Multilevel and Energy-Efficient Partial Computation Offloading in Heterogeneous Edge Intelligence

Baoyu Xu, Yancheng Ruan, Chenghu Qiu, Shuibing He, *Member, IEEE*, Feng Shu, *Senior Member, IEEE*, Xiaoyang Kang, *Member, IEEE*, and Lihua Zhang, *Member, IEEE*

***Abstract*—Due to the diversity of edge devices (EDs) and applications, edge systems are heterogeneous and have been applied in artificial intelligence fields, such as smart factories and intelligent transportation, which is called heterogeneous edge intelligence. Many studies employ computation offloading to transfer processing data from resource-scarce EDs to resource-rich edge servers. These studies primarily focus on the overall resource consumption of homogeneous edge systems, neglecting the system heterogeneity and the details of resource consumption. In this article, we construct a system model from a parallel perspective for the heterogeneous edge system with different processors, memory, and applications, which perceives the cost of energy and delay from three levels: system, application, and component. A hybrid metaheuristic algorithm combined with a greedy rule, hybrid mutation, and whale optimization algorithm (GHMWOA) is proposed to realize partial computation offloading. A partial offloading architecture of heterogeneous edge intelligence is proposed to validate our model and algorithm with real-world hardware and software. Experiment results not only show GHMWOA outperforms multiple classical optimization algorithms in minimizing energy consumption, but also discover on which system component energy consumption depends, and how properties of application and system influence the cost of energy.**

***Index Terms*—Heterogeneous edge intelligence, multilevel energy analysis, parallel processing, partial computation offloading.**

Received 25 September 2024; revised 17 December 2024; accepted 30 December 2024. Date of publication 13 January 2025; date of current version 23 May 2025. This work was supported by the Shanghai Municipal Science and Technology Major Project under Grant 2021SHZDZX0103. *(Corresponding authors: Lihua Zhang; Xiaoyang Kang.)*

Baoyu Xu is with the Academy for Engineering and Technology, Fudan University, Shanghai 200433, China, and also with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China (e-mail: byxu@shu.edu.cn).

Yancheng Ruan and Chenghu Qiu are with the School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China (e-mail: 18918694350@shu.edu.cn; 2532725615@shu.edu.cn).

Shuibing He is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: heshuibing@zju.edu.cn).

Feng Shu is with the School of Electrical and Information Engineering, University of Sydney, Sydney, NSW 2050, Australia (e-mail: feng.shu@outlook.com).

Xiaoyang Kang is with the Laboratory for Neural Interface and Brain Computer Interface, MOE Frontiers Center for Brain Science, Academy for Engineering and Technology, Fudan University, Shanghai 200433, China (e-mail: xiaoyang_kang@fudan.edu.cn).

Lihua Zhang is with the Academy for Engineering and Technology, Fudan University, Shanghai 200433, China (e-mail: lihuazhang@fudan.edu.cn).

Digital Object Identifier 10.1109/JIOT.2025.3529185

I. Introduction

EDGE intelligence leverages artificial intelligence (AI) for edge inference or edge training on edge systems, enhancing the capabilities of edge systems in data processing and privacy protection, etc. [1] However, AI applications are often data-intensive, latency-sensitive, or energy-consuming, posing challenges for resource-constrained edge devices (EDs) with low computing power and short battery time, which cannot support the resource demands of AI applications. An effective approach to mitigate the issue is to offload data or tasks from resource-scarce EDs to resource-rich edges or clouds, which is called computation offloading.

In general, the edge system is heterogeneous. Heterogeneity is not only embodied in network architecture [2], [3], but also exists in hardware [4], such as processors and memory, as well as in software [5], including applications and operating systems. Most research related to computation offloading mainly focuses on the differences in network architectures and rarely takes that of hardware and software into account. The general approach adopted by these studies is to only use the CPU as the processor, while the latency and energy consumption of applications are calculated based on CPU cycles [6], [7], [8], [9]. In practical edge systems, however, processors are not limited to CPUs and include GPUs, ARM processors (which are commonly used in mobile devices), and other types of accelerators or coprocessors. As a result, neglecting the heterogeneity of edge systems will construct impractical models for those systems.

In addition, ignoring the heterogeneity of edge systems in computation offloading often results in unfeasible offloading schemes based on energy consumption or delay. This phenomenon stems from the uncertain sources of energy and delay in heterogeneous systems. In terms of the delay, when an edge system processes applications, it performs both computing and memory operations. Since the fact that the transmission rate of memory is often slower than the computation rate of the processor, the latency of memory operations can be significantly lower than that of computing operations. Therefore, it is unreasonable to calculate the delay of the computing operations alone. Considering the energy consumption, when processors are in an idle state, a system still consumes a certain amount of energy (called static energy in our work). Therefore, identifying the sources of delay and energy consumption is crucial for designing a more accurate offloading scheme for the heterogeneous edge system.

In this article, we attempt to build a more practical and accurate system model and design an offloading algorithm to minimize energy consumption for heterogeneous edge intelligence. Here are some difficulties. The energy and delay are consumed by different components of the edge system, such as processor, memory, and network, but these components often interact with each other, which makes the calculation of energy consumption and delay more complex. For example, the actual computing capability of the processor often depends on the transmission performance of memory. Therefore, it is inaccurate to calculate the delay based solely on the processor's computing capability. Furthermore, it is challenging to design a partial offloading architecture for heterogeneous edge intelligence and obtain real-time data on energy consumption from real-world components during the offloading process. The objective function of offloading optimization is usually nonlinear or discontinuous, which leads to high complexity and requires effective optimization algorithms. In our work, contributions to address these difficulties as follows.

1) The roofline method is adopted to analyze the actual computing ability of heterogeneous EDs. Based on this, we design a heterogeneous edge system model (HESM) from a parallel perspective. The model takes into account the energy and delay consumed in processors, memory, network, and other components and addresses the independence of these components. Ultimately, the model can perceive the cost of energy and delay from three levels: system, application, and component.
2) A partial offloading algorithm greedy rule, hybrid mutation (HM), and whale optimization algorithm (GHMWOA) based on the whale optimization algorithm (WOA) is designed to obtain an optimal offloading decision. The algorithm leverages the excellent search performance of WOA, adopts penalty functions to eliminate constraints, and utilizes a greedy rule and HM to further enhance the offloading performance.
3) A partial offloading architecture for heterogeneous edge intelligence is proposed. For EDs, an energy profiling method is designed to acquire energy parameters and has been implemented on NVIDIA Jetson devices. The work provides theoretical and practical foundations for the application of HESM and GHMWOA in heterogeneous edge intelligence.
4) The test data comes from diverse deep learning (DL) models and heterogeneous EDs in the real world. Through comparisons with multiple classical algorithms, GHMWOA achieves better performance in minimizing system energy (SE) consumption and maintaining the stability of optimal values. In addition, the data from three-level energy consumption and delay facilitate the design and improvement of the system.

The remainder of this article is organized as follows. Section II introduces related work. Section III gives the HESM in detail. Section IV proposes the GHMWOA. Section V presents a partial offloading architecture for heterogeneous edge intelligence. Experimental results are shown in Section VI. Finally, conclusions are provided.

II. Related Work

A. Computation Offloading in Heterogeneous Edge

Due to the diversity of EDs, edge servers (ESs), applications, and networks, edge computing systems are often heterogeneous. Edge intelligence adopting AI applications is a special scenario of edge computing. Currently, most work related to computation offloading in edge computing is based on a homogeneous software and hardware environment. Next, we will introduce the related work in edge computing, including edge intelligence.

In [5], [6], [7], [8], [9], [10], [11], [12], [13], and [14], computation offloading is studied within a given terminal-edge-cloud or terminal-edge network structure. These works either do not focus on processor components or treat them as homogeneous devices. Typically, the CPU is considered the sole processing unit (PU), and the cost of computation offloading is calculated based on CPU cycles. In addition, Ren et al. [2] focused on the time-varying characteristics of the network and construct a system model for hybrid networks. Yu et al. [3] considered the heterogeneous network of aerial edge computing (AEC).

In literature [15], general-purpose servers and field-programmable gate arrays (FPGAs) are adopted at the edge to provide computing resources for edge services. This work designs offloading models and algorithms for servers and FPGAs to achieve offloading. The work does not take into account the differences of EDs, and FPGAs are often designed for specific applications. Works of [13] and [16] abstract the processor performance as the ability to process frames per unit time and determine the energy consumption based on the number of frames. The method can build a unified mathematical model for heterogeneous processors, but it is only applicable to frame-based edge computing scenarios and does not address the interference between different components in heterogeneous edge systems. Consequently, the method can not discover the sources of energy consumption in general heterogeneous edge systems.

Some studies focus on heterogeneous applications. Lin et al. [5] considered that tasks from different applications have diverse resource demands, particularly storage needs, and proposes an application-aware offloading algorithm to achieve minimal latency. In [12], tasks are classified into dedicated tasks and generic tasks. Dedicated tasks are urgent tasks that can not be offloaded and must be executed by the mobile devices, while other tasks are considered general tasks that can be offloaded to ESs/data center. Xue et al. [17] classified tasks into delay-sensitive and delay-tolerant categories, and builds models, respectively, employing different offloading algorithms. Notably, some studies focus on applications based on AI [8], [13], [16], [18]. In [16], the researcher focuses on the convolutional neural network (CNN) and proposes an analytics accuracy model by considering the resolution and sampling rate of the CNN model. To achieve higher accuracy, the work also designs an offloading strategy. However, the heterogeneity of edge systems is not considered in these works.

TABLE I
SUMMARY OF RELATED WORK ON COMPUTATION OFFLOADING IN EDGE COMPUTING SYSTEM

Ref.	Heterogeneous	Optimization method	Optimization objective	Edge intelligence
[2]	Network	Stochastic gradient descent	Energy consumption	No
[3]	Network	Deep reinforcement learning	Delay	No
[5]	Application	Branch-and-bound method	Energy consumption	No
[6]	No	Reinforcement learning	Communication cost and delay	No
[7]	No	Deep reinforcement learning	Task processing delay	No
[8]	No	Branch-and-bound method	Delay	**Yes**
[9]	No	Ant colony optimization	Delay and energy consumption	No
[10]	No	Deep Q-network (DQN)	The reward based on the distance of users	No
[11]	No	Deep reinforcement learning	Energy consumption	No
[12]	Application	Approach based on KKT	Energy consumption	No
[13]	No	Simulated Annealing (SA)	Delay and frame rate	**Yes**
[14]	No	A belief-based method	Delay	No
[15]	FPGA	Relaxation and rounding	Delay	No
[16]	No	Lyapunov optimization	Average accuracy and energy consumption	**Yes**
[17]	Application	Lyapunov optimization	Resource of MEC and cloud	No
[18]	No	A snapshot-based method	Delay	**Yes**
[19]	No	PSO, SA and DE	Energy consumption	No
[20]	No	Gene-inspired algorithm	Delay and CPU resources	No
[21]	No	Grey wolf optimization	Delay and energy consumption	No
[22]	No	Whale optimization algorithm	Energy consumption and privacy protection	No
[23]	Application	SA and DQN	Delay and energy consumption	No
ours	**Edge device and server, application**	**Greedy, hybrid mutation, and WOA**	**Energy consumption**	**Yes**

B. Computation Offloading Algorithm

In edge computing, offloading algorithms include meta-heuristic algorithm, reinforcement learning, and their variants. Additionally, some classical algorithms are also used in the offloading process, such as branch and bound, dynamic programming, etc. Our work pays more attention to meta-heuristic algorithms. A brief summary of related works is as follows.

Combining greedy algorithm and simulated annealing (SA), in [13], a meta-heuristic offloading algorithm is designed to improve the monitoring frame rate and reduce detection latency for real-time power monitoring. In [19], a hybrid meta-heuristic algorithm is employed to minimize energy consumption. In [9] ant colony optimization (ACO) is adopted to find optimal offloading decisions that achieve multiobjective optimization values for latency, energy consumption, and load balancing. In [20], a genetic-inspired meta-heuristic algorithm (GIMA) is utilized to schedule workflow tasks into resource-sufficient MEC servers. In [21], gray wolf optimization (GWO) is adopted to achieve optimization of energy consumption and latency in edge systems. In [22], a privacy-preserving computation offloading scheme based on the WOA is proposed. In [23], an SA offloading algorithm based on deep Q-network (DQN) and IoB is adopted in the medical Internet, which can significantly increase system utilization, and reduce latency and energy consumption.

Table I summarizes the related work on computation offloading in edge computing systems, with a particular focus on heterogeneous system, optimization methods, and optimization objectives. Meanwhile, we investigate AI in edge computing, aiming to validate the model and algorithms proposed in this article within a heterogeneous edge intelligence that integrates AI and heterogeneous edge systems.

III. SYSTEM MODEL AND PROBLEM FORMULATION

This section describes how to build a system model for a heterogeneous edge system, and formalizes edge offloading as an optimization problem.

TABLE II
SUMMARY OF NOTATIONS

Notation	Description
N	Number of edge edvices (EDs)
E/SE	System energy
AE	Application energy
CE	Component energy
μ_i	Proportion of bandwidth in channels for ED i
λ_i	Ratio of the amount offloaded to its total amount for application i
c_i	Peak floating-point performance of ED i
b_i	Peak memory bandwidth of ED i
ρ_i^f	Energy cost per flop in ED i
ρ_i^m	Energy cost per mop in ED i
p_i^{static}	A fixed constant power in ED i
c_j^e	Peak floating-point performance of ES j
b_j^e	Peak memory bandwidth of ES j
F_i	Number of flops in application i
M_i	Number of mops in application i
OI_i	Operational Intensity of application i
c_i^a	Attainable performance of application i executed in the edge device i
T_i^f	Times to execute the flops of application i locally
T_i^m	Times to execute the mops of application i locally
T_i^l	Local delay of appllication i
R_i	Transmission rate of ED i
d_i	Network distance from ED i to its AP
S	Total number of network channel
$T_{i,j}^f$	Times to execute flops of application i on ES j
$T_{i,j}^m$	Times to execute mops of application i on ES j
T_i^r	Remote delay of application i
T_i^t	Transmission delay application i
T_i	Delay to execute application i in edge system
C_{max}	Peak floating-point performance of edge cluster
e^{tran}	Total energy of transmission for all EDs
e^p	Total energy of data processing for all EDs

A. System Architecture

We consider an edge system that consists of N EDs and an edge cluster. The edge cluster is composed of H ESs that are interconnected through high-speed networks and provide computing and storage resources for EDs through an access point (AP). Data are transferred between EDs and ESs via a wireless network. The architecture of the system is shown in

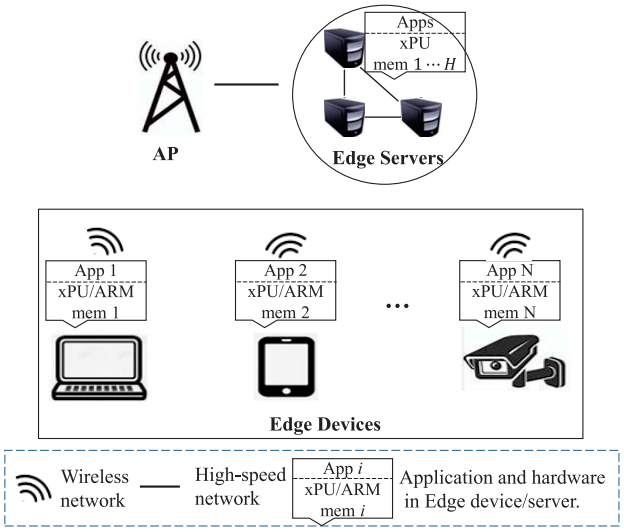


Fig. 1. Architecture of edge system. In the system, the processors and memories of EDs and ESs are heterogeneous, and EDs run different applications.

Fig. 1, where xPU is a generic term for different PUs, such as CPU and GPU.

In the system, EDs are heterogeneous in processor and memory. The energy and delay costs of their single operation are different. Each ED has five attributes and is defined as $ED_i = \{c_i, b_i, \rho_i^f, \rho_i^m, p_i^{\text{static}}\}, i \in \{1..N\}$. Let c_i and b_i denote peak floating-point performance (flops/second) and peak memory bandwidth (bytes/second). Let ρ_i^f and ρ_i^m represent energy cost of a single float-point operation (flop) and memory operation (mop), respectively. p_i^{static} denotes a fixed constant power regardless of whether ED i is idle or not. ESs are heterogeneous. Let c_j^e and b_j^e ($j \in \{1..H\}$) denote the peak floating-point performance and peak memory bandwidth of ES j.

B. Application Model

In our system, applications are heterogeneous. The application from ED i is defined as $APPi = \{F_i, M_i\}, i \in \{1..N\}$. Let F_i and M_i represent the number of flop and mop, respectively, that application i performs to process a unit of data (e.g., one frame), where M_i are equal to the total number of transferred words. To take advantage of the parallelism of EDs and ESs, all applications are data partition oriented and submitted in EDs.

Let OI_i (Flops/Bytes) denotes the operational intensity (OI) of application i, which is obtained by

$$\text{OI}_i = \frac{F_i}{M_i}. \tag{1}$$

Note OI is an intrinsic property of the application. For some applications, OI is not a constant and depends on problem size, such as FFT [24].

Let c_i^a denotes attainable performance of application i executed in the ED. According to Roofline model [24], c_i^a depends on the memory bandwidth and OI and is no higher than the peak Floating-Point performance of ED i. Therefore, c_i^a is given by

$$c_i^a = \min\{c_i, b_i \times \text{OI}_i\}. \tag{2}$$

C. Delay Model

1) Transmission Delay: In the edge system, the network adopts a frequency-division duplex mode. Let S denotes the total number of network channels, W denotes the bandwidth of the channel, respectively. The transmission rate of ED i is defined as R_i. According to the paper [19], it is given as follows:

$$R_i = S\mu_i W \log_2\left(1 + \frac{P_i^t d_i^{-v} \mid \gamma \mid^2}{N_0}\right) \tag{3}$$

where d_i denotes a distance from ED i to its serving wireless AP. v and d_i^{-v} denote pass loss exponent and path loss between ED i and its serving AP. γ denotes the channel fading coefficients. N_0 denotes white Gaussian noise power. P_i^t is transmit power of ED i. μ_i denotes the proportion of bandwidth in channels for ED i.

In time slot t, let s_i and λ_i denote the speed of sampling data and the ratio of the amount offloaded to its total amount, respectively. For ED i, the amount of downlink data is often ignored since it is much less than the amount of uplink data [25]. The amount of data uploaded is $\beta_1 \lambda_i s_i t r_i$, where β_1 is the overhead in data transmission in an uplink channel and r_i is the size of the unit data. Therefore, transmission delay is defined as T_i^t, as follows:

$$T_i^t = \frac{\beta_1 \lambda_i s_i t r_i}{R_i}. \tag{4}$$

2) Local Delay: While application i is executed, flops, mops, and network transmission overlap maximally in the best case. Suppose T_i^f and T_i^m are times to execute the flops and mops of application i locally, that is, on the ED, respectively. The local delay in the best case, call T_i^l, is

$$T_i^l = \max\{T_i^f, T_i^m, T_i^t\}. \tag{5}$$

T_i^f and T_i^m are given by the following equations:

$$T_i^f = \frac{(1-\lambda_i) s_i t F_i}{c_i^a} \tag{6}$$

$$T_i^m = \frac{(1-\lambda_i) s_i t M_i}{b_i}. \tag{7}$$

Substituting (4), (6), and (7) into formula (5), local delay can be calculated.

3) Remote Delay: When applications are offloaded to heterogeneous ESs, they should be scheduled to appropriate ESs to reduce remote delay (RD). Heterogeneity existing in ESs implies that applications exhibit different performance while they are executed in different ESs. Let $c_{i,j}^a$ denotes the attainable performance of the application i in the ES j. Similar to the ED, $c_{i,j}^a$ is given by the following equation:

$$c_{i,j}^a = \min\{c_j^e, b_j^e \times \text{OI}_i\}. \tag{8}$$

During the scheduling process of applications, using a policy of fastest-execution (FE) ES priority, ES j with the highest

$c^a_{i,j}$ is first chosen for application i to decrease RD as much as possible. With the amount of offloaded data increasing, a single ES may face overload. To maintain load balance among ESs, a method based on the greedy algorithm in the knapsack problem [26] is designed for migrating applications between ESs and is outlined as follows.

Step 1: Sort the applications on the overloaded server in descending order based on the amount of offloading data, and further sort the applications in ascending order based on the execution time per unit data.

Step 2: According to the arranged order, migrate the application that caused the ES overload, called the overloaded application, to another ES that offers the best computing performance among those capable of meeting the resource requirement of the application.

The above scheduling strategy is referred to as FEG-HES. Given application i is scheduled to ES j with FEG-HES. Let T^r_i denotes the RD of application i in the best case. T^r_i can be obtained by

$$T^r_i = \max\{T^f_{i,j}, T^m_{i,j}\} \tag{9}$$

where $T^f_{i,j}$ and $T^m_{i,j}$ are times to execute flops and mops of application i on ES j, respectively

$$T^f_{i,j} = \frac{\lambda_i s_i t F_i}{c^a_{i,j}} \tag{10}$$

$$T^m_{i,j} = \frac{\lambda_i s_i t M_i}{b^e_j}. \tag{11}$$

Considering the parallel execution of the ED and ES, the delay of application i is defined as T_i, as follows:

$$T_i = \max\{T^l_i, T^r_i + T^t_i\}. \tag{12}$$

In addition, during the offloading process, the floating-point operations offloaded to the edge cannot exceed the processing capability of the edge per unit time. Let $C_{\max}$ denotes the peak floating-point performance of the edge cluster. The following constraints need to be met:

$$\sum_{i=1}^{N} F_i \min\left(\lambda_i s_i, R_i/r_i\right) \le C_{\max}. \tag{13}$$

D. Energy Model

Different from the ES, the battery life of the EDs often is low. Therefore, we pay more attention to the energy consumption of the ED. Energy consumption of EDs mainly consists of three parts: 1) data transmission energy; 2) data processing energy; and 3) static energy. The middle is mainly generated by the two basic operations of data processing: flop and mop.

Suppose the static energy is linear in T^l_i. For all EDs, The total energy of data processing in slot t is defined as e^p, as follows:

$$e^p = \sum_{i=1}^{N} e^p_i = \sum_{i=1}^{N} ts_i(1-\lambda_i)\left(\rho^f_i F_i + \rho^m_i M_i\right) + T^l_i P^{\text{static}}_i. \tag{14}$$

The total energy of transmission for all EDs is

$$e^{tran} = \sum_{i=1}^{N}\left(P^0_i + k^t_i S \mu_i P^t_i\right) T^t_i \tag{15}$$

where P^0_i and k^t_i denote idle power and amplifier coefficient for the network module, respectively. Submitting (4) into (15), the total energy of transmission can be obtained.

Let E denotes the total energy consumed in all EDs, called SE. It is as follows:

$$E = e^p + e^{tran}. \tag{16}$$

E. Problem Formulation

According above models, the optimization problem is formulated as

$$\begin{aligned} P1: \arg\min_{\lambda,\mu} \quad & E \\ & \lambda = \{\lambda_1, \ldots, \lambda_, \ldots, \lambda_N\} \\ & \mu = \{\mu_1, \ldots, \mu_i, \ldots, \mu_N\} \end{aligned} \tag{17}$$

s.t.

$$C1: 0 \le \lambda_i \le 1, i = 1, 2, \ldots, N \tag{18}$$

$$C2: 0 \le \mu_i \le 1, i = 1, 2, \ldots, N \tag{19}$$

$$C3: \sum_{i=1}^{N} \mu_i \le 1 \tag{20}$$

$$C4: \sum_{i=1}^{N} F_i \min\left(\lambda_i s_i, R_i/r_i\right) \le C_{\max} \tag{21}$$

$$C5: T_i \le T_{\max}, i = 1, 2, \ldots, N. \tag{22}$$

For each ED, constraints 18 and 19 restrict the offloading ratio of data and the proportion of bandwidth to be variables between 0 and 1. Constraint 20 guarantees that the sum of the proportion of bandwidth of all EDs does not exceed 1, meaning that the total network bandwidth of all EDs does not exceed the system's network bandwidth. Constraint 21 ensures that the flops required for the data offloaded to the edge do not exceed the peak floating-point performance of the edge cluster. Constraint 22 ensures the delay of application i does not exceed an upper bound $T_{\max}$ which can be adjusted without affecting the optimization of energy consumption. According to (14)–(16), E is nonlinear with respect to λ and μ so the optimization problem (17) is a nonlinear programming (NLP) problem that is NP-hard in nature [27]. Based on (5), (14), and (16), E is often discontinuous, especially at the boundary of the domain, which implies that some gradient-based optimization methods may struggle to achieve optimal solutions. Therefore, we propose a hybrid metaheuristic method based on WOA to address the problem in the following section.

IV. Optimal Solution

Since $P1$ is a constrained optimal program, a penalty function method [28] is adopted to convert the constrained optimization into an unconstrained optimization.

Let x denotes a solution which is a vector of decision variables (μ and λ). $G(x)$ is defined as the total penalty of all constraints if they are not met. $G(x)$ is given as follows:

$$G(x) = \sum_{i=1}^{\pi_{\neq}} \left[\max\{0, -g_i(x)\}\right]^{\gamma_1} + \sum_{j=1}^{\pi_{=}} \mid h_j(x) \mid^{\gamma_2} \tag{23}$$

where $g_i(x)$ and $h_j(x)$ denote inequality and equality constraints in $P1$. $\pi_{\neq}$ and $\pi_{=}$ denote the number of inequality and equality constraints. The values of γ_1 and γ_2 are two positive constants and set as 1 or 2 in general. In detail, each equality constraint is converted into $h_j(x) = 0$, while each inequality constraint is converted into $g_i(x) \geq 0$. For example, constraint $C3$ is transformed into $1 - \sum_{i=1}^{N} \mu_i \geq 0$.

Let $F(x)$ denotes a modified object function as follows:

$$F(x) = E + \varphi G(x) \tag{24}$$

where φ is a large positive penalty parameter. If all constraints are met, $\varphi G(x)$ is equal to zero. Until now, an unconstrained optimization that transformed from $P1$ is expressed as follows:

$$P2 : \arg\min_{\lambda,\mu} \{F(x) = E + \varphi G(x)\}. \tag{25}$$

A. Whale Optimization Algorithm

The WOA was proposed by Mirjalili and Lewis [29]. The algorithm seeks to find the optimal solution to a problem by simulating the hunting behaviors of whales in the ocean, such as encircling prey and bubble-net attacking. Below is a detailed introduction to the WOA.

In WOA, the position of each whale represents a solution (x) and is iteratively updated to search for the globally optimal solution (x_{best}). The actions in the search process include encircling prey, bubble-net attacking, and searching for prey.

In encircling prey, the position of each whale in iteration $n+1$ is updated as follows:

$$x(n+1) = x_{\text{best}}(n) - A * |C * x_{\text{best}}(n) - x(n)|. \tag{26}$$

A and C are calculated by following equations:

$$A = (2\alpha_1 - 1) * 2\left(1 - n/\mathscr{M}\right) \tag{27}$$

$$C = 2\alpha_2 \tag{28}$$

where α_1 and α_2 are random numbers between [0, 1], and $\mathscr{M}$ is the maximum number of iterations.

In bubble-net attacking, the position of each whale in iteration $n+1$ is expressed as follows:

$$x(n+1) = |x_{\text{best}}(n) - x(n)| \exp(bl) \cos(2\pi l) + x_{\text{best}}(n) \tag{29}$$

where b is the logarithmic spiral shape parameter and l is a random number in the range of $[-1, 1]$.

When $|A| > 1$, the whale commences searching for prey. The action can be described by the following model:

$$x(n+1) = x_{\text{rand}}(n) - A * |C * x_{\text{rand}}(n) - x(n)| \tag{30}$$

where x_{rand} denotes a random whale selected from the population.

Algorithm 1 PRIBG

Input:
populization size: $\mathscr{P}$
the number of EDs: N
$ED_i, APP_i, i \in [1, N]$
Output:
Populization(pop)

for $i = 1$ to N **do**
  computing e_i^p with (14) under $\lambda_i = 0$
end for
$EDorder \leftarrow$ Sort EDs according to their e_i^p
for $k = 1$ to $\mathscr{P}$ **do**
  for $j = 1$ to $N - 1$ **do**
    $randNum[j] \leftarrow rand(0, 1)$
  end for
  $Sort(randNum)$
  $tmp \leftarrow 0$
  for $i = 1$ to $N - 1$ **do**
    $randVal[i] \leftarrow randNum[i] - tmp$
    $tmp = randNum[i]$
  end for
  $randVal[N] = 1.0 - randNum[N-1]$
  $Sort(randVal)$
  for $i = 1$ to N **do**
    $pop[k].\mu_{EDorder[i]} \leftarrow randVal[EDorder[i]]$
    $pop[k].\lambda_{EDorder[i]} \leftarrow rand(0, 1)$
  end for
end for
return pop

Algorithm 2 HM

Input:
current iteration cout: n
maximum number of iteration: $\mathscr{M}$
switching factor of mutations: sf
Output:
new position: x

if $n/\mathscr{M} \leq sf$ **then**
  $x^{SBX\_p}, x^r \leftarrow$ two random positions in the population
  $x^{SBX\_o} \leftarrow SBX(x^{SBX\_p}, x^r)$ with (31)
  $x \leftarrow x^{SBX\_o}$
else
  $x^{DE\_p}, x^{r1}, x^{r2} \leftarrow$ three random positions in the population
  $x^{DE\_o} \leftarrow DE(x^{DE\_p}, x^{r1}, x^{r2})$ with (33) and crossover
  $x \leftarrow x^{DE\_o}$
end if
return x

B. Position Random Initialization Based on the Greedy Rule

To improve the convergence speed, each whale is initialized with a feasible solution generated by a position random initialization algorithm based on a greedy rule (PRIBG).

In PRIBG, a simple greedy rule is to enable the ED with high energy consumption to offload as much data as possible to reduce the total energy consumption. Since the

Algorithm 3 GHMWOA

Input:
current iteration cout: n
maximum number of iterations: $\mathscr{M}$
switching factor of mutations: sf
Output:
optimal solution: x_{best}

```
1: Initialize population with PRIBG
2: while n ≤ M do
3:   for k = 1 to P do
4:     if rand(0, 1) < 0.5 then
5:       Get a new solution with HM
6:     else
7:       p ← rand(0, 1)
8:       if p < 0.5 then
9:         if |A| ≤ 1 then
10:          Calculate a new solution with (26)
11:        else
12:          Calculate a new solution with (30)
13:        end if
14:      else
15:        Calculate a new solution with (29)
16:      end if
17:    end if
18:  end for
19:  Evaluate the E of all solutions according to (16)
20:  Update x_best with optimal solution in iteration n
21:  n ← n + 1
22: end while
23: return x_best
```

amount of offloaded data depends on the proportion of network bandwidth for ED, PRIBG generates a set of random values within [0, 1] and ensures that their sum equals 1. Then, the larger random value is assigned to the ED with higher energy consumption. The PRIBG is shown in Algorithm 1.

In Algorithm 1, lines 1 and 2 calculate the energy consumption of each ED without offloading. Line 4 sorts all EDs in ascending order based on their energy consumption. Lines 6–15 generate N uniformly distributed random values whose sum equals 1. Line 16 sorts the values in ascending order. Line 18 assigns values to the decision variable μ of individual k using the greedy rule.

C. Hybrid Mutation

To enhance search performance by improving the diversity of solutions, We adopt simulated binary crossover (SBX) and differential evolution (DE) based on mutation to generate new solutions. A periodic switching mechanism is employed between the two mutations. How to implement the two perturbation strategies is introduced in the following.

In the SBX, two whale positions are selected randomly from the population, denoted as x^{SBX_p} (parent solution) and x^r. The offspring solution x^{SBX_o} of x^{SBX_p} is generated using the following equation:

$$x_i^{SBX\_o} = 0.5 \times \left[(1+\beta)x_i^{SBX\_p} + (1-\beta)x_i^r\right] \tag{31}$$

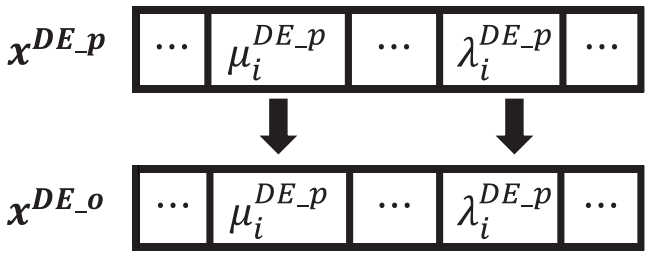


Fig. 2. Crossover with a random Gene position in DE. where x^{DE_o} is a child solution derived from x^{DE_p}, and the gene position i is randomly selected.

where β is obtained from (32). The constant dis represents the distribution factor and is set to 2 in this article

$$\beta = \begin{cases} (2 \times \text{rand}(0,1))^{1/(1+\text{dis})}, & \text{rand}(0,1) \le 0.5 \\ \left(1/(2 - 2 \times \text{rand}(0,1))\right)^{1/(1+\text{dis})}, & \text{otherwise.} \end{cases} \tag{32}$$

In the DE, three whale positions first are selected randomly from the populations, denoted as x^{DE_p} (parent solution), x^{r1}, and x^{r2}. The offspring solution x^{DE_o} of x^{DE_p} is generated according to the following formula:

$$x_i^{DE\_o} = x_i^{DE\_p} + F \times \left(x_i^{r1} - x_i^{r2}\right). \tag{33}$$

Here, F represents the scaling factor. Then, randomly select a gene from the offspring solution x^{DE_o} and replace its value with the value at the corresponding position in its parent solution x^{DE_p}, ensuring that the offspring solution retains genetic information from the parent solution. The process is called as the crossover operation and is shown in Fig. 2.

In optimization problems, adopting a constant mutation cannot guarantee the acquisition of the optimal solution, while adaptive selection of mutation requires significant time to search for the best mutation. Therefore, this article adopts an HM, which utilizes a parameter called a switching factor (sf) [30] ranging from 0 to 1 to determine the execution frequency for SBX and DE. HM is shown in Algorithm 2.

Combining PRIBG and HM, an improved WOA (GHMWOA) is presented in Algorithm 3. In the algorithm, line 1 initializes the position of each whale using PRIBG. During the search process, the position of each whale is updated with WOA or HM. Lines 10, 12, and 15 represent the actions of the whales performing encircling prey, searching for prey, and bubble-net attacking, respectively. After updating the position of each whale, Lines 19 and 20 search for the optimal position (optimal solution) of all whales in the current iteration based on their E. After $\mathscr{M}$ iterations, the algorithm obtains the optimal solution x_{best}. Next, we analyze the computational complexity of GHMWOA. In each iteration, HM requires $O(N)$ time, lines 10 or 15 consume $O(N)$ time, and lines 19 and 20 require $O(\mathscr{P})$ time. Thus, the complexity of one iteration is $O(\mathscr{P}N)$. Since the PRIBG is based on the greedy rule, its complexity is $O(\mathscr{P}N\log_2 N)$. Therefore, the computational complexity of the GHMWOA is $O(\mathscr{P}N\log_2 N + \mathscr{P}\mathscr{M}N)$.

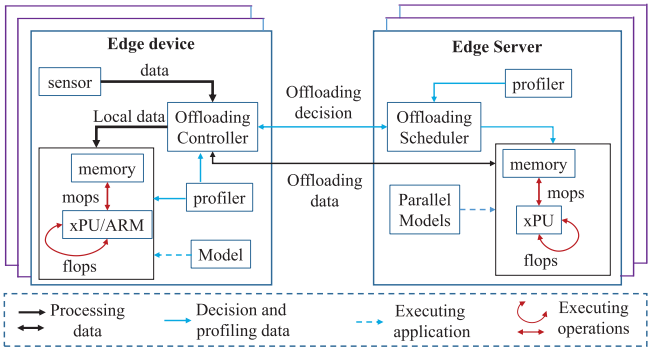


Fig. 3. Partial offloading architecture on edge intelligence with heterogeneous EDs which run DL models to perform inference on data from sensor.

V. Partial Offloading Architecture for Heterogeneous Edge Intelligence

To apply HESM and GHMWOA to heterogeneous edge intelligence, we summarize the applications in edge intelligence scenarios, propose a partial offloading architecture of heterogeneous edge intelligence, and design an energy profiling method for EDs.

Edge intelligence is an edge computing system that leverages AI to enhance the capabilities of data processing, data protection, task scheduling, etc. and it has experienced explosive growth in recent years. Like most edge computing systems, edge intelligence is heterogeneous.

Machine learning (ML) is a typical method for achieving AI [1], and it overcomes the limitation of early AI approaches, such as the knowledge-base approach, where it is difficult for humans to identify all the tacit knowledge required to perform complex tasks. ML aids in generating reliable and repeatable decisions by learning from previous computations and extracting patterns from massive databases. DL is an ML method based on artificial neural networks (ANNs). Compared to simple ANNs, deep learning networks consist of multiple hidden layers and utilize advanced neurons with special operations, such as convolution. The characteristics enable deep learning networks to accept raw input data and spontaneously complete corresponding learning tasks, which often makes DL models outperform shallow ML models and traditional data analysis methods [31]. Currently, there are various deep learning architectures, including CNNs, recurrent neural networks (RNNs), and generative adversarial networks (GANs). CNNs and GANs are typically applied to image and video processing, while RNNs are often used for time series forecasting [32]. The data they collect are usually in the form of frames or time steps (FTs). We adopt the models of the three networks as the main applications in our work.

Combining the system architecture of edge computing and the DL model, a brief overview of the partial offloading architecture for heterogeneous edge intelligence is provided in Fig. 3. In the architecture, the sensor of ED captures data at a specific rate, called FTs per second (FTPS) in our work. The offloading scheduler and controller execute an offloading optimization algorithm to determine the offloading ratio (λ) of the sampled data and the proportion of network bandwidth (μ) for all EDs. Local data are processed by the ED, while the remaining data are offloaded to ESs for processing. A profiler in ES obtains the APP_i according to [33], ρ_i^f, ρ_i^m, and P_i^{static} of ED_i is real-time acquired by an energy profiling method as shown in Algorithm 4.

Algorithm 4 Energy Profiling

Input:

APP_i: the $\{F_i, M_i\}$ of the model runing in ED i

Output:

$\rho_i^f, \rho_i^m, P_i^{static}$ of ED_i

1: Start the power monitoring tool and obtain $P_i^{static}, P_{pro}^{static}, P_{mem}^{static}$
2: Record the instantaneous power of the processor (P_k^{pro}) and memory (P_k^{mem}) in time point k
3: $t_1 \leftarrow$ the system time
4: Run one frame or time step by the model
5: $t_2 \leftarrow$ the system time
6: Stop the power monitoring tool.
7: Calculate the average processor power (P_{pro}^{ave}) and average memory power (P_{mem}^{ave}) with (34) and (35)
8: $E_{total}^f \leftarrow (P_{pro}^{ave} - P_{pro}^{static}) * (t_2 - t_1)$
9: $E_{total}^m \leftarrow (P_{mem}^{ave} - P_{mem}^{static}) * (t_2 - t_1)$
10: $\rho_i^f \leftarrow E_{total}^f / F_i$
11: $\rho_i^m \leftarrow E_{total}^m / M_i$

In the algorithm, the static power (P_i^{static}) includes the static power of the processor (P_{pro}^{static}) and the static power of the memory (P_{mem}^{static}). P_k^{pro} and P_k^{mem} record the processor power and memory power at time point k, called instantaneous powers. When the ED is idle, P_k^{pro} and $P_k^{mem}(k = 1, 2, \ldots)$ are equal to P_{pro}^{static} and P_{mem}^{static}, respectively. When the ED executes an application, P_k^{pro} and P_k^{mem} will momentarily surge. Considering that the power monitoring tool can not record the instantaneous power at every moment but can only capture a power value at a fixed time interval, the average power P_{pro}^{ave} and P_{mem}^{ave} of processor and memory are used as a substitute for instantaneous powers to calculate energy consumption. The equations are as follows:

$$P_{pro}^{ave} = \sum_{k=1}^{tp} P_k^{pro} / tp \tag{34}$$

$$P_{mem}^{ave} = \sum_{k=1}^{tp} P_k^{mem} / tp \tag{35}$$

where tp denotes the number of time points between t_1 and t_2. In DL models, F_i and M_i represent the number of flops and mops required to process one frame or time step. Therefore, energy consumption information for single flop and mop is obtained by executing one frame or time step in line 4. If the application programming interface (API) provided by certain devices can not capture the instantaneous power of the processor and memory, it is necessary to develop specific test cases. For instance, a memory copy case is utilized to analyze the energy consumption of a single mop.

The energy profiling method has already been implemented on the NVIDIA Jetson device, and the power monitoring tool

TABLE III
PARAMETERS OF EDS AND ESS, WHERE PROCESSOR PEAK AND MEMORY BANDWIDTH (MEM. BW.) IS VENDOR'S CLAIMED PEAK. ρ^f AND ρ^m ARE THE ENERGY COST OF A SINGLE FLOP AND MOP, RESPECTIVELY. P^{static} IS THE STATIC POWER

EDs/ES	Processor	Processor Peak(Gflop/s)	Mem. bw.(GB/s)	ρ^f(pJ/fop)	ρ^m(pJ/mop)	P^{static}(w)
Arndale ARM	A15	27.2	12.8	107	386	5.5
Arndale GPU	T-604	72	12.8	84.2	518	1.28
NUC CPU	I3	57.6	25.6	14.7	418	16.5
Nvidia TX2	A57	96	59.7	36.6	612.3	1.9
Intel Xeon Phi	5110P	2020	320	-	-	-
Nvidia Fermi	GF100	1580	192	-	-	-
Nvidia Kepler	GK140	3530	192	-	-	-

TABLE IV
FLOPS AND MOPS OF DL MODELS

No.	Models	Input dimension	Flops (Mflops)	Mops (Mbytes)
1	densenet161	224×224	8000	345
2	caffenet	224×224	724	236
3	squeezenet1-0	224×224	873	35
4	vgg-vd-19	224×224	20000	611
5	resnet50	224×224	4000	201
6	mcn-mobilenet	224×224	579	54
7	googlenet	224×224	2000	77
8	SE-BN-Inception	224×224	2000	89
9	Senet	224×224	21000	787
10	resnet-101	224×224	8000	325
11	Mbed-ANT	$3 \times 1M$	2233	34
12	ST-LSTM	$20 \times 35 \times 256$	13700	54
13	ISS-LSTM	$20 \times 35 \times 256$	8680	21.8
14	Swin-GAN	32×32	4115.2	30.2
15	Trans-GAN	256×256	15474.8	38.4

TABLE V
COMPARISON OF SE (J) IN THREE ES SCHEDULING STRATEGIES UNDER $H = 4$, FTPS $= 30$, AND NETDIS $= 100$ (M)

N	R-HES	FE-HES	FEG-HES
10	11.32671	15.15795	**9.97708**
15	27.90442	24.71553	**23.48769**
20	60.11492	46.87274	**46.77402**
25	75.887	71.463889	**71.405601**
30	86.88624	95.34788	**80.98211**
35	105.67322	106.7487	**104.71989**
40	145.51915	132.34192	**126.29032**

was developed based on the command "jtop." The method can be integrated into the profiler in EDs.

VI. PERFORMANCE EVALUATION

A. System, Application, and Algorithm Setting

In this section, we adopt real-world processers and DL models.

EDs are configured with Cortex-A15 (A15), Intel Core I3-3217U (I3), and Mali T-604 (T-604). ESs are equipped with Nvidia GF100, Nvidia GK140, and Intel 5110P, respectively. The processors are listed in [34]. In addition, we obtain energy consumption per flop and mop and static power in NVIDIA Jetson TX2 configured with ARM cortex-A57 (A57) as follows. The energy cost per flop, energy cost per mop, and static power are 36.6 (pJ), 612.3 (pJ), and 1.9 (W), respectively. Table III enumerates the parameters of all EDs and ESs.

Fifteen DL models, listed in Table IV, are utilized to validate the offloading strategy. Among them, Models 1 to 10 are CNNs [35], Models 11 to 13 are RNNs [36], [37],

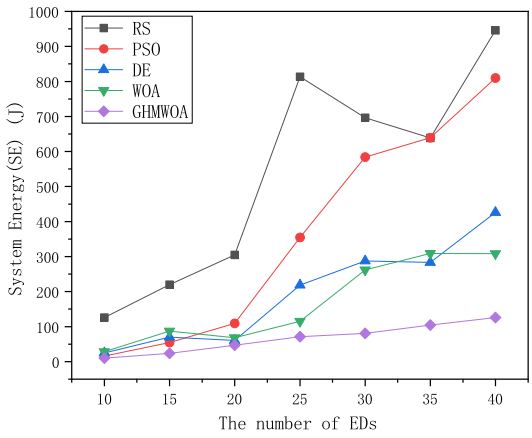


Fig. 4. Comparison of SE in five algorithms as the number of EDs increases under FTPS $= 30$ and netdis $= 100$ (m).

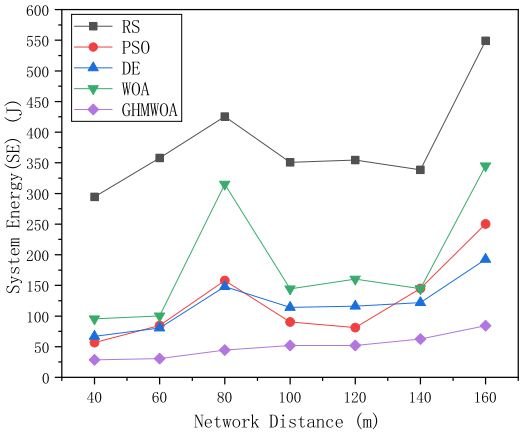


Fig. 5. Comparison of SE in five algorithms as network distance increases under $N = 20$ and FTPS $= 30$.

Models 14 and 15 are GANs [38], [39], and Model 15 also incorporates a transformer network.

The models have different network depths and structural characteristics, making them suitable for various task requirements. Densenet161 adopts a network structure with 161 layers of dense connections, enabling efficient feature reuse. Caffenet and Squeezenet1-0 focus on lightweight design, making them suitable for resource-constrained environments. Vgg-vd-19 uses small convolution kernels and a deep structure (19 layers) to extract features in depth. Resnet50 and Resnet-101, with their residual structures, alleviate the vanishing gradient problem. They have 50 and 101 layers, respectively, and are

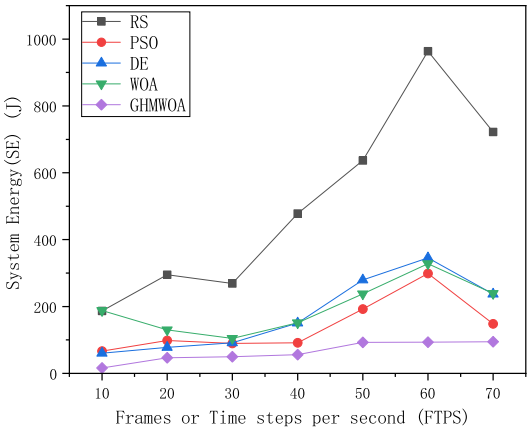


Fig. 6. Comparison of SE in five algorithms as FTPS increases under N = 20 and netdis = 100 (m).

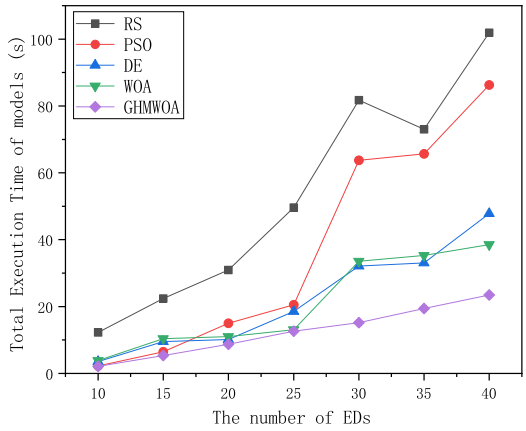


Fig. 7. Total execution time of models in five algorithms as the number of EDs increases under FTPS = 30 and netdis = 100 (m).

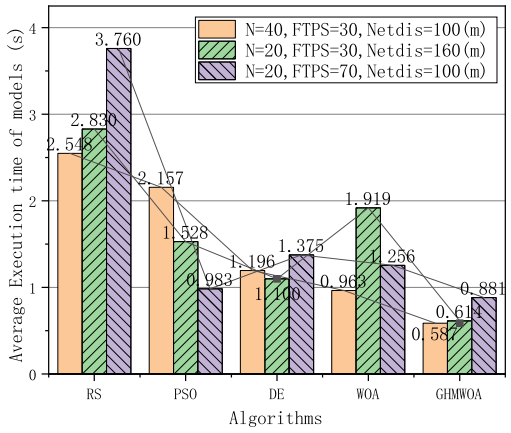


Fig. 8. Average execution time of models optimized with five algorithms under the three most resource-intensive cases, respectively.

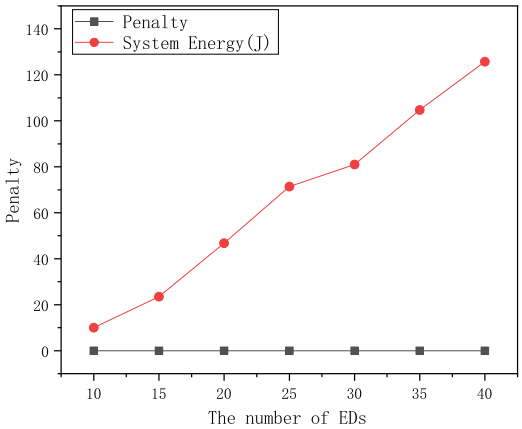


Fig. 9. Penalty of GHMWOA as the number of EDs increases under FTPS = 30 and netdis = 100(m).

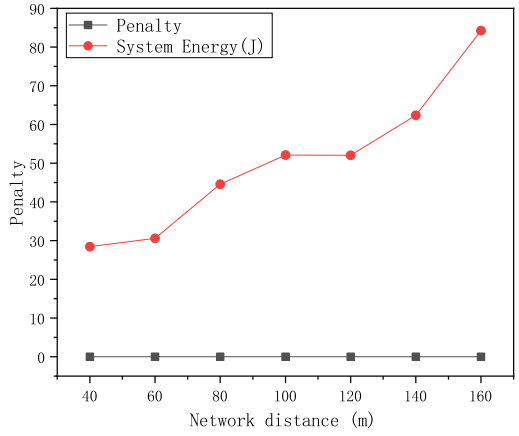


Fig. 10. Penalty of GHMWOA as network distance increases under N = 20 and FTPS = 30.

suitable for complex tasks. Mcn-mobilenet employs depthwise separable convolutions to reduce computational load and model size significantly. Googlenet, through its inception module, combines multiscale convolutions to achieve efficient feature extraction. SE-BN-Inception and Senet further incorporate attention mechanisms to enhance the model's focus on key features. For sequential data, Mbed-ANT combines the attention mechanism with gated recurrent units (GRU) to extract features. ST-long short-term memory (LSTM) optimizes performance by replacing dense weights with sparse ones, while ISS-LSTM requires first training an over-parameterized dense model, then gradually pruning and reallocating weights to obtain a sparse LSTM model. Swin-GAN and Trans-GAN combine GANs with Transformer architectures, demonstrating strong performance in image generation tasks. Swin-GAN, based on the Swin-Transformer block, uses a shifted window self-attention mechanism to capture both contextual semantics and low-level texture features. Trans-GAN introduces a novel grid self-attention mechanism to mitigate memory bottlenecks in high-resolution generation tasks.

According to [40], network parameters can be set as follows. $W = 10$ MHz, $\gamma = 0.98$, $N_0 = 1.6 \times 10^{-11}$, $v = 4$, $\beta_1 = 1$, $P_i^t = 0.1W$, $P_i^0 = 0.4W$, and $k_i^t = 18$. The time slot is equal to one second. In GHMWOA, $sf = 0.7$, $dis = 2$, and $F = 0.4$.

B. Comparison of Algorithm Performance

In the experiment, fifteen different models are invoked by several heterogeneous EDs with equal opportunity. The tests focus on the characteristics of the EDs, including the number of EDs (N), the network distance (netdis) from ED to AP, and the FTPS of EDs. The benchmark algorithms for comparison include random search (RS) [41], particle swarm optimization (PSO) [42] based on population evolution, DE [43] based

TABLE VI
COMPONENT ENERGY(J) UNDER N = 15, FTPS = 30, AND NETDIS = 100 (M)

Models	FE	ME	STE	NE	TE	FE-N	ME-N	SE-N	TE-N
densenet161	1.42904	0.23312	5.23199	0.49441	7.38856	25.68	4.18917	48.52941	78.39858
caffenet	0.25633	2.49129	4.99500	0.13397	7.87659	0.31928	3.1032	6.22188	9.64436
squeezenet1-0	2.11426	0.57032	0.4464	0	3.13098	2.11426	0.57032	0.4464	3.13098
vgg-vd-19	2.89195	1.54983	1.56384	0.36953	6.37515	21.96	11.76865	11.875	45.60365
resnet50	1.87591	0.35658	5.35449	0.49241	8.07938	12.84	2.44065	24.26471	39.54536
mcn-mobilenet	0.25534	0.71005	4.97578	0	5.94117	0.25534	0.71005	4.97578	5.94117
googlenet	2.87007	0.71280	0.95088	0.34924	4.88298	5.05200	1.25471	1.06667	7.37338
SE-BN-Inception	2.196	1.71425	1.1875	0	5.09775	2.196	1.71426	1.1875	5.09776
Senet	0.06495	0.00921	2.5781	0.30799	2.96024	67.41	9.55616	127.3897	204.3559
resnet-101	0.06602	0.07997	11.599	0.39954	12.14452	3.528	4.27347	68.75	76.55147
Mbed-ANT	2.96382	0.29111	0.62577	0.02333	3.90403	5.64056	0.55403	1.19093	7.38552
ST-LSTM	5.49465	0.37992	2.97127	0.86868	9.71452	15.0426	1.04011	8.13438	24.21709
ISS-LSTM	0.51679	0.00491	6.60966	0.90346	8.03482	27.8628	0.26471	52.65441	80.78192
Swin-GAN	0	0	0.78225	0.02143	0.80367	1.8148	0.3971	35.365	37.5769
Trans-GAN	0	0	0.8644	0.42765	1.29205	39.08935	0.62572	8.25323	47.9683

TABLE VII
COMPONENT DELAY(s) UNDER N = 15, FTPS = 30, AND NETDIS = 100 (M)

Models	FD	MD	STD	ND	ED	TD	FD-N	MD-N	STD/TD-N
densenet161	0.49101	0.04394	0.95127	0.95127	0.06421	1.01548	8.82353	0.78964	8.82353
caffenet	0.30273	0.21682	0.30273	0.27547	0.00447	0.30273	0.37708	0.27008	0.37708
squeezenet1-0	0.34875	0.08011	0.34875	0	0	0.34875	0.34875	0.08011	0.34875
vgg-vd-19	0.82307	0.03949	0.82307	0.66202	0.14759	0.82307	6.25	0.29984	6.25
resnet50	0.64455	0.06721	0.97354	0.97354	0.02903	1.00257	4.41177	0.46005	4.41177
mcn-mobilenet	0.30156	0.06180	0.30156	0	0	0.30156	0.30156	0.0618	0.30156
googlenet	0.47342	0.10012	0.74287	0.74287	0.00734	0.75021	0.83333	0.17624	0.83333
SE-BN-Inception	0.625	0.04368	0.625	0	0	0.625	0.625	0.04368	0.625
Senet	0.02232	0.00174	0.46875	0.46875	0.1783	0.64704	23.16177	1.8013	23.16177
resnet-101	0.07797	0.00696	0.70297	0.70297	0.06672	0.76969	4.16667	0.37193	4.16667
Mbed-ANT	0.48889	0.04089	0.48889	0.04955	0.00901	0.48889	0.93042	0.07782	0.93042
ST-LSTM	1.56382	0.00968	1.56382	1.48818	0.07390	1.56382	4.28125	0.0265	4.28125
ISS-LSTM	0.17757	0.00093	1.20176	1.20176	0.0724	1.27416	9.57353	0.0499	9.57353
Swin-GAN	0	0	0.04741	0.04741	0.03497	0.08238	2.14333	0.03456	2.14333
Trans-GAN	0	0	0.67531	0.67531	0.13151	0.80683	6.44783	0.08789	6.44783

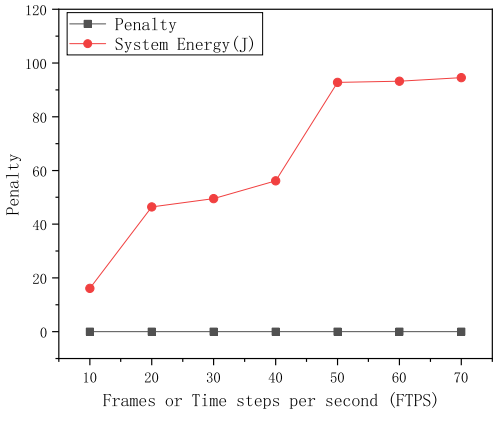


Fig. 11. Penalty of GHMWOA as FTPS increases under N = 20 and netdis = 100 (m).

on genetic evolution, and WOA [29]. During the testing process, we not only focus on the amount of SE consumed but also consider the stability of its value. To more accurately compare the performance of algorithms, the four benchmark algorithms do not adopt the PRIBG method, but use a set of the proportions of network bandwidth identical to those of GHMWOA to initialize the solutions. $\mathscr{P} = 50$ and $\mathscr{M} = 200$ in all algorithms.

Fig. 4 illustrates the changes in SE consumption as the number of EDs increases from 10 to 40, with five different algorithms for offloading. Overall, all algorithms are capable of implementing the offloading process. However, the GHMWOA achieves the lowest SE consumption, with its value showing a relatively stable increase as the number of EDs increases. In contrast, notable fluctuations are observed in the values obtained by the RS, DE, and WOA. Furthermore, the RS, which adopts a random solution generation approach, fails to achieve minimal energy consumption.

Fig. 5 demonstrates the changes in SE consumption with the increase in netdis among the five algorithms. Fig. 6 shows the changes in SE consumption with the increase in FTPS across the five algorithms. Similar to Fig. 3, GHMWOA consistently achieves better performance. Specifically, GHMWOA (an improved version of WOA) significantly outperforms WOA. For instance, compared to WOA, GHMWOA achieves a 59% reduction in the cost of SE when the number of EDs equals 40.

Fig. 7 illustrates the changes in the total execution time (TET) of models within the system after offloading optimization with five different algorithms, as the number of EDs increases from 10 to 40. The figure shows that GHMWOA achieves the lowest TET and maintains a stable growth trend as the number of EDs increases.

Fig. 8 presents the average execution time (AET) of models in the system after offloading optimization with five different algorithms for the most resource-intensive cases in our experiments. By comparison, it can be observed that GHMWOA

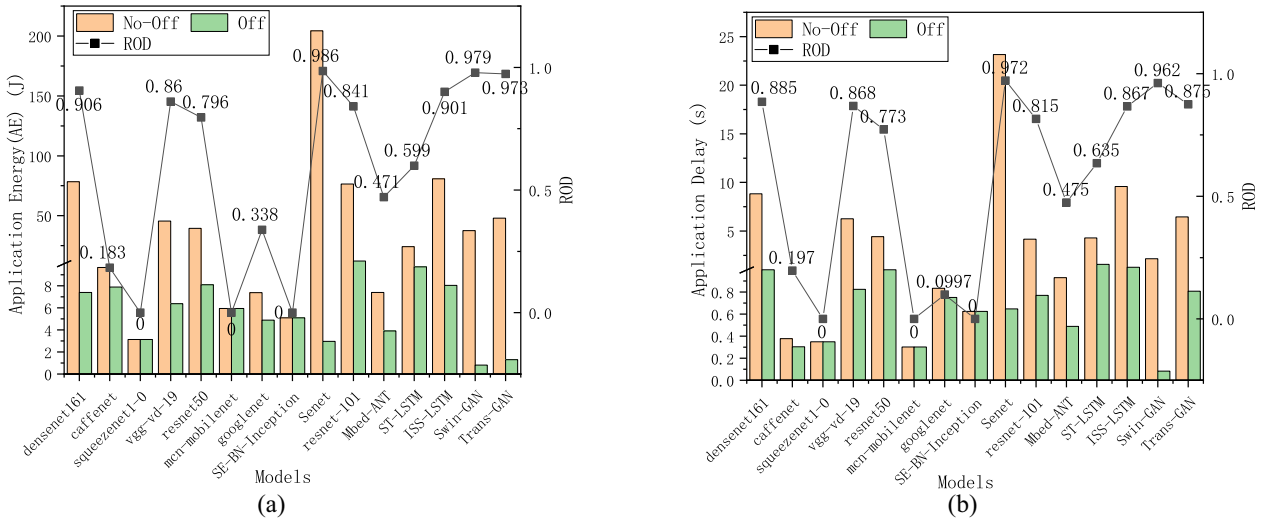


Fig. 12. Application-level energy and delay under N = 15, FTPS = 30, and netdis = 100 (m). (a) Application energy (AE) comparison between nonoffloading and offloading cases. (b) Application delay (AD) comparison between nonoffloading and offloading cases.

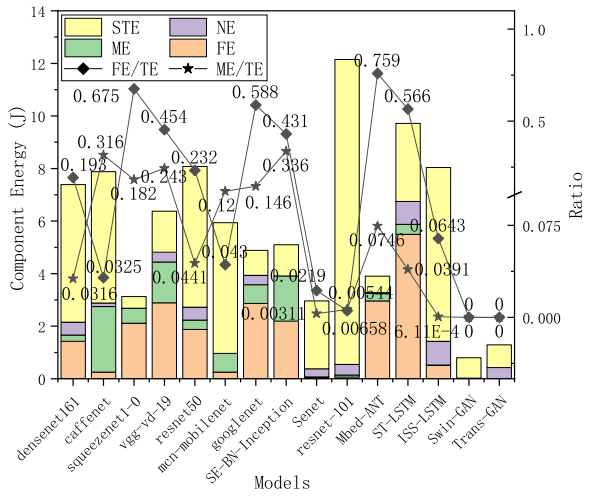


Fig. 13. Component energy in models under N = 15, FTPS = 30 and netdis = 100 (m).

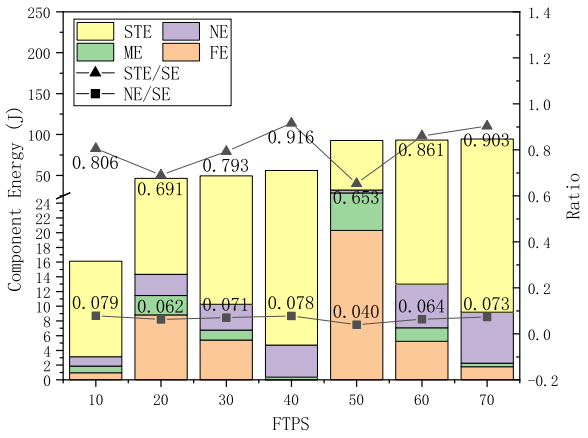


Fig. 14. Component energy in edge system as FTPS increases under N = 20 and netdis = 100(m).

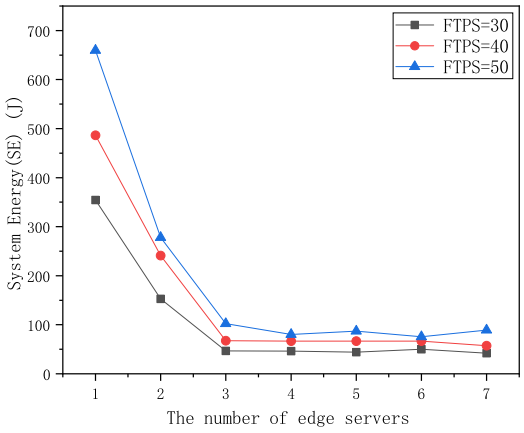


Fig. 15. Comparison of SE as the number of ESs increases under N = 20 and netdis = 100 (m).

consistently achieves shorter AET. Notably, the AET is below 1 s with GHMWOA, which indicates that the EDs are capable of processing sampled data within a short period despite the cases demanding more system resources. Therefore, the edge system optimized by GHMWOA demonstrates better real-time performance.

Figs. 9–11 show the SE and penalty of GHMWOA as the number of EDs, netdis, and FTPS vary. They indicate that the penalty of GHMWOA is always 0, which demonstrates that the solutions found by GHMWOA satisfy all the constraints.

To validate the performance of FEG-HES, it is compared with a random scheduling strategy (R-HES) and a scheduling strategy of FE ES priority (FE-HES) under heterogeneous ESs. In R-HES, the ES for applications is selected randomly. If an ES becomes overloaded, the overloaded applications in the ES are randomly migrated to another ES that meets the resource requirements of the applications. In FE-HES, applications are preferentially scheduled to the ES with the highest performance. If an ES becomes overloaded, its overloaded applications are migrated to another server that not only meets the resource requirements of the overloaded applications but also offers optimal performance for the applications. Table V shows that the SE of FEG-HES is lower than that of R-HES

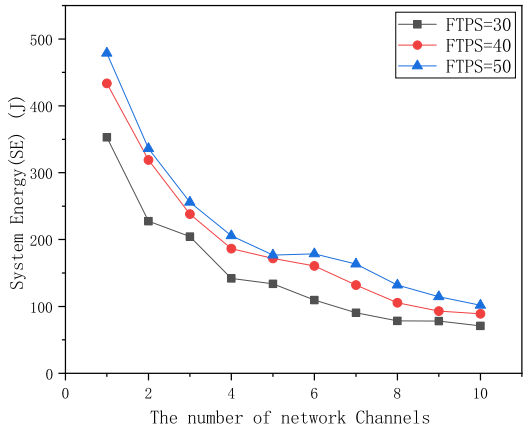


Fig. 16. Comparison of SE as the number of network channels increases under N = 20 and netdis = 100 (m).

and FE-HES as EDs increase from 10 to 40 under GHMWOA algorithm.

C. Evaluations at the Application and Component Levels

Although the optimization goal of this article is to achieve the minimal cost of SE, we can obtain energy and delay costs at the application and component by analyzing the flops, mops, and network transmissions. The three-level details of energy consumption and delay facilitate an understanding of the sources of energy and delay.

Fig. 12 presents the cost of application-level energy and delay when the number of EDs is 15, the FTPS is 30, and the netdis is 100 (m). "No-off," "off," and "ROD" represent the energy or latency without offloading, the energy or delay with offloading, and the reduction ratio between them, respectively. The figure shows that GHMWOA can generally reduce the energy consumption of applications. In particular, when an application has more fops and mops, the ROD is greater. On the contrary, the effect of offloading is not obvious. The reason is that executing models with more fops and mops on the EDs will consume more energy, so the models have more opportunities for offloading to achieve a lower cost of SE.

Fig. 13 illustrates the energy consumption of various components during the offloading process of DL models. Specifically, FE and ME represent the energy costs associated with performing flops and mops, respectively, on the processors and memories of EDs. NE denotes the energy cost in network transmission, while STE signifies static energy. TE stands for the total energy consumption of each model. The figure reveals that, apart from FE, other energy consumptions occupy a substantial proportion. For instance, in the ResNet series of models, including ResNet50 and ResNet-101, STE constitutes the primary energy cost. In CaffeNet and MCN-MobileNet, ME exceeds FE, particularly pronounced in CaffeNet. Consequently, considering only the energy consumption of processor components, such as CPUs, would be inaccurate.

Table VI enumerates the detailed energy consumption of components from Fig. 13. Similarly, Table VII lists the delays incurred in each component, including flop delay (FD), mop delay (MD), static delay (STD), network delay (ND), edge delay (ED), and total delay (TD). The notation "-N" indicates the energy consumption and delay when offloading does not occur. These data discover on which energy consumption of system components depends. For instance, Table VI shows that the energy consumption of densenet161 and CaffeNet primarily comes from STE. Since STE is proportional to local delay, Table VII indicates that the STE of densenet161 depends on ND, while that of CaffeNet depends on FD, which refers to the delay of the processor component.

Fig. 14 illustrates the variation in component energies as the FTPS increases when the number of EDs is set to 20. The component energy is calculated as the sum of energy consumption from its corresponding component across all EDs. The figure indicates that STE accounts for more than 65% of the SE. Additionally, the value of ME is approximate to FE when FTPS equals 10. These phenomena underscore the inaccuracy of analyzing energy consumption based solely on the processor component in practical scenarios. Furthermore, it highlights the significance of multilevel energy consumption modeling and analysis in edge systems.

D. Effects of Edge Servers and Network Channels

The computing capability at the edge is provided by ESs. Fig. 15 presents SE with respect to the number of ESs. By turning up FTPS in tests, the demand for computing power is increased to evaluate the effect of ESs. As shown in the figure, SE reduces as the number of ESs increases. The reason is that more data need to be executed in ED locally when fewer ESs can not provide enough computing resources. In addition, the SE consumption tends to stabilize as the number of ESs exceeds 3. The reason is SE does not depend on the computing resource but on other resources, such as network resources when the number of ESs increases to a certain level and is sufficient.

Further, we evaluate the effects of network channels on SE. Fig. 16 shows SE falls with the channel rising. The reason is that the transmission rate is proportional to the number of network channels. Increasing the network channel facilitates to transmission of more data to the ES so that energy consumption for solving data in EDs decreases. Similar to Fig. 15, the cost of SE tends to stabilize when the network channels are sufficient, for example, the test in FTPS = 30. The evaluation provides valuable data for constructing a practical edge system.

VII. Conclusion

In this article, we propose a system model (HESM) for the heterogeneous edge system and design GHMWOA to obtain optimal offloading decisions for minimizing SE consumption, which combines the characteristics of application and edge, covering the application complexity (flop and mop), FTPS, network, processor, memory, etc. A partial offloading architecture for heterogeneous edge intelligence, including an energy profiling method, provides theoretical and practical foundations for the application of HESM and GHMWOA in heterogeneous edge intelligence. The offloading mechanism

can analyze the details of energy consumption and delay at three levels: system, application, and component.

Real-world DL models, EDs, and ESs are employed in experiments. The results show that the GHMWOA outperforms four classic benchmark algorithms. Compared with WOA, for example, the GHMWOA reduces the cost of the SE by 59% when the number of EDs is 40. Meanwhile, the results also demonstrate the significance of multilevel energy analysis. System-level energy analysis can determine the merits of offloading methods, as well as the impact of system networks and computing capabilities on offloading. Application-level energy analysis reveals that applications with fewer flops and mops have fewer opportunities for offloading. Component-level analysis of energy and delay can identify the sources and causes of application and SE consumption. These analysis results provide valuable insights for the improvement and design of energy-efficient systems. Despite the adoption of computation offloading, it is difficult to balance the workload between the ED and ES because of limited network resources when the amount of data increases sharply. In the future, data compression and edge caching should be adopted to optimize the offloading process.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

[1] D. Xu et al., "Edge intelligence: Empowering intelligence to the edge of network," *Proc. IEEE*, vol. 109, no. 11, pp. 1778–1837, Nov. 2021.

[2] C. Ren, W. Song, and X. Lyu, "Hybrid learning of predictive mobile-edge computation offloading under differently-aged network states," *Future Gener. Comput. Syst.*, vol. 156, pp. 301–312, Jul. 2024.

[3] H. Yu, S. Leng, and F. Wu, "Joint cooperative computation offloading and trajectory optimization in heterogeneous UAV-swarm-enabled aerial edge computing networks," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 17700–17711, May 2024.

[4] S. Khan, Z. Jiangbin, M. Irfan, F. Ullah, and S. Khan, "An expert system for hybrid edge to cloud computational offloading in heterogeneous MEC–MCC environments," *J. Netw. Comput. Appl.*, vol. 225, May 2024, Art. no. 103867.

[5] R. Lin, X. Guo, S. Luo, Y. Xiao, B. Moran, and M. Zukerman, "Application-aware computation offloading in edge computing networks," *Future Gener. Comput. Syst.*, vol. 146, pp. 86–97, Sep. 2023.

[6] H. Zhou, Z. Wang, H. Zheng, S. He, and M. Dong, "Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1326–1338, May/Jun. 2023.

[7] Z. Xue, C. Liu, C. Liao, G. Han, and Z. Sheng, "Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 72, no. 5, pp. 6709–6722, May 2023.

[8] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. 2nd ACM/IEEE Symp. Edge Comput. (SEC)*, 2017, pp. 2573–2574.

[9] Y. Sun, Z. Wu, K. Meng, and Y. Zheng, "Vehicular task offloading and job scheduling method based on cloud-edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 12, pp. 14651–14662, Dec. 2023.

[10] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 111–118, Jul. 2019.

[11] J. Huang, H. Gao, S. Wan, and Y. Chen, "AoI-aware energy control and computation offloading for industrial IoT," *Future Gener. Comput. Syst.*, vol. 139, pp. 29–37, Feb. 2023.

[12] Z. He, Y. Xu, D. Liu, W. Zhou, and K. Li, "Energy-efficient computation offloading strategy with task priority in cloud assisted multi-access edge computing," *Future Gener. Comput. Syst.*, vol. 148, pp. 298–313, Nov. 2023.

[13] Y. Huang, Y. Lu, F. Wang, X. Fan, J. Liu, and V. C. Leung, "An edge computing framework for real-time monitoring in smart grid," in *Proc. IEEE Int. Conf. Ind. Internet (ICII)*, 2018, pp. 99–108.

[14] H. Ko, J. Kim, D. Ryoo, I. Cha, and S. Pack, "A belief-based task offloading algorithm in vehicular edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 5, pp. 5467–5476, May 2023.

[15] Y. Liu, Y. Mao, Z. Liu, F. Ye, and Y. Yang, "Joint task offloading and resource allocation in heterogeneous edge environments," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 7318–7334, Jun. 2024.

[16] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE INFOCOM*, 2020, pp. 257–266.

[17] Z. Zhou, S. Yu, W. Chen, and X. Chen, "CE-IoT: Cost-effective cloud-edge resource provisioning for heterogeneous IoT applications," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8600–8614, Sep. 2020.

[18] H. J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning Web apps in the edge server environment," in *Proc. 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 1492–1499.

[19] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3774–3785, Mar. 2021.

[20] L. Ma, Y. Zhang, J. Zhou, and G. Zhang, "A gene-inspired metaheuristic for scheduling workflow tasks in mobile edge computing-supported cyber–physical systems," *J. Syst. Archit.*, vol. 151, Jun. 2024, Art. no. 103136.

[21] A. Abbas, A. Raza, F. Aadil, and M. Maqsood, "Meta-heuristic-based offloading task optimization in mobile edge computing," *Int. J. Distrib. Sensor Netw.*, vol. 17, no. 6, pp. 1–11, 2021.

[22] Z. Liu, J. Wang, Z. Gao, and J. Wei, "Privacy-preserving edge computing offloading scheme based on whale optimization algorithm," *J. Supercomput.*, vol. 79, pp. 3005–3023, Feb. 2023.

[23] X. Yuan et al., "A MEC offloading strategy based on improved DQN and simulated annealing for Internet of Behavior," *ACM Trans. Sensor Netw.*, vol. 19, no. 2, pp. 1–20, 2022.

[24] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[25] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *Proc. IEEE Global Commun. Conf. GLOBECOM*, 2016, pp. 1–6.

[26] H. Lee and Y. Eun, "Advertisement revenue and exposure optimization for digital screens in subway networks using smart card data," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 24095–24104, Dec. 2022.

[27] A. Daghayeghi and M. Nickray, "Decentralized computation offloading via multi-agent deep reinforcement learning for NOMA-assisted mobile edge computing with energy harvesting devices," *J. Syst. Archit.*, vol. 151, Jun. 2024, Art. no. 103139.

[28] A. Panda and S. Pani, "A symbiotic organisms search algorithm with adaptive penalty function to solve multi-objective constrained optimization problems," *Appl. Soft Comput. J.*, vol. 46, pp. 344–360, Sep. 2016.

[29] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[30] R. Sengupta and S. Saha, "Reference point based archived many objective simulated annealing," *Inf. Sci.*, vol. 467, pp. 725–749, Oct. 2018.

[31] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electron. Markets*, vol. 31, pp. 685–695, Sep. 2021.

[32] K. Heinrich, P. Zschech, C. Janiesch, and M. Bonin, "Process data properties matter: Introducing gated convolutional neural networks (GCNN) and key-value-predict attention networks (KVP) for next event prediction with deep learning," *Decis. Support Syst.*, vol. 143, Apr. 2021, Art. no. 113494.

[33] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5353–5360.

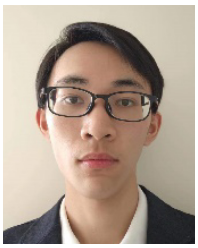
[34] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, "Algorithmic time, energy, and power on candidate HPC compute building blocks," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2014, pp. 447–457.

[35] S. Albanie. "Estimates of memory consumption and FLOP counts for various convolutional neural networks." 2018. [Online]. Available: https://github.com/albanie/convnet-burden

[36] A. Jagannath and J. Jagannath, "Embedding-assisted attentional deep learning for real-world RF fingerprinting of Bluetooth," *IEEE Trans. Cogn. Commun. Netw.*, vol. 9, no. 4, pp. 940–949, Aug. 2023.
[37] S. Liu, I. Ni'mah, V. Menkovski, D. C. Mocanu, and M. Pechenizkiy, "Efficient and effective training of sparse recurrent neural networks," *Neural Comput. Appl.*, vol. 33, pp. 9625–9636, Aug. 2021.
[38] S. Wang, Z. Gao, and D. Liu, "Swin-GAN: Generative adversarial network based on shifted windows transformer architecture for image generation," *Vis. Comput.*, vol. 39, pp. 6085–6095, Dec. 2023.
[39] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two pure transformers can make one strong GAN, and that can scale up," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, Feb. 2021, pp. 14745–14758.
[40] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[41] L. Deng and S. Liu, "Deficiencies of the whale optimization algorithm and its validation method," *Expert Syst. Appl.*, vol. 237, Mar. 2024, Art. no. 121544.
[42] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw. (ICNN)*, 1995, pp. 1942–1948.
[43] R. Storn and K. Price, "Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, pp. 341–359, Dec. 1997.

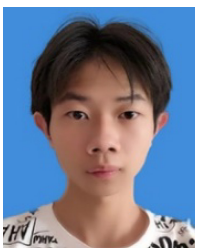
Baoyu Xu received the M.S. degree in computer science from Shanghai University, Shanghai, China, in 2008.

He is currently a Staff with the School of Computer Engineering and Science, Shanghai University. His research interests include parallel computing, edge computing, and optimization.



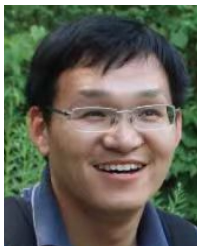
Yancheng Ruan is currently pursuing the B.S. degree in computer science and technology with the School of Computer Science and Engineering, Shanghai University, Shanghai, China.

His research interests include deep learning and millimeter wave radar.



Chenghu Qiu is currently pursuing the B.S. degree from Shanghai University, Shanghai, China.

His research interests include mobile edge computing and deep reinforcement learning.



Shuibing He (Member, IEEE) received the Ph.D. degree in computer science and technology from Huazhong University of Science and Technology, Wuhan, China, in 2009.

He is currently a ZJU100 Young Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research areas include intelligent computing, memory and storage systems, and processing-in-memory.

Dr. He is a member of the ACM.

Feng Shu (Senior Member, IEEE) received the B.E. degree in electrical engineering from Shanghai Jiaotong University, Shanghai, China, in 2002, the M.Eng. degree in electrical engineering from Nanyang Technological University, Singapore, in 2004, and the Ph.D. degree in electrical engineering from The University of Melbourne, Parkville, VIC, Australia, in 2007.

He is an Associate Professor with the School of Electrical and Information Engineering, University of Sydney, Sydney, NSW, Australia. Prior to that, he was with Fudan University, Shanghai, China. Before returning to academia, he had held various Research and Development positions in the microelectronics industry for nearly ten years in Europe. His current research interests include nanometrology, high precision mechatronics, and machine intelligence.

Xiaoyang Kang (Member, IEEE) received the Ph.D. degree in electronic science and technology from Shanghai Jiao Tong University, Shanghai, China, in 2016.

He completed Postdoctoral Research with Ecole Polytéchnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, from 2016 to 2018. He is currently a Tenure-Track Associate Professor with the Laboratory for Neural Interface and Brain Computer Interface, Engineering Research Center of Al & Robotics, Ministry of Education, Shanghai Engineering Research Center of Al and Robotics, MOE Frontiers Center for Brain Science, State Key Laboratory of Medical Neurobiology, Institute of Al and Robotics, Academy for Engineering and Technology, Fudan University, Shanghai. His research interests include artificial intelligence, neural engineering and brain computer interface, and their interdisciplinary applications.

Lihua Zhang (Member, IEEE) received the Ph.D. degree in control theory and engineering from Tsinghua University, Beijing, China, in 2000.

He is currently a Professor with the Institute of AI and Robotics, Fudan University, Shanghai, China. His research interests include artificial intelligence and its interdisciplinary applications.