DRL-Enabled Computation Offloading for AIGC Services in IoIT-assisted Edge Computing Networks

Xingxing Zhang, Shaobo Li, Jianhang Tang, Keyu Zhu, Yang Zhang, Biplab Sikdar, *Senior Member, IEEE*

***Abstract*—The widespread application of AIGC services has driven demand for efficient computational resources, making effective task scheduling and computation offloading in edge computing environments a critical research topic. However, the high computational requirements and low latency demands of AIGC services, combined with the limitations of edge computing, present challenges for existing offloading methods, such as unstable decision-making in dynamic task environments and resource overloading. Here, we propose a decentralized AIGC task offloading architecture within an IoT-assisted edge computing network to optimize the quality of AIGC services. In this architecture, we define a multi-objective joint optimization problem for AIGC task offloading, aiming to simultaneously optimize key performance metrics such as task latency, energy efficiency, and load balancing. To address this problem, we introduce an improved Proximal Policy Optimization (PPO)-based deep reinforcement learning (DRL) algorithm, named TOPPO. By incorporating a policy update step size constraint and a clipping mechanism, TOPPO significantly enhances the stability of the training process and reduces fluctuations during policy updates. Additionally, the algorithm integrates an LSTM model to improve its ability to handle temporal dependencies. Through continuous interaction between the model and the environment, the offloading strategy is iteratively updated to ensure that diverse AIGC tasks are efficiently executed on IoT devices or edge servers. Extensive simulations and performance evaluations demonstrate that the proposed method achieves significant improvements in task latency, energy consumption, and load management during AIGC task processing.**

***Index Terms*—AIGC services, computation offloading, DRL, IoIT, edge computing network**

I. Introduction

AI-GENERATED Content (AIGC) is a specific category of artificial intelligence that is trained on large datasets to learn patterns in data distributions, enabling it to generate new and unique content that resembles the training data [1]. With the rapid development of AIGC technologies, applications of AIGC are showing tremendous potential across multiple domains [2], [3]. However, AIGC tasks are typically computationally intensive, requiring significant data processing and complex model inference to generate content. Meanwhile, AIGC services typically require real-time responsiveness to support instant user interactions, which imposes strict demands on real-time performance and low latency [4]. However, traditional cloud-based solutions often fail to meet these demands, especially in scenarios that involve real-time processing and data-driven decision-making, where issues like high communication latency and scalability limitations of cloud computing become particularly pronounced [5], [6]. In typical Intelligent Internet of Things (IoIT) scenarios—such as smart homes, smart factory, and smart transportation—edge computing offers the advantage of offloading computing tasks to edge nodes near users [7]. This approach can significantly reduce latency and alleviate computational pressure on central cloud servers [8]. By designing a task offloading framework to address the unique demands of AIGC tasks in edge computing environments, we aim to enhance AIGC service performance and improve user experience.

However, the heterogeneity of edge devices, their limited computational capacity, and the high computational and real-time requirements of AIGC tasks make efficient offloading in an edge computing environment a complex problem [9]. In recent years, Deep Reinforcement Learning (DRL) has shown significant potential in optimizing decision-making processes within dynamic environments [10]–[12]. DRL is capable of learning and adapting to the complex changes in edge computing environments, offering intelligent strategies for task offloading and resource management. Nevertheless, most existing research focuses on general computing tasks and has not fully addressed the specific demands of AIGC services, such as real-time performance and computational intensity. As a result, while AIGC has made remarkable advancements in text and image generation, its application in IoIT-assisted edge computing is still in its early stages.

Moreover, existing research predominantly focuses on optimizing a single objective or striking a balance between energy consumption and latency, often overlooking the need for joint optimization of energy consumption, latency, and load balancing [13]–[16]. The nature of AIGC tasks highlights a close interdependence and trade-off between these three factors. For instance, reducing energy consumption may lead to increased latency, while prioritizing latency optimization can place a heavier load on certain nodes, thereby compromising overall system stability. Consequently, strategies that address only a single optimization objective often fall short of meeting the complex requirements of real-world applications. Thus, achieving joint optimization of energy consumption, latency, and load balancing through DRL methods, to ensure the

Xingxing Zhang is with State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China; and the Department of Electrical and Computer Engineering, National University of Singpore, Singpore.

Shaobo Li is with State Key Laboratory of Public Big Data, Guizhou University and Guizhou Institute of Technology, Guiyang, China, e-mail: (lishaobo@gzu.edu.cn).

Jianhang Tang and Keyu Zhu are with State Key Laboratory of Public Big Data, Guizhou University, Guiyang, China.

Yang Zhang is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.

Biplab Sikdar is with the Department of Electrical and Computer Engineering, National University of Singpore, Singpore 119077.

Manuscript received October 16, 2024; revised December 26, 2024.

efficient execution of AIGC tasks in edge computing environments, has emerged as a critical research direction—one that still lacks sufficient investigation and systematic exploration.

We propose a decentralized cloud framework for AIGC task offloading in IoIT-assisted edge computing networks (ECN), aimed at optimizing AIGC service quality by minimizing the long-distance transmission delays between intelligent devices (InDes) and the central cloud. Within this framework, we introduce a DRL-based AIGC task offloading method to address the challenges of efficient task scheduling and resource allocation in IoT-assisted ECN. The proposed method leverages DRL techniques to dynamically manage computational resources across heterogeneous edge nodes, optimizing the AIGC task offloading process while considering real-time processing, energy efficiency, and load balancing constraints. This approach effectively enhances the performance and efficiency of AIGC services. The study not only provides a robust solution for integrating AIGC services in IoIT environments but also makes significant contributions to the broader field of intelligent task scheduling and resource management. The main contributions of this research are as follows:

1) We propose a novel cloud-free computational offloading framework tailored for AI-generated content (AIGC) tasks in IoT-assisted ECN. By eliminating the need for long-distance data transmission between smart devices and central clouds, our framework significantly reduces transmission latency, thereby enhancing the overall quality of AIGC services. In this approach, both AIGC models and offloading decision algorithms are trained directly on high-performance edge servers(ESs), avoiding the inefficiencies of transferring data to remote cloud infrastructures. The inference of AIGC models can be flexibly executed either on InDes or at the ESs, depending on the specific task or network conditions. This strategy effectively minimizes transmission delays and optimizes the performance and quality of AIGC task execution.
2) We propose a DRL-based optimization method for AIGC task offloading (TOPPO). To begin with, we define an AIGC task offloading optimization problem, which, unlike other approaches, considers not only energy consumption and latency but also load balancing. In the reward function design of the Markov Decision Process(MDP) for DRL, we introduce a load balancing metric, which adds complexity to the optimization problem and places higher demands on the stability of the algorithm. To address this challenge, the TOPPO algorithm adopts a clipping mechanism that limits the amplitude of policy updates, thereby preventing instability due to the high dimensionality introduced by load balancing optimization. Furthermore, in the policy and value networks of TOPPO, we introduce an LSTM layer to capture the temporal features of task offloading, enabling the model to better adapt to dynamic edge environments. Through the combination of multi-dimensional reward function design, clipping mechanism, and temporal neural network, our method not only outperforms traditional DRL offloading algorithms in complex AIGC task environments but also provides superior handling of load fluctuations and dynamic resource changes, ultimately achieving higher AIGC service quality.
3) Extensive simulation experiments demonstrate that the proposed TOPPO offloading algorithm significantly improves performance during AIGC computing offloading. Specifically, it reduces energy consumption by 74.57% compared to traditional round-robin methods and decreases delay by approximately 66%, while boosting CPU utilization by around 85%. This increased CPU utilization enables more efficient handling of AIGC tasks in multi-task scenarios, significantly enhancing processing capacity. Additionally, compared to six representative DRL algorithms, TOPPO shows an average reward improvement of 8.36% to 19.97%, indicating its superior adaptability and efficiency in meeting the unique demands of AIGC tasks. These improvements underscore TOPPO's effectiveness in optimizing energy consumption, latency, and load balancing, ultimately providing a more robust solution for AIGC service quality in dynamic environments.

The remainder of this research is organized as follows: Section II reviews the related works. Section III presents the system framework and problem formulation. Section IV details the proposed method and algorithm, along with the corresponding theoretical performance analysis. Section V evaluates the effectiveness of our approach, and Section VI concludes the study.

II. Related Work

A. AIGC Service Offloading Framework

In recent years, many scholars in related fields have explored how to optimize offloading strategies to reduce task execution delays and enhance the performance of edge computing systems [17]. Yuan et al. [18] proposed a task offloading method based on Deep Q-Networks (DQN) to optimize offloading decisions in edge computing. Chao Fang et al. [19] introduced a novel Deep Reinforcement Learning (DRL)-assisted task offloading and resource allocation scheme, TORA-DRL, aimed at minimizing power consumption in cloud-edge collaborative environments. Chao Li et al. [20] developed a dynamic offloading scheme based on the Markov Decision Process (MDP) to efficiently manage computational tasks among heterogeneous devices. Lihua Ai et al. [21] introduced a Reinforcement Learning (RL) algorithm that predicts current Channel State Information (CSI) from historical CSI to derive the optimal task offloading strategy. However, most of these methods primarily focus on general computational tasks and do not address the unique requirements of AI-generated content (AIGC) services, such as real-time processing and computation-intensive tasks.

Despite the significant advancements AIGC has made in generating text, images, and videos, its application in IoIT-assisted edge computing remains in its early stages. Many AIGC applications, such as virtual assistants and real-time content generation, demand high computational resources and

low latency challenges that traditional cloud-based solutions often struggle to meet. Edge computing(EC) plays a crucial role in managing and processing AIGC tasks with high computational requirements. In recent years, researchers have proposed cloud-edge hybrid frameworks, where AIGC model training is handled by the central cloud, while inference and execution are performed on base stations (BS) equipped with edge servers (ES) located near user devices, thereby reducing latency for AIGC services [22]–[24]. For instance, Siyuan Li et al. [22] proposed a generative model-driven industrial AIGC collaborative edge learning framework and introduced an AIGC computation offloading model to ensure the execution of heterogeneous AIGC tasks on edge servers. Weizhe Zeng et al. [25] proposed a latency-aware parallel offloading AIGC framework that partitions multi-modal content and offloads certain diffusion tasks to multiple servers, thereby reducing latency in edge environments. Jianan Zhang et al. [23] proposed a cloud-edge hybrid architecture to support AIGC services. While such hybrid frameworks utilize the powerful computing capabilities of the central cloud to process complex tasks, they still encounter challenges like high communication latency and imbalanced resource utilization.

B. DRL_based Computing Offloading

Deep Reinforcement Learning (DRL) has shown great potential in decision-making problems within complex, dynamic environments, particularly in task scheduling and computation offloading in edge computing. Siyuan Li et al. [22] proposed a computation offloading method based on a attention-enhanced multi-agent reinforcement learning (AMARL) algorithm, aiming to optimize the total system latency for edge-based AIGC task completion, thereby supporting generative model-driven edge learning. Shahidani, FR et al. [26] proposed a RL-based fog scheduling algorithm that improves load balancing and reduces response time. Jialin Lu et al. [13] introduced an A2C-DRL real-time task scheduling technique tailored for stochastic edge environments, incorporating a load-balancing factor to evaluate the model's capability in maintaining balanced loads. Changfu Xu [27] formulated the AIGC computation offloading problem in edge computing as an online integer linear programming problem and then proposed a novel AIGC task scheduling algorithm based on diffusion DRL (DDRL), termed DDRL-ATS, to solve this problem, aiming to minimize task offloading latency. Weijun Cheng et al. [28] proposed a task offloading method based on Deep Q-Networks (DQN), which dynamically selects edge nodes to minimize latency and energy consumption. Similarly, Haiyuan Li et al. [29] developed a task offloading strategy aimed at maximizing long-term benefits in terms of both delay and energy efficiency.

Moreover, several studies have integrated DRL with multi-objective optimization, proposing adaptive task offloading schemes for edge computing [30]–[32]. As summarized in Table 1, most of the existing approaches focus on optimizing individual objectives, such as energy consumption, latency, or load balancing. While some attempt to jointly optimize energy consumption and latency, they often overlook the joint optimization of all three factors. Given the interdependencies and trade-offs between energy consumption, latency, and load balancing in AIGC task offloading, this study will focus on the simultaneous optimization of these three aspects.

TABLE I: A Comparison of Related Studies

Reference	Objectives				Environment
	Latency	Energy	Loading	Completion Rate	
[18]	✓	✓	✗	✗	Edge-Enabled IoMT
[26]	✓	✗	✓	✗	FC
[13]	✗	✗	✓	✓	Edge-cloud
[31]	✗	✗	✓	✓	EN
[29]	✓	✓	✗	✗	EC
[32]	✓	✓	✗	✓	EC
[14]	✓	✓	✗	✗	MEC
[30]	✓	✓	✗	✓	MEC
Our work	✓	✓	✓	✓	IoIT-assisted ECN

In summary, while existing research has made progress in computation offloading, the application of DRL, and resource management in EC, there remains a gap in addressing the unique requirements of AIGC services in IoT-assisted ECN, particularly in the joint optimization of latency, energy consumption, and load balancing. This study aims to fill that gap by proposing a DRL-based computation offloading framework that intelligently optimizes task offloading and resource allocation. The framework is designed to meet the real-time processing and high-efficiency computational demands of AIGC tasks, thereby improving the performance of AIGC services in IoT-assisted edge computing networks.

III. System Framework for AIGC Services in IoIT-assisted ECN and AIGC Computing Offloading Optimization Model

A. System Framework for AIGC Services in IoIT-assisted ECN

We focus on the offloading of AIGC tasks in IoIT-assisted ECN and propose a novel cloud-free offloading framework (AIECOF), as illustrated in Figure 1. By leveraging IoIT's self-learning and autonomous decision-making capabilities, this framework enhances the intelligence and collaborative optimization of ECN. It also optimizes the quality of AIGC services by reducing the long-distance transmission latency between InDes and central cloud servers.

In current framework, AIGC models are typically trained in the central cloud, while inference is carried out at the edge and terminal layers. Although this hybrid scheduling framework utilizes the central cloud's powerful computational resources to handle complex tasks, transmitting data from the edge to the central cloud can introduce additional latency and higher bandwidth consumption. This becomes especially problematic when dealing with sensitive data, as it poses increased security risks. For AIGC applications that require real-time responsiveness, such latency can degrade the quality of service. AIGC tasks, such as real-time image generation and intelligent voice interaction, often demand real-time interactivity and a high degree of data locality. These tasks rely on fast and accurate data processing. Therefore, in IoIT-assisted ECN environments, AIGC is well-suited to a decentralized edge offloading framework. This approach enables faster data processing near the point of data generation, significantly

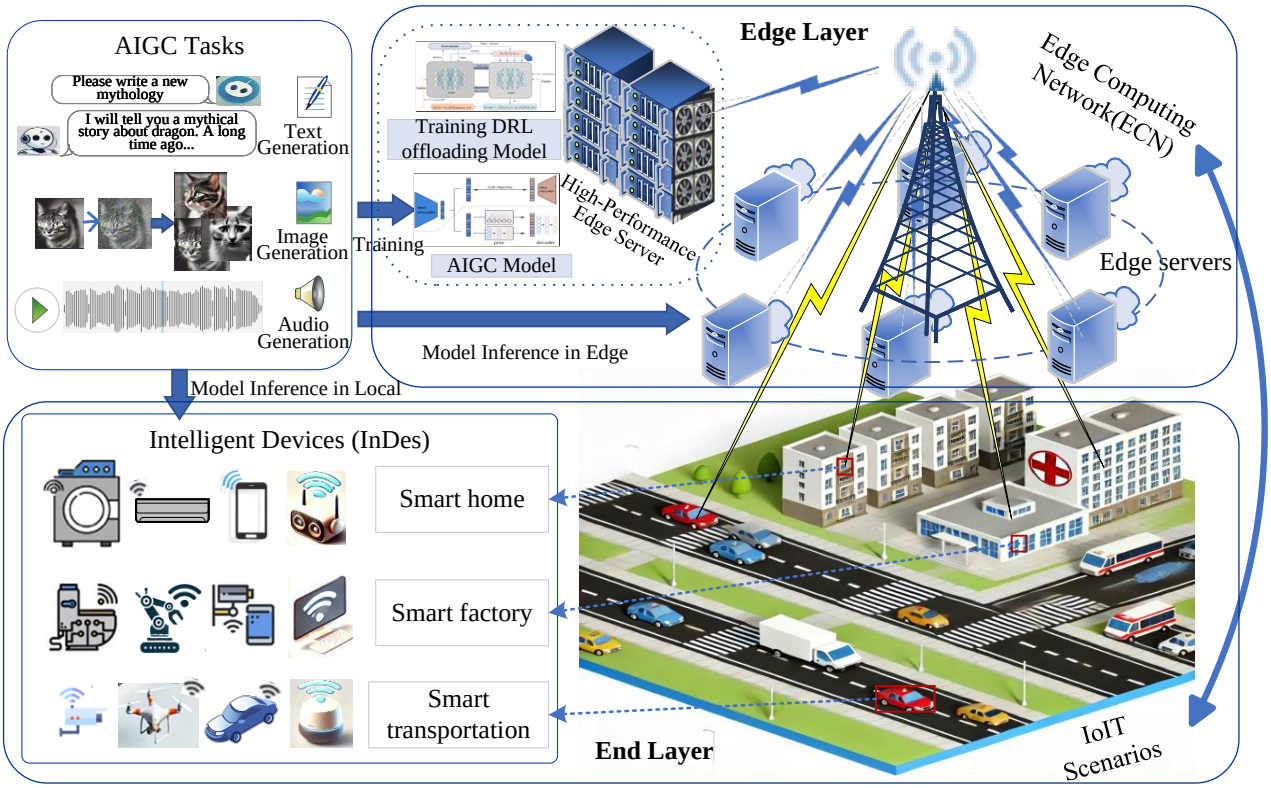


Fig. 1: A novel decentralized cloud offloading framework for optimizing AIGC services in IoIT-assisted ECN (AIECOF)

reducing latency, enhancing data security, and improving the efficiency of AIGC task processing. Ultimately, this framework optimizes the overall system performance and responsiveness.

As shown in Figure 1,the application scope of AIECOF is extensive, spanning smart homes, smart manufacturing, intelligent transportation, and many other fields. In these scenarios, devices capable of autonomously processing AIGC tasks are referred to as intelligent devices(InDes). In AIECOF, we define a set of InDes as $M = \{1, 2, ..., M\}$ and a set of edge servers(ESs) as $N = \{1, 2, ..., N\}$. There are M InDes and N ESs. We define the set of all time slots as $T = \{T_1, T_2, ..., T_k\}$, where the time interval between adjacent time slots is $\Delta t = T_k - T_{k-1}$. A user may request an AIGC task from any InDe at any time, and these task requests are random. This means that any InDe can generate an AIGC task at any given time. Therefore, we assume that AIGC tasks are generated according to a uniform distribution and follow a first-in, first-out (FIFO) principle for task processing.

The AIGC tasks we consider primarily include text generation tasks (e.g., chatbots) with task sizes of approximately 1 Mbits to 3 Mbits, image or visual content generation tasks (e.g., AI painting) ranging from 5 Mbits to 10 Mbits, and audio generation tasks (e.g., AI music or smart home voice control) ranging from 3 Mbits to 10 Mbits. Once an AIGC task is initiated, the InDe m must independently execute a task offloading strategy based on its state within the AIECOF framework—either processing the task locally or offloading it to an ES n via a wireless channel.

Each InDe's offloading strategy consists of two stages. The first stage determines whether the AIGC task needs to be offloaded. If offloading is necessary, the second stage selects the most suitable ES for processing the task. The key symbols used in this study are summarized in Table 2.

B. AIGC Services Computing Offloading System Model

In IoIT, IoT devices must have self-learning, self-adaptive, and autonomous decision-making capabilities to effectively meet the demands of delivering AIGC services to users. In this study, we focus on the indivisible task offloading problem during the inference process of AIGC services on InDes. Unlike the approach of offloading an entire AIGC task to an ES, we consider that tasks can either be processed locally on the InDe or offloaded to a nearby ES within an IoIT-assisted ECN for execution. Our goal is to address two key issues:

1) Should the AIGC task be offloaded to an ES?
2) Which ES should the task be offloaded to in order to optimize overall performance in terms of energy consumption, latency, and load balancing?

Based on these two considerations, we design the AIGC task offloading decision-making process.

We describe the task generated by InDe m at timestamp t as

$$\Phi_{m,t} = \{\mu_{m,t}, x_{m,t}, \rho_{m,t}, \tau_{\max}\},$$

where $\mu_{m,t}$ represents the unique index of the AIGC task, with $\mu_{m,t} \in \mathbb{Z}^{++}$. $x_{m,t}$ denotes the size of the AIGC task (in

TABLE II: Main Notations and Their Meanings

Symbol	Meaning
M, N, T	InDe, ES, Timestamp
f_m^{InDe}	Computational capability of InDe m at time t
f_n^{ES}	Computational capability of ES n
$\mu_{m,t}$	Unique index of AIGC task at InDe m at time t
$L_{m,t}$	AIGC task size
$\eta_{m,t}$	Offloading decision at InDe m at timestamp t
$\xi_{m,n,t}$	Whether the AIGC task is offloaded to the ES n
$\tau_{\max}$	Maximum allowable execution time for AIGC task
$D_m^{\text{wait}}(t)$	Waiting delay of the computing queue in InDe m at timestamp t
$D_m^{\text{comp}}(t)$	Computing delay in InDe m at time t
$D_m(t)$	The total delay of computing tasks: $L_{m,t}/f_m^{\text{InDe}}$
$E_m^{\text{wait}}(t)$	Static energy consumption during AIGC task waiting in InDe
$E_{m,t}^{\text{comp}}$	Dynamic energy consumption during AIGC task execution in InDe
$E_{m,t}^{\text{InDe}}$	Energy consumed for executing the task generated by InDe m at time t locally
$D_{m,n}^{\text{trans}}(t)$	Transmission delay
$E_{m,n}^{\text{trans}}(t)$	Transmission energy consumption
$r_{m,n}$	Transmission rate
$D^{\text{edge}}(t)$	The total delay of all tasks across all ESs at all timestamps
$E_n^{\text{wait}}(t)$	Static energy consumption during AIGC task waiting in ES
$E_{n,t}^{\text{comp}}$	Dynamic energy consumption for task execution by ES at time t
$E_{n,t}^{\text{ES}}$	Energy consumed for executing the task processed by ES n at time t
$L_{n,t}^{\text{edge}}$	The load of the ES at timestamp t
ρ_m	Computing density of the InDe
$\lambda_1, \lambda_2, \lambda_3, \lambda_4$	Latency sensitivity weight, Energy consumption task dropout weight, and loading weight

Mbits), $\rho_{m,t}$ is the computational density (in cycles/bit), and $\tau_{\max}$ is the maximum tolerable time for the AIGC task (in seconds). If task $\Phi_{m,t}$ is not fully transmitted or executed by the deadline at timestamp $t_{\text{end}} = t + \tau_{\max} - 1$, the task will be dropped.

The AIGC tasks generated at each timestamp vary in terms of data size, computational resource requirements, and processing time.

We define a binary decision variable $\eta_{m,t} = \{0, 1\}$ to indicate whether the AIGC task is offloaded. If the AIGC task is executed on the InDe, $\eta_{m,t} = 1$. If the task is offloaded to an ES, then $\eta_{m,t} = 0$.

Next, we define the variable $\xi_{m,n,t} = \{0, 1\}$ to decide which ES the AIGC task is offloaded to for processing. When the AIGC task from InDe m is offloaded to ES n, $\xi_{m,n,t} = 1$; otherwise, $\xi_{m,n,t} = 0$. Let $\xi_{m,t} = (\xi_{m,n,t}, n \in N)$. The offloading constraint is given by:

$$\sum_{n \in N} \xi_{m,n,t} = \mathbf{1}(\eta_{m,t} = 0), \tag{1}$$

where $m \in M$, $n \in N$, and $t \in T$.

1) Intelligent Device (InDes) Model In IoIT:

a) Delay model of local computing queue: When $\eta_{m,t} = 1$, the AIGC task $\mu_{m,t}$ is executed locally on the InDe. When a task is generated, it is placed in the computational queue of the InDe $Q_m^{\text{comp}} = \{\Phi_{m,1}, \Phi_{m,2}, \cdots, \Phi_{m,t}\}$ and is executed in a first-in-first-out (FIFO) manner. Therefore, if the previous task has not been completed, the next task will experience a waiting delay before it can start execution. We define the waiting delay as:

$$D_m^{\text{wait}}(t) = D_m^{\text{end}}(t-1) - t + 1, \tag{2}$$

where $D_m^{\text{wait}}(t) < \tau_{\max}$, meaning that the waiting delay must be less than the maximum tolerable time for the AIGC task.

Additionally, when the task starts computing on the InDe, it will experience a computational delay, which is the time required to process the task on the device. The computational delay is calculated by rounding up the required time to process the task. We define this as:

$$D_m^{\text{com}}(t) = \left\lceil \frac{x_{m,t}\eta_{m,t}}{f_m^{\text{InDe}} \cdot \frac{\Delta t}{\rho_m}} \right\rceil, \tag{3}$$

where $x_{m,t}\eta_{m,t}$ represents the data size of the AIGC task in the computational queue of InDe m. The term f_m^{InDe} represents the computational capability of the InDe, usually measured in GHz (the amount of computation processed per second). Δt represents the time step, and ρ_m is the computational density of the device.

The total delay for task $\mu_{m,t}$ is the sum of the waiting delay and the computational delay. The timestamp at which the task is either processed or discarded is defined as:

$$D_m^{\text{end}}(t) = \min \begin{cases} t + D_m^{\text{wait}}(t) + D_m^{\text{com}}(t), & D_m^{\text{wait}}(t) < \tau_{\max} \\ t + \tau_{\max} - 1, & D_m^{\text{wait}}(t) \geq \tau_{\max} \end{cases} \tag{4}$$

In summary, the total delay is the sum of the delays across different time steps and different InDes, as follows:

$$D_{\text{total}}^{\text{InDe}}(t) = \sum_{t=1}^{T} \sum_{m=1}^{M} \left(D_m^{\text{wait}}(t) + D_m^{\text{com}}(t) \right). \tag{5}$$

b) Energy Consumption model of computing queue: When the AIGC task $\mu_{m,t}$ in the computational queue experiences waiting time, it incurs static energy consumption, denoted as $E_{m,t}^{\text{wait}}$. When the AIGC task is processed on the InDe, dynamic energy consumption is generated, denoted as $E_{m,t}^{\text{comp}}$. Therefore, the total energy consumption for task $\mu_{m,t}$ from request to completion at different timestamps is calculated as follows:

$$E_{m,t}^{\text{InDe}} = E_{m,t}^{\text{wait}} + E_{m,t}^{\text{comp}}. \tag{6}$$

In the InDe's task queue, the static energy consumption during the waiting period can be computed using the following formula:

$$E_{m,t}^{\text{wait}} = \int_{t}^{D_m^{\text{end}}(t-1)} P_{\text{wait}}^{\text{InDe}} \, dt = P_{\text{wait}}^{\text{InDe}} \cdot D_{m,t}^{\text{wait}}, \tag{7}$$

where $P_{\text{wait}}^{\text{InDe}}$ represents the static power, which is constant.

To further optimize the local execution energy consumption, Dynamic Voltage and Frequency Scaling (DVFS) technology is used. This technology flexibly adjusts the CPU operating frequency, denoted as f_{ue}^i, and the supply voltage V_{cir}. the dynamic power consumption $P_{\text{comp}}^{\text{InDe}}$ is given by the formula [33]:

$$P_{\text{comp}}^{\text{InDe}} = V_{\text{cir}}^2 f_m^{\text{InDe}} = c \left(f_m^{\text{InDe}} \right)^3. \tag{8}$$

The equation shows that dynamic power consumption is proportional to the square of the voltage V_{cir}, and that CPU frequency is approximately linearly related to the supply voltage [32]. The constant $c = 10^{-26}$ reflects t he capacitance switching characteristics of the chip architecture. Based on these relationships, the dynamic computational energy consumption of a task can be calculated using the following equation:

$$E_{m,t}^{\text{comp}} = c \left(f_m^{\text{InDe}}\right)^2 x_{m,t} \rho_{m,t}, \tag{9}$$

where f_m^{InDe} represents the CPU frequency, $x_{m,t}$ is the task size, and $\rho_{m,t}$ is the computational density.

$$E_{m,t}^{\text{comp}} = P_{\text{comp}}^{\text{InDe}} \cdot D_{m,t}^{\text{InDe}} = c \left(f_m^{\text{InDe}}\right)^2 x_{m,t} \rho_{m,t}. \tag{10}$$

Finally, the total energy consumption across all InDes can be summarized as:

$$E_{\text{InDe}} = \sum_{t=1}^{T} \sum_{m=1}^{M} \left(E_{m,t}^{\text{wait}} + E_{m,t}^{\text{comp}}\right). \tag{11}$$

c) Loading model of computing queue in InDe: In IoIT-assisted ECN, CPU utilization is an important metric for measuring loading and resource usage efficiency. We define the CPU utilization of an InDe as $L_{n,t}^{\text{InDe}}$, which is calculated as the ratio between the computational resource demand and the available computational resources, using the following formula:

$$L_{n,t}^{\text{InDe}} = \min \left\{ 0.98, \frac{\rho_{m,t} \cdot x_{m,t}}{f_n^{\text{InDe}} \times (t - t' + 1)} \right\}, \tag{12}$$

where $\rho_{m,t} \cdot x_{m,t}$ represents the computational demand of the AIGC task. The size and computational requirements of the AIGC task are the key factors determining resource usage. Here, f_n^{InDe} denotes the computational capability of the InDe (in CPU cycles per second), and $(t - t' + 1)$ represents the time the AIGC task has been executing on the InDe, calculated from the task's start time to the current time step.

The execution of an AIGC task not only depends on its computational demands and the computational capacity of the InDe, but also on the length of time the task has been executing. Thus, we incorporate the time dimension, denoted by $t - t' + 1$, to represent how long the task has been running on this InDe. By calculating the task's execution time in conjunction with the server's computational capacity, we can determine whether the task's computational demands can be met within the given time frame.

Additionally, the computational capacity of the InDe is limited. Therefore, we must assess whether the node can complete the task within the allotted time based on its computational capability. If $L_{n,t}^{\text{InDe}} < 1.0$, indicating that the task does not fully utilize the InDe's resources. However, the utilization is capped at 1.0 (100%), indicating that the InDe is working at full capacity. To prevent system overload due to the limited computational capacity of InDes, the CPU utilization is capped at 0.98, leaving a 0.02 buffer to provide fault tolerance and safeguard against overload.

d) Delay and Energy Consumption model of transmission queue: At the beginning of timestamp t, when the task $\Phi_{m,t}$ of InDe m needs to be offloaded to ES n, i.e., the offloading decision is $\xi_{m,n,t} = 1$, the AIGC task $\mu_{m,t}$ must first enter the transmission queue $Q_{m,n}^{\text{trans}}$ of InDe m to wait for ES n to process it. The InDe sends the AIGC task in the transmission queue to the ES through its wireless network interface. This transmission process is carried out over orthogonal channels to prevent interference.

The transmission rate $r_{m,n}^{\text{trans}}$ (in bits per second) is calculated using the following formula:

$$r_{m,n}^{\text{trans}} = W_{m,n} \log_2 \left(1 + \frac{|h_{m,n}|^2 P_m}{\sigma^2}\right), \tag{13}$$

where $W_{m,n}$ represents the bandwidth allocated to each communication channel, $|h_{m,n}|^2$ is the channel gain between m and n, P_m represents the transmission power of InDe m, and σ^2 is the noise power received at ES n.

We define the transmission delay of the task as follows:

$$D_{m,n}^{\text{trans}}(t) = \frac{x_{m,t}(1 - \eta_{m,t})}{r_{m,n}^{\text{trans}}} \Delta t, \tag{14}$$

where $x_{m,t}(1-\eta_{m,t})$ is the size of the AIGC task data arriving at the transmission queue, and $r_{m,n}^{\text{trans}}$ is the transmission rate from InDe m to ES n. It is important to note that if the transmission delay exceeds the maximum tolerable time of the task during transmission, the task will be dropped.

The total transmission delay across all timestamps is given by:

$$D_{\text{total}}^{\text{trans}}(t) = \sum_{t=1}^{T} \sum_{m=1}^{M} D_{m,n}^{\text{trans}}. \tag{15}$$

The energy consumption $E_{m,t}^{\text{trans}}$ for offloading the AIGC task from InDe m to ES n is calculated using the following formula:

$$E_{m,t}^{\text{trans}} = P_{m,n}^{\text{trans}} \cdot \left(\frac{x_{m,t}(1 - \eta_{m,t})}{r_{m,n}^{\text{trans}} \Delta t}\right), \tag{16}$$

where $P_{m,n}^{\text{trans}}$ represents the transmission power, and $x_{m,t}(1 - \eta_{m,t})$ is the task's data size.

The total transmission energy consumption across different timestamps and InDes for offloading AIGC tasks is calculated by summing the energy consumption of all tasks in the transmission queue, as follows:

$$E_{\text{total}}^{\text{trans}} = \sum_{t=1}^{T} \sum_{m=1}^{M} E_{m,t}^{\text{trans}}. \tag{17}$$

2) Edge Servers (ES) Model In ECN:

a) AIGC task computing queue delay model in ES: When $\eta_{m,t} = 0$ and $\xi_{m,t} = 1$, the AIGC task $\mu_{m,t}$ will be offloaded to ES n for computation. However, it first passes through the transmission queue $Q_{m,n}^{\text{trans}}$ to check if the condition $D_{m,n}^{\text{trans}}(t) < \tau_{\max}$ is satisfied. If this condition is met, the task reaches the computation queue of the edge server $Q_n^{\text{EScomp}} = \{\Phi_{m,1}, \Phi_{m,2}, \ldots, \Phi_{m,t}\}$, where it waits to be processed.

The delay for processing the AIGC task includes both the waiting delay and the computation delay, and it is calculated using the following formula:

$$D_{m,n}^{\text{edge}}(t) = \left(D_n^{\text{end}}(t-1) - t + 1\right) + \left\lceil \frac{x_{m,t}(1-\eta_{m,t})}{f_n^{\text{edge}}} \right\rceil, \tag{18}$$

where f_n^{edge} represents the computational capability of ES n (in CPU cycles per second).

The total delay for all ESs across all time is given by:

$$D_{\text{total}}^{\text{edge}}(t) = \sum_{t=1}^{T} \sum_{n=1}^{N} D_{m,n}^{\text{edge}}. \tag{19}$$

b) Energy Consumption model in ES: When the AIGC task $\mu_{m,t}$ in the ES's computational queue is waiting, it incurs static energy consumption, denoted as $E_{n,t}^{\text{wait}}$. When the AIGC task is being processed on the ES, it incurs dynamic energy consumption, denoted as $E_{n,t}^{\text{comp}}$.

The static energy consumption of an AIGC task while waiting in the ES's computation queue can be calculated using the following formula:

$$E_{n,t}^{\text{wait}} = \int_{t}^{D_n^{\text{end}}(t-1)} P_{\text{wait}}^{\text{edge}}\, dt = P_{\text{wait}}^{\text{edge}} \cdot D_{n,t}^{\text{wait}}, \tag{20}$$

where $P_{\text{wait}}^{\text{edge}}$ represents the static power (which is constant).

According to references [33], the dynamic power consumption $P_{\text{comp}}^{\text{edge}}$ is calculated as follows:

$$P_{\text{comp}}^{\text{edge}} = V_{\text{cir}}^2 f_n^{\text{edge}} = c\left(f_n^{\text{edge}}\right)^3, \tag{21}$$

the dynamic computational energy consumption of a task can be derived as:

$$E_{n,t}^{\text{edge}} = P_{\text{comp}}^{\text{edge}} \cdot D_{n,t}^{\text{edge}} = c\left(f_n^{\text{edge}}\right)^2 x_{m,t}\rho_{m,t}, \tag{22}$$

Thus, the total energy consumption for task $\mu_{m,t}$, from entering the transmission queue to completing execution on the ES, is calculated as follows:

$$E_{n,t}^{\text{edge}} = E_{n,t}^{\text{wait}} + E_{n,t}^{\text{comp}}. \tag{23}$$

Finally, the total energy consumption across all smart devices can be summarized as:

$$E_{\text{InDe}} = \sum_{t=1}^{T} \sum_{n=1}^{N} E_{n,t}^{\text{edge}}. \tag{24}$$

After the AIGC task is processed by the ES, the results are sent back to the InDe. However, since the downlink data transmission rate from the ES to the InDe is higher than the uplink transmission rate, we ignore the time delay, energy consumption, and load resulting from transmitting the processed AIGC results back to the smart device.

C. Problem formulation

We focus on the task offloading problem for AIGC inference in IoIT-assisted ECN. Our goal is to minimize task processing delay, long-term system energy consumption, and load variance, while also reducing the number of failed task offloads, all within the constraint of the maximum tolerable time. The objective function is defined as:

We focus on the task offloading problem for AIGC inference in IoIT-assisted ECN. Our goal is to minimize task processing delay, long-term system energy consumption, and load variance, while also reducing the number of failed task offloads, all within the constraint of the maximum tolerable time. The objective function is defined as:

$$F_{m,t} = \begin{cases} \lambda_1\left(D_m^{\text{wait}}(t) + D_m^{\text{com}}(t)\right) + \lambda_2\left(E_{m,t}^{\text{InDe}}\right) \\ +\lambda_3\mu_{m,t}^{\text{dead}} + \lambda_4 L_{n,t}^{\text{InDe}}, & \text{if } \eta_{m,t} = 1 \\ \lambda_1\left(D_{m,n}^{\text{trans}}(t) + D_{m,n}^{\text{edge}}(t)\right) + \lambda_3\mu_{m,t}^{\text{dead}} \\ +\lambda_2\left(E_{m,t}^{\text{trans}} + E_{n,t}^{\text{edge}}\right), & \text{otherwise} \end{cases} \tag{25}$$

Where $\mu_{m,t}^{\text{dead}}$ represents the task failure rate, and $\lambda = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$, $\lambda \in (0,1)$, represents the weights for average delay, energy consumption, task drop rate, and load variance, respectively, used to balance multiple metrics. The minimization problem can be formulated as:

$$\min \mathbb{E}\left[\frac{1}{T}\frac{1}{M}\sum_{t=1}^{T}\sum_{m=1}^{M} F_{m,t}(\eta_{m,t})\right], \tag{26}$$

Subject to constraints (1)-(4),and

c1: $1 \le m \le M,\ 1 \le n \le N,\ \forall t \in T$

c2: $\eta_{m,t} \in \{0,1\},\ \forall m \in M$

c3: $\xi_{m,n,t} \in \{0,1\},\ \forall m \in M,\ \forall n \in N$

c4: $D_m^{\text{wait}}(t) + D_m^{\text{com}}(t) < \tau_{\max}, \quad \text{if } \eta_{m,t} = 1$

c5: $D_{m,n}^{\text{trans}}(t) + D_{m,n}^{\text{edge}}(t) < \tau_{\max}, \quad \text{if } \xi_{m,n,t} = 1$

c_1 represents the constraints on task execution timestamps, the number of smart devices, and the number of ESs. c_2 and c_3 ensure that AIGC tasks must either be processed on the smart device or offloaded to an ES. c_4 and c_5 specify that task latency must not exceed the maximum tolerable time.

IV. TOPPO_Based Offloading Method In IoIT_assised ECN

The objective function of the AIGC task offloading problem, as described by equation (26), is highly complex, making it difficult to find the optimal solution using traditional optimization methods. We have designed the TOPPO algorithm based on Deep Reinforcement Learning (DRL) to effectively address this issue. In this section, we provide a detailed explanation of the state, action, and reward functions within the Markov Decision Process (MDP) for the AIGC task offloading problem, and introduce the proposed TOPPO-based AIGC computation offloading algorithm.

A. Markov Decision Process

1) state: At timestamp t, the InDe m detects the current state of the AIGC task, including the task characteristics, the edge server (ES) features, and the load status of the queue. The state of the AIGC task is represented by $\Phi_{m,t} = \{\mu_m, x_{m,t}, \rho_{m,t}, T_{\max}\}$, where μ_m is the task index, $x_{m,t}$ is

the task size, $\rho_{m,t}$ is the computational requirement, and $T_{\max}$ is the maximum tolerable time. $D_m^{\text{wait}}(t)$ represents the waiting time for device m in the queue at time t, corresponding to the task being processed. The characteristics of the edge server $ES_{n,t}$ are denoted by $\{f_n^{\text{edge}}|Q_n^{\text{ES-comp}}(t-1)\}$, where f_n^{edge} represents the computational power of the ES and $Q_n^{\text{ES-comp}}(t-1)$ represents the queue load from the previous time slot. Therefore, for $n \in N$ and $m \in M$, the state of the InDe m at time t is expressed as:

$$s_{m,t} = \{\mu_m, x_{m,t}, \rho_{m,t}, T_{\max}, D_m^{\text{wait}}(t), f_n^{\text{edge}}, Q_n^{\text{ES-comp}}(t-1)\}$$

, this state incorporates the features of both the AIGC task and the edge server (ES), as well as the load of the computation queue.

2) Action: The AIGC task offloading strategy consists of two levels: the first level requires the smart device to determine whether the current task should be offloaded, while the second level selects the most appropriate edge server to execute the task if offloading is needed. Thus, the action can be described as:

$$a_{m,t} = (\eta_{m,t}, \xi_{m,t}))$$

, where, $\eta_{m,t}$ indicates whether the task is offloaded. A value of 1 means the task is executed locally, while 0 means it is offloaded to the edge node.$\xi_{m,t}$ indicates whether to select edge server n for task execution. A value of 1 means the edge server is selected, while 0 means it is not selected.

3) Reward function: To optimize energy consumption, task delay, task completion rate, and load balancing of factor in the AIGC task offloading process, the reward function is designed to comprehensively account for these four key metrics during decision-making. First, at time slot T_k, the average task delay ℓ_D and the average energy consumption ℓ_E for processing AIGC tasks are calculated.

$$\ell_D = \sum_{t=1}^{T_k} \frac{r_m^D(t)}{T_k}, \tag{27}$$

$$\ell_E = \sum_{t=1}^{T_k} \frac{r_m^E(t)}{T_k}, \tag{28}$$

Where, $r_m^D(t)$ and $r_m^E(t)$ represent the sets of energy consumption and delay, respectively, for InDe m at various time stamps

The weighted penalty value, incorporating energy consumption, delay, task offloading success rate, and load balancing factor, is calculated using the following formula:

$$\begin{cases} \sigma_{\text{E}} & = -\log_2\left(\frac{r_m^E(t)}{\ell_E} + 1 \times 10^{-10}\right), \\ \sigma_{\text{D}} & = 1 - \frac{\ell_D}{D_{\max}}, \\ \sigma_{\text{F}} & = 1 - \frac{g}{G}, \\ \sigma_{\text{L}} & = \frac{1}{M}\sum_{m=1}^{M}\left(L_{m,t}^{\text{InDe}} - \overline{L_{m,t}^{\text{InDe}}}\right)^2, \end{cases} \tag{29}$$

Where, g represents the total number of failed offloading tasks, G is the total number of offloaded tasks, and $D_{\max}$ denotes the maximum tolerable delay. $r_m^E(t)$ refers to the most recent energy consumption data for device m.

Based on the penalty terms for different optimization metrics, integrating energy consumption σ_E, delay σ_D, task success rate σ_F, and load balancing factor σ_L, the final reward function $r(t)$ can be expressed as follows in Equation (30):

$$r(t) = \lambda_1\sigma_E + \lambda_2\sigma_D + \lambda_3\sigma_F + \lambda_4\sigma_L, \tag{30}$$

where λ_1, λ_2, λ_3, and λ_4 are the weight coefficients for energy consumption, delay, task success rate, and load balancing, respectively. The reward function for the edge device $R_m(t)$ is given by:

$$R_m(t) = r(s_{m,t}, a_{m,t}). \tag{31}$$

This function calculates the reward by observing the current state and action at time t, considering energy consumption, delay, and load balancing. The objective of the reward is to maximize the overall reward $r(t)$ for InDe.

Through this MDP model, the AIGC task offloading problem can be efficiently addressed in an IoT-based computing environment. By factoring in the dynamic network environment and available resources, smart devices can adjust their offloading decisions in real time to optimize performance, including minimizing energy consumption, reducing latency, improving task success rates, and maximizing CPU utilization.

B. TOPPO_based AIGC task offloading algorithm

We propose an AIGC task offloading algorithm called TOPPO, which combines the efficient policy optimization of PPO algorithm with the temporal dependency processing capabilities of LSTM networks.As illustrated in Figure 2, the dynamic changes in the AIGC task offloading environment directly impact the offloading decisions within the TOPPO algorithm. By incorporating LSTM networks into the policy network (ActorLSTM) and value network (CriticLSTM) of TOPPO, the algorithm can capture and leverage the complex temporal dependencies present during the task offloading process. Additionally, we introduce a step size limit and clipping mechanism for policy updates to enhance the stability of the training process and reduce fluctuations during policy updates. Through continuous interaction with the AIGC task offloading environment, the algorithm continuously learns and optimizes the offloading policy, resulting in higher task success rates, reduced latency, energy savings, and improved load balancing.

Since LSTM can effectively capture both the short-term and long-term dependencies of task states and environmental changes, it enables TOPPO to more accurately predict task demands and the load status of edge nodes in dynamic environments, thereby enhancing the robustness of decision-making. To leverage this capability, we integrate an LSTM layer into the first layer of both the policy network and value network in the original PPO structure, allowing it to process time-series information obtained from the environment.

At each time step, the state sequence (a sequence of historical states) is fed into the LSTM, which outputs the features

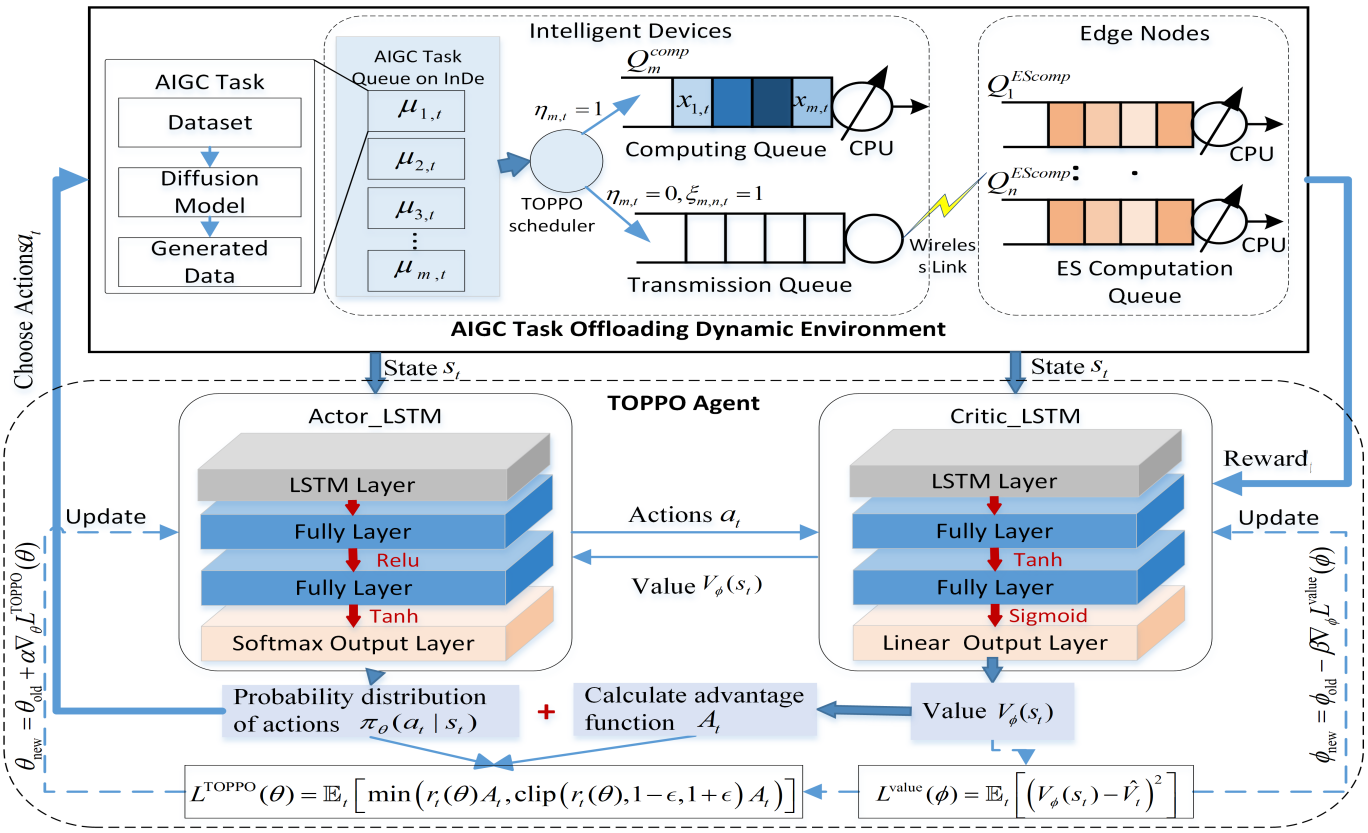


Fig. 2: TOPPO Algorithm Overview for Efficient AIGC Task Offloading

corresponding to the last time step. The time-series features produced by the LSTM are then concatenated with the current state s_t observed from the environment. This combined feature set is subsequently processed through two fully connected layers in sequence.

As shown in Figure 2, the policy network first outputs the current state s_t of the environment. The offloading action $a_{m,t} = (\eta_{m,t}, \xi_{m,t})$is sampled according to the probability distribution $\pi_\theta(a_t \mid s_t)$. This action $a_{m,t} = (\eta_{m,t}, \xi_{m,t})$ determines the offloading decision for the AIGC task. The environment then returns an immediate reward r_t, which is a function of the reward function, delay, and penalty for offloading. Simultaneously, the state transition function $P(s_{t+1} \mid s_t, a_t)$ returns a new state s_{t+1}, which is used as the input state for the policy network in the next iteration.

For each task offloading, the system collects the four elements (s_t, a_t, r_t, s_{t+1}), which include the previous state s_t, the action executed a_t, the immediate reward r_t, and the next state s_{t+1}. These data are used for subsequent policy updates and value estimation.

The value network uses the immediate reward r_t and the estimated value of the next state s_{t+1} to calculate the Temporal Difference (TD) error, as shown below:

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t), \tag{32}$$

The objective of the value network is to optimize the state value function $V(s_t)$ by minimizing the Temporal Difference (TD) error. Therefore, the loss function for the value network is typically defined as:

$$L^{\text{value}}(\phi) = \mathbb{E}_t\left[\left(V_\phi(s_t) - \hat{V}(s_t)\right)^2\right], \tag{33}$$

where, $V_\phi(s_t)$ is the current estimate of the value network for state s_t. The target value $\hat{V}(s_t) = r_t + \gamma V_\phi(s_{t+1})$ is calculated using the immediate reward and the estimated value of the next state. The loss function measures the mean squared error between the current estimate of the value network and the target value.

By computing the gradient of the loss function with respect to the parameters and using gradient descent, the value network parameters are updated as shown in Equation (34):

$$\phi_{\text{new}} = \phi_{\text{old}} - \beta \nabla_\phi L^{\text{value}}(\phi), \tag{34}$$

where β is the learning rate, controlling the step size of each update. The value network gradually adjusts its parameters by minimizing the squared TD error, allowing the network to more accurately estimate the long-term value of each state.

After updating the value network, the advantage function A_t is calculated based on the updated value network output and serves as the basis for updating the policy network. The advantage function A_t evaluates the advantage of the current action in the context of the AIGC task offloading decision relative to the expected policy. This value is crucial for policy network optimization. The Generalized Advantage Estimation

(GAE) is an efficient method for capturing state-value changes and is used in TOPPO to compute the advantage function A_t:

$$A_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}, \tag{35}$$

When the advantage function is computed, the policy network is updated using the clipped objective function from PPO:

$$L^{\text{TOPPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip} \left(r_t(\theta), 1-\epsilon, 1+\epsilon \right) A_t \right) \right], \tag{36}$$

where A_t is the advantage function calculated via GAE, ensuring that the policy update captures multi-step advantages during the offloading process; $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of the new and old policies for the given state-action pair, used for policy step updates.

To control the update magnitude and prevent drastic changes during a single policy update, ϵ is the clipping parameter that restricts the update range, ensuring policy stability and robustness.

Finally, the policy network weights are updated using the following parameter update rule:

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \nabla_\theta L^{\text{TOPPO}}(\theta), \tag{37}$$

where α is the learning rate, which determines the step size for each update.

The TOPPO algorithm, through stable policy updates and efficient value evaluation, effectively adapts to dynamic environmental changes in AIGC task offloading. By incorporating the LSTM module, it efficiently captures temporal dependencies in complex and dynamic task offloading environments, optimizing offloading decisions. Through continuous interaction with the environment, it learns improved policies and value estimates, leading to significant improvements in task offloading decisions in terms of latency, energy consumption, and load balancing.

The pseudocode for the AIGC task offloading algorithm TOPPO is shown in Algorithm 1.

Algorithm 1 TOPPO Algorithm for AIGC Task Offloading

Initialize ActorLSTM parameters θ and CriticLSTM parameters ϕ
for each episode $e = 1$ to $e_{\max}$ **do**
  Reset the AIGC task offloading environment to obtain initial state s_0
  Initialize actions for all Intelligent devices (InDes)
  while not done **do**
    for each InDe m in InDes **do**
      Pass state $s_{m,t}$ to the ActorLSTM network
      Compute action probability $\pi_\theta(a_{m,t}|s_{m,t})$ and sample action $a_{m,t}$
      Execute action $a_{m,t}$, obtain reward $r_{m,t}$, and observe next state $s_{m,t+1}$
      Store current observation $\langle s_{m,t}, a_{m,t}, r_{m,t}, s_{m,t+1} \rangle$
      Pass the current state s_t to CriticLSTM
      Compute state value $V_\phi(s_t)$ and $V_\phi(s_{t+1})$
      Use in Eq.32 to compute temporal-difference error
      Update CriticLSTM using Eq.33
      Update ϕ in Eq.34 using the gradient descent
      Use Eq.35 to compute Generalized advantage estimations A_t
      Use the clipped objective function $L^{\text{TOPPO}}(\theta)$ in Eq.36 to update ActorLSTM
      Update θ using the gradient descent method in Eq.37.
    end for
  end while
end for

C. Computational Complexity Analysis

To evaluate the operational efficiency and resource requirements of the TOPPO algorithm at different scales, we conducted a computational complexity analysis. In Algorithm 1, the complexity of initializing the parameters θ and ϕ of the ActorLSTM and CriticLSTM networks is $\mathcal{O}(1)$. This is a constant initialization operation and can be ignored.

During the episode loop, we iterate over each episode, from $e = 1$ to $e_{\max}$, with a linear complexity of $\mathcal{O}(e_{\max})$, which depends on the number of episodes. In each episode, the TOPPO algorithm resets the task offloading environment and obtains the initial state s_0. The computational complexity also needs to consider the T time steps within each episode, so the complexity of the time step loop is $\mathcal{O}(T)$.

Additionally, at each time step, actions are selected and network updates are performed on M intelligent devices (InDes), resulting in a device loop complexity of $\mathcal{O}(M)$. For each intelligent device, the ActorLSTM forward pass, action sampling, and execution have a complexity of $\mathcal{O}(1)$, advantage function calculation, and network updates via backward propagation are performed. We assume the complexity of the ActorLSTM network update operation is $\mathcal{O}(D)$. Since both the ActorLSTM network and CriticLSTM network contain network update operations in the TOPPO algorithm, the complexity is $2 \times \mathcal{O}(D)$.

In summary, the total computational complexity of the TOPPO algorithm is:

$$\mathcal{O}(2e_{\max}TMD),$$

where, $\mathcal{O}(D) = \mathcal{O}(H^2 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3)$, $\mathcal{O}(D)$ includes the complexity of one LSTM layer and three fully connected layers. H represents the number of hidden units in the LSTM, a is the input dimension of each fully connected layer, and b is the output dimension of each fully connected layer.

V. RESULT AND ANALYSIS

In this section, we first introduce the experimental setup, followed by the evaluation and analysis of the performance of the proposed TOPPO method. Finally, we compare and analyze the proposed method against several benchmark algorithms.

A. Experimental Setup and Application Scenario

We simulated an intelligent factory scenario to evaluate the DRL-based AIGC task offloading s trategy. T he experimental environment includes two types of smart devices: production line controllers and high-resolution defect detection cameras. These devices have computational capabilities ranging from $3 \sim 5$ GHz and can generate AIGC tasks based on production demands. For example, production line controllers process real-time operational data to generate detailed production process reports, including critical data such as production speed, temperature, and quality metrics, as well as predictive analysis based on historical data. Defect detection cameras analyze high-resolution product images to generate annotated defect analysis reports, identifying potential defects (e.g., surface scratches, misalignment) and providing recommendations for equipment adjustments.

AIGC tasks generally have high computational density and large data volume characteristics. Therefore, the computational density $\rho_{m,t}, m \in M, t \in T$ is set in the range of [2000, 3000] Megacycles to address the computational needs of complex content generated by AIGC. Meanwhile, the factory's internal network is connected via a high-speed industrial wireless network with a transmission rate of 14 Mbps, efficiently supporting task offloading. T he t ask d ata s ize $x_{m,t}$ ranges from $2 \sim 8$ Mbits, covering the diverse output requirements of generated text reports and annotated defect images. The main parameter settings are summarized in Table 3.

TABLE III: Parameters settings of simulation experiments

Parameter	Value
M	{20, 30, 40, 50, 60}
N	8
Δt	0.1s
$f_m^{InDe}, \forall m \in M$	[3.0, 5.0] GHz
$f_n^{edge}, \forall n \in N$	[30, 41.8] GHz
$r_{m,n}^{trans}, \forall m \in M, \forall n \in N$	14 Mbps
$x_{m,t}, \forall m \in M, \forall t \in T$	[2.0, 10.0] Mbits
$\rho_{m,t}, \forall m \in M, \forall t \in T$	[2000, 3000] Megacycles
P_{wait}^{InDe} and P_{wait}^{edge}	0.1 W [32]
$P_{m,n}^{trans}, \forall m \in M, \forall n \in N$	2.3 W [32]
AIGC-Task arrival probability	0.6

The processing of each AIGC task is optimized by a dynamic scheduling strategy driven by TOPPO. After task generation, the device determines whether to execute the task locally or offload it to an edge server based on the current computational load, the number of tasks, and network conditions. The TOPPO selects the optimal task processing strategy by evaluating parameters such as device energy consumption and edge server resource availability in real-time. Offloaded tasks are transmitted to the most suitable edge server for processing, and the results are promptly returned to the originating device. For tasks executed locally, the device leverages its limited computational resources to ensure timely completion, achieving a dynamic balance among latency, energy consumption, and CPU utilization.

B. Performance and Convergence Analysis of The Proposed Algorithm

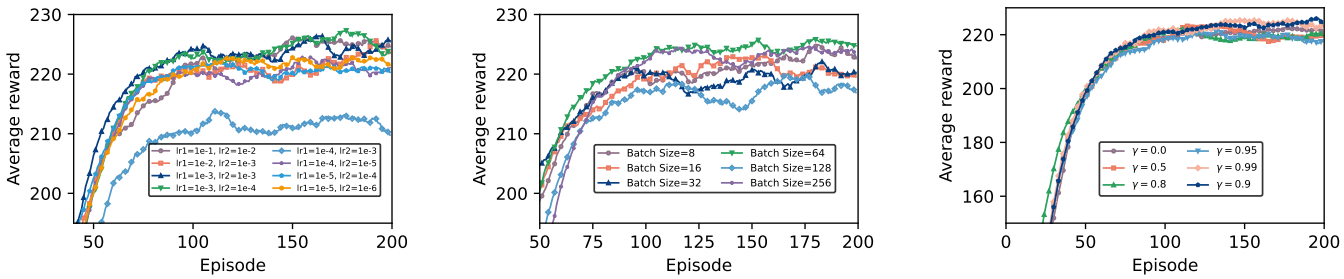
Although the convergence of the PPO algorithm has already been thoroughly demonstrated in theoretical terms in existing work [34], our proposed TOPPO algorithm introduces an LSTM network structure within both the policy and value networks. This addition brings extra state variables and increases the number of parameters (e.g., hidden layer dimensions, memory cell size), making gradient calculations more complex and requiring careful hyperparameter tuning to ensure the convergence of the proposed algorithm. Improper hyperparameter selection could lead to unstable convergence or even failure to converge. Therefore, We conducted experiments under a scenario involving 30 AIGC InDes ($M = 30$) and 8 ESs ($N = 8$) to validate the convergence of the proposed algorithm. Additionally, we analyzed the impact of key hyperparameters—learning rate (lr), batch size, and discount factor γ—on the convergence behavior, observing the stability and speed of convergence across various configurations.

In the proposed TOPPO, the policy network determines the optimal task offloading decisions, deciding whether to offload AIGC tasks and to which edge server. Meanwhile, the value network estimates the long-term rewards of offloading decisions, ensuring efficient utilization of system resources. For the TOPPO algorithm to converge effectively, certain hyperparameters play a critical role in balancing convergence speed and stability. One of these key hyperparameters is the learning rate, which controls the update speed for the policy and value networks. If the learning rate is set too high, the offloading strategy may become unstable; if it is too low, the optimization process could slow, risking premature convergence at suboptimal points. Similarly, Larger batch sizes produce more stable updates, promoting smooth convergence to an optimal offloading strategy, while smaller batch sizes may accelerate convergence but with increased risks of instability and oscillations in the learning process. By analyzing these hyperparameters, we aim to refine the TOPPO algorithm's convergence, enabling it to reach a stable and optimal solution effectively.

To determine the optimal learning rate (lr) and batch size for the AIGC task offloading problem, we conducted a series of experiments. The experimental results are shown in Figure 3. lr1 represents the learning rate of the policy network, and lr2 represents the learning rate of the value network. As seen in figure 3(a), when lr1=1e-3 and lr2=1e-3, or lr1=1e-3 and lr2=1e-4, TOPPO achieves higher rewards compared to other configurations. However, in the early stages of training, lr1=lr2=1e-3 demonstrates faster convergence. Therefore, we set the learning rates for both the policy and value networks to 1e-3 to balance rapid and stable convergence.

As show in figure 3(b), a batch size of 64 produces the highest reward, optimizing energy consumption, delay and load for AIGC task offloading. This batch size also leads to smooth convergence, avoiding the fluctuations seen with smaller batch sizes and promoting efficient learning. As a result, we selected a batch size of 64 for subsequent experiments.

The discount factor γ in deep reinforcement learning is

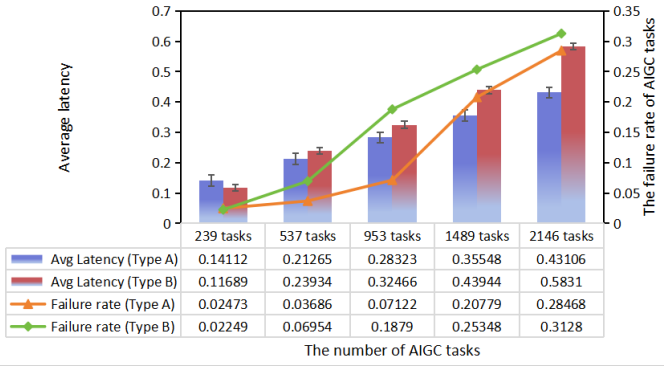


(a) Learning rate (b) Batch size (c) Discount Factor

Fig. 3: Convergence of the proposed algorithm under different key hyperparameters

crucial for balancing immediate rewards with long-term returns. It determines the extent to which an agent considers future rewards when making decisions. In the context of AIGC task offloading, selecting an appropriate γ helps the agent balance immediate computation delays with long-term system performance in offloading decisions. Experiment results shown in Figure 3(c) illustrate that when $\gamma = 0$, TOPPO focuses solely on immediate rewards, entirely ignoring future returns, leading to quicker, short-sighted convergence but lower overall performance. At $\gamma = 0.5$, the agent prioritizes short-term performance but shows signs of more balanced convergence. At $\gamma = 0.9$, TOPPO achieves a balance between present rewards and future returns, leading to optimal performance and consistent convergence. Finally, at $\gamma = 0.99$, TOPPO focuses almost entirely on long-term rewards, which promotes convergence towards an optimal solution for long-term performance, albeit with slower convergence in initial stages.

To explore the impact of the number of production process report generation tasks and defect analysis report generation tasks on the optimization metrics of TOPPO (average latency and task failure rate), we conducted extensive experiments by varying the number of tasks generated within 100 timestamps. As shown in Figure 4, production process report generation tasks are represented as type A AIGC tasks, with each task size ranging from 2 to 4 Mbits, while defect analysis report generation tasks are defined as type B AIGC tasks, with each task size ranging from 5 to 8 Mbits. The average latency and task failure rate for both production process report generation tasks and defect analysis report generation tasks increased as the number of tasks grew.



	239 tasks	537 tasks	953 tasks	1489 tasks	2146 tasks
Avg Latency (Type A)	0.14112	0.21265	0.28323	0.35548	0.43106
Avg Latency (Type B)	0.11689	0.23934	0.32466	0.43944	0.5831
Failure rate (Type A)	0.02473	0.03686	0.07122	0.20779	0.28468
Failure rate (Type B)	0.02249	0.06954	0.1879	0.25348	0.3128

Fig. 4: Performance of Average Latency and Failure Rate for AIGC Tasks (Type A and Type B)

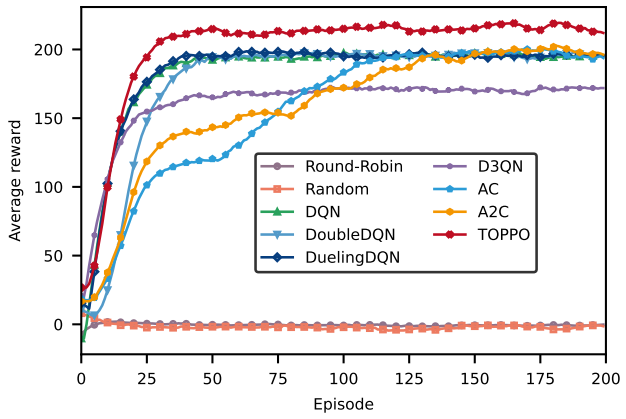


Fig. 5: Comparison of Average Reward Across Different Methods

C. Compare Methods

To highlight the practicality and effectiveness of deep reinforcement learning-based methods for AIGC task offloading optimization, we compared them with traditional methods such as Round-Robin and Random. Additionally, to demonstrate the superior optimization capability of the proposed TOPPO algorithm, we conducted comparisons with other DRL-based methods, including DQN [35], Double DQN [36], Dueling DQN [37],D3QN [38], AC [39], and A2C [32]. Experimental results show that the TOPPO algorithm exhibits strong representativeness and comparability.

Our evaluation metrics include energy consumption, delay, load, and task failure rate. By considering these four key metrics for AIGC services, we aim to derive the optimal AIGC task offloading and scheduling solution.

The reward value is a comprehensive metric that integrates four performance indicators: energy consumption, delay, load, and task failure rate. As shown in Figure 5, the reward values of our proposed TOPPO are better than those of the other eight baseline algorithms. As shown in Table IV, our method

achieves approximately a 100% improvement compared to traditional methods like Round-Robin and Random, demonstrating significant enhancements. When compared to similar deep reinforcement learning algorithms, TOPPO shows improvements of 8.36% over DQN, 12.08% over Double DQN, 8.43% over Dueling DQN, 19.97% over D3QN, 11.61% over AC, and 8.82% over A2C. Additionally, TOPPO converges faster, reaching a near-optimal solution when the episode count reaches 40.

The rapid convergence of TOPPO highlights its efficiency in adapting to the offloading environment and finding optimal task allocations quickly. In contrast, the slower convergence of AC and A2C indicates higher sensitivity to parameter tuning and less stability in the early training stages, whereas TOPPO's structured learning approach enables stable convergence even with relatively fewer episodes.

TABLE IV: Comparison of different algorithms on various objectives and rewards

Algorithm	**Objective**				**Reward**
	Delay	**Energy consumption**	**Dropout**	**Loading**	
Round-Robin	0.781	3869.83	0.160	0.1248	-0.29
Random	0.830	4118.00	0.163	0.1138	-5.77
DQN	0.337	1404.67	0.071	0.7370	193.73
DoubleDQN	0.297	1196.33	0.073	0.7459	189.39
DuelingDQN	0.285	1120.17	0.075	0.7375	192.12
D3QN	0.401	1744.00	0.083	0.6510	171.43
AC	0.279	1031.17	0.074	0.7400	182.08
A2C	0.276	1032.00	0.078	0.7230	189.05
TOPPO	**0.265**	**984.17**	**0.063**	**0.7925**	**205.99**

In the experimental evaluation of energy consumption for AIGC task offloading, as illustrated in Figure 6(a), the proposed TOPPO algorithm demonstrated clear advantages over all baseline algorithms. Specifically, compared to the traditional Round-Robin and Random algorithms, TOPPO reduced energy consumption by 74.57% and 76.10%, respectively. When compared with reinforcement learning-based methods such as DQN, DoubleDQN, and DuelingDQN, TOPPO achieved energy savings of 29.94%, 17.73%, and 12.14%, respectively. Furthermore, TOPPO outperformed actor-critic-based methods, offering a 4.56% reduction in energy consumption compared to AC and a 4.64% reduction compared to A2C. These results highlight the superior energy efficiency of TOPPO, which consistently minimizes energy consumption across both traditional and advanced baselines, making it a highly effective solution for AIGC task offloading in energy-constrained edge computing environments.

As illustrated in Figure 6(b), the proposed TOPPO algorithm shows significant improvements in delay optimization for AIGC task offloading, outperforming all baseline methods. In particular, compared to traditional scheduling methods like Round-Robin and Random, TOPPO achieves a remarkable reduction in delay by 66.11% and 68.09%, respectively, demonstrating its ability to substantially reduce latency in task offloading scenarios. Among more advanced deep reinforcement learning-based methods, TOPPO also delivers considerable delay savings, reducing delay by 21.48% compared to DQN, 10.91% compared to DoubleDQN, and 7.09% compared to DuelingDQN. These results highlight TOPPO's efficiency in managing the dynamic nature of task offloading while maintaining low latency. Additionally, TOPPO surpasses actor-critic-based methods such as AC and A2C. Although the delay reduction compared to AC (4.84%) and A2C (3.96%) is relatively smaller, TOPPO still optimizes performance, demonstrating its robustness across different algorithm types. In conclusion, TOPPO consistently reduces delays across a range of algorithms, making it a highly effective solution for AIGC task offloading and ensuring efficient task processing in latency-sensitive environments.

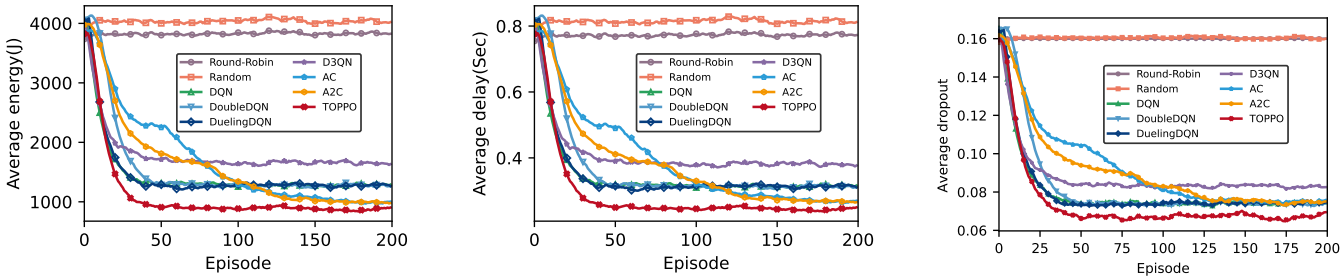
As shown in Figure 6(c), the proposed TOPPO algorithm has a lower AIGC task failure rate compared to the other 8 baseline algorithms. This indicates that our method effectively improves the success rate of AIGC task completion.

We conducted a comparative experiment to evaluate the CPU utilization before and after load optimization using TOPPO. The experimental results, as shown in Figure 7, demonstrate that in the early stages before TOPPO optimization, the number of AIGC tasks executed locally on intelligent devices was relatively low, leaving many devices underutilized, resulting in a suboptimal CPU utilization of only 42.57%. However, after applying TOPPO optimization, there was a significant improvement in load distribution, with the average CPU utilization increasing to 79.25%.

To verify the generalization ability and stability of the proposed method, we increased the number of AIGC service intelligent devices M (i.e., the number of AIGC tasks) and conducted comparative experiments against 8 baseline algorithms. As shown in Figure 8, when the number of tasks is small, the AC series algorithms outperform the proposed TOPPO algorithm. However, when $M \geq 30$, the average reward of the TOPPO method consistently exceeds that of the 8 baseline algorithms. This demonstrates that the proposed algorithm can effectively generalize to offloading and scheduling scenarios with a larger number of AIGC service devices.

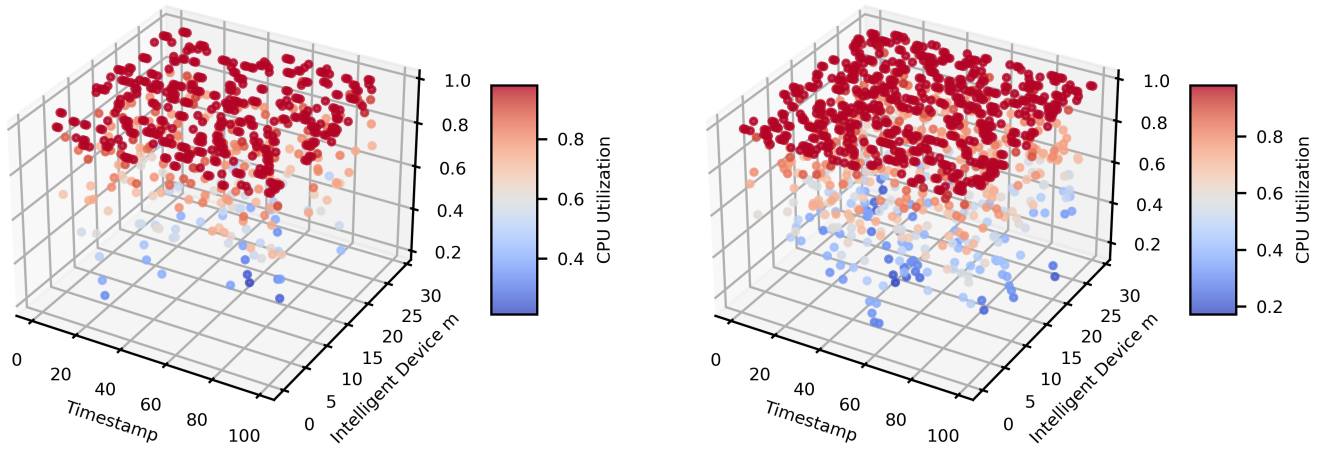
To validate the robustness of the TOPPO method across heterogeneous computational capacities, we incorporated a series of experiments where we varied the computational capabilities of the edge servers. As shown in Figure 9, under varying computational capacities of edge servers, TOPPO achieves superior solutions compared to other baseline algorithms across multiple metrics, including energy consumption, latency, load, and task success rate. Experimental results demonstrate that TOPPO is minimally impacted by device computational power, demonstrating strong adaptability.

The joint optimization of energy consumption and workload enables AIGC tasks to achieve low latency and high efficiency in applications such as smart homes, intelligent transportation, and smart factories. By reducing device energy consumption, operational time is extended, while workload optimization improves the system's real-time responsiveness, ensuring rapid execution of AIGC tasks. In intelligent transportation, optimized AIGC tasks enable real-time analysis and decision-making, enhancing traffic safety and efficiency. In smart factories, this optimization supports a higher number of concurrent tasks, reducing costs and enhancing automation



(a) Energy Consumption (b) Latency (c) AIGC Task Failure Rate

Fig. 6: Comparison of Energy Consumption and Latency Convergence Across Different Methods



(a) CPU Utilization - Early Stage of Optimization (b) CPU Utilization - Later Stage of Optimization

Fig. 7: Comparison of CPU Utilization Across Different Stages of TOPPO Optimization

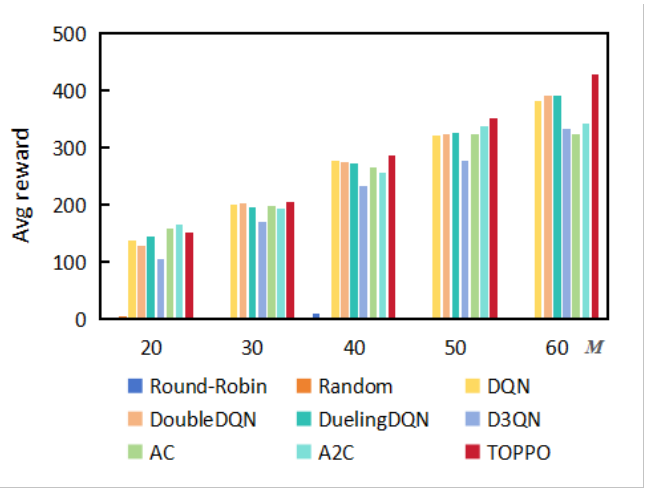
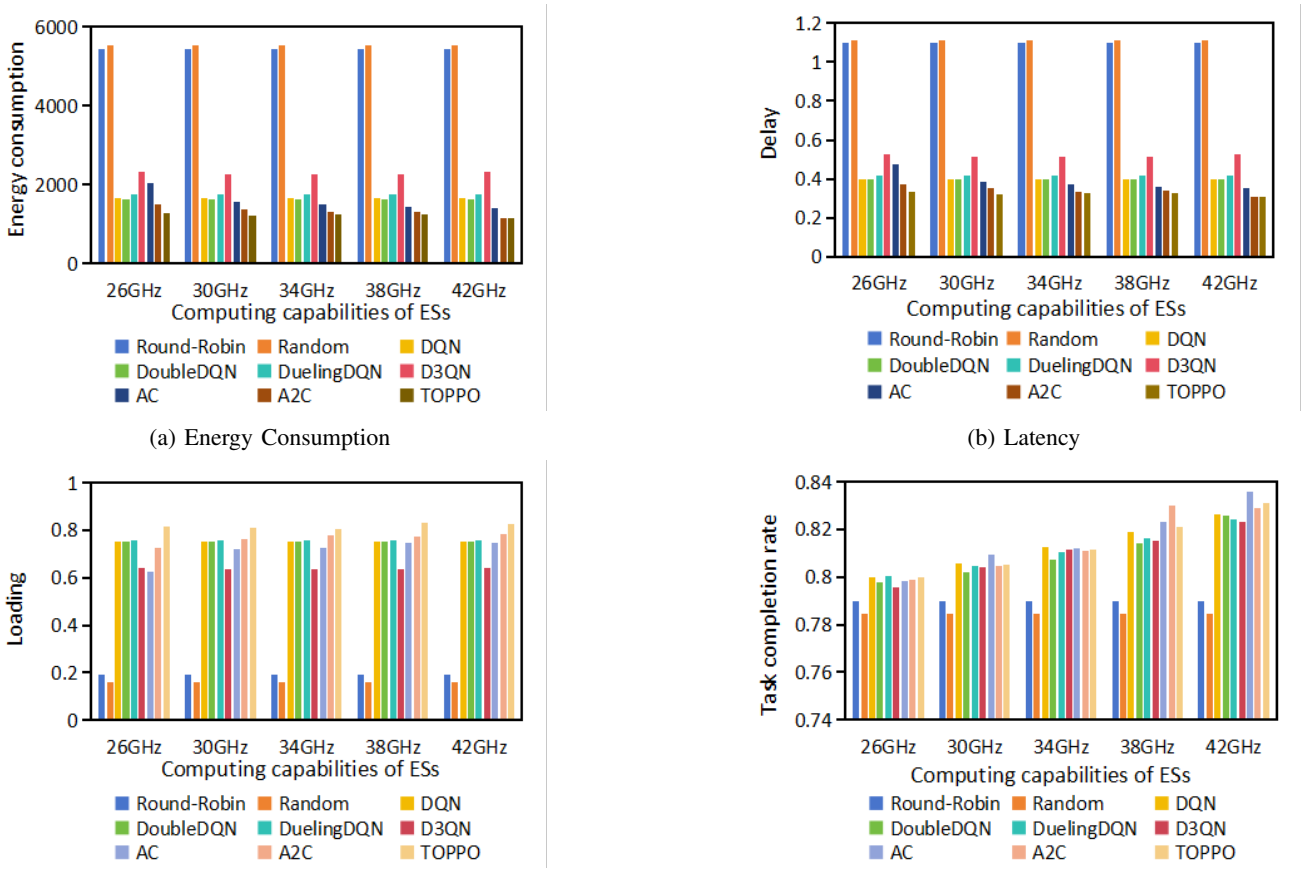


Fig. 8: Comparison of Reward for Different Methods and different number of InDe

levels. Overall, the optimization of energy consumption and workload provides a more efficient and scalable execution environment for AIGC tasks, driving intelligent applications toward sustainable development.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed an AIGC task offloading framework (AIECOF) for the IoIT-assisted ECN. Based on this framework, we formulated the AIGC task offloading problem as a complex mathematical optimization problem. To solve this, we introduced a novel AIGC task offloading algorithm (TOPPO), which integrates the efficient policy optimization of Proximal Policy Optimization (PPO) with the temporal dependency handling capabilities of Long Short-Term Memory (LSTM). The integration of LSTM not only enhances TOPPO's adaptability to dynamic environments but also optimizes energy consumption, reduces latency, and ensures load balancing in the AIGC task offloading process. When tested with $m = 30$ and $n = 8$, the TOPPO offloading algorithm delivered approximately 100% higher average cumulative rewards compared to traditional methods like RR and random algorithms. Additionally, when compared to six state-of-the-art DRL algorithms, TOPPO demonstrated significant improvements, achieving up to 43.56% energy savings and reducing latency by as much as 33.90%. Moreover, it boosted the combined average reward for energy efficiency, latency, task success rate, and loading by over 8.36%. Even as the number of AIGC tasks increased, TOPPO consistently outperformed



(a) Energy Consumption

(b) Latency

(c) Loading

(d) Task Completion Rate

Fig. 9: Comparison of different performance indicators under different computing capabilities of ESs

other methods, showcasing robust performance.

In future work, we will explore the performance of the TOPPO algorithm in larger-scale AIGC task offloading scenarios. By integrating Transformer models, TOPPO is expected to achieve new breakthroughs in adaptability and robustness for large-scale AIGC task offloading, further extending its application to broader task scheduling scenarios in edge computing.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under grant No.52275480; in part by the Guizhou Provincial Department of Science and Technology Project under grant Nos.QKHZYD[2023]002, QKHP-GCC[2023]001, QKHP-KXJZ[2024]002; and in part by the Guiyang Science and Technology Platform Construction Project under Grant No.ZKHT[2023]7; and in part by China Scholarship Council [No.202306670004].

References

[1] J. Wang, H. Du, D. Niyato, J. Kang, S. Cui, X. S. Shen, and P. Zhang, "Generative ai for integrated sensing and communication: Insights from the physical layer perspective," *IEEE Wireless Communications*, 2024.

[2] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen *et al.*, "Unleashing the power of edge-cloud generative ai in mobile networks: A survey of aigc services," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 2, pp. 1127–1170, 2024.

[3] J. Wang, H. Du, D. Niyato, Z. Xiong, J. Kang, B. Ai, Z. Han, and D. I. Kim, "Generative artificial intelligence assisted wireless sensing: Human flow detection in practical communication environments," *IEEE Journal on Selected Areas in Communications*, 2024.

[4] Y. Liu, X. Lin, S. Li, G. Li, Q. Mao, and J. Li, "Towards multi-task generative-ai edge services with an attention-based diffusion drl approach," *arXiv preprint arXiv:2405.08328*, 2024.

[5] M. Y. Akhlaqi and Z. B. M. Hanapi, "Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions," *Journal of Network and Computer Applications*, vol. 212, p. 103568, 2023.

[6] A. Naouri, H. Wu, N. A. Nouri, S. Dhelim, and H. Ning, "A novel framework for mobile-edge computing by optimizing task offloading," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 13 065–13 076, 2021.

[7] J. Wang, H. Du, Y. Liu, G. Sun, D. Niyato, S. Mao, D. I. Kim, and X. Shen, "Generative ai based secure wireless sensing for isac networks," *arXiv preprint arXiv:2408.11398*, 2024.

[8] B. Lai, J. Wen, J. Kang, H. Du, J. Nie, C. Yi, D. I. Kim, and S. Xie, "Resource-efficient generative mobile edge networks in 6g era: Fundamentals, framework and case study," *IEEE Wireless Communications*, vol. 31, no. 4, pp. 66–74, 2024.

[9] C. Xu, J. Guo, J. Zeng, S. Meng, X. Chu, J. Cao, and T. Wang, "Enhancing ai-generated content efficiency through adaptive multi-edge collaboration," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2024, pp. 960–970.

[10] Q. Liu, Z. Tian, N. Wang, and Y. Lin, "Drl-based dependent task offloading with delay-energy tradeoff in medical image edge computing," *Complex & Intelligent Systems*, pp. 1–22, 2024.

[11] B. He, H. Li, and T. Chen, "Drl-based computing offloading approach for large-scale heterogeneous tasks in mobile edge computing," *Concurrency and Computation: Practice and Experience*, vol. 36, no. 19, p. e8156, 2024.

[12] K. Jia, H. Xia, R. Zhang, Y. Sun, and K. Wang, "Multi-agent drl for edge computing: A real-time proportional compute offloading," *Computer*

Networks, vol. 252, p. 110665, 2024.
[13] J. Lu, J. Yang, S. Li, Y. Li, W. Jiang, J. Dai, and J. Hu, "A2c-drl: Dynamic scheduling for stochastic edge-cloud environments using a2c and deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 11, no. 9, pp. 16 915–16 927, 2024.
[14] M. K. Mondal, S. Banerjee, D. Das, U. Ghosh, M. S. Al-Numay, and U. Biswas, "Towards energy-efficient and cost-effective task offloading in mobile edge computing for intelligent surveillance systems," *IEEE Transactions on Consumer Electronics*, 2024.
[15] T. Baker, Z. Al Aghbari, A. M. Khedr, N. Ahmed, and S. Girija, "Editors: Energy-aware dynamic task offloading using deep reinforcement transfer learning in sdn-enabled edge nodes," *Internet of Things*, vol. 25, p. 101118, 2024.
[16] Z. Gao, L. Yang, and Y. Dai, "Large-scale computation offloading using a multi-agent reinforcement learning in heterogeneous multi-access edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 6, pp. 3425–3443, 2022.
[17] J. Ren, T. Hou, H. Wang, H. Tian, H. Wei, H. Zheng, and X. Zhang, "Collaborative task offloading and resource scheduling framework for heterogeneous edge computing," *Wireless Networks*, vol. 30, no. 5, pp. 3897–3909, 2024.
[18] X. Yuan, Z. Zhang, C. Feng, Y. Cui, S. Garg, G. Kaddoum, and K. Yu, "A dqn-based frame aggregation and task offloading approach for edge-enabled iomt," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 3, pp. 1339–1351, 2022.
[19] C. Fang, Z. Hu, X. Meng, S. Tu, Z. Wang, D. Zeng, W. Ni, S. Guo, and Z. Han, "Drl-driven joint task offloading and resource allocation for energy-efficient content delivery in cloud-edge cooperation networks," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 12, pp. 16 195–16 207, 2023.
[20] C. Li, J. Xia, F. Liu, D. Li, L. Fan, G. K. Karagiannidis, and A. Nallanathan, "Dynamic offloading for multiuser muti-cap mec networks: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2922–2927, 2021.
[21] L. Ai, B. Tan, J. Zhang, R. Wang, and J. Wu, "Dynamic offloading strategy for delay-sensitive task in mobile-edge computing networks," *IEEE Internet of Things Journal*, vol. 10, no. 1, pp. 526–538, 2022.
[22] S. Li, X. Lin, H. Xu, K. Hua, X. Jin, G. Li, and J. Li, "Multi-agent rl-based industrial aigc service offloading over wireless edge networks," *arXiv preprint arXiv:2405.02972*, 2024.
[23] J. Zhang, Z. Wei, B. Liu, X. Wang, Y. Yu, and R. Zhang, "Cloud-edge-terminal collaborative aigc for autonomous driving," *IEEE Wireless Communications*, vol. 31, no. 4, pp. 40–47, 2024.
[24] D. Wang, H. Zhu, C. Qiu, Y. Zhou, and J. Lu, "Distributed task offloading in cooperative mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 7, pp. 10 487–10 501, 2024.
[25] W. Zeng, J. Zheng, H. Wang, Q. Sun, R. Cao, S. Ji, J. Ren, and L. Gaol, "Delay-aware parallel offloading aigc service in edge computing," in *2024 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*. IEEE, 2024, pp. 208–213.
[26] F. Ramezani Shahidani, A. Ghasemi, A. Toroghi Haghighat, and A. Keshavarzi, "Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm," *Computing*, vol. 105, no. 6, pp. 1337–1359, 2023.
[27] C. Xu, "Phd forum abstract: Diffusion-based task scheduling for efficient ai-generated content in edge networks," in *2024 23rd ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2024, pp. 333–334.
[28] W. Cheng, X. Liu, X. Wang, and G. Nie, "Task offloading and resource allocation for industrial internet of things: A double-dueling deep q-network approach," *IEEE Access*, vol. 10, pp. 103 111–103 120, 2022.
[29] H. Li, K. D. R. Assis, S. Yan, and D. Simeonidou, "Drl-based long-term resource planning for task offloading policies in multiserver edge computing networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4151–4164, 2022.
[30] L. Zhao, E. Zhang, S. Wan, A. Hawbani, A. Y. Al-Dubai, G. Min, and A. Y. Zomaya, "Meson: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 4259–4272, 2023.
[31] I. Ullah, H.-K. Lim, Y.-J. Seok, and Y.-H. Han, "Optimizing task offloading and resource allocation in edge-cloud networks: a drl approach," *Journal of Cloud Computing*, vol. 12, no. 1, p. 112, 2023.
[32] K. Zhu, S. Li, X. Zhang, J. Wang, C. Xie, F. Wu, and R. Xie, "An energy-efficient dynamic offloading algorithm for edge computing based on deep reinforcement learning," *IEEE Access*, vol. 12, pp. 127 489–127 506, 2024.
[33] J. Feng, L. Liu, Q. Pei, and K. Li, "Min-max cost optimization for efficient hierarchical federated learning in wireless edge networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2687–2700, 2021.
[34] M. Holzleitner, L. Gruber, J. Arjona-Medina, J. Brandstetter, and S. Hochreiter, "Convergence proof for actor-critic methods applied to ppo and rudder," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLVIII: Special Issue In Memory of Univ. Prof. Dr. Roland Wagner*. Springer, 2021, pp. 105–130.
[35] M. Ibrahim, A. Alsheikh, and R. Elhafiz, "Resiliency assessment of power systems using deep reinforcement learning," *Computational Intelligence and Neuroscience*, vol. 2022, no. 1, p. 2017366, 2022.
[36] A. Staffolani, V.-A. Darvariu, P. Bellavista, and M. Musolesi, "Rlq: Workload allocation with reinforcement learning in distributed queues," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 3, pp. 856–868, 2023.
[37] A. Chraibi, S. Ben Alla, A. Touhafi, and A. Ezzati, "A novel dynamic multi-objective task scheduling optimization based on dueling dqn and per," *The Journal of Supercomputing*, vol. 79, no. 18, pp. 21 368–21 423, 2023.
[38] H. Hu, D. Wu, F. Zhou, X. Zhu, R. Q. Hu, and H. Zhu, "Intelligent resource allocation for edge-cloud collaborative networks: A hybrid ddpg-d3qn approach," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 8, pp. 10 696–10 709, 2023.
[39] B. Wang, L. Liu, and J. Wang, "Task offloading optimization based on actor-critic algorithm in vehicle edge computing," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2023, pp. 687–692.