A Multimodel Edge Computing Offloading Framework for Deep-Learning Application Based on Bayesian Optimization

Zidi Zhao, Hong Zhang, Liqiang Wang, and Haijun Huang

***Abstract*—With the rapid development of the Internet of Things (IoT), data generated by IoT devices are also increasing exponentially. The edge computing has alleviated the problems of limited network and transmission delay when processing tasks of IoT devices in traditional cloud computing. And with the popularity of deep learning, more and more terminal devices are embedded with artificial intelligence (AI) processors for higher processing capability at the edge. However, the problems of deep-learning task offloading in a heterogeneous edge computing environment have not been fully investigated. In this article, a multimodel edge computing offloading framework is proposed, using NVIDIA Jetson edge devices (Jetson TX2, Jetson Xavier NX, and Jetson Nano) and GeForce RTX GPU servers (RTX3080 and RTX2080) to simulate the edge computing environment, and make binary computational offloading decisions for face detection tasks. We also introduce a Bayesian optimization algorithm, namely, modified tree-structured Parzen estimator (MTPE), to reduce the total cost of edge computation within a time slot including response time and energy consumption, and ensure the accuracy requirements of face detection. In addition, we employ the Lyapunov model to obtain the harvesting energy between time slots to keep the energy queue stable. Experiments reveal that MTPE algorithm can achieve the globally optimal solution in fewer iterations. The total cost of multimodel edge computing framework is reduced by an average of 17.94% compared to a single-model framework. In contrast to the double deep Q-network (DDQN), our proposed algorithm can decrease the computational consumption by 23.01% for obtaining the offloading decision.**

***Index Terms*—Bayesian optimization, deep learning, edge computing, Lyapunov drift function, modified tree-structured Parzen estimator (MTPE), multimodel.**

I. Introduction

IN RECENT years, the variety and number of Internet of Things (IoT) devices have been growing rapidly, according to *Statista* [1], the number of IoT devices worldwide is forecast to almost triple from 8.74 billion in 2020 to more than 25.4 billion IoT devices in 2030. IoT devices in the consumer segment, such as smartphones, connected (autonomous) vehicles, asset tracking & monitoring, account for around 60 percent of all connected IoT devices in 2020. The explosive growth of IoT devices and the development of 5G technology have subsequently promoted their applications in Internet of Medical Things (IoMT) [2], Internet of Vehicles (IoV) [3] and other scenarios [4], [5], [6]. Unfortunately, various data generated by IoT devices are not just literal data and pictures, but also include streaming video that take up a bunch of resources and are time-critical. In intelligent scenarios, although cloud platforms may provide higher computing capacity, there are many issues with them, such as limited network resources [7], transmission delay [8], privacy leakage [9] and other problems [10], which could be considerably challenging for IoT applications. The concepts of edge computing [11], fog computing [12], and cloudlet [13] have been put forward to alleviate these problems by moving computing resources from the cloud to the edge to reduce cloud computing load beforehand.

The common three-layer edge computing framework, also known as the "Cloud–Edge–End" framework [14], is shown in Fig. 1. The traditional end device is a device that only has the function of data collection, and the edge server nearby performs data processing tasks. However, with the upgrade of end devices, they can also perform edge computing with embedded artificial intelligence (AI) processors [15], such as surveillance camera [16], unmanned aerial vehicle (UAV) [17], which further relieve the pressure of the cloud. Furthermore, edge and cloud computing are not mutually exclusive. In large scenarios, numerous edge servers can form an edge cloud closer to the terminal, which can be a part of cloud computing.

An AI application task can be divided into several subtasks, which can be offloaded according to the capability of edge servers and subtask complexity. For example, a face recognition task can be roughly divided into four subtasks: 1) image acquisition; 2) image preprocessing; 3) face detection; and 4) face recognition. The first three steps can be handled by edge devices, whereas the feature extraction results are usually transmitted to the cloud for face matching and recognition. Koubaa et al. [18] compared the time consumption of face recognition tasks in cloud, edge and hybrid architectures, and found that the hybrid architecture took the shortest time, which

Manuscript received 2 April 2023; accepted 22 May 2023. Date of publication 26 May 2023; date of current version 9 October 2023. The work of Zidi Zhao, Hong Zhang, and Haijun Huang was supported in part by the Natural Science Foundation of Hebei Province of China under Grant F2019201361, and in part by the Science and Technology Research Project of Hebei Higher Education Institutions under Grant QN2020133. The work of Liqiang Wang was supported in part by NSF under Grant NSF-1952792. *(Corresponding author: Hong Zhang.)*

Zidi Zhao, Hong Zhang, and Haijun Huang are with the School of Cyber Security and Computer, Hebei University, Baoding 071002, Hebei, China (e-mail: zhaozidi@stumail.hbu.edu.cn; hzhang@hbu.edu.cn; huanghaijun@stumail.hbu.edu.cn).

Liqiang Wang is with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: liqiang.wang@ucf.edu).

Digital Object Identifier 10.1109/JIOT.2023.3280162

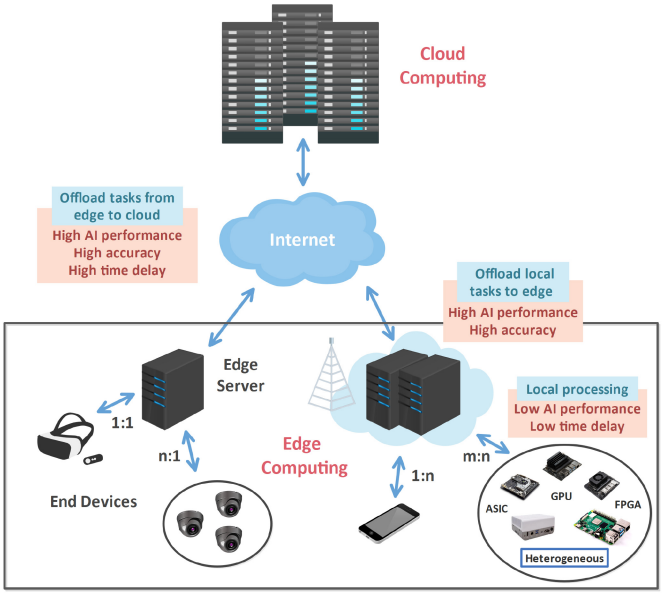

Fig. 1. Edge computing framework.

means the cloud and edge can complement each other and jointly improve task processing efficiency. There are four edge computing offloading schemes between end devices and edge servers, which are one-to-one, one-to-many, many-to-one, and many-to-many [19]. These schemes can be used in various offloading scenarios, which fully embodies the flexibility and scalability of edge computing.

However, there are still several open issues and challenges in edge computing offloading.

1) *Model Selection:* Due to the heterogeneous environment of the "Cloud–Edge–End" framework, a single deep-learning model may not fit all computing platforms well. For example, GPU resources of end devices cannot support the demands of sophisticated neural networks such as Resnet50. Then, we need to choose certain lightweight models suitable for heterogeneous end devices to process AI computing requests.
2) *Tradeoff of Accuracy and Efficiency:* In practice, the number of tasks generated by end devices in different time slots is not uniform. As computation scale up, the accuracy disparity of different models becomes increasingly pronounced. Lower end devices may not support the high-accuracy requirement, while offloading to the edge server with a high-accuracy model diminishes the efficiency. Therefore, the accuracy and efficiency at various task levels cannot be balanced by deploying a single model on edge servers.
3) *Complex Offloading Algorithm:* With the expansion of the computation scale, offloading algorithms such as metaheuristic approach and deep reinforcement learning become much more complex than before in terms of initialization parameters and iterations.
4) *Energy Supply:* The total cost of task offloading within a single time slot is often considered in previous research for edge computing. Yet how to supply energy among time slots to ensure the stability of energy queue is still a critical problem that needs to be resolved.

In response to these shortcomings and challenges, we propose a multimodel edge computing framework for a many-to-many offloading scheme to improve the efficiency and accuracy of AI applications on edge IoT devices, keep energy consumption within an ideal range, create a more intelligent edge computing environment, and make end and edge devices highly cohesive. Here, we take face recognition with a wide range of scenarios as an example, because face recognition is conducted on a variety of end and edge devices, such as stations, commercial districts and military bases, meanwhile, face recognition often requires good performance on accuracy, efficiency, and energy consumption. In our experiment, we deploy face detection models on different Jetson devices and edge servers to construct edge computing environments. In particular, we propose a modified Bayesian optimization algorithm named modified tree-structured Parzen estimator (MTPE) to guide the offloading decision.

The offloading indicators are regarded as the parameters that need to be configured by MTPE to minimize the time and energy consumption of face detection, maintain the accuracy at an ideal level, and improve the Quality of Experience (QoE). Finally, we ensure energy stability by minimizing the upper bound of the Lyapunov drift function, and determining the optimal values of the initial energy and the energy replenished between time slots.

The main contributions of this article are summarized as follows:

1) We propose a novel edge computing offloading framework based on multiple models conduct extensive experiment on real end devices including Jetson GPUs and edge servers with different backbone networks to evaluate our framework on various scenarios. Deploying multiple models in edge computing environment can extremely balance the accuracy and efficiency at different task levels.
2) We introduce an MTPE algorithm to minimize the offloading cost of the whole edge computing environment in a single time slot. The optimized objectives such as accuracy, time and energy consumption are fully considered in our algorithm to choose the global optimal solution as possible by establishing the probability density function.
3) We also provide a dynamic energy adjustment algorithm to keep the energy queue stable. The Lyapunov drift function is employed to describe the energy change of the dynamic energy queue and minimize these fluctuations by limiting the upper bound of the drift function. To obtain the harvesting energy among time slots and the limitation of the initial energy, we calculate the minimum point of a unary quadratic function, which keeps the energy adequate throughout the processing cycle.

The remainder of this article is organized as follows. The related work is summarized in Section II. The system model and our algorithm are introduced in Section III. Experimental results are given in Section IV, and the conclusion and future work are presented in Section V.

II. Related Work

At present, there are many research achievements in the field of edge computing. In addition to the common research on offloading problems, deep-learning technology in AI applications has been widely integrated into the edge computing framework. Wang et al. summarized the concepts of edge intelligence and intelligent edge in [20].

For edge intelligence, most of the research on the application of embedded AI devices in edge computing systems is related to object detection and recognition. Li et al. [21] proposed a driver fatigue detection system based on CNN and tested the performance of the detection system in real driving scenarios. In this article, Jetson Nano was used as an edge computing device for real-time detection to improve robustness and accuracy of the system. Chang et al. [15] presented a wearable assistive system based on AI edge computing technology and adopted deep-learning technology for real-time recognition of zebra crossing images. Liu et al. [22] designed a food recognition system based on edge computing to make an accurate dietary assessment and overcome the problems of system delay and low battery life of mobile devices in mobile cloud computing. Both [3] and [17] studied the traffic video surveillance system based on edge computing platform. Wan et al. [3] offloaded vehicle detection tasks to edge nodes with Jetson TX2, while the system designed in [17] was embedded in the UAV to track and detect vehicles in real-time. Neto et al. [23] proposed a distributed system for video analysis, which divided the heavy processing of large-scale video streams into various machine learning tasks and deployed these tasks as data processing workflows on edge devices equipped with neural network hardware accelerators. Rajavel et al. [6] proposed a video surveillance system based on edge computing for object tracking and behavior recognition in IoMT. In this article, the detection of moving objects was improved by combining background subtraction and the DNN algorithm, which brought robustness and intelligence to the distributed video surveillance system. These studies fully demonstrated the wide applications of deep learning in edge computing frameworks. However, they only considered single model to solve deep-learning problems rather than multimodel systems when offloading, which did not take the tradeoff between efficiency and accuracy into account.

For intelligent edge, how to design an edge computing system with intelligent offloading scheme under the multiple constraints of network, communication, computing power, and energy consumption [24], [25], [26], [27] is the key to improve the efficiency of edge computing. Common offloading decision approaches include AI-based approaches [28] (e.g., deep Q-network [29], [30], [31], [32]), Lyapunov optimization [33], [34], [35], [36], [37], metaheuristic algorithm [38], [39], [40], [41], [42], (non-)convex optimization [43], [44], etc. Tu et al. [29] proposed the online predictive offloading algorithm based on double deep Q-network (DDQN) and long short-term memory networks for cost minimization, which integrates the processing latency, processing energy consumption and the task throw rate of latency-sensitive tasks. In [33], a blockchain-enabled IoT–Edge–Cloud computing architecture that benefits both from mobile cloud computing and mobile-edge computing (MEC) was proposed. The authors derived an adaptive offloading-decision algorithm EEDTO by utilizing the Lyapunov optimization problem such that the energy consumption of the IoT device can be minimized when only sacrificing a little delay. Natesha and Guddeti [38] designed a service placement strategy based on metaheuristic hybrid algorithms MGAPSO and EGAPSO. They implemented a two-level fog computing framework developed by docker and container techniques to minimize service costs and ensure the Quality of Service (QoS) for Industrial IoT (IIoT) applications. Xue et al. [39] proposed an efficient offloading scheme for DNN inference acceleration in a three-layer collaborative environment. For migration plan, algorithm PSO-GA is applied to obtain the distribution of DNN layers under the server with the lowest migration delay, and for uploading plan, a layer merge uploading algorithm is proposed to obtain DNN partitions and their upload order with efficient DNN query performance. Deng et al. [43] studied the application of MEC in the air-to-ground-integrated wireless network. They optimized the offloading decision of the DNN model, resource allocation, and UAV route based on energy consumption and resource constraints. It i s worth noting that the UAV deploys well trained DNN models with different model input sizes to satisfy various QoS requirements of IoT devices. These studies considered various influencing factors in edge offloading, such as delay, energy consumption, and communication resources. The algorithm needs to tune a large number of parameters. Moreover, the tuning process for the proposed algorithms such as reinforcement learning is very slow, which cannot guarantee timeliness in real-time scenarios.

How to combine deep-learning tasks in real scenes with edge computing, minimize task delay, energy consumption, task discarding, and cost payment and maximize computing speed and energy efficiency through computational offloading are the critical problems in making edge computing come into practice. Based on the above summary of the existing work, deep learning has been widely applied in edge computing, but how to offload deep-learning tasks under the heterogeneous edge computing framework remains to be discussed. In addition to the time and energy consumption that most people focus on, ensuring the accuracy of AI application services is also an issue that cannot be ignored, which also motivates the work of this article.

III. System Model and Solution Algorithm

In this section, we introduce the edge computing offloading architecture that contains different types of edge devices running multiple models. We first introduce edge system architecture including model selection, response time and energy consumption. Then, we formulate the problem within a time slot with the offloading decision matrix and provide the solving process of the MTPE algorithm, which is improved based on TPE [45]. Finally, we provide the energy adjustment algorithm to keep the energy queue stable between time slots. The notations used in this article are summarized in Table I.

TABLE I
SUMMARY OF KEY NOTATIONS

Notation	Description
$\mathcal{M}$	set of edge servers
$\mathcal{K}$	set of inference models deployed on the edge servers
$\mathcal{N}$	set of end devices
$\mathcal{T}$	set of time slots
$\mathcal{S}$	set of tasks generated by $\mathcal{N}$ in each time slot
s_t	task set in time slot t
Q_k	set of the accuracy of model k
R_t	task generation rate within a time slot
D_n^l	response time of end device n
D_m^k	response time of server m using model k
E_n^l	energy consumption of executing task on end device n
E_m^k	energy consumption of executing task on edge server m
D_m^t	time consumption of transferring task to edge server m
E_m^t	energy consumption of transferring task to edge server m
D_n^e	total time consumption of executing task on edge server m
E_n^e	total energy consumption of executing task on edge server m
E_n^{max}	maximum energy consumption for a task
p_l	power consumption of end device
p_e	power consumption of edge server
p_t	transmission power
p_t^{max}	maximum transmission power
A_n, B_n^m	offloading indicators
x_t	group of offloading indicators in time slot t
E_t	total energy consumption of executing task in time slot t
E_t^{max}	maximum total energy consumption in time slot t
e_t	harvesting energy between each time slot
e_t^{max}	maximum harvesting energy
B_0	initial energy within $\mathcal{T}$
B_t	remaining energy at the end of time slot t

A. Edge System Architecture

As depicted in Fig. 2, we consider a multimodel edge computing offloading framework with embedded edge devices (namely, end devices) and edge servers. The entire offloading process is mainly divided into three stages: 1) *stage 1:* the end devices transmit the information of the generated tasks to a central server containing the scheduler within a single time slot; 2) *stage 2:* the target node and model for the task are returned to each end device by the central server after scheduling tasks using the scheduler's offloading method; and 3) *stage 3:* the end devices offload the tasks to the target nodes, and the target nodes send back the results after processing the tasks. The set of edge servers is denoted as $\mathcal{M} = \{1, 2, \ldots, M\}$, and the set of face detection inference models deployed on the edge servers is denoted as $\mathcal{K} = \{1, 2, \ldots, K\}$. $\mathcal{N} = \{1, 2, \ldots, N\}$ is the set of embedded edge devices, and only one inference model in $\mathcal{K}$ is deployed on each edge device. From Fig. 2, an edge cloud composed of edge servers connects to various types of end devices and handles their uplink transmission tasks. The same type of end devices can deploy different models, while different types of end devices can also deploy the same model. Therefore, various model deployment scenarios are fully considered in our edge computing framework.

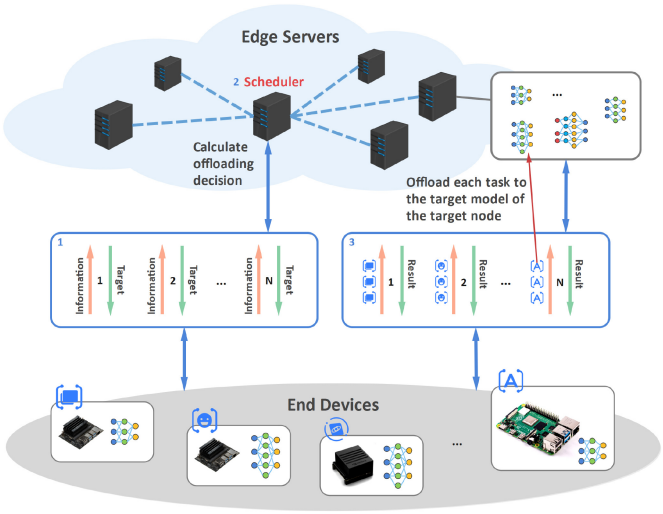

Fig. 2. Multimodel edge computing offloading framework.

The inference process of face detection can be executed on both edge and end devices. But the face detection models with different backbone networks have certain differences in efficiency and accuracy as well as the response time. For each backbone network, we analyze the accuracy $Q_k = \{q_e, q_m, q_h\}$, $k \in \mathcal{K}$ at easy, medium, and hard task levels, respectively, which is officially classified in data set "WIDER FACE" [46]. Assume that tasks $\mathcal{S} = \{s_1, s_2, \ldots, s_T\}$ are generated in Bernoulli distribution within a time slot $\mathcal{T} = \{1, 2, \ldots, T\}$, and the task generation rate R_t of all end devices determines the specific offloaded model on each edge device within a time slot, where $R_t = (|s_t|/N)$.

If R_t is at the easy task level, it means few end devices generate tasks within a time slot so that the overall tasks are easy to process, and the models with high accuracy may be selected to offload. Conversely, R_t at the hard task level means tasks are hard to process, and the models with the appropriate accuracy may be selected. Even if it reduces processing efficiency, the necessary accuracy must be guaranteed. The condition of model selection based on task generation rate and accuracy is shown in

$$\begin{cases} q_e \geq q_1, & \text{if } R_t < R_{\text{low}} \\ q_m \geq q_2, & \text{if } R_{\text{low}} \leq R_t < R_{\text{high}} \\ q_h \geq q_3, & \text{if } R_{\text{high}} \leq R_t. \end{cases} \tag{1}$$

As R_t increases, the accuracy of the task on each model decreases. For the accuracy of selected models for offloading, the accuracy threshold satisfies $q_1 \geq q_2 \geq q_3 \geq q_{\text{base}}$, that is, models that meet the accuracy constraints can be selected for offloading. R_{low} and R_{high} are denoted to classify the complexity of the tasks, which have been determined

in data set "WIDER FACE." $R_t < R_{\text{low}} = 0.8$ for easy tasks, $R_{\text{low}} \le R_t < R_{\text{high}} = 0.95$ for medium tasks, and $R_t \ge R_{high}$ for hard tasks.

We examine actual face detection inference tasks on end devices and edge servers, record and calculate the time and energy consumption of each task processed on different devices. D_n^l is the local response time on end device n with a single model, while D_m^k is the time of task executed on the edge server m using model k, in which $n \in \mathcal{N}$, $m \in \mathcal{M}$.

Then the energy consumption of local execution is as follows:

$$E_n^l = D_n^l p_l \tag{2}$$

where p_l is the power consumption on an end device. The energy consumption of edge execution is calculated similarly

$$E_m^k = D_m^k p_e \tag{3}$$

where p_e is the power consumption on an edge server.

If the task needs to be executed on an edge server, the time of data transfer should also be considered. We denote D_m^t as the transmission time consumed by offloading a frame image to server m and then returning the result to the end device. The energy consumption of data transmission E_m^t is shown in

$$E_m^t = D_m^t p_t \tag{4}$$

where p_t is the transmission power, $0 < p_t \le p_t^{\max}$, and $p_t^{\max}$ is the maximum transmission power between the server and the end device. Equation (5) shows the total time and energy consumption for executing a task on the edge server

$$\begin{aligned} D_n^e &= D_m^k + D_m^t \\ E_n^e &= E_m^k + E_m^t. \end{aligned} \tag{5}$$

B. Multiobjective Optimization Within Time Slot

Each computing task can be processed on an end device or offloaded to an edge server with better computing performance and higher precision models. We denote $A_n = \{a_{n_0}, a_{n_1}, \ldots, a_{n_m}, \ldots, a_{n_M}\}$ and $B_n^m = \{b_n^{m_1}, b_n^{m_2}, \ldots, b_n^{m_k}, \ldots, b_n^{m_K}\}$ as the offloading indicators. $a_{n_0} = 1$ means the task generated on edge device n is processed locally, and $a_{n_m} = 1$ means the task is offloaded to the server m. $b_n^{m_k} = 1$ shows that task generated on edge device n is offloaded to the model k of the server m.

For each end device n, only one parameter is set to 1 in A_n, and only one parameter is set to 1 in B_n^m if the task is offloaded to the server m, then the relationships in (6) and (7) must be satisfied

$$a_{n_0} + \sum_{m=1}^{M} a_{n_m} = 1 \quad \forall n \in \mathcal{N} \tag{6}$$

$$\sum_{k=1}^{K} b_n^{m_k} = \begin{cases} 0, & \text{if } a_{n_0} = 1 \\ 1, & \text{if } a_{n_m} = 1 \end{cases} \quad \forall n \in \mathcal{N}\ \forall m \in \mathcal{M}. \tag{7}$$

Fig. 3 is a binary offloading matrix, showing the offloading situation of tasks within a time interval. The large matrix has N rows, representing the offloading situation of each edge device in $\mathcal{N}$. The first column expresses whether the tasks are executed locally, and columns 2 through $M+1$ show whether the tasks are offloaded to the edge servers. For example, column $M+1$ in row N values 1 means end device N offloads its task to server M. Note that each row has only one column value of 1, and all others are 0. The small matrix represents the model selection of each server. As shown in Fig. 3, the second column in the model matrix values 1, which represents inference with the second model is selected. Similarly, it has only one column with a value of 1.

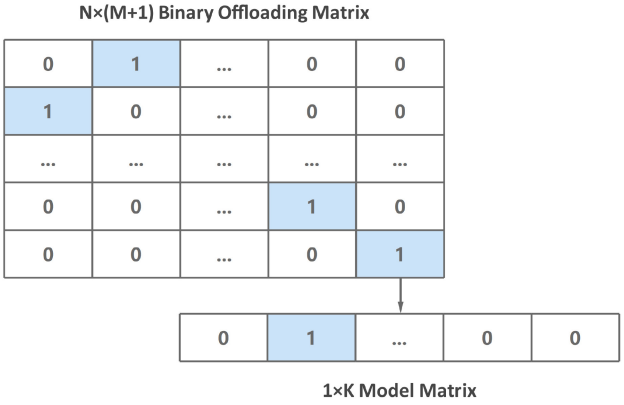

Fig. 3. Binary offloading matrix.

After constructing the system model and the offloading decision matrix, the time and energy consumption of the task generated by device n within a time slot are shown in

$$T_n = a_{n_0} D_n^l + \sum_{m=1}^{M} a_{n_m} \sum_{k=1}^{K} b_n^{m_k} D_n^e \tag{8}$$

$$E_n = a_{n_0} E_n^l + \sum_{m=1}^{M} a_{n_m} \sum_{k=1}^{K} b_n^{m_k} E_n^e \tag{9}$$

which are adjusted to the same order of magnitude when calculating.

The offloading indicators are used to limit the selection of delay and energy consumption. Since the offloading decision in each time slot satisfies (6) and (7), the results calculated by T_n and E_n are the time and energy consumption under the current decision. The objective function of the multimodel edge computing framework for task offloading within a time slot is set in (10). The time and energy consumption are considered comprehensively, and the goal is to minimize the total cost of all the tasks, where x is the offloading decision matrix

$$\begin{aligned} \mathbf{P1} \quad & \arg\min_{x=[A_n, B_n^m]} f(x) = \sum_{n=1}^{N} (\alpha T_n + \beta E_n) \\ \text{s.t.} \quad & (1), (6), (7), (8), (9), && (10) \\ & A_n, B_n^m \in \{0, 1\}\ \forall n \in \mathcal{N} && (10a) \\ & T_n \le t\ \forall n \in \mathcal{N} && (10b) \\ & E_n \le E_n^{\max}\ \forall n \in \mathcal{N} && (10c) \\ & 0 < p_t \le p_t^{\max} && (10d) \\ & \alpha + \beta = 1. && (10e) \end{aligned}$$

Constraint (1) keeps the overall accuracy to a great level in different task complexity. Constraints (6), (7), and (10a) denote

the binary selection indicators and limit the values they can take. Constraints (8) and (9), respectively, state the time and energy consumed by the task of end device n in an offloading period. Constraint (10b) denotes that each task should be completed within a time slot. Constraint (10c) states that the energy consumption of each task should be less than the maximal energy consumption. Constraint (10d) is the limit of transmission power. Constraint (10e) is to assign the weight of time and energy consumption to the total cost.

Bayesian optimization is often used to tune hyperparameters in machine learning models, which is generally regarded as a black box optimization problem. During the tuning process, we only observe its outputs based on the given inputs, and there are no restrictions of concavity and convexity for the objective problem. Compared with Grid Search and Randomized Search, the parameter space and computation of Bayesian optimization are greatly reduced, and Bayesian optimization does not require a large number of initial samples in contrast to metaheuristic algorithms. Due to these advantages, we choose Bayesian optimization to solve our offloading decision problem, which can also be viewed as a black box function. The tree-structured Parzen estimator (TPE) is a classic sequential model-based Bayesian optimization algorithm, which constructs probability models to improve the system performance based on historical measurements.

In this article, we design an MTPE algorithm that can optimize initial parameters and dynamically adjust quantile along with iterations. Compared with traditional Bayesian Optimization based on the Gaussian process, MTPE based on the Gaussian mixture model can achieve better results with higher efficiency and even solve discrete parameters' optimization.

With the observed experience set $\{(x_1, f(x_1)), x_2, f(x_2)), \ldots, (x_q, f(x_q))\}$, MTPE defines $\mathrm{p}(x|f(x))$ using two probability density functions in (11) to divide the parameter space into good part and bad part. Note that x is the solution of a group of offloading indicators

$$p(x|f(x)) = \begin{cases} l(x), & \text{if } f(x) < f(x)^* \\ g(x), & \text{if } f(x) \geq f(x)^* \end{cases} \tag{11}$$

where $f(x)^*$ is chosen to be some quantile γ of the observed target function values, so that quantile γ satisfies $p(f(x) < f(x)^*) = \gamma$. The probabilistic surrogate models $l(x)$ and $g(x)$ are tree-structured hierarchical processes constructed by adaptive Parzen estimators and can be applied to discrete-valued variables. Observations $\{x_i\}$ is used to form density $l(x)$ such that corresponding objective function $f(x_i)$ is in the good part. Conversely, $g(x)$ is the density function of the bad part.

Here, just for the sake of the derivation, let $y = f(x)$. After the above estimation by the probability density function, we adopt the expected improvement (EI) function as the acquisition function to collect the next observation point. The acquisition process is shown in

$$\arg\min_x EI_{y^*}(x) = \arg\min_x \mathbb{E}\big[y - y^*\big]. \tag{12}$$

Since we minimize the objective function in (10) and y is less than y^* for the next point, (12) can be rewritten in

$$\arg\max_x EI_{y^*}(x) = \arg\max_x \mathbb{E}\big[\max\big(y^* - y, 0\big)\big]. \tag{13}$$

According to the Bayes formula, posterior probability $p(y|x)$ can be calculated by prior probability $p(y)$ and conditional probability $p(x|y)$, i.e., $p(y|x) = [p(x|y)p(y)/p(x)]$. Therefore, the above problem is derived as follows:

$$\begin{aligned} EI_{y^*}(x) &= \int_{-\infty}^{\infty} \max\big(y^* - y, 0\big) p(y|x) dy \\ &= \int_{-\infty}^{y^*} \big(y^* - y\big) \frac{p(x|y)p(y)}{p(x)} dy \\ &= \frac{\gamma y^* - \int_{-\infty}^{y^*} y p(y) dy}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}} \\ &\propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1}. \end{aligned} \tag{14}$$

According to (13) and (14), maximizing EI means to select the next observation point with the minimum value of $(g(x)/l(x))$. So, we would like select point x with lower probability under $g(x)$ and higher probability under $l(x)$.

However, the TPE algorithm starts from a randomly generated solution, which is an NP-hard Optimization problem with a large parameter space. Another issue is the fixed quantile also makes the result fall into a local optimization early. Our proposed algorithm in Algorithm 1 has the following improvements.

1) It is known that the inference speeds of edge servers are much faster than that of end devices. Therefore, when initializing the observation points, we set the offloading indicators to allow a certain proportion (50-50 policy based on our experiments) of tasks to perform on the edge servers. It can dramatically reduce the number of iterations, compared with selecting the initial parameters randomly. The overall process is described in lines 1–5 of Algorithm 1.
2) To avoid falling into local optimization, the initial value of quantile γ is 0.4 by default, and changes dynamically with the output of each iteration. In each iteration, we choose bottom-k ($k = \lceil \gamma \cdot I_c \rceil$) from Z to generate set Z_l, and the rest are in set Z_g. For the next iteration, if the objective function value of the selected point is less than that of the previous iteration, γ is set to reduce by 10% of the decline degree f_c and stop changing until γ is less than 0.25. The overall process is described in lines 6–22 of Algorithm 1.

C. Energy Optimization Between Time Slots

Through the optimization in the above section, we can figure out the lowest cost for each time slot. To keep the energy stable during time $\mathcal{T}$, we employ the Lyapunov model to construct a dynamic energy queue and obtain the energy that needs to be newly supplied between two adjacent time slots.

Algorithm 1 MTPE Algorithm

Input:
I: number of iterations
I_c: number of candidates per iteration
γ: quantile
s_t: task set in time slot t
Output: x minimizes $f(x)$ in Z
Initialization: $Z = \emptyset$

for $i = 1$ to I_c **do**
  set x to randomly offload 50% of s_t to the servers
  calculate $f(x)$ according to Eq. 10
  $Z \leftarrow Z \cup \{(x, f(x))\}$
end for
for $i = 1$ to I **do**
  $k = \lceil \gamma \cdot I_c \rceil$
  $Z_l \leftarrow$ select $(x, f(x))$ with bottom-k values in Z
  $Z_g \leftarrow Z \setminus Z_l$
  construct $l(x)$ with Z_l and $g(x)$ with Z_g
  $C \leftarrow \{(x_c, f(x_c))|x_c \sim l(x), c = 1, ..., I_c\}$
  $x \leftarrow \arg\max_{x \in C} \frac{l(x)}{g(x)}$
  $Z \leftarrow Z \cup \{(x, f(x))\}$
  if $i > 1$ and $f(x) < f(x_{pre})$ **then**
    if $\gamma > 0.25$ **then**
      $f_c = \frac{f(x_{pre}) - f(x)}{f(x_{pre})}$
      $\gamma = (1 - 0.1 f_c)\gamma$
    end if
  end if
  $x_{pre} = x$
  $I_c = I_c + 1$
end for
return x minimizes $f(x)$ in Z

Assume that the remaining energy B_t varies at the end of time slot t in the following:

$$B_t = B_{t-1} - E_t + e_t \tag{15}$$

where E_t is the total energy consumption in time slot t, and e_t is the energy that is replenished at the end of each time slot.

Then, we introduce the Lyapunov optimization, defined in

$$L(B_t) = \frac{1}{2} B_t^2. \tag{16}$$

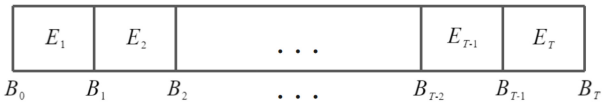

Fig. 4. Energy queue.

The Lyapunov drift can be described in

$$\Delta(B_t) = \mathbb{E}\big[L(B_t) - L(B_{t-1})|B_t\big]. \tag{17}$$

Then, we infer an upper bound of $\Delta(B_t)$, which can be found in Lemma 1 [34].

Lemma 1: For the end of each time slot, the upper bound of the Lyapunov drift function $\Delta(B_t)$ is

$$\Delta(B_t) \le \mathbb{E}\big[B_{t-1}(e_t - E_t)|B_t\big] + C \tag{18}$$

where C is a constant, and is denoted as

$$C = \frac{\big(e_t^{\max}\big)^2 + \big(E_t^{\max}\big)^2}{2} \tag{19}$$

limit $0 < e_t \le e_t^{\max}$ and $0 < E_t \le E_t^{\max}$.

To keep the energy queue dynamically stable, we minimize the upper bound in time $\mathcal{T}$, which can be transformed into

$$\mathbf{P2} \quad \arg\min_{e_t} \frac{1}{T} \sum_{t=1}^{T} B_{t-1}(e_t - E_t). \tag{20}$$

Fig. 4 shows the dynamic change of the energy queue within time $\mathcal{T}$, each square represents a time slot, and B_0 is the initial energy.

The detailed derivation of calculating e_t is shown in (21), at the bottom of the page, where $C_1 = \sum_{i=1}^{T-1} \sum_{j=i+1}^{T} E_i E_j$, $C_2 = E_1 + E_2 + \cdots + E_T$. From the deduced result, we find that this is a unary quadratic function with e_t as the independent variable. Therefore, value e_t that satisfies (20) can be obtained by computing the minimum point of the unary quadratic function in

$$\begin{aligned} e_t &= -\frac{TB_0 - (T-1)C_2}{T(T-1)} \\ &= \frac{C_2}{T} - \frac{B_0}{T-1}. \end{aligned} \tag{22}$$

Due to $e_t > 0$ and the sufficiency of initial energy in time $\mathcal{T}$, the range of B_0 is $E_1 < B_0 < (T - 1/T)C_2$. Then, the initial energy and the harvesting energy between time slots can be solved by maintaining energy stability in the whole task processing cycle.

$$\begin{aligned} \sum_{t=1}^{T} B_t(e_t - E_t) &= B_0(e_t - E_1) + B_1(e_t - E_2) + \cdots + B_{T-1}(e_t - E_T) \\ &= B_0(e_t - E_1) + (B_0 - E_1 + e_t)(e_t - E_2) + \cdots + \big[B_0 - E_1 - \cdots - E_{T-1} + (T-1)e_t\big](e_t - E_T) \\ &= Te_tB_0 - e_t\big[(T-1)E_1 + (T-2)E_2 + \cdots + E_{T-1}\big] + (1 + 2 + \cdots + T - 1)e_t^2 \\ &\quad - B_0(E_1 + E_2 + \cdots + E_T) - e_t[E_2 + 2E_3 + \cdots + (T-1)E_T] + C_1 \\ &= \frac{T(T-1)}{2} e_t^2 - (T-1)e_t[E_1 + E_2 + \cdots + E_T] + Te_tB_0 - B_0(E_1 + E_2 + \cdots + E_T) + C_1 \\ &= \frac{T(T-1)}{2} e_t^2 - [(T-1)C_2 - TB_0]e_t + C_1 - C_2B_0 \end{aligned} \tag{21}$$

TABLE II
NVIDIA JETSON BOARD SPECIFICATION

	Jetson Nano	Jetson TX2	Jetson Xavier NX
CPU	Quad (4)-Core ARM Cortex-A57 MPCore	Hexa (6)-core processor with 2x NVIDIA Denver2 64-Bit CPU and 4x ARM Cortex-A57 cores	6-core NVIDIA Carmel ARMv8.2 64-bit CPU 6 MB L2+4 MB L3
GPU	128-core Maxwell GPU	256-core Pascal GPU	Volta GPU with 384 NVIDIA CUDA cores and 48 Tensor cores
AI Performance	472 GFLOPs (FP16)	1.33 TFLOPs (FP16)	6 TFLOPs (FP16)&21 TOPs (INT8)
Memory	4 GB 64-bit LPDDR4 @25.6GB/s	8 GB 128-bit LPDDR4 @59.7GB/s	8 GB 128-bit LPDDR4 @51.2GB/s
Power	5-10W	7.5-15W	10-15W
Jetpack version	4.4	4.4	4.4
Ubuntu Linux	18.04.5	18.04.5	18.04.5
CUDA version	10.2	10.2	10.2
CuDNN version	8.0	8.0	8.0
Archiconda version	3-0.2.3	3-0.2.3	3-0.2.3

Algorithm 2 Dynamic Energy Adjustment Algorithm

Input:

$\mathcal{M} = \{1, 2, ..., m, ..., M\}$: set of edge servers
$\mathcal{K} = \{1, 2, ..., k, ..., K\}$: set of models
$\mathcal{N} = \{1, 2, ..., n, ..., N\}$: set of end devices
$\mathcal{T} = \{1, 2, ..., t, ..., T\}$: set of time slots
B_0: initial energy
e_t: harvesting energy

1: set $\mathcal{S} = \{s_1, s_2, ..., s_T\}$ by Bernoulli distribution
2: set $\mathcal{D} = \emptyset$, $\mathcal{E} = \emptyset$
3: **for** $t = 1$ to T **do**
4: **for** each task in s_t **do**
5: import D_n^l, D_m^k, D_m^t from the actual test
6: $D_n^e = D_m^k + D_m^t$
7: $\mathcal{D} \leftarrow \mathcal{D} \cup \{D_n^l, D_n^e\}$
8: calculate E_n^l, E_m^k, E_m^t according to Eq. 2-4
9: $E_n^e = E_m^k + E_m^t$
10: $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_n^l, E_n^e\}$
11: **end for**
12: // call the MTPE algorithm of Alg. 1
13: x_t = MTPE (s_t, $\mathcal{D}$, $\mathcal{E}$)
14: put x_t into (10b) to get $E_t = \sum_{n=1}^{N} E_n$
15: $B_t = B_{t-1} - E_t + e_t$
16: **end for**

The dynamic change of energy during the whole process is shown in Algorithm 2. In each time slot t, the time consumption D_n^l, D_n^e and energy consumption E_n^l, E_n^e of executing generated tasks on the end devices and edge servers are calculated in lines 4–11. Then, the MTPE algorithm from Algorithm 1 constructs the objective function and returns the optimal offloading decision. Therefore, we can obtain the total energy consumption E_t of time slot t. At the end of each time slot t, Algorithm 2 calculates energy e_t by (22) and updates energy queue B_t. Algorithms 1 and 2 solve the problem of minimizing the total cost of offloading within a time slot and the problem of keeping the energy queue stable between the time slots in the whole cycle $\mathcal{T}$, respectively, which correspond to **P1** and **P2**.

TABLE III
EDGE SERVER SPECIFICATION

	GeForce RTX 3080	GeForce RTX 2080
CUDA Cores	8704	2944
GPU Memory	10GB GDDR6X	8GB GDDR6
Enforced Power Limit	320W	215W
CentOS Linux	8.5.2111	8.5.2111
Driver version	510.47	470.86
CUDA version	11.6	11.4
CuDNN version	8.2.1	8.2.1
Anaconda version	4.8.2	4.8.2

IV. EXPERIMENTAL RESULTS

In this section, we first conduct experiments to compare the performance of end devices and edge servers. Then we test the inference capability of various face detection models including response time and accuracy on real equipment. Finally, we evaluate our proposed MTPE algorithm compared with the TPE algorithm, two metaheuristic algorithms and Randomized Search.

A. AI Application on NVIDIA Jetson and RTX GPU

In all experiments, we choose Jetson Xavier NX, Jetson TX2 and Jetson Nano from the NVIDIA Jetson series as embedded edge devices, and use GeForce RTX 3080 and GeForce RTX 2080 as edge servers. Different features and configurations of these NVIDIA Jetson end devices are shown in Table II. Theoretically, the AI performance of Jetson Xavier NX is roughly 12.71 times that of Jetson TX2 and 4.51 times that of Jetson Nano. We install the same system versions on all three types of boards, ensuring that our experimental results only rely on hardware conditions. Since the driver version on each edge server is limited by the GPU model, which is shown in Table III, GeForce RTX 3080 has higher requirements for the GPU driver version.

First, we train the single-shot face detection model Retinaface [47] of various backbone networks on the GeForce RTX 3080 server using the face detection benchmark data

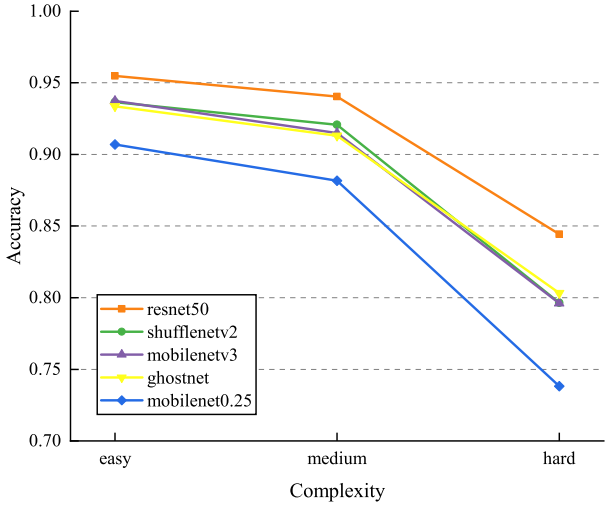

Fig. 5. Accuracy of each model on different difficulty tasks.

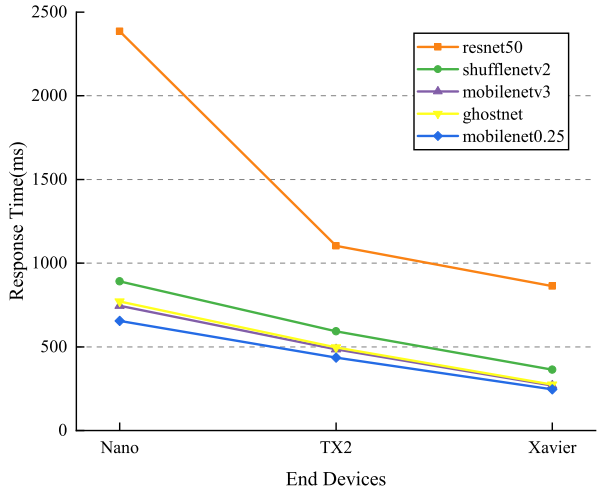

Fig. 6. Response time of each end device.

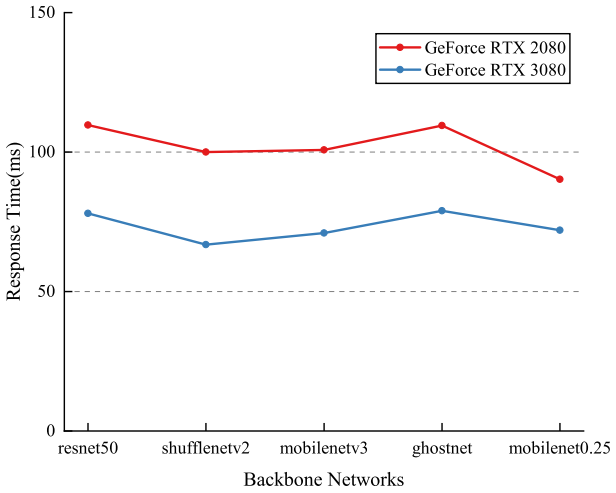

Fig. 7. Response time of each model on edge servers.

set WIDER FACE [46]. The backbone networks including Mobilenet0.25, Mobilenetv3, Ghostnet, Shufflenetv2, and Resnet50 are fully employed in our experiments. By combining extra-supervised and self-supervised multitask learning, RetinaFace can perform pixel-level localization of faces at various scales. After model training, we estimate the performance of all kinds of backbone networks on three complex task levels, which is classified by data set WIDER FACE according to the number of images. The exact accuracy results of each model on the easy, medium and hard WIDER FACE testing sets are shown in Fig. 5. The accuracy of each model on the easy testing set is above 90%, especially over 95% of Resnet50. On the medium testing set, the accuracy remains stable without significant reduction, but only Mobilenet0.25 decreases below 90%. However, the accuracy of all models on the hard testing set are greatly reduced, yet only Resnet50 shows an acceptable result (84.43%), compared to 80% or even below 75% of the others.

Second, we also evaluate the response time on end devices and edge servers, which is the detected time of each image. The inference performance of different models on each testing equipment is shown in Figs. 6 and 7.

From Fig. 6, the response time of each model decreases gradually along with the increase of capability of the end devices, the model with backbone Mobilenet0.25 is the fastest among all models. By correlated with Fig. 5, it shows that the model with relatively higher accuracy takes a longer time for inference. For Resnet50, as a complex backbone network, it takes far more time to execute tasks on end devices than other lightweight networks. Especially, the reference time of Resnet50 on the Jetson Nano is more than 2 s, so it is unsuitable to deploy on end devices.

From Fig. 7, the response time of each model is relatively low and similar. Compared with end devices, Resnet50 runs more efficiently on edge servers, which presents the dual advantages of high accuracy and low latency. Therefore, we consider deploying one lightweight model on each end device and multiple models including Resnet50 on edge servers.

B. Results and Analysis of MTPE Algorithm

In this part of the experiments, we verify the feasibility of the multimodel edge computing framework and estimate the performance of the MTPE algorithm compared with others. $p_l = 10$ W, $p_e = 100$ W, and $p_t = 1$ W, which are based on the actual test data and rated power of devices. The time slot t is $1s$, and the energy consumption limited for a task within a time slot $E_n^{\max}$ is $10J$. According to the accuracy measured by different complex tasks, q_1 and q_2 are set to 0.9, and q_3 is 0.8. Therefore, models with accuracy more than 0.9 are accepted for easy and medium tasks, whereas models with accuracy better than 0.8 are selected for hard tasks. Due to the high requirements for real-time performance in actual applications, the weight α of T_n should be set to more than 0.5. The larger α is easily conducive to reduce the total response time. However, the lowest response time is often accompanied by the highest energy consumption. Initially, α is adjusted to 0.8 to balance response time and energy consumption simultaneously.

To consider multiple deployment scenarios in our experiments, end devices come into a group of 12, and each group contains three types of devices mentioned in Table II and four

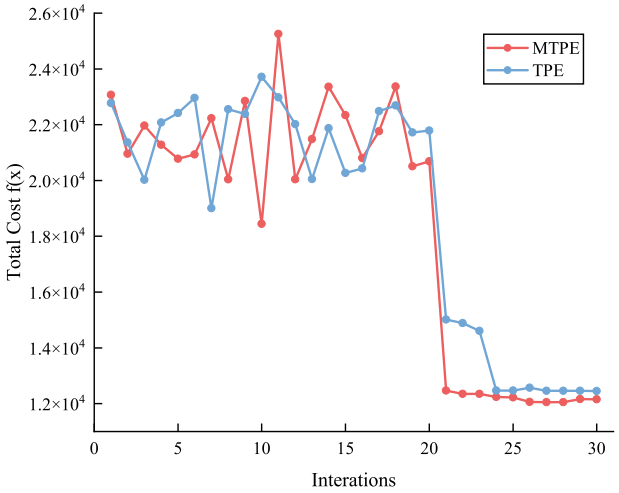

Fig. 8. Convergence of MTPE and TPE ($\alpha = 0.8, K = 5, N = 60$).

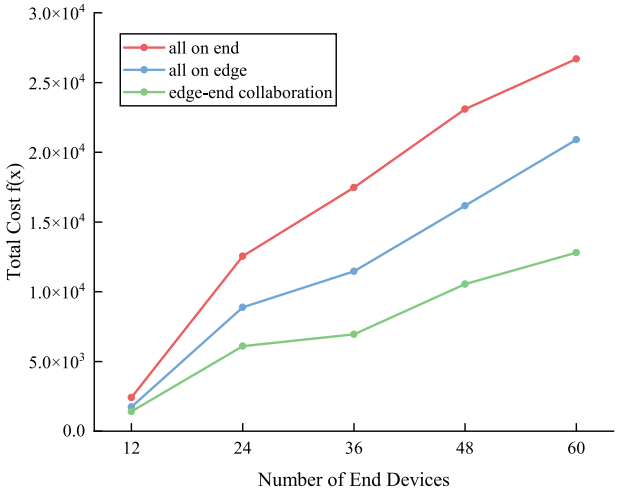

Fig. 9. Total cost for different offloading environment ($\alpha = 0.8, K = 5$).

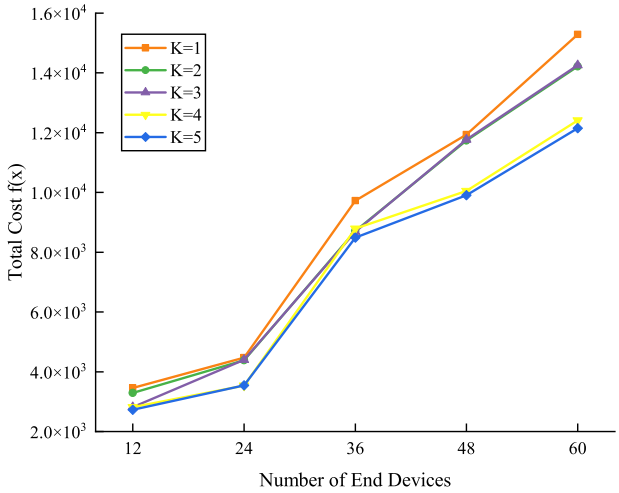

Fig. 10. Total cost for different number of models ($\alpha = 0.8$).

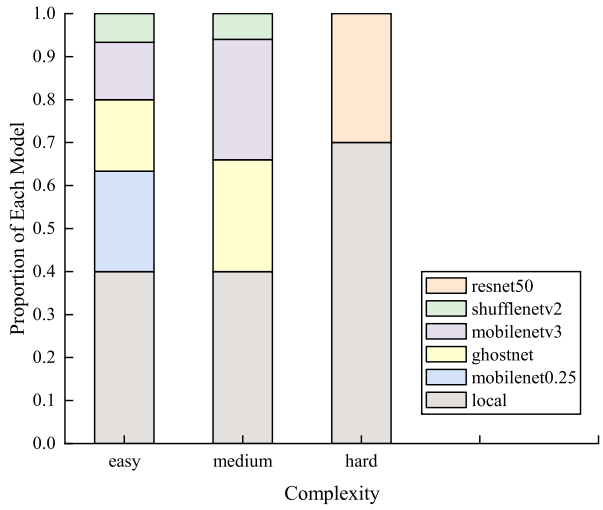

Fig. 11. Model selection ($\alpha = 0.8, K = 5, N = 60$).

types of lightweight models in each type of end device. The number of edge servers M is 4, which is enough for our edge computing framework. In Fig. 8, we set $N = 60$, five groups of end devices, and the number of inference models $K = 5$ to obtain the convergence of the iterative process for the MTPE algorithm and TPE algorithm. We find that the total costs of both algorithms start to drop significantly after 20 iterations, but MTPE converge faster than TPE to search the optimal value, and the optimal value is also smaller. Since the results stabilize after 20 iterations, we set the number of iterations of the MTPE algorithm to 30 in our training process, by default.

When the number of end devices N is varied from 12 to 60 with the fixed number of models $K = 5$, the total costs of processing tasks generated by different numbers of devices under three offloading environments are calculated in Fig. 9. We observe that with the increase of end devices and the number of tasks, the total cost of task consumption gradually increases. However, it is obvious that the cost of edge-end collaboration is 36.10% and 54.03% lower than offloading all tasks to the servers or end devices, respectively. Therefore, edge computing can dramatically reduce computing costs.

Next, we compare the total cost of deploying different numbers of models on the servers. N still ranges from 12 to 60, and the number of models K varies from 1 to 5 as well as the complexity of the model added from difficulty to easiness. As shown in Fig. 10, as more models are deployed, the total cost tends to decrease. Multimodel framework with $K = 5$ is reduced by an average of 17.94% compared to a single-model framework, reaching up to 21.86% when $N = 60$. Hence, auxiliary deployment of lightweight models on edge servers can further ease the computing burden, and the effect is more obvious with the increase of tasks.

The model selection under different task complexities with the number of end devices $N = 60$ is shown in Fig. 11. In general, the advantage of multimodel is that lightweight models can meet basic requirements without selecting complex models with long response time. All five models can be selected at the easy task level. However, with the increase of task complexity, the number of selected models decreases to meet the requirement of accuracy. At the hard task level, only Resnet50 can be selected for edge devices.

To further observe the influence of the two factors of time and energy consumption on the experimental results, we vary

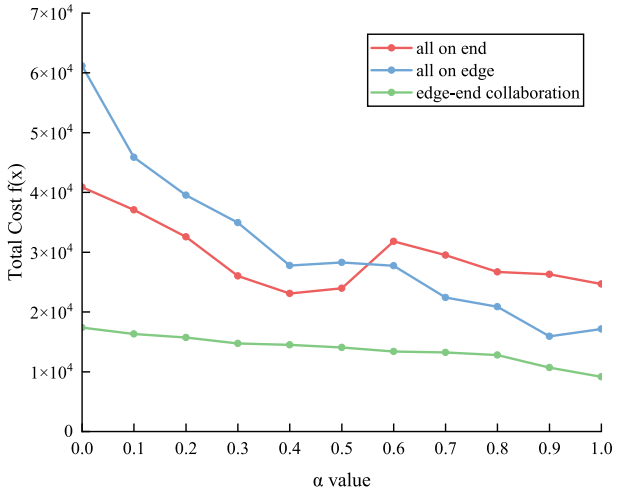

Fig. 12. Total cost for different α Value ($K = 5, N = 60$).

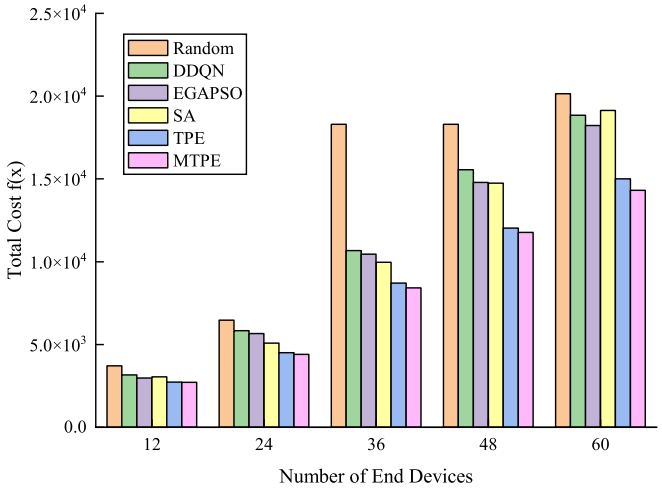

Fig. 13. Contrast experiment ($\alpha = 0.8, K = 5$).

the weight α of the objective function $f(x)$ from 0 to 1 with 0.1 in Fig. 12. We find that the total cost of edge-end collaboration keeps decreasing steadily as the value of α increases. But with $\alpha = 0.5$ as the boundary, the total cost of executing all tasks on end devices locally starts to be higher than that of offloading all tasks to servers which maintains a sharp downward trend. It can be concluded that the edge servers are beneficial to reduce the total response time, while the end devices contribute to reducing the total energy consumption. For this reason, the total cost of edge-end collaboration can be effectively reduced and maintained at a stable level.

Finally, we compare our algorithm with the latest DDQN algorithm [29], metaheuristic hybrid algorithm EGAPSO [38], SA and the original TPE algorithm. The result in contrast to the baseline algorithm, Randomized Search, is shown in Fig. 13 with $\alpha = 0.8$, $K = 5$. Our MTPE algorithm can obtain the optimal solution by using fewer initialization decision parameters. It can be concluded that our algorithm MTPE is superior to the DDQN algorithm and the metaheuristic algorithm for solving the offloading problem in multimodel edge computing framework, which reduces the total cost by 37.79%, 23.01%, 3.14%, 20.09%, and 19.90% on average compared with Randomized Search, DDQN, TPE, EGAPSO, and SA, respectively.

V. Conclusion and Future Work

In this article, we proposed a multimodel edge computing offloading framework, using embedded edge devices NVIDIA Jetson and GeForce RTX GPU servers to simulate the edge computing environment of real AI applications. We comprehensively considered the accuracy, time, and energy consumption of inference tasks. To work out the lowest total cost of the system within a time slot, we put forward a Bayesian optimization algorithm using MTPE, and theoretically indicated that our algorithm can find an optimal solution under less iterations. To ensure the stability of the energy queue between time slots, we also employed the Lyapunov drift function to solve the harvesting energy between the time slots. Through comparative experiments, we verified that the multimodel edge computing offloading framework achieved satisfactory results in communication cost and computation cost, and ensured the high accuracy of inference tasks. Compared with the original TPE algorithm, the state-of-the-art DDQN algorithm and the metaheuristic algorithms, EGAPSO and SA, our algorithm was superior for solving offloading problems at the lowest cost. In the future, we will concentrate on load balancing, including the scheduling and migration of containers on edge devices to fully utilize the computing, storage, and network resources, which will also improve the reliability of the edge computing environment.

References

[1] "Statista: Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030." 2020. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] M. A. Rahman and M. S. Hossain, "An Internet-of-Medical-Things-enabled edge computing framework for tackling COVID-19," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 15847–15854, Nov. 2021.

[3] S. Wan, S. Ding, and C. Chen, "Edge computing enabled video segmentation for real-time traffic monitoring in Internet of Vehicles," *Pattern Recognit.*, vol. 121, Jan. 2022, Art. no. 108146.

[4] L. Fan and L. Zhang, "Multi-system fusion based on deep neural network and cloud edge computing and its application in intelligent manufacturing," *Neural Comput. Appl.*, vol. 34, no. 5, pp. 3411–3420, 2022.

[5] M. A. Guillén et al., "Performance evaluation of edge-computing platforms for the prediction of low temperatures in agriculture using deep learning," *J. Supercomput.*, vol. 77, no. 1, pp. 818–840, 2021.

[6] R. Rajavel, S. K. Ravichandran, K. Harimoorthy, P. Nagappan, and K. Ramasubramanian, "IoT-based smart healthcare video surveillance system using edge computing," *J. Ambient Intell. Humanized Comput.*, vol. 13, no. 6, pp. 3195–3207, 2022.

[7] Z. Ali, Z. H. Abbas, G. Abbas, A. Numani, and M. Bilal, "Smart computational offloading for mobile edge computing in next-generation Internet of Things networks," *Comput. Netw.*, vol. 198, Oct. 2021, Art. no. 108356.

[8] A. S. Mohammed, K. Venkatachalam, S. Hubálovskỳ, P. Trojovskỳ, and P. Prabu, "Smart edge computing for 5 g/6 g satellite IOT for reducing inter transmission delay," *Mobile Netw. Appl.*, vol. 27, pp. 1050–1059, Feb. 2022.

[9] A. I. Tahirkheli et al., "A survey on modern cloud computing security over smart city networks: Threats, vulnerabilities, consequences, countermeasures, and challenges," *Electronics*, vol. 10, no. 15, p. 1811, 2021.

[10] M. M. Sadeeq, N. M. Abdulkareem, S. R. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "IoT and cloud computing issues, challenges and opportunities: A review," *Qubahan Acad. J.*, vol. 1, no. 2, pp. 1–7, 2021.

[11] W. Shi, C. Jie, Z. Quan, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[12] S. Yi, L. Cheng, and Q. Li, "A survey of fog computing: Concepts, applications, and issues," in *Proc. Workshop Mobile Big Data (Mobidata)*, 2015, pp. 37–42.

[13] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *J. Netw. Comput. Appl.*, vol. 62, pp. 18–40, Feb. 2016.

[14] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surveys*, vol. 52, no. 6, pp. 1–36, 2019.

[15] W.-J. Chang, L.-B. Chen, C.-Y. Sie, and C.-H. Yang, "An artificial intelligence edge computing-based assistive system for visually impaired pedestrian safety at zebra crossings," *IEEE Trans. Consum. Electron.*, vol. 67, no. 1, pp. 3–11, Feb. 2021.

[16] X. Kong et al., "Real-time mask identification for COVID-19: An edge-computing-based deep learning framework," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 15929–15938, Nov. 2021.

[17] N. Balamuralidhar, S. Tilon, and F. Nex, "MultEYE: Monitoring system for real-time vehicle detection, tracking and speed estimation from UAV imagery on edge-computing platforms," *Remote Sens.*, vol. 13, no. 4, p. 573, 2021.

[18] A. Koubaa, A. Ammar, A. Kanhouch, and Y. Alhabashi, "Cloud versus edge deployment strategies of real-time face recognition inference," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 1, pp. 143–160, Jan.-Feb. 2022.

[19] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102781.

[20] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[21] X. Li, J. Xia, L. Cao, G. Zhang, and X. Feng, "Driver fatigue detection based on convolutional neural network and face alignment for edge computing device," *Proc. Inst. Mech. Eng. D, J. Automobile Eng.*, vol. 235, nos. 10–11, pp. 2699–2711, 2021.

[22] C. Liu et al., "A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 249–261, Mar./Apr. 2018.

[23] A. R. Neto, T. P. Silva, T. Batista, F. C. Delicato, P. F. Pires, and F. Lopes, "Leveraging edge intelligence for video analytics in smart city applications," *Information*, vol. 12, no. 1, p. 14, 2020.

[24] M. Lapegna, W. Balzano, N. Meyer, and D. Romano, "Clustering algorithms on low-power and high-performance devices for edge computing environments," *Sensors*, vol. 21, no. 16, p. 5395, 2021.

[25] T. V. Pham, N. N. Q. Tran, H. M. Pham, T. M. Nguyen, and T. Ta Minh, "Efficient low-latency dynamic licensing for deep neural network deployment on edge devices," in *Proc. 3rd Int. Conf. Comput. Intell. Intell. Syst.*, 2020, pp. 44–49.

[26] A. Khakimov et al., "Flexible architecture for deployment of edge computing applications," *Simulat. Model. Pract. Theory*, vol. 114, Jan. 2022, Art. no. 102402.

[27] Z. Shahbazi and Y.-C. Byun, "Improving transactional data system based on an edge computing–blockchain–machine learning integrated framework," *Processes*, vol. 9, no. 1, p. 92, 2021.

[28] S. Iftikhar et al., "AI-based fog and edge computing: A systematic review, taxonomy and future directions," *Internet Things*, vol. 21, Apr. 2023, Art. no. 100674.

[29] Y. Tu, H. Chen, L. Yan, and X. Zhou, "Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT," *Future Internet*, vol. 14, no. 2, p. 30, 2022.

[30] J. Wang and L. Wang, "Mobile edge computing task distribution and offloading algorithm based on deep reinforcement learning in Internet of Vehicles," *J. Ambient Intell. Humanized Comput.*, vol. 2021, pp. 1–11, Aug. 2021.

[31] X. Li, "A computing offloading resource allocation scheme using deep reinforcement learning in mobile edge computing systems," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–12, 2021.

[32] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.

[33] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.

[34] Z. Chang, L. Liu, X. Guo, and Q. Sheng, "Dynamic resource allocation and computation offloading for IoT fog computing system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3348–3357, May 2021.

[35] Z. Ning et al., "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1277–1292, Jun. 2021.

[36] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.

[37] J. Zhang et al., "Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3688–3699, Apr. 2019.

[38] B. Natesha and R. M. R. Guddeti, "Meta-heuristic based hybrid service placement strategies for two-level fog computing architecture," *J. Netw. Syst. Manage.*, vol. 30, no. 3, pp. 1–23, 2022.

[39] M. Xue, H. Wu, R. Li, M. Xu, and P. Jiao, "EosDNN: AN efficient offloading scheme for DNN inference acceleration in local-edge-cloud collaborative environments," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 1, pp. 248–264, Mar. 2022.

[40] 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.

[41] L. Kuang, T. Gong, S. OuYang, H. Gao, and S. Deng, "Offloading decision methods for multiple users with structured tasks in edge computing for smart cities," *Future Gener. Comput. Syst.*, vol. 105, pp. 717–729, Apr. 2020.

[42] G. Li, Y. Liu, J. Wu, D. Lin, and S. Zhao, "Methods of resource scheduling based on optimized fuzzy clustering in fog computing," *Sensors*, vol. 19, no. 9, p. 2122, 2019.

[43] C. Deng, X. Fang, and X. Wang, "UAV-enabled mobile-edge computing for AI applications: Joint model decision, resource allocation, and trajectory optimization," *IEEE Internet Things J.*, vol. 10, no. 7, pp. 5662–5675, Apr. 2023.

[44] Q. Tang, Z. Fei, B. Li, and Z. Han, "Computation offloading in LEO satellite networks with hybrid cloud and edge computing," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 9164–9176, Jun. 2021.

[45] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011, pp. 1–9.

[46] "WIDER FACE: A face detection benchmark." 2017. [Online]. Available: http://shuoyang1213.me/WIDERFACE/index.html

[47] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-shot multi-level face localisation in the wild," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 5203–5212.

Zidi Zhao is currently pursuing the M.S. degree in computer technology with the School of Cyber Security and Computer, Hebei University, Baoding, China.

Her current research interests include edge computing, machine learning, and Internet of Things.

Hong Zhang received the Ph.D. degree in computer science from the University of Central Florida, Orlando, FL, USA, in 2018.

He is an Assistant Professor with the School of Cyber Security and Computer, Hebei University, Baoding, China. His research interest is the design and analysis of distributed systems for edge computing and big data. In terms of design, he is currently working on developing the high-performance platforms of edge computing and big data. As for analysis, he focuses on improving performance, scalability, resilience, security, and load balancing of edge computing and distributed machine learning.

Liqiang Wang received the Ph.D. degree in computer science from Stony Brook University, Stony Brook, NY, USA, in 2006.

He is currently an Professor with the Department of Computer Science, University of Central Florida, Orlando, FL, USA. He is the Director of Big Data Lab. He was a Faculty Member with the Department of Computer Science, University of Wyoming, Laramie, WY, USA, from 2006 to 2015. He was a Visiting Research Scientist with IBM T.J. Watson Research Center, Ossining, NY, USA, from 2012 to 2013. His research focuses on big data computing and analytics techniques in the following aspects: 1) improving accuracy and security of big data analysis models; 2) optimizing performance and scalability of big data processing and parallel computing systems, including multithreading, HPC, Cloud, and GPU platforms; and 3) using program analysis and deep learning techniques to detect and prevent programming errors and execution anomaly in big data and/or parallel programs.

Prof. Wang received the NSF CAREER Award in 2011 and the Castagne Faculty Fellowship from 2013 to 2015.

Haijun Huang is currently pursuing the M.S. degree in computer technology with the School of Cyber Security and Computer, Hebei University, Baoding, China.

His current research interests include edge computing, information security, and blockchain.