Evaluation and Evolution**
3 Draw the current population $\mathbb{P}$ from $\mathcal{N}(\mu, \Sigma)$ to replace the old $\mathbb{P}$ except the injected RL individual.
4 **Population Evaluation:** Sample data with size J from $\mathcal{D}$ to compute fitness $\{f(\theta_i, \theta'_i)\}_{i=1}^{k}$ using Equation (14).
5 **Elite Selection:** Select top individuals with highest fitness as elites.
6 Update μ and Σ with the top half of P using Equation (15).
7 ② **Elite Interaction:** Elites interact with environment and store experiences in $\mathcal{D}$.
8 ③ **GCN-Based DQN Optimization: Interaction and Gradient Optimization**
9 **RL Interaction:** The RL agent interacts with the environment and stores experiences in $\mathcal{D}$.
10 **RL Optimization:** Observe the current system state $s(t)$ obtained through GCN and compute the Q-values based on (16)
11 With probability ϵ, select a random action $a(t)$; otherwise, select $a(t) = \arg\max_a Q_{\theta}^{\text{GCN}}(s(t), a(t))$.
12 Execute action $a(t)$ and observe reward $r(t)$ and next state $s(t+1)$.
13 Store transition $(s(t), a(t), r(t), s(t+1))$ in the pool.
14 **Gradient Optimization:** Sample a random minibatch of transitions $(s(t), a(t), r(t), s(t+1))$ from the pool.
15 Perform gradient descent to optimize the $Q_{\theta_{\text{RL}}}^{\text{GCN}}$ based on (19)
16 ④ **RL Injection:** Inject the optimized $Q_{\theta_{\text{RL}}}^{\text{GCN}}$ and $Q_{\theta'_{\text{RL}}}^{\text{GCN}}$ into the population P.
17 Update $Q_{\theta'_{\text{RL}}}^{\text{GCN}}$ periodically.
18 Update $Q_{\theta'_i}^{\text{GCN}}$ in P with $Q_{\theta_i}^{\text{GCN}}$ every H generations.

After CERL is trained and deployed, it is capable of responding quickly to the dynamic state of the edge

computing environment according to the time slot, including the current arrivals of tasks and the availability of resources for the DCI system. Thus, it reacts quickly to the environment and generates the corresponding partition decisions, computation resource allocation, and communication resource allocation in real-time. Therefore, the optimal solution $\{(\mathbf{D}^*, \mathbf{U}^*), \mathbf{R}^*, \mathbf{W}^*\}$ and the corresponding minimum system cost of the DCI system can be obtained.

4.5. Algorithm Complexity Analysis

The computational complexity is predominantly influenced by the architecture of the GCN and the evolution strategy. For a GCN with $\mathcal{G}$ layers, where each layer contains u_g nodes in the g-th layer, the computational complexity of one forward pass is computed as $\mathcal{O}(Y) = \mathcal{O}(\sum_{g=1}^{G} u_g u_{g+1})$ [28]. For the CEM-based evolution phase with population size k and sample size J, the computational complexity is $\mathcal{O}(kJY)$. The DQN optimization with batch size $\mathcal{B}$ for t iterations requires $\mathcal{O}(Y\mathcal{B}t)$ computations. Therefore, the total computational complexity for training is $\mathcal{O}(kJY + Y\mathcal{B}t)$. For the execution phase, the decision-making process only requires one forward pass of the GCN-based Q-network, resulting in a complexity of $\mathcal{O}(Y)$.

5. Experimental Study

5.1. Experimental Settings

As shown in Figure 3, the device model considered consists of 6 convolutional (Conv) layers, one Transformer (Trans) layer and one fully connected (FC) layer, where $L = 8$. The input and output of each layer are stored in a floating point matrix of size $\{F_q^L\} = \{3 \times 64 \times 64, 4 \times 32 \times 32, 64 \times 16 \times 16, 64 \times 8 \times 8, 64 \times 8 \times 8, 64 \times 4 \times 4, 64 \times 2 \times 2, 64 \times 1 \times 1, 9\}$. Since the storage size of a floating-point number is 64 bits (that is, 8 Bytes), the output data size is in bits for each layer $\{O_q^L\} = \{4 \times 32 \times 32, 64 \times 16 \times 16, 64 \times 8 \times 8, 64 \times 8 \times 8, 64 \times 4 \times 4, 64 \times 2 \times 2, 64 \times 1 \times 1, 9\} \times 64$.

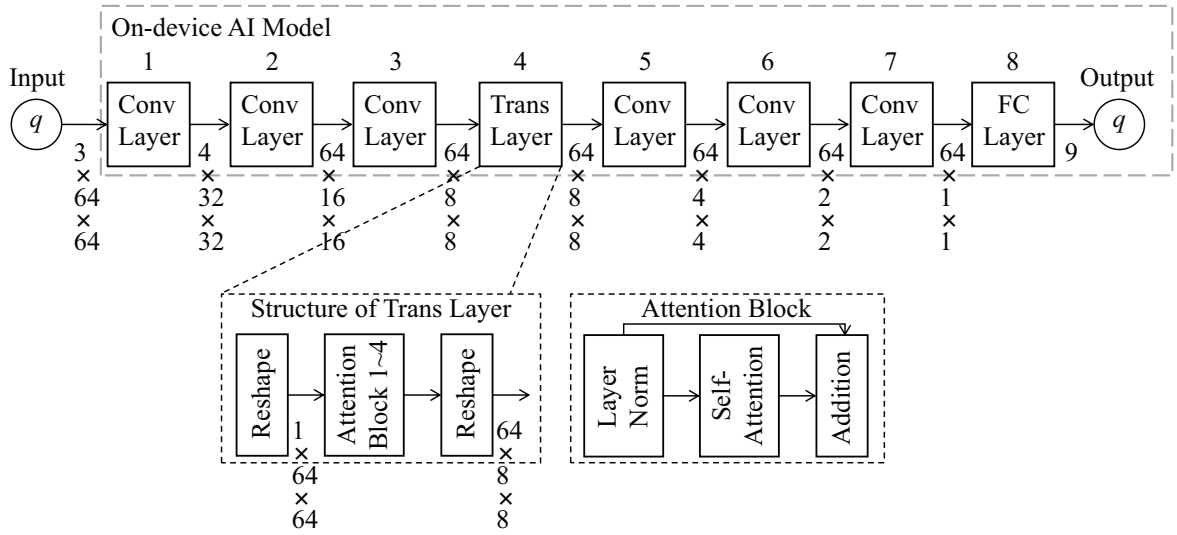

Figure 3. The considered on-device model, where $L = 8$.

In each convolutional layer, the output of the previous layer is processed using Conv2D kernels (4 kernels for the first layer and 64 kernels for the remaining layers), followed by a batch normalization and a ReLu activation. The Transformer layer processes features through 4 attention units, each with 8 heads and a dimension of 32. For the convolutional layer, each convolution kernel processes the current input by performing F_q^{l-1} convolution operations, each of which contains 17 floating-point multiply/add calculations. Let ξ be the average computational resource cycles required per calculation. Then, the computation workload of each convolutional layer can be calculated as $O_q^l = F_q^{l-1} \times 64 \times 17 \times \xi$. The result of the last convolutional layer is processed by a fully connected layer, which involves $(64 + 63) \times 9$ floating-point multiply/add calculations to obtain the final individual output. Therefore, the computing workload of the FC layer is $(64 + 63) \times 9 \times \xi$. The number of multiply-add operations for the transformer layer is quantified as 45.4164×10^6 using the calflops tool [29] in Python. Then, the computation workload for each layer is $\{C_q^L\} = \{3 \times 64 \times 64 \times 4 \times 17, 4 \times 32 \times 32 \times 64 \times 17, 64 \times 16 \times 16 \times 64 \times 17, 45.4164 \times 10^6, 64 \times 8 \times 8 \times 64 \times 17, 64 \times 4 \times 4 \times 64 \times 17, 64 \times 2 \times 2 \times 64 \times 17, 127 \times 9\} \times \xi$. As reported in [30], a typical modern CPU with an on-chip FPU requires 7 and 4 CPU cycles for floating-point multiply and add operations, respectively. Therefore, to account for the computational workload of other operations, such as ReLU activation and

max-pooling, we set a slightly larger value of $\xi = 10$.

The GCN-based Q-network consists of 3 graph convolutional layers, with hidden dimensions of 64, 128, and 256 respectively. Each layer is followed by ReLU activation and batch normalization. For CEM optimization, population size $k = 50$ and select the top as elite, sample size $J = 64$, $H = 1$. We perform experiments with varying numbers of EDs: $N = 5, 10, \ldots, 50$. The inference tasks generated by each ED follow the Poisson distribution. The detailed experimental settings are as follows [6,9]: The computational resources of the ED are [0.1, 1.0] GHz, and the MEC server is 10 GHz. The processing delay tolerance is $T = 3$ s. The channel bandwidth B is set to 20 MHz, the transmission power p is 1 W, and the Gaussian noise power δ^2 is -113 dBm. The computational efficiency constant k_0 is set to 10^{-26}. The channel gain $g = d^{-4}$, where d is the propagation distance. The weight coefficients are $\omega_e = 0.8$ and $\omega_d = 0.2$. The effectiveness of CERL is evaluated using the metrics of system cost and inference task completion rate.

5.2. Effectiveness of CERL

In order to evaluate the effectiveness of CERL, we first compare our CERL with two reinforcement learning methods. The first is the Double Deep Q-Network (DDQN), which stabilizes DQN training by using two separate networks to decouple the selection and evaluation of Q-values, making it a standard baseline in deep reinforcement learning. The second is D^{2}QN [31], a bi-level optimization framework that combines Dueling-DQN, Double-DQN with adaptive parameter space noise. We select DDQN as it represents a fundamental value-based RL approach, while D^{2}QN serves as a strong baseline that has demonstrated effectiveness in MEC optimization scenarios.

5.2.1. Comparison with RL Methods

Figure 4 shows the comparison of convergence performance with two RL methods under various numbers of EDs. As illustrated in the figure, CERL consistently achieves lower system costs compared to both D^2 QN and DDQN on all network scales. When $N = 10$, CERL converges to a system cost of approximately 2, while D^{2}QN and DDQN converge to approximately 5 and 7, respectively. D^{2}QN outperforms DDQN because it incorporates adaptive parameter space noise and Dueling-DQN, leading to more efficient exploration and better resource allocation decisions. As the number of EDs increases, the cost of the system naturally increases due to the increased complexity of resource management and task coordination. However, CERL maintains its superior performance and shows better scalability. For example, with $N = 50$, CERL achieves a stable system cost of about 40, while both D^{2}QN and DDQN converge to significantly higher values around 70.

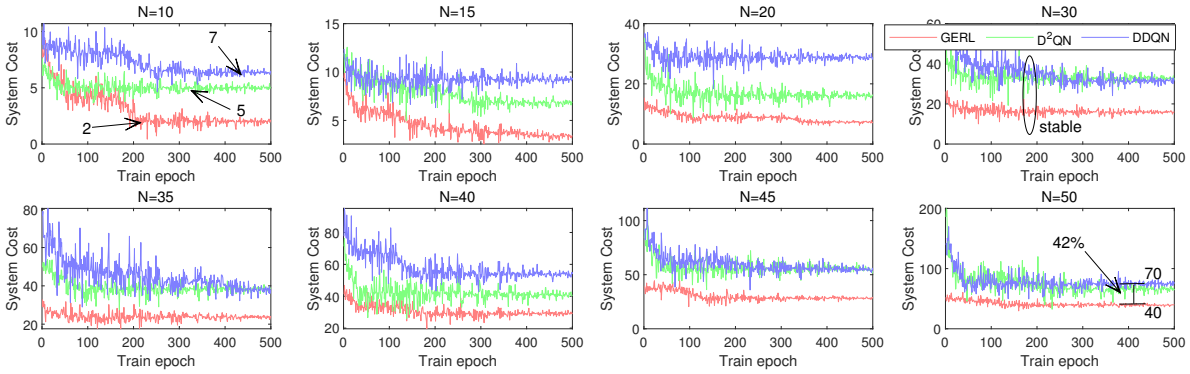

Figure 4. Convergence of the CERL comparison with RL methods.

Furthermore, CERL exhibits faster and more stable convergence behavior. It typically reaches stable performance within 200 training epochs, whereas D^{2}QN and DDQN show more fluctuations and require more training epochs to converge. This advantage becomes more pronounced as the network scale increases, particularly in scenarios with $N \geq 30$. CERL, by using elite individuals to guide the search, discovers optimal partition decisions and resource allocation strategies in complex scenarios with a large number of EDs.

Moreover, Figure 5 shows the variation in the utilization of computational resources in three time slots during the optimization process for $N = 20$. Several important conclusions can be drawn: First, the effect of resource optimization is evident. Most computing nodes show a significant reduction in the remaining resources after optimization. For example, in time slot 1, the remaining resource of ED 11 decreases from 92% to 2%, indicating that CERL effectively allocated inference tasks, leading to better resource utilization. Second, the results demonstrate dynamic resource management capability. The remaining resources vary significantly across different time slots (for example, ED 7 shows 14%, 93%, and 74% in the three time slots), but CERL consistently adjusts its

optimization strategy according to these time-varying conditions. Third, CERL achieves effective load balancing. The optimization process often first uses nodes with more resources remaining, which results in a more balanced distribution of the computational load after optimization. This balanced distribution helps prevent system bottlenecks and improves overall efficiency. These results collectively demonstrate CERL's ability to dynamically optimize resource allocation while maintaining system performance and efficiency.

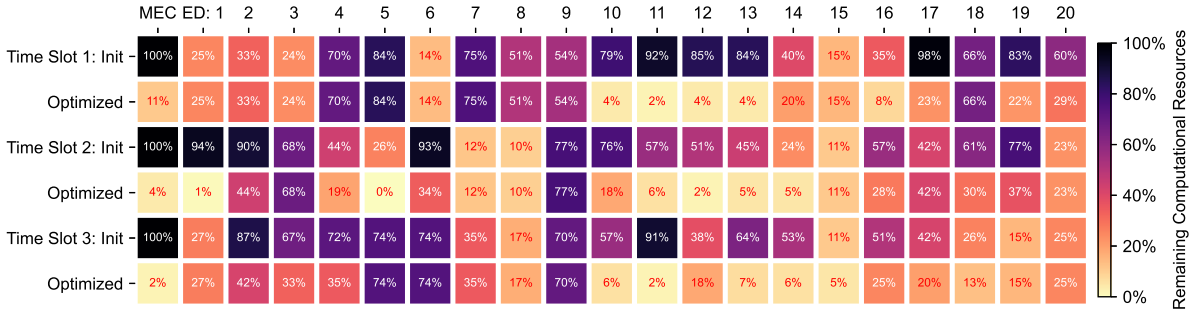

Figure 5. The utilization of computational resources of CERL over time.

Furthermore, Figure 6 compares the system cost and collaboration counts for $N = 30$ among CERL, D^{2}QN, and DDQN. Collaboration count refers to the total number of times that all tasks are distributed and processed on multiple computing nodes. It should be noted that a higher number of collaborations does not necessarily lead to better performance, as excessive partitioning can introduce additional communication overhead and resource consumption. CERL achieves the lowest system costs with appropriate collaboration counts. This indicates that CERL can intelligently determine when collaboration is beneficial, avoiding unnecessary partitioning that could increase communication overhead. In contrast, D^{2}QN shows higher system costs despite similar collaboration counts, suggesting less efficient collaboration decisions. DDQN exhibits the highest system costs with varying collaboration counts, indicating its inability to effectively balance the trade-off between collaboration benefits and associated costs. These results demonstrate that the effectiveness of collaborative inference depends not just on the number of collaborations, but on making intelligent decisions about when and how to partition tasks. CERL's superior performance stems from its ability to identify truly beneficial collaboration opportunities while avoiding unnecessary partitioning that could increase system overhead.

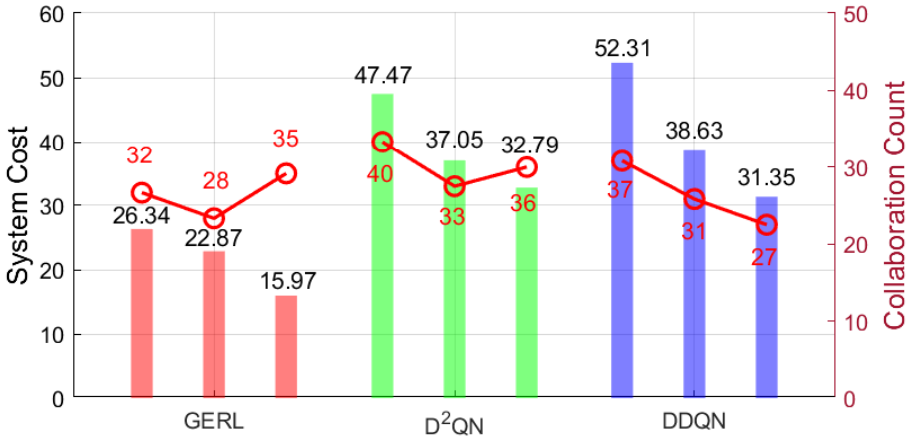

Figure 6. Comparison of system costs and collaboration counts for $N = 30$.

5.2.2. Comparison with Benchmark Schemes

In this subsection, we evaluate CERL by comparing its performance with three benchmarks: LC, where all inference tasks are performed locally; EC, where all EDs attempt to send their inference tasks to the MEC server; and Random, where all EDs randomly select either LC or OC.

For the system cost shown in Figure 7, CERL consistently maintains the lowest cost by intelligently distributing inference tasks across multiple nodes, achieving a reduction of 65. 7% compared to LC. Although EC performs better than LC by avoiding local computing limitations, it still shows higher costs than CERL due to excessive server congestion and connection constraints. This demonstrates that neither pure local computing nor complete edge offloading is optimal for DNN inference in edge computing environments.

In terms of the task completion rate shown in Figure 7, CERL maintains a consistent completion rate 100% through effective collaborative computing, showing a 57.5% improvement over LC. The declining completion rates of both LC and EC. The LC approach relies primarily on local inference and exhibits poor performance due to the resource-intensive nature of DNN models that exceed the computational capabilities of EDs. This is evidenced by its high system cost and low completion rate, highlighting the challenge of executing complex DNN models solely on resource-constrained devices. These results validate the necessity of balanced collaborative inference.

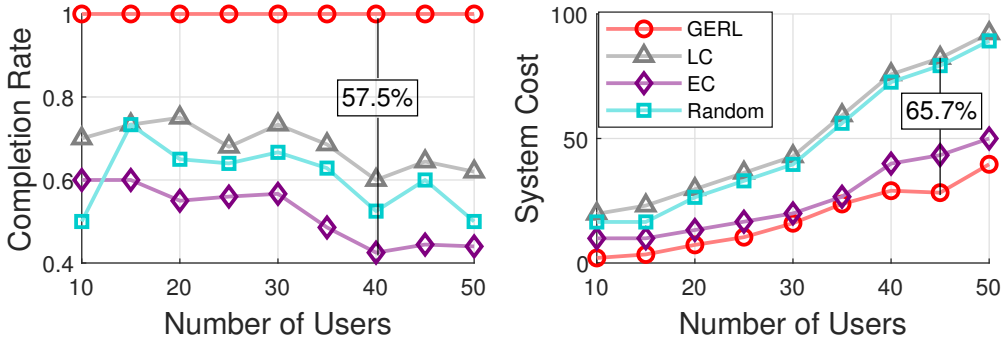

Figure 7. System cost and task completion rate comparison with benchmarks.

These results demonstrate that CERL effectively addresses the limitations of both local-only and server-only approaches through intelligent task partitioning and collaborative inference, significantly improving both system efficiency and reliability. Moreover, in the context of $N = 50$ ED, the average training time for each epoch in CERL is approximately 20 seconds, while the single inference time after training is around 25 milliseconds. This illustrates the feasibility of CERL within the DCI system and confirms the efficiency of real-time decision-making.

In summary, CERL demonstrates superior performance in terms of both system cost reduction and inference task completion rate, indicating its strong capability to handle resource-constrained edge computing scenarios where complex DNN models need to be efficiently executed through collaborative inference.

6. Conclusions

In this paper, we have addressed the challenges of efficient distributed co-inference in dynamic edge computing environments through a novel collaborative computing paradigm. The proposed DCI system highlights the potential of distributed intelligence, enabling computing nodes to collaboratively address complex inference tasks. Our extensive experiments have validated that this approach effectively balances system performance and resource utilization, demonstrating the viability of collaborative edge inference for real-world applications. Our work contributes to the ongoing development of distributed collaborative inference as an effective approach for deploying advanced on-device models in resource-constrained edge devices.

While the CERL framework proposed in this paper has achieved initial success in reducing system costs and enhancing task completion rates, it is crucial to explore its potential limitations and look toward future research directions. One direction worthy of in-depth exploration is the integration of this framework with Federated Learning (FL) technology. The core advantage of FL lies in its ability to enable nodes to collaboratively train models without sharing raw data, thereby offering a new paradigm for distributed inference that enhances both efficiency and privacy. Recent studies have applied FL to the field of IoT cybersecurity, aiming for privacy preservation and real-time threat detection [32], which confirms the value of this technology in enhancing distributed systems. In summary, investigating the deep integration of the CERL framework with privacy-preserving technologies such as FL to build more robust, secure, and efficient distributed AI systems at the edge constitutes a valuable subject for future research.

Author Contributions

L.T.: Conceptualization, Methodology, Software, Investigation, Writing—Original Draft; S.G.: Conceptualization, Supervision, Funding Acquisition, Writing—Review & Editing; P.Z.: Software, Validation; Z.K.: Resources, Writing—Review & Editing. All authors have read and agreed to the published version of the manuscript.

Funding

This work was supported by the National Natural Science Foundation of China (No. 62272069, 62202072), the Fundamental Research Funds for the Central Universities (No.2024CDJGF-036) and Natural Science Key

Foundation of Chongqing (No. CSTB2024NSCQ-LZX0129).

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Use of AI and AI-assisted Technologies

During the preparation of this work, the author(s) used Google's Gemini to review and edit the content for language, formatting, and clarity. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the published article.

References

1. Tang, S.; Yu, Y.; Wang, H.; et al. A Survey on Scheduling Techniques in Computing and Network Convergence. *IEEE Commun. Surv. Tutorials* **2024**, *26*, 160–195.
2. Walia, G.K.; Kumar, M.; Gill, S.S. AI-Empowered Fog/Edge Resource Management for IoT Applications: A Comprehensive Review, Research Challenges, and Future Perspectives. *IEEE Commun. Surv. Tutorials* **2024**, *26*, 619–669.
3. Duan, S.; Wang, D.; Ren, J.; et al. Distributed Artificial Intelligence Empowered by End-Edge-Cloud Computing: A Survey. *IEEE Commun. Surv. Tutorials* **2023**, *25*, 591–624.
4. Kar, B.; Yahya, W.; Lin, Y.D.; et al. Offloading Using Traditional Optimization and Machine Learning in Federated Cloud–Edge–Fog Systems: A Survey. *IEEE Commun. Surv. Tutorials* **2023**, *25*, 1199–1226.
5. Liu, Z.; Song, J.; Qiu, C.; et al. Hastening Stream Offloading of Inference via Multi-Exit DNNs in Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2024**, *23*, 535–548.
6. Tan, L.; Kuang, Z.; Zhao, L.; et al. Energy-Efficient Joint Task Offloading and Resource Allocation in OFDMA-Based Collaborative Edge Computing. *IEEE Trans. Wirel. Commun.* **2022**, *21*, 1960–1972.
7. Wang, J.; Cao, C.; Wang, J.; et al. Optimal Task Allocation and Coding Design for Secure Edge Computing With Heterogeneous Edge Devices. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2817–2833.
8. Tang, X.; Chen, X.; Zeng, L.; et al. Joint Multiuser DNN Partitioning and Computational Resource Allocation for Collaborative Edge Intelligence. *IEEE Internet Things J.* **2021**, *8*, 9511–9522.
9. Li, X.; Bi, S. Optimal AI Model Splitting and Resource Allocation for Device-Edge Co-Inference in Multi-User Wireless Sensing Systems. *IEEE Trans. Wirel. Commun.* **2024**, *23*, 11094–11108.
10. Liang, H.; Sang, Q.; Hu, C.; et al. DNN Surgery: Accelerating DNN Inference on the Edge Through Layer Partitioning. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3111–3125.
11. Li, J.; Liang, W.; Li, Y.; et al. Throughput Maximization of Delay-Aware DNN Inference in Edge Computing by Exploring DNN Model Partitioning and Inference Parallelism. *IEEE Trans. Mob. Comput.* **2023**, *22*, 3017–3030.
12. Tan, L.; Guo, S.; Zhou, P.; et al. Multi-UAV-Enabled Collaborative Edge Computing: Deployment, Offloading and Resource Optimization. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 18305–18320.
13. Xu, C.; Guo, J.; Li, Y.; et al. Dynamic Parallel Multi-Server Selection and Allocation in Collaborative Edge Computing. *IEEE Trans. Mob. Comput.* **2024**, *23*, 10523–10537.
14. Li, Y.; Zeng, D.; Gut, L.; et al. DNN Partitioning and Assignment for Distributed Inference in SGX Empowered Edge Cloud. In Proceedings of the IEEE 44th International Conference on Distributed Computing Systems (ICDCS), Jersey City, NJ, USA, 23–26 July 2024.
15. Mohammed, T.; Joe-Wong, C.; Babbar, R.; et al. Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Virtual, 6–9 July 2020.
16. Eng, K.X.; Xie, Y.; Pereira, M.; et al. A Vision and Proof of Concept for New Approach to Monitoring for Safer Future Smart Transportation Systems. *Sensors* **2024**, *24*, 6018.
17. Liu, H.; Fouda, M.E.; Eltawil, A.M.; et al. Split DNN Inference for Exploiting Near-Edge Accelerators. In Proceedings of the IEEE International Conference on Edge Computing and Communications (EDGE), Shenzhen, China, 7–13 July 2024.
18. Dong, C.; Hu, S.; Chen, X.; et al. Joint Optimization With DNN Partitioning and Resource Allocation in Mobile Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3973–3986.
19. Zeng, L.; Chen, X.; Zhou, Z.; et al. CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices. *IEEE/Acm Trans. Netw.* **2021**, *29*, 595–608.
20. He, W.; Guo, S.; Guo, S.; et al. Joint DNN Partition Deployment and Resource Allocation for Delay-Sensitive Deep Learning Inference in IoT. *IEEE Internet Things J.* **2020**, *7*, 9241–9254.

21. Zeng, Q.; Du, Y.; Huang, K.; et al. Energy-Efficient Resource Management for Federated Edge Learning With CPU-GPU Heterogeneous Computing. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7947–7962.
22. Tan, L.; Guo, S.; Zhou, P.; et al. HAT: Task Offloading and Resource Allocation in RIS-Assisted Collaborative Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2024**, *11*, 4665–4678.
23. Li, P.; HAO, J.; Tang, H.; et al. Value-Evolutionary-Based Reinforcement Learning. Forty-first International Conference on Machine Learning. In Proceedings of the Forty-First International Conference on Machine Learning, Vienna, Austria, 21–27 July 2024.
24. de Boer, P.T.; Kroese, D.P.; Mannor, S.; et al. A Tutorial on the Cross-Entropy Method. *Ann. Oper. Res.* **2005**, *134*, 19–67.
25. Larrañaga, P.; Lozano, J.A. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
26. Li, P.; Hao, J.; Tang, H.; et al. Bridging Evolutionary Algorithms and Reinforcement Learning: A Comprehensive Survey on Hybrid Algorithms. *IEEE Trans. Evol. Comput.* **2024**, *29*, 1707–1728.
27. Zhang, H.; Shao, J.; Jiang, Y.; et al. State Deviation Correction for Offline Reinforcement Learning. *Proc. AAAI Conf. Artif. Intell.* **2022**, *36*, 9022–9030.
28. Sun, W.; Zhang, H.; Wang, R.; et al. Reducing Offloading Latency for Digital Twin Edge Networks in 6G. *IEEE Trans. Veh. Technol.* **2020**, *69*, 12240–12251.
29. MrYxJ. Calflops: A FLOPs and Params Calculate Tool for Neural Networks in PyTorch Framework, 2023. Available online: https://github.com/MrYxJ/calculate-flops.pytorch (accessed on 20 October 2025).
30. Coldwell, J. Latencies for Typical Modern Processor. Available online: http://www.phys.ufl.edu/coldwell/MultiplePrecision/fpvsintmult.htm (accessed on 25 October 2025).
31. Tan, L.; Kuang, Z.; Gao, J.; et al. Energy-Efficient Collaborative Multi-Access Edge Computing via Deep Reinforcement Learning. *IEEE Trans. Ind. Inform.* **2023**, *19*, 7689–7699.
32. Rahmati, M.; Pagano, A. Federated Learning-Driven Cybersecurity Framework for IoT Networks with Privacy Preserving and Real-Time Threat Detection Capabilities. *Informatics* **2025**, *12*, 62.