An Energy-Efficient Partition and Offloading Method for Multi-DNN Applications in Edge-End Collaboration Environments

No Author Given

No Institute Given

Abstract. Deep Neural Networks (DNNs) have emerged as the preferred solution for Internet of Things (IoT) applications, owing to their remarkable performance capabilities. However, the inherent complexity of DNNs presents significant challenges for IoT devices that are constrained by limited computational power and battery life. To adeptly navigate the demands of intricate inference tasks, edge computing is leveraged, enabling collaborative inference of DNNs between IoT devices and edge servers. However, existing research rarely focus simultaneously on the power consumption of IoT devices, the latency of collaborative inference and the cost of edge servers. Moreover, current research seldom takes into account the deployment of multiple DNN applications on IoT devices, a critical factor for adapting to increasingly complex edge-end collaborative environments. This research focuses on optimizing the inference power consumption of multiple DNN applications deployed on IoT devices in larger-scale edge-end collaboration environments, under the constraints of maximum End-to-End latency and the cost of edge servers. To address this issue, we propose the Greedy Genetic Algorithm, which leverages a combination of greedy strategy and Genetic Algorithm. The performance of our proposed method is extensively evaluated through experiments, demonstrating its superiority in achieving lower inference power consumption with fewer iterations compared to existing solutions.

Keywords: DNN inference · Edge Computing · DNN Partitioning and Offloading · Energy Efficiency · Greedy Genetic Algorithm

1 Introduction

Deep Learning (DL) approaches have become increasingly popular in a variety of Industrial Internet of Things (IIoT) applications [2]. Owing to the constrained computational capabilities inherent in IoT devices, they are ill-equipped independently infer complex DNN applications. Consequently, a prevalent approach in current research is to leverage edge servers to alleviate the computational burden, which denoted as the edge computing paradigm [7].

This paradigm situates computation in close proximity to data originators, which serves to diminish End-to-End (E2E) latency and bolster privacy safeguards [12]. Furthermore, for IoT devices that are predominantly powered by

batteries, edge computing can markedly curtail computational energy consumption, thereby conferring a significant advantage in terms of device maintenance frequency [1,10]. Against this backdrop, the partitioning and offloading of DNNs have emerged as a significant research direction [5]. This approach optimizes the utilization of computational resources by offloading different parts of the DNN model to IoT devices and edge servers, effectively reducing E2E latency and the power consumption of IoT devices.

In IIoT scenarios, imposing cost constraints on edge servers involved in collaborative inference is an important consideration [13]. However, existing research does not fully address the edge-device collaboration issue, failing to adequately consider the interrelationship between the power consumption of IoT devices, maximum E2E latency, and the cost of edge servers. For example, [8] overlooks the impact of end device power consumption and server costs. Meanwhile, [4] employs a weighted aggregation of various factors for multi-objective optimization, a method that requires recalibrating parameters through iterative experimentation tailored to the nuances of each application scenario.

Additionally, most studies focus on deploying a single DNN application for each IoT devices [8,11]. However, as IIoT smart systems become increasingly sophisticated, a single DNN model can no longer meet the diverse and demanding requirements of tasks, necessitating the collaborative operation of multiple complex DNN models. Moreover, the number of IoT devices is gradually increasing, making support for multiple applications and larger-scale IoT devices also crucial. However, formulating the partition offloading strategy commonly framed as a Mixed-Integer Nonlinear Programming (MINLP) problem, which recognized as belonging to the NP-Hard class [9]. This problem is not only highly complex but also comes with an enormous search space. When supporting multiple applications and larger-scale IoT devices, the search space expands further, exacerbating the complexity of the problem. This makes generating strategies for DNN model partitioning and offloading even more challenging. Therefore, an efficient solving algorithm is necessary.

Motivated by the above, this paper proposed the Greedy Genetic Algorithm (GGA). This algorithm combines a greedy strategy with Genetic Algorithms (GA) to tackle the issue of energy-efficient partitioning and offloading (EPOMA) for multiple DNN applications within cooperative edge environments. The GGA is designed with a comprehensive consideration of the interdependencies among the power consumption of IoT devices, the maximum E2E latency, and the costs associated with edge servers. It facilitates the deployment of a greater number of IoT devices, each capable of hosting multiple DNN applications, thereby enhancing the integration and application of deep learning technologies in the IIoT. The novelties of our proposed approach are the following:

– We introduce an optimization problem that seeks to minimize the inference power consumption of multiple DNN applications across IoT devices, subject to constraints on maximum E2E latency and edge server costs. The formulation of this problem incorporates a broader spectrum of IoT devices and edge servers, with the flexibility to deploy multiple DNN applications

on each IoT device. The holistic power consumption of the IoT devices is utilized as the key performance indicator for this optimization endeavor.
– This paper presents the GGA algorithm to address the optimization problem we have proposed, which utilizing a greedy strategy to reduce the problem space, thereby enabling the achievement of lower inference power consumption with fewer iterations.
– The efficacy of GGA has been substantiated through rigorous numerical experimentation. The findings reveal that the GGA surpasses current methodologies with respect to energy efficiency. Additionally, the effectiveness of the greedy strategy has been corroborated through ablation experiments.

The remainder of this paper is structured as follows. Section 2 provides a summary of the relevant literature. Section 3 defines the energy-efficient Multi-DNN application partition and offloading problem in edge-end collaboration environment. Section 4 introduces our proposed algorithm to tackle this problem. The methodologies and results of our experimental evaluation of the algorithm are detailed in Section 5. Finally, Section 6 synthesizes the findings of this study and posits avenues for future inquiry and advancement.

2 Related Work

In IIoT scenarios, there are several aspects that can be optimized in the collaborative inference of DNNs between IoT devices and edge servers, including the energy consumption, E2E latency, edge server costs, and data transmission volume. However, existing works often fail to consider all these factors appropriately. In this section, we review some of the algorithms from recent years based on the optimization objectives and summarize the factors they consider.

In terms of reducing task offloading latency, [8] proposed an improved particle swarm genetic algorithm (IPSGA) to achieve the optimal offloading strategy. The algorithm uses the variable acceleration coefficient with the number of iterations and the inertia weight with the success rate as the feedback parameters to improve the particle swarm optimization algorithm, and the GA is improved with the adaptive crossover probability and the adaptive mutation probability, aiming to minimize task offloading latency while considering the limitations of computing resources. [3]proposed a joint method by a self-adaptive DNN partition with cost-effective resource allocation to facilitate collaborative computation between IoT devices and edge servers, which can be proved to ensure the overall rental cost within an upper bound above the optimal solution while guaranteeing the latency for DNN-based task inference.

The research previously introduced has achieved efficient DNN inference, which is significant for the collaborative reasoning of edge devices. However, for IoT devices powered by batteries, the pursuit of the fastest inference speed may lead to unnecessary energy consumption, thereby increasing the maintenance frequency of the devices.

In the realm of multi-objective optimization, a multi-objective optimization problem was proposed in [11], which jointly considers offloading decisions, al-

location of communication and computation resources to minimize latency and cost. This study introduced a particle swarm optimization (PSO) based computation offloading (PSOCO) algorithm to obtain the Pareto-optimal solutions to the multi-objective optimization problem, while this method did not take into account energy consumption. [6] presented a container-based DNN partitioning placement and resource allocation strategy, which takes into account the varying processing capabilities, memory, and battery levels of heterogeneous IoT devices, aiming to simultaneously optimize E2E latency, service probability, and energy consumption. In [4], researchers jointly optimized the design of task partitioning and offloading to minimize the cost for each mobile edge device, including computational latency, energy consumption, and the price paid to servers. [14] designed a Deep Neural Network partitioning and Task Offloading (DPTO) algorithm based on Deep Reinforcement Learning, jointly optimizing the energy and latency issues of DNN partitioning and task offloading. However, this method did not consider the cost of the edge servers.

Although the aforementioned multi-objective optimization algorithms have incorporated the main optimization factors, the concurrent optimization of several objectives can engender superfluous complexity within the algorithms themselves. When applied to real-world settings with a substantial number of devices and intricate scenarios, these algorithms are prone to encountering protracted solution durations and the arduous task of identifying the global optimum.

In the IIoT scenario, the pursuit of minimal E2E latency is not always necessary. Different DNN applications, depending on the usage scenario, can meet real-time requirements as long as they do not exceed a certain E2E latency threshold. Moreover, the cost of edge servers is also a factor that cannot be ignored to ensure the feasibility and economic viability of the solution.

Therefore, our work considers minimizing the overall inference power consumption of multiple DNN applications on IoT devices under the constraints of maximum E2E latency and edge server costs. By adhering to the maximum E2E latency constraints, different DNN applications are enabled to meet their respective real-time requirements, which is of significant importance for extending the maintenance cycle and improving overall energy efficiency of devices.

3 Problem Modelling

This section provides a detailed exposition of the problem scenario, with meticulous modeling of the various factors involved in the problem, including multiple DNN applications, a range of IoT devices, several edge servers, the power consumption of IoT devices, E2E latency, and the cost of edge servers, all aimed at optimizing the power consumption across all IoT devices.

DNN Applications. For all DNN applications, each application corresponds to a DNN model. Let $\mathbb{D} = \{D_1, D_2, ..., D_n\}$ denote N different DNN models in the environment, where $D_i =< F_{D_i}, M_{D_i}, Data_{D_i}^{input}, Data_{D_i}^{result} >$

- $F_{D_i} = \{f_1, f_2, ..., f_{l_{D_i}}\}$ represents the floating point operations(FLOPs) for each layer.

- $M_{D_i} = \{m_1^{D_i}, m_2^{D_i}, ..., m_{l_{D_i}}\}$ represents the memory requirement for computation for each layer.
- $Data_{D_i}^{input} = \{s_1^{D_i}, s_2^{D_i}, ..., s_{l_{D_i}}^{D_i}\}$ is the size of input data for each layer.
- $Data_{D_i}^{result}$ is the size of result data.

and l_{D_i} denotes the layer number of D_i. Furthermore, this paper utilizes the torchstat library to perform an analysis on each DNN model, thereby obtaining three key attributes: the FLOPs for each layer, the memory requirements for computation and the size of the input data.

In this paper, our focus is on the impact of CPU and memory on inference performance. Therefore, we solely consider these two types of resources. The floating point operations per second (FLOPS) of a CPU can be calculated by Equation 1, where *cores* denotes the number of CPU cores, *cycles/second* signifies the number of cycles per core per second, and $FLOPs/cycle$ represents the number of floating-point operations per cycle per core [1]. Furthermore, for the sake of analysis, we consider the system to be quasi-static, with the wireless channels remaining unchanged during DNN applications inferencing [8].

$$FLOPS = cores \times \frac{cycles}{secend} \times \frac{FLOPs}{cycle} \tag{1}$$

Edge Servers. Let $\mathbb{E} = \{E_1, E_2, ..., E_s\}$ denote S edge servers in the environment, where $E_i =< FLS_i^{edge}, Mem_i^{edge}, num^{core}, P_i^{core}, P_i^{mem}, P_i^{traffic} >$. In this notation, FLS_i^{edge} represent the FLOPS of edge servers, Mem_i^{edge} indicates the maximum available memory, num^{core} is the core number, P_i^{core}, P_i^{mem} and $P_i^{traffic}$ represent the price per core-second of CPU, the price per MB-second of memory and the price per Mbps per second of traffic, respectively.

IoT Devices. Let $\mathbb{I} = \{I_1, I_2, ..., I_k\}$ denote K IoT devices in the environment, where $I_i =< FLS_i^{end}, P_i^I, P_i^M, P_i^T, Mem_i^{end}, B_i, Dep_i, ML_i, MC_i >$, v denotes the number of DNN applications will deploy on IoT device I_i.

- FLS_i^{end} represent the FLOPS of I_i.
- P_i^I, P_i^M, P_i^T represents the idle power, max power, transmitting power of I_i.
- Mem_i^{end} is the maximum available memory of I_i.
- $B_i = \{B_{i,1}, B_{i,2}, ..., B_{i,s}\}$ is the bandwidth to edge servers.
- $Dep_i = \{a_1^i, a_2^i, ..., a_v^i\}$ is the set of DNN applications to be deployed on I_i.
- $ML_i = \{l_1^i, l_2^i, ..., l_v^i\}$ is the maximum latency for each application.
- $MC_i = \{c_1^i, c_2^i, ..., c_v^i\}$ is the maximum inference cost for each application.

End to End Latency. For the j-th DNN application with $l_{a_j^i}$ layers on IoT device I_i, we selects the partition point p_j^i, where the initial p_j^i layers of the DNN task are processed locally, while the subsequent layers from $p_j^i + 1$ to $l_{a_j^i}$ are processed in a selected edge server, assume that $pos_j^i = x$, indicating that we have chosen edge server E_x for this task.

[1] https://en.wikipedia.org/wiki/FLOPS

The E2E latency can be expressed as $T^{total}_{a^i_j}$, as shown in Equation 2. This includes the local execution time $T^{I_i}_{a^j_i}$, the execution time on the edge server $T^{E_x}_{a^i_j}$ and the data transmission time between IoT device and edge server $T^{tran}_{a^i_j}$.

$$T^{total}_{a^i_j} = T^{I_i}_{a^i_j} + T^{E_x}_{a^i_j} + T^{tran}_{a^i_j} \tag{2}$$

After the application a^i_j is partitioned, the computational latency for offloading to IoT device I_i and edge server E_x represented by $T^{I_i}_{a^i_j}$ and $T^{E_x}_{a^i_j}$, respectively, which can be calculated by Equation 3 and 4, where f_{la} represents the FLOPs of layer la in a^i_j, $\theta^{a^i_j}_{I_i}$ and $\theta^{a^i_j}_{E_x}$ represents the percentage of computational resources allocated for application a^i_j on IoT device and edge server, respectively.

$$T^{I_i}_{a^i_j} = \sum_{la=1}^{p^i_j} \frac{f_{la}}{FLS^{end}_i \theta^{a^i_j}_{I_i}} \tag{3}$$

$$T^{E_x}_{a^i_j} = \sum_{la=p^i_j+1}^{l_{a^i_j}} \frac{f_{la}}{FLS^{edge}_x \theta^{a^i_j}_{E_x}} \tag{4}$$

$T^{tran}_{a^i_j}$ represent the time required for intermediate data transmission to the edge server and result data transmission to the IoT device, which can be computed by Equation 5, where T_{ex} represents the latency associated with establishing network connections, data link transmission, and so on.

$$T^{tran}_{a^i_j} = \frac{s^{a^i_j}_{p^i_j} + Data^{result}_{a^i_j}}{B_{i,x}} + T_{ex} \tag{5}$$

Edge Server Cost. Let $C_{a^i_j}$ represent the computation, memory, and data transmission cost of layers $p^i_j + 1$ to l of model a^i_j on edge server E_x, each calculated by multiplying the respective prices by the time duration, which can be expressed as Equation 6, where m_{la} represents the memory requirement for computation of layer la.

$$C_{a^i_j} = T^{E_x}_{a^i_j}(P^{core}_x num^{core} \theta^{a^i_j}_{E_x} + \sum_{la=p^i_j}^{l_{a^i_j}} m^{a^i_j}_{la}) + T^{tran}_{a^i_j} B_{i,x} \tag{6}$$

IoT Devices Energy Consumption. Let $E_{a^i_j}$ represent the energy consumption of inference from layer 1 to p^i_j of model a^i_j on end device I_i and the energy consumption of uploading the intermediate output data of the model to edge server E_x, each obtained by multiplying the corresponding power with

time. which can be expressed as Equation 7.

$$E_{a_j^i} = T_{a_j^i}^{I_i}(P_i^I + P_i^M \theta_{I_i}^{a_j^i}) + (\frac{s_{p_j^i}^{a_j^i}}{B_{i,x}} + T_{ex})P_i^T \tag{7}$$

Optimization Problem. We consider minimizing the energy consumption of all IoT devices under constraints of maximum E2E latency, edge server cost, computational capacity and memory resources. To achieve this, it is necessary to optimize the partition points of each DNN application, the location where the model is to be offloaded, the allocation of CPU resources for edge servers and IoT devices, represented by P, POS, θ_E and θ_I, respectively. The optimization problem is formulated as follows:

$$\min_{P,POS,\theta_E,\theta_I} \quad \sum_{i=1}^{k}\sum_{j=1}^{v} E_{a_j^i} \tag{8}$$

$$\text{s.t.} \quad C_{a_j^i} \le c_j^i, \quad T_{a_j^i}^{total} \le l_j^i \quad i = 1, \ldots, k, \quad j = 1, \ldots, v \tag{9}$$

$$\sum_{j=1}^{v} \theta_{I_i}^{a_j^i} \le 1 \quad i = 1, \ldots, k \tag{10}$$

$$\sum_{j=1}^{v}\sum_{la=1}^{p_j^i} m_{la}^{a_j^i} \le Mem_i^{end} \quad i = 1, \ldots, k \tag{11}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{v} Ind(pos_j^i, x)\theta_{E_x}^{a_j^i} \le 1, \quad x = 1, \ldots, s \tag{12}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{v}\sum_{la=p_j^i+1}^{l_{a_j^i}} Ind(pos_j^i, x) m_{la}^{a_j^i} \le Mem_x^{edge}, \quad x = 1, \ldots, s \tag{13}$$

$$0 \le \theta_{I_i}^{a_j^i}, \theta_{E_x}^{a_j^i} \le 1, \quad 1 \le pos_j^i \le s, \quad 0 \le p_j^i \le l_{a_j^i} \tag{14}$$

where Equation 9 represents the cost constraints for edge servers across various applications and the maximum E2E latency constraints for each application. Equation 10 and 11 indicate the computational capacity and memory constraints for IoT devices, respectively. Additionally, Equation 12 and 13 address the computational and memory capacity constraints for edge servers, respectively. The indicator function $Ind(pos, x)$ returns 1 when the offloading position p_j^i of DNN model a_j^i is equal to x, and returns 0 when x does not equal to pos.

4 Proposed Algorithm

In this section, we describe our proposed GGA. The algorithm flowchart is shown in Fig. 1. The key advantage of our algorithm is that during the GA iteration

process, a portion of the optimization variables is solved using a greedy strategy, which narrows down the problem space and makes the search more efficient. Consequently, it becomes easier to find the optimal solution during iterations.

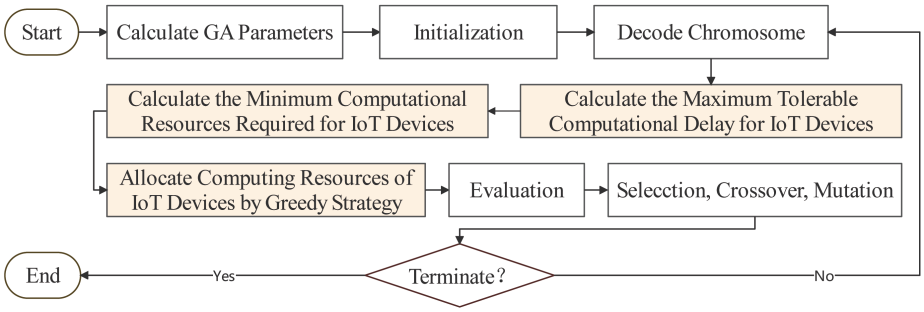

Fig. 1. Overall of Greedy Genetic Algorithm.

To transform the constrained optimization problem into an unconstrained one, we employ the penalty function method, which is commonly used in intelligent optimization algorithms to constrain the objective function. The fitness function is used to evaluate the performance of heuristic algorithms. After each iteration, the fitness value is calculated for all individuals in the population to assess the quality of a single chromosome. The fitness function F can be represented by Equation 15, where the fitness function consists of the objective function plus the penalty function, and ρ represents the penalty coefficient.

$$\begin{aligned} Fit = \sum_{i=1}^{k}\sum_{j=1}^{v}(E_{a_j^i} + \rho(\max(0, C_{a_j^i} - c_j^i) \\ + \max(0, T_{a_j^i}^{total} - l_j^i) + \max(0, \sum_{j=1}^{v}\sum_{la=1}^{p_j^i} m_{la}^{a_j^i} - Mem_i^{end}) \\ + \max(0, \sum_{i=1}^{k}\sum_{j=1}^{v}\sum_{la=p_j^i+1}^{l_{a_j^i}} Ind(pos_j^i, x) m_{la}^{a_j^i} - Mem_x^{edge}))) \end{aligned} \tag{15}$$

Algorithm 1 shows the process of our proposed Greedy Genetic Algorithm, the initial population size of the genetic algorithm is denoted as GA_{ps}, which can be expressed by Equation 16. The maximum number of iterations for the genetic algorithm, GA_{maxi}, can be represented by Equation 17.

$$GA_{ps} = 100 + 100(Dep^{num}/30) \tag{16}$$

$$GA_{maxi} = 50 + 2Dep^{num} \tag{17}$$

Algorithm 1 Greedy Genetic Algorithm

Input: DNN set D, IoT devices set I, Edge servers set E.
Output: The optimal strategy $best_x$.

Calculate $Dep^{num}, GA_{ps}, GA_{maxi}$;
Initialize chromosome populations, $best_x$, $GA_i = 1$;
while $GA_i \leq GA_{maxi}$ **do**
  Decode P,POS,θ_E from chromosome;
  Calculate T_I^{max}, R_I^{min};
  if sum(R_I^{min}) < 1 **then**
    Initialize list $tempR$;
    for $r_i \in R_I^{min}$ **do**
      Append $\sqrt{r_i}/sum(T_I^{max})$ to $tempR$;
    end for
    if all elements in $tempR$ no less than the corresponding element in R_I^{min} **then**
      $R_I = tempR$;
    else
      $R_I =$ Normalize(R_I^{min});
    end if
  else
    $R_I =$ Normalize(R_I^{min});
  end if
  Calculate Fit;
  Update $best_x$ as the best chromosome;
  Selecting the next generation of chromosome individuals;
  Execute two point crossover;
  Execute uniform variation;
  $GA_i = GA_i + 1$;
end while
return $best_x$

The process of initializing the GA population begins with tallying the number of DNN applications for deployment on all IoT devices, which denoted by Dep^{num}. Subsequently, we generate the partition points P, offloading positions POS and the proportion of CPU resource allocation on edge server θ_{E_x} according to constraints 14 for all Dep^{num} applications, these elements are then concatenated into a vector, encoded by Gray code, to form an initial chromosome. This procedure is repeated GA_{ps} times to fully initialize the GA population.

During each iteration, chromosome decoding provides the partition points, the offloading positions, and the percentage of server computational resource allocation for each DNN application. By applying Equations 4 and 5 in conjunction with Constraint 9, the maximum permissible computational latency for applications on IoT devices is ascertained, which denoted by T_I^{max}. Subsequently, Equation 3 is employed to deduce the minimum proportion of computational resources that need to be allocated to the IoT devices, which denoted by R_I^{max}. With Equations 7 and 3 in mind, it shows that the value of $E_{a_j^i}$, given fixed parti-

tion points, offloading locations, and server computational resource distribution ratios for DNN applications, is dependent on the IoT device's computational resource allocation ratio, $\theta_{I_i}^{a_j^i}$, as $\theta_{I_i}^{a_j^i}$ increases, $E_{a_j^i}$ decreases.

Therefore, to minimize the power consumption of IoT devices, it is necessary to maximize the computing resources of IoT devices, ideally reaching 100%. Computational resources are redistributed based on the maximum tolerable latency for each application and are assessed against the minimum required allocations. If the new allocation suffices, the final distribution is made in proportion. Otherwise, the minimum allocations are normalized to yield a holistic offloading strategy and resource allocation plan for all models.

5 Experimental Evaluation

This section presents the experiment design and the experiment results to evaluate the performance of GGA. We evaluate the performance of all competing algorithms in terms of fitness value based on the resource allocation solutions found by the respective algorithms.

5.1 Experimental Setup

Simulation Environment. In our simulation experiments, we emulated the Raspberry Pi 4B with 1G and 4G memory versions. Their computing capabilities, idle power, maximum power consumption and transmission power consumption were evaluated by solving a set of dense linear algebraic equations with N variables using the Gaussian elimination method provided by the Linear system package. This method is used to assess the FLOPS and to record the energy consumption of IoT devices.

We considered simulating the 2-core, 4G memory instance of Alibaba Cloud's Edge Node Service (ENS), with the CPU of E5-2680 v3 (Haswell), and we refer to their documentation to calculate the costs of edge server CPU, memory, and public network bandwidth per second. Based on the number of cores and Equation 1, we determined the computing capability of this instance. Table 1 shows the configuration of the IoT devices.

Table 1. Configurations of IoT Devices and an Edge Server

Device	Computing capability (GFLOPS)	Memory (GB)	Idle Power (W)	Trans. Power (W)	Max. Power (W)
Raspberry Pi 4B(1G)	9.92	1	2.9	3.1	7.9
Raspberry Pi 4B(4G)	9.69	4	2.8	3.0	8.2
Edge Server	80	4	-	-	-

In addition, we employ prevalent DNN models in the experiment encompass ResNet18, ResNet50, VGG11, VGG16, AlexNet, and GoogleNet [4,15]. Each

DNN implementation is derived from the torchvision.models standard library and has been carefully scrutinized using the torchsummary function within ptflops to acquire comprehensive details, including the architecture, computational intensity, and inter-layer data transmission. On each IoT device, one of ResNet18, ResNet50, VGG11, or VGG16 is executed in conjunction with AlexNet and GoogleNet. Consequently, three DNN applications are inferred on each IoT device at the same time.

Experimental Strategy. This paper compares GGA to three baseline methods: IPSGA [8], PSOCO [11] and GA. The IPSGA is based on PSO algorithm, where during the PSO iteration, a subset of particles are treated as chromosomes. Subsequently, the application of GA iterations facilitates the discovery of superior particles, which in turn augments the overall search capability of the algorithm. PSOCO also based on the PSO algorithm, has been empirically demonstrated to surpass the performance of the GA within the context of the scenarios presented in their corresponding scholarly work. As our algorithm is an enhancement based on GA, it is appropriate to use GA as a control in ablation studies, thereby incorporating it into the baseline comparisons.

Since the aforementioned algorithms all require random initialization of the starting solutions, to enhance the reliability of the experimental results and minimize errors, we conduct the experiments three times with fixed random seeds of 1, 2, and 3, respectively. We then take the best solution from these three trials to evaluate the performance of the algorithms.

Experiment Setting. The GGA and GA algorithms are initialized with a mutation rate set to 0.001 and a tournament size of 5. The IPSGA and PSOCO algorithms, which are tailored to address distinct optimization issues and scenarios, require parameters that diverge from those outlined in their original publications. Through rigorous experimental evaluation, the IPSGA algorithm has been calibrated with the following parameters: $w = 0.65$, $c1 = 1$, $c2 = 0.5$, $ga_{mini} = 10$, $ga_{maxi} = 20$, $ga_{ps}^{min} = 5$, $ga_{ps}^{max} = 10$, $ga_{num}^{max} = 10$, and $ga_{num}^{min} = 5$. The PSOCO algorithm, on the other hand, has been fine-tuned with parameters of $w = 0.65$, $c1 = 1$, and $c2 = 0.5$.

All algorithms use the same number of iterations and initial solution scale, as determined by Equation 16 and 17, and set the penalty coefficient $\rho = 1e5$. Taking into account the IIoT scenarios and the limited computational capabilities of edge devices, the maximum E2E latency for the ResNet18, ResNet50, VGG11, and VGG16 are limited to 3 seconds, with a edge server cost capped at 5e-5. For the AlexNet and GoogleNet, the maximum E2E latency is limited to 1 second, with a edge server cost capped at 3e-5. The termination condition is the number of evolutionary generations reaching the pre-specified maximum number of iterations.

5.2 The Impact of the Number of End Devices

To validate the effectiveness of GGA under a larger number of IoT devices, the experiment sets up IoT devices to increase from 5 to 100 units, while controlling the server resources to three instances of edge servers. The bandwidth between IoT devices and the servers are set at 150Mbps. The iteration processes with 5 and 100 IoT devices are shown in Fig. 2(a) and Fig. 2(b), respectively.

Fig. 2(a) indicates that when there are only 5 IoT devices, PSOCO, IPSGA, GA, and GGA can all identify partitioning and offloading strategies with relatively low power consumption. However, fig. 2(b) and fig. 2(c) evident that as the number of IoT devices increases, only GA and GGA remain effective. PSOCO and IPSGA become trapped in local optima, resulting in strategies that do not meet the maximum E2E latency or cost constraints.

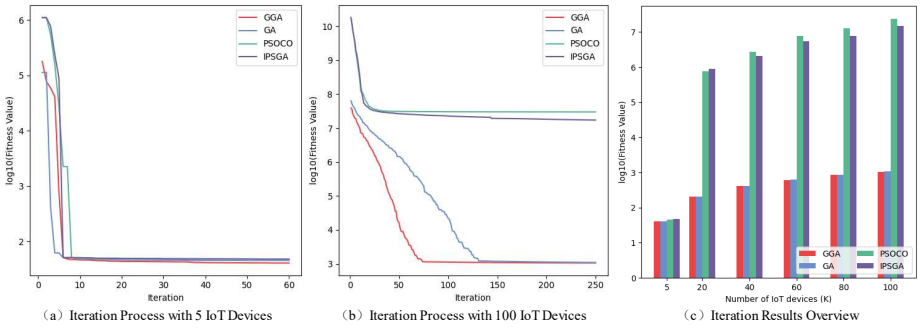

Fig. 2. Experimental results under different numbers of IoT devices.

Compared to GA, GGA can achieve the same performance with fewer iterations. Specifically, when there are 100 IoT devices, GGA reaches the same performance as GA after 250 iterations in just 139 iterations, reducing the number of iterations by 44.4%. After 250 iterations, the strategy generated by GGA has lower power consumption than that of GA, with values of 1052.57 and 1089.29, respectively, representing a 3.37% reduction in power consumption. This demonstrates that notwithstanding an escalation in the quantity of IoT devices, our greedy strategy can still effectively reduce the search space, thereby finding better solutions in fewer iterations.

5.3 The Impact of Link Bandwidth

To validate the effectiveness of GGA under different network bandwidth conditions, the server resources are controlled to consist of three edge server instances, with 50 IoT devices. The network bandwidth is increased from 100 Mbps to 300 Mbps. The iteration process at 100Mbps and 300Mbps is shown in Fig. 3(a) and Fig. 3(b), respectively, indicated that IPSGA and PSOCO, when dealing with 50

IoT devices, cannot find the global optimal solution even with increased network bandwidth. In contrast, GGA and GA are capable of escaping local optima.

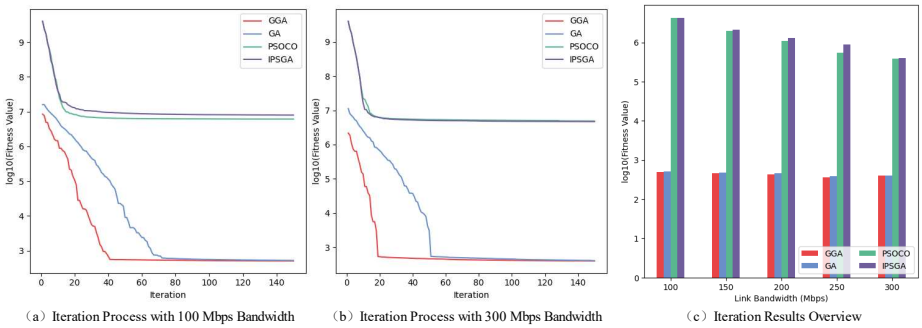

Fig. 3. Experimental Results Under Different Link Bandwidth.

From Fig. 2(c), it is evident that as the link bandwidth increases from 100 to 250 Mbps, the total energy consumption gradually decreases. However, when the bandwidth reaches 300 Mbps, the total energy consumption begins to rise. This is because higher bandwidth leads to increased data transmission costs, and due to cost constraints, the algorithms tend to split the DNN model data at layers with less transmission, resulting in increased computational tasks on the IoT devices and consequently higher power consumption.

Compared to GA, GGA requires fewer iterations to achieve the same effect. Specifically, at a bandwidth of 250 Mbps, GGA reaches the performance of GA's 150th iteration after just 110 iterations, reducing the number of iterations by 26.6%. Moreover, under the same number of iterations, the strategy generated by GGA has lower energy consumption than that of GA, with values of 405.46 and 416.10, respectively, representing a 2.55% reduction in energy consumption. This demonstrates that our greedy strategy can still effectively reduce the search space under different bandwidth, thereby finding better solutions in fewer iterations.

5.4 The Impact of Edge Server Computing Resources

To verify the effectiveness of GGA when server resources vary, such as in situations of abundance or scarcity, the number of edge servers is increased from 1 to 5, with 50 IoT devices, and the bandwidth between IoT devices and servers are set at 150 Mbps. The iteration processes with 1 and 5 edge servers are shown in Fig. 4(a) and Fig. 4(b), respectively, indicated that IPSGA and PSOCO, when dealing with 50 IoT devices, cannot find the global optimal solution even with increased edge servers. In stark contrast, the GGA and GA algorithms have demonstrated an adeptness at eluding local optima, thereby showcasing their potential for uncovering more efficient solutions.

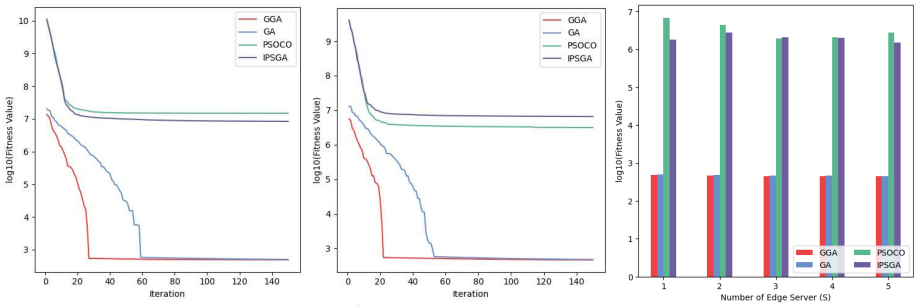

Fig. 4. Experimental results under different number of edge servers.

From Fig. 4(c), we can observe that after the number of edge servers increases to 3, due to the cost constraints of each application, more abundant computational resources do not lead to a significant reduction in the power consumption of the end devices. Compared to GA, GGA can achieve the same effect with fewer iterations. Specifically, when there are 3 edge servers, GGA reaches the performance of GA's 150th iteration after just 107 iterations, reducing the number of iterations by 28.6%. Moreover, under the same number of iterations, the strategy generated by GGA has lower power consumption than that of GA, with values of 459.40 and 471.88, respectively, representing a 2.64% reduction in power consumption. This proves that our greedy strategy can still effectively reduce the search space under different computational resource conditions, thereby finding better solutions in fewer iterations.

6 Conclusion and Future Work

This paper introduces an optimization problem within the scope of the IIoT, focusing on the minimization of inference power consumption for multiple DNN applications across an extensive array of IoT devices. This problem is addressed under the constraints of maximum E2E latency and the costs associated with edge server utilization. To tackle this issue, the GGA is proposed, which harnesses the iterative process of genetic algorithms to determine the partitioning points for DNN applications and the corresponding offloading strategies. Subsequently, a greedy strategy is implemented for the allocation of computational resources to IoT devices, thereby effectively condensing the search space. The experimental outcomes indicate that the GGA is superior to the baseline algorithm, as it reduces the energy consumption of IoT devices in a significantly reduced number of iterations.

Potential future research endeavors may involve refining problem modeling, and based on the proposed partitioning and offloading method, supporting deployment strategies for multiple instances of the same DNN application, instead of being limited to a single instance.

References

1. Abrar, M., Ajmal, U., Almohaimeed, Z.M., Gui, X., Akram, R., Masroor, R.: Energy efficient uav-enabled mobile edge computing for iot devices: A review. IEEE Access **9**, 127779–127798 (2021)
2. Chang, Z., Liu, S., Xiong, X., Cai, Z., Tu, G.: A survey of recent advances in edge-computing-powered artificial intelligence of things. IEEE Internet of Things Journal **8**(18), 13849–13875 (2021)
3. Dong, C., Hu, S., Chen, X., Wen, W.: Joint optimization with dnn partitioning and resource allocation in mobile edge computing. IEEE Transactions on Network and Service Management **18**(4), 3973–3986 (2021)
4. Gao, M., Shen, R., Shi, L., Qi, W., Li, J., Li, Y.: Task partitioning and offloading in dnn-task enabled mobile edge computing networks. IEEE Transactions on Mobile Computing **22**(4), 2435–2445 (2023)
5. Kakolyris, A.K., Katsaragakis, M., Masouros, D., Soudris, D.: Road-runner: Collaborative dnn partitioning and offloading on heterogeneous edge systems. In: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6 (2023)
6. Kim, T., Park, H., Jin, Y., Lee, S.S., Lee, S.: Partition placement and resource allocation for multiple dnn-based applications in heterogeneous iot environments. IEEE Internet of Things Journal p. 9836–9848 (Jun 2023)
7. Kong, L., Tan, J., Huang, J., Chen, G., Wang, S., Jin, X., Zeng, P., Khan, M., Das, S.K.: Edge-computing-driven internet of things: A survey. ACM Comput. Surv. **55**(8) (dec 2022)
8. Li, C., Chai, L., Jiang, K., Zhang, Y., Liu, J., Wan, S.: Dnn partition and offloading strategy with improved particle swarm genetic algorithm in vec. IEEE Transactions on Intelligent Vehicles pp. 1–11 (2023)
9. Liu, G., Dai, F., Xu, X., Fu, X., Dou, W., Kumar, N., Bilal, M.: An adaptive dnn inference acceleration framework with end–edge–cloud collaborative computing. Future Generation Computer Systems **140**, 422–435 (2023)
10. Liu, J., Pang, Y., Ding, H., Cai, Y., Zhang, H., Fang, Y.: Optimizing iot energy efficiency on edge (eee): A cross-layer design in a cognitive mesh network. IEEE Transactions on Wireless Communications **20**(4), 2472–2486 (2021)
11. Luo, Q., Li, C., Luan, T.H., Shi, W.: Minimizing the delay and cost of computation offloading for vehicular edge computing. IEEE Transactions on Services Computing **15**(5), 2897–2909 (2022)
12. Nain, G., Pattanaik, K., Sharma, G.: Towards edge computing in intelligent manufacturing: Past, present and future. Journal of Manufacturing Systems **62**, 588–611 (2022)
13. Wang, P., Xu, J., Zhou, M., Albeshri, A.: Budget-constrained optimal deployment of redundant services in edge computing environment. IEEE Internet of Things Journal **10**(11), 9453–9464 (2023)
14. Zhang, J., Ma, S., Yan, Z., Huang, J.: Joint dnn partitioning and task offloading in mobile edge computing via deep reinforcement learning. Journal of Cloud Computing **12** (08 2023)
15. Zhang, X., Mounesan, M., Debroy, S.: Effect-dnn: Energy-efficient edge framework for real-time dnn inference. In: 2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). pp. 10–20 (2023)