BIRP: Batch-aware Inference Workload Redistribution and Parallel Scheme for Edge Collaboration

Hesheng Sun*
Nanjing University
Nanjing, China
hesheng.sun@smail.nju.edu.cn

Xinyi Chen†
Nanjing University
Nanjing, China
mf20090002@smail.nju.edu.cn

Zhuzhong Qian*
Nanjing University
Nanjing, China
qzz@nju.edu.cn

Zengji Li*
Ning Chen*
zoukiri180@gmail.com
ningc@smail.nju.edu.cn
Nanjing University
Nanjing, China

Tuo Cao*
Suwei Xu*
tuocao@smail.nju.edu.cn
suwei@smail.nju.edu.cn
Nanjing University
Nanjing, China

Yitong Zhou*
yitong.zhou@smail.nju.edu.cn
Nanjing University
Nanjing, China

ABSTRACT

The inference workload redistribution is a technique for evacuating inference requests from hot edges to idle edges in edge collaborative systems, thereby achieving inference workload balancing for inference on different edges. However, with the continuous development of edge accelerators, the resource utilization of edge accelerators in executing inference requests in series is often low, and when executing multiple inference requests in parallel, it faces uncertain execution delays, different response-time Service Level Objectives (SLOs), and the generality of inference workloads in heterogeneous edge collaborative systems. To address these issues, for the first time in the domain of inference workload redistribution, we propose a **B**atch-aware **I**nference workload **R**edistribution and **P**arallel execution scheme, called **BIRP**, to reduce the additional latency caused by waiting for a single inference task during serial execution, thereby improving the overall inference accuracy. **BIRP** uses the Multi-Armed Bandit (MAB) algorithm to adjust hyper-parameters of the Throughput Improvement Ratio (TIR) function online for improving the overall inference accuracy. For nonlinear terms in the problem, BIRP uses a piecewise linear approximation to convert it into a Quadratic Programming (QP) problem, ensuring the effectiveness of **BIRP** in theory. We prototype **BIRP** on an edge collaborative system composed of three heterogeneous edges. Based on real inference workload trace, we validate the superiority of our algorithm compared to the state-of-the-art model selection-based inference workload redistribution algorithm, with an overall inference loss reduction of at least 32.9% and the failure rate of SLO has been reduced to 19.8% of alternatives.

CCS CONCEPTS

• **Computer systems organization** → *Grid computing*.

KEYWORDS

DNN inference, edge collaboration, workload redistribution

ACM Reference Format:
Hesheng Sun, Xinyi Chen, Zhuzhong Qian, Zengji Li, Ning Chen, Tuo Cao, Suwei Xu, and Yitong Zhou. 2023. BIRP: Batch-aware Inference Workload Redistribution and Parallel Scheme for Edge Collaboration. In *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605615

1 INTRODUCTION

[1]Edge intelligence is a distributed computing paradigm that brings computation and data storage closer to the edge of the network, catering to the increasing demand for low-latency and high-accuracy Deep Neural Network (DNN) inference processing [8, 25, 35, 38]. The utilization of DNN inference for extracting information features from various data types such as images, sounds, and sequential data [20], as well as leveraging the proximity of edges [4], has led to extensive applications of edge intelligence in diverse domains, including the Industrial Internet of Things(IIoT) [11], augmented reality [5], smart cities [3], and so on.

However, one of the key challenges in edge intelligence is the issue of DNN inference request *workload imbalance* [21], where some edge devices experience heavy loads while others have lighter loads, leading to inefficient resource utilization and reduced system performance. To address this, researchers have proposed edge collaborative *workload redistribution* techniques, leveraging the proximity of edge devices and DNN Inference characteristics to improve system performance and resource utilization [1, 7, 10, 19, 30, 33].

These inference workload redistribution techniques are typically designed for uncertain edge-to-edge collaborative systems,

*State Key Laboratory for Novel Software Technology, the Department of Computer Science and Technology, Nanjing University, China
†School of Foreign Studies, Nanjing University, China

ICPP 2023, August 07–10, 2023, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0843-5/23/08…$15.00
https://doi.org/10.1145/3605573.3605615

[1]Corresponding author: Zhuzhong Qian (qzz@nju.edu.cn) and Hesheng Sun (hesheng.sun@smail.nju.edu.cn)

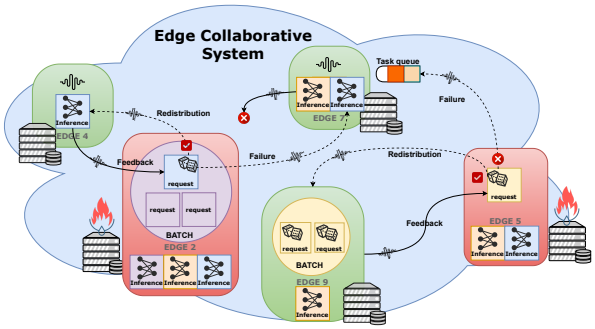


Figure 1: Edge Collaborative System

involving intelligent decision algorithms based on reinforcement learning [33], mixed-integer programming [1, 30], or game theoretic principles [10]. Then the workload redistribution results are determined based on the latency and computation requirements of different inference requests. There are also some studies [19, 29] that aim to balance the tradeoff between inference accuracy and inference latency by employing different versions of inference models. In addition, some research [7] focuses on predicting the distribution of workloads to provide a decision-making basis for the placement of edge servers and the real-time placement of inference models.

The algorithms for workload redistribution effectively balance the workload among edge devices and improve the system performance of edge devices. However, the execution of these inference workloads is commonly performed in a serialized manner [1, 7, 10, 19, 30, 33]. With the continuous upgrading and iteration of edges [26–28], a single DNN inference model executed at the edge often fails to fully leverage the Streaming Multiprocessors (SMs) of edge accelerators, resulting in limitations in the utilization of system resources and performance improvement[34]. As shown in Table. 1, we execute edge inference sequentially with different accelerators for various tasks such as image recognition, object detection, and text classification using different models. When implementing BERT [9], YOLOv4-t [2], and ResNet-18 [16] models, the utilization rates of CPU, GPU, and Neural Processing Unit (NPU) are limited to 29.2%, 72.4%, and 31.2% respectively.

The inherent constraints of resource utilization in the sequential execution of DNN inference models on different accelerators pose a significant barrier to enhancing system performance. However, these limitations also present valuable opportunities for improving system efficiency through the adoption of parallel execution approaches. But parallel execution of redistributed inference tasks faces numerous problems:

Firstly, what types of inference workloads should be parallelized? There have been approaches that propose kernel-level inference workload parallelization techniques [15], as well as methods for GPU virtualization [6]. However, these approaches often rely on specific hardware configurations. For edge collaborative systems, it is important to search for multi-workload parallel inference techniques that are applicable to heterogeneous edges and various frameworks.

Secondly, how much impact does parallel have on the original inference workloads? In inference workload redistribution techniques, it is often necessary to confirm the maximum workload that each device can bear before making decisions in order to maximize resource utilization and system performance [7, 19]. When using multi-workload parallel inference, it is challenging to obtain this information in advance due to issues such as unordered hardware execution during parallelization.

Thirdly, under what workload conditions should multi-workload parallel inference be employed? In order to achieve optimal system performance, an application often maps multiple inference models, where larger models generally result in higher inference accuracy but also higher overhead costs [19, 29]. It is necessary to strike a balance between the higher inference accuracy achieved by executing large models and the high resource utilization during the parallel execution of small models.

In order to address the new challenges brought by multi-workload parallel inference, we propose **adopting a batch-aware parallel approach to redistribute inference workloads,** and accordingly adjusting the inference models and corresponding batch sizes allocated to different edges within the edge collaborative system.

For instance, as Fig. 1 shows, five edges within the system and three types of inference applications: blue, yellow, and purple. Edges 7 and 9 are relatively idle, with a small number of tasks being executed on edge 7. However, edges 2 and 5 are currently under high load and require a redistribution of inference workload. For edge 2, the purple or blue inference request can be transferred, while for edge 5, the choice is between edge 7 or edge 9. The batch-aware approach suggests that requests of the same type of inference should be executed in the same batch (same color) on the same edge. Therefore, in this example, edge 5 will redistribute its own request to edge 9, and edge 2 will redistribute the blue task to edge 4.

Specifically, first, we merge inference requests belonging to the same intelligent application into a single request vector, allowing these requests to be executed in parallel using the weights of the same inference model within a time slot. We measure the performance improvement of the inference model after merging requests using the concept of the Throughput Improvement Ratio (TIR).

Then, based on the edge collaborative system limits, accuracy parameters of different versions of inference models, and the TIR function, we construct an inference workload redistribution optimization problem for multiple applications based on model version selection and batch size selection, with the goal of optimizing the overall inference accuracy. For solving the overall optimization problem, we use a learning-based approach to obtain the hyperparameters of the TIR function during execution. Then a piecewise linear fitting approach is used to adjust the optimization problem structure based on the hyperparameters chosen at each time step, transforming the original problem into a quadratic integer programming problem.

Finally, we implement our algorithm on an edge collaborative system with three heterogeneous edges. Compared to state-of-the-art inference workload redistribution method [19], the algorithm has achieved significant improvement in terms of overall inference accuracy and the response-time Service Level Objective (SLO).

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

- For the first time in the domain of inference workload redistribution, we propose a **B**atch-aware **I**nference workload

redistribution and **P**arallel execution scheme, called **BIRP**, to reduce the additional latency caused by waiting for a single inference task during serial execution, thereby improving the overall inference accuracy.
- We propose a dynamic batch decision-making method for uncertain TIR functions based on the Mulit-Armed Bandit (MAB) strategy, which can decouple the problem of inference workload redistribution in different time slots in an online manner. This method can guarantee optimality in long-term execution.
- We prototype BIRP on an edge collaborative system composed of three heterogeneous edges. Based on real inference workload trace, we validate the superiority of our algorithm compared to the state-of-the-art model selection-based inference workload redistribution algorithm OAEI [19], with an overall inference loss reduction of at least 32.9% and the failure rate of SLO has been reduced to 19.8% of OAEI [19].

2 BACKGROUND AND MOTIVATION

2.1 Background

DNN inference. Deep neural networks (DNN) is a type of machine learning model, characterized by multiple layers of interconnected neurons[20]. Nowadays, DNN inference has achieved significant success in applications such as image classification [16], object detection [2], text classification [9], and more.

Edge accelerators. Hardware accelerators with parallel acceleration capabilities are needed for processing DNN inference models, often with multiple Streaming Multiprocessors (SMs). Edge accelerators have also emerged gradually [13, 18, 23]. Compared to accelerators in the cloud, edge accelerators prioritize energy efficiency, miniaturization, and optimization for the DNN inference phase [26].

Workload redistribution. Workload redistribution refers to the edge collaborative system scheduler that can set a time slot smaller than the inference response-time SLO and adjust the workload of each edge in the edge collaborative system every time slot, thus meeting the SLO of intelligent applications and balancing the inference workloads of different edges to avoid the occurrence of stragglers [19, 33]. However, some small inference models are unable to utilize all SMs of edges, and the waiting time for inference requests during serial execution leads to low utilization of various resources at the edge.

2.2 Motivation

Edge resource low utilization. As in Table. 1, we implement different inference models on Huawei Atlas 200DK [18] and Jetson Nano [23] respectively. We use Ascend Tensor Compiler (ATC) [18] to deploy inferences on Huawei Atlas 200DK and TensorRT [22] on Jetson Nano, such that these inferences can be highly optimized and invoked by Python. Four commonly used inference models are considered here, as shown in Table 1: Yolov4-tiny [2], Yolov4-normal [2], Resnet-18 [16], and BERT [9].

We measure the average Frame Per Second (FPS) for image inferences, while for BERT FPS means the number of texts analyzed in one second. In addition, we also measure the utilization of various computational resources of the edge accelerators. As shown in Table. 1, Atlas uses Neural Processor Unit (NPU) as the accelerator of computation-intensive bottleneck, while Nano uses GPU as the accelerator. For small models such as Yolov4-t, GPU and NPU usage can not meet the 75% at both Atlas and Nano. Even large models like BERT do not exceed 50% CPU utilization on two different devices. When implementing BERT, YOLOv4-t, and ResNet-18 models, the utilization rates of CPU, GPU, and Neural Processing Unit (NPU) are limited to 29.2%, 72.4%, and 31.2% respectively.

Batch-aware multi-workload inference. The batch-aware approach combines multiple inference requests of the same type into one request vector and shares the weight parameters of the DNN inference model to achieve parallel execution of multiple inference workloads. General frameworks such as PyTorch and TensorFlow support dynamic batch inference, and some specialized device inference frameworks also support batch-aware inference, such as ATC and TensorRT. But how do different models behave with batch-aware methods in terms of throughput improvement on different edges?

To address this question, we first implement three image recognition models, GoogleNet, LeNet, and ResNet-18, on Jetson Nano with varying batch sizes, denoted as b, and measure the number of task batches completed within a fixed time, denoted as n. The throughput under different batch sizes can be calculated as $n * b$. We use the throughput at batch size 1 as the baseline throughput and describe the impact of batch size growth on throughput enhancement using the **Throughput Improvement Ratio (TIR)**, which is defined as

$$TIR_{model}(b) = \frac{throught(b)}{throught(1)}. \tag{1}$$

The experimental results are shown in Figure. 2. We conducted five experiments for each batch of each model. In the three figures of Figure. 2, blue dots represent the raw data of (batch-size, TIR). The green line represents the relationship between TIR and batch size, which can be represented by a constant function as the batch size increases. The red line represents the fitting curve of the TIR function when the batch size is less than the threshold value b_0.

This experiment's results demonstrate that: despite the differences in models and equipment, **the variation pattern of TIR remains the same**: a piecewise function that includes a power function and a constant function can well fit the functional relationship between the TIR and the batch size.

In other words, the relationship between the TIR_j^k of a model j on device k and the batch size b can be described as a piecewise function as follows:

$$TIR_{jk} = \begin{cases} b^{\eta_j^k}, & b \le \beta_j^k, \\ C_j^k, & b > \beta_j^k. \end{cases} \tag{2}$$

In the above expression, η_j^k, β_j^k, and C_j^k are three coefficients related to the edge k and the selected model j, representing the power function growth coefficient, the threshold for the transition of the relationship, and the maximum achievable TIR improvement, respectively.

Due to the numerous types and models of applications applied in edge collaborative systems and the heterogeneity of edge types participating in collaboration, it is difficult for us to predict the function representation of different models on heterogeneous edges

Table 1: Inference Resource Usage and Performance upon Heterogeneous Edges

Inference	Edge Types	CPU Usage (%)	GPU Usage (%)	NPU Usage (%)	NPU Core Usage (%)	Average FPS
Yolov4-t	Jetson Nano	97.9	72.4	/	/	23.6
Yolov4-t	Atlas 200DK	99.1	/	12.6	31.2	64.6
Yolov4-n	Jetson Nano	37.5	99.9	/	/	4.4
Yolov4-n	Atlas 200DK	45.5	/	3.1	71.5	18.7
ResNet-18	Jetson Nano	99.9	61.2	/	/	32.2
ResNet-18	Atlas 200DK	99.9	/	11.2	25.1	78.8
BERT	Jetson Nano	29.2	98.5	/	/	1.1
BERT	Atlas 200DK	36.7	/	0.0	82.3	9.1

for TIR beforehand. Thus, we need to propose a method that allows for the dynamic determination of the TIR function coefficients during the process of inference workload redistribution.

3 SYSTEM MODEL

First, we present a batch-aware inference workload redistribution architecture for edge collaborative systems. Based on this, the decision variables of the system are introduced. Next, we present the key constraints in this framework, including network, memory, and computation capacity. Finally, we provide the objectives and overall formulation of the problem.

3.1 Decision Variables

As Figure. 1 shows, we consider an edge system with $\mathcal{K} = \{1, \ldots, K\}$ edges, where each edge is responsible for a specific area that generates $\mathcal{I} = \{1, \ldots, I\}$ different types of intelligent applications. The size of the inference workload generated in the area of edge k by application i during time slot t is denoted by r^t_{ik}. Each intelligent application i may have J_i inference models, with corresponding numbers denoted by $j_i \leq J_i$.

There are a total of T time slots in which workload inference redistribution needs to be performed. At the beginning of each time slot t, the cloud-edge interface determines the distribution of each edge inference workload, the type of deployed DNN inference models, and their corresponding batch size on each edge.

Inference workload decisions. The quantity of inference workload generated by the intelligent application i on device k at time t and transferred to device $k^{'}$ is described as $y^t_{ikk'}$. At each time t, the inference workload distribution of all devices in the edge collaborative system will be recalculated for each intelligent application i.

The quantity of inference workload transferred from each device k satisfies the following relationship:

$$\forall t, i, k, \sum_{k'} y^t_{ikk'} = r^t_{ik}. \tag{3}$$

Inference model and batch size decisions. To handle the redistributed workload, at each time t and for each edge k, we need to make decisions on which inference models should be deployed and what parallel features (batch size) each inference model corresponds to for each intelligent application i.

The variable $x^t_{ijk} \in \{0, 1\}$ is used to indicate whether the corresponding inference model j of application i is deployed on edge k during time slot t. The variable $b^t_{ij} \in N$ is used to represent the batch size corresponding to the deployed inference model j of application i on edge k during time slot t.

Only after deploying the inference model can we make decisions about the batch size. Therefore, these two decision variables have the following constraints:

$$\forall t, k, i, j, b^t_{ijk}(x^t_{ijk} - 1) = 0, b^t_{ijk} \geq x^t_{ijk}. \tag{4}$$

Decision variables relationship. Once the inference workload distribution is finished, it should be ensured at the current time slot t that all workloads transferred to edge k for intelligent application i can be completed by the inference models deployed on that edge. In other words,

$$\forall t, k, i, \sum_j x^t_{ijk} b^t_{ijk} = \sum_{k'} y^t_{ik'k}. \tag{5}$$

3.2 System Limits

The edge collaborative system is subject to resource limitations in three aspects: memory constraints, computational capacity limitations, and wireless bandwidth constraints.

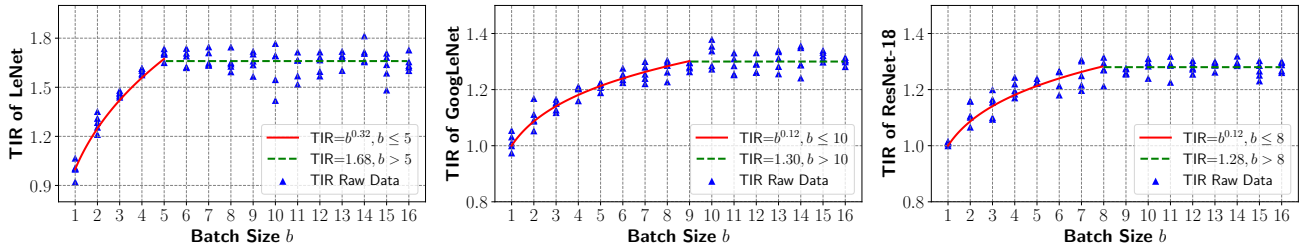
Memory constraints. At each time slot, in order to avoid the overhead of switching the inference model between memory and storage, we load all the inference models into the memory of the edge accelerator and execute each inference in a time-sliced manner. The memory required for the weights of the DNN inference model j_i is δ_{j_i}, and the memory required for intermediate variables when the batch size is 1 is μ_{j_i}. Therefore, for each time slot t and edge k, we have the following equation:

$$\forall t, k, \sum_i \sum_j x^t_{ijk}(\delta_{j_i} + \mu_{j_i} b^t_{ijk}) \leq M_k, \tag{6}$$

where M_k denoted the memory of edge k.

Computational capacity limitations. There are many works [36, 37] that predict the computation latency of a single inference on different devices. We use the method proposed in reference [36]. $\gamma^k_{j_i}$ is used to represent the computation latency of inference j_i on edge k. According to the aforementioned definition of TIR functions Eq. 2, when inference j_i executed in a batch-aware manner, the overall computation time is

$$f^k_{j_i}(b^t_{ijk}) = \frac{b^t_{ijk}\gamma^k_{j_i}}{TIR^k_{j_i}(b^t_{ijk})} = \begin{cases} \gamma^k_{j_i}(b^t_{ijk})^{1-\eta^k_{j_i}}, & b^t_{ijk} \leq \beta^k_{j_i}, \\ \gamma^k_{j_i} b^t_{ijk}/C^k_{j_i}, & b^t_{ijk} > \beta^k_{j_i}, \end{cases} \tag{7}$$



(a) The LeNet TIR fitting function. **(b) The GoogLeNet TIR fitting function.** **(c) The ResNet-18 TIR fitting function.**

Figure 2: The fitting results of different inference models TIR function.

Thus, the sum of computation time for all inference models on an edge k should not exceed the time slot duration τ:

$$\forall t, k, \sum_i \sum_j x_{ijk}^t f_{j_i}^k(b_{ijk}^t) \le \tau. \tag{8}$$

Wireless bandwidth constraints. For each edge k at time slot t, the wireless bandwidth constraints are represented as

$$\begin{aligned} \forall t, k, \sum_i \sum_j \xi_{j_i} [x_{ijk}^t - x_{ijk}^{t-1}]^+ + \\ \sum_i \zeta_i \sum_{k', k' \neq k} (y_{ikk'}^t + y_{ik'k}^t) \le N_k^t. \end{aligned} \tag{9}$$

The first term represents the network resource consumption brought by model changes, where ξ_{j_i} represents the cost of inference j_i in network transmission, and $[a]^+$ represents function $maxa, 0$. The second term represents the network resource consumption brought by the redistribution of inference workload, where ζ_i represents the cost of forwarding an inference request i. N_k^t represents the total network resources at edge k at time slot t.

3.3 Problem Formulation

Optimization goals. Our optimization objective is to achieve the highest inference accuracy for all inference workloads within the edge collaborative system, which means

$$\min \sum_t \sum_k \sum_i \sum_j loss_{ij} x_{ijk}^t b_{ijk}^t, \tag{10}$$

where $loss_{j_i}$ is used to denote inference error of model j_i. The lower the inference error, the higher the inference accuracy. Therefore, the global goal is to minimize the overall loss.

Problem formulation. Taking the above equations into account, the optimization problem can be written as:

$$\begin{aligned} \mathcal{P}_0 : \min_{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{b}} \sum_t \sum_k \sum_i \sum_j loss_{ij} x_{ijk}^t b_{ijk}^t, \\ s.t. \quad Eq.\ 3 - Eq.\ 9, \end{aligned}$$

where $\boldsymbol{x}, \boldsymbol{y}$, and $\boldsymbol{b}$ are tensors with dimensions $[T, |\mathcal{K}|, |\mathcal{I}|, \max\{J_i\}]$, $[T, |\mathcal{I}|, |\mathcal{K}|, |\mathcal{K}|]$, and $[T, |\mathcal{K}|, |\mathcal{I}|, \max\{J_i\}]$ respectively.[2]

4 ALGORITHM DESIGN

The difficulty in solving problem $\mathcal{P}_0$ mainly comes from three aspects: firstly, three hyperparameters in the TIR function are unknown and need to be dynamically obtained; secondly, there are constraints coupling between the various time slots; thirdly, the computational condition constraints belong to piecewise functions and contain non-linear terms;

[2]If the number of corresponding models for application i is less than $\max\{J_i\}$, for the models that do not actually exist, we assign the corresponding parameters a value of either 0 or positive infinity. to prevent them from being selected. Thus it is equivalent to the non-existence of those models.

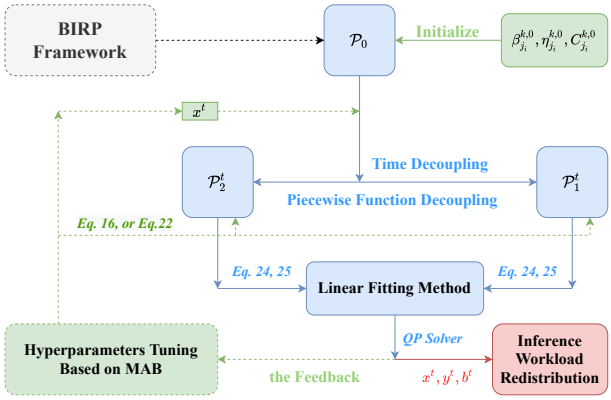


Figure 3: BIRP framework (relationship between proposed problems and algorithms).

In order to address these issues, we first decouple the optimization problem into an online problem based on the resolutions obtained in the previous time slot. Next, for problems with uncertain hyperparameters, we balance exploration and exploitation based on the idea of multi-armed bandits in order to find the long-term optimal solution. Finally, for problems with non-linear terms, we propose a linear fitting method based on the Mean Value Theorem to simplify the solution process for each time slot, without causing too much loss in inference accuracy.

4.1 Optimization Problem Decouple

The coupling of $\mathcal{P}_0$ involves two aspects: the decision variables and the results of the solution at the previous time slot that can affect the formulation of the current problem.

Piecewise functions decoupling. Eq. 7 and Eq. 8 are determined by piecewise functions, demonstrating the influence of decision variables on the selection of the expression during solving $\mathcal{P}_0$.

We note that in the piecewise function Eq. 7, the second segment is a linear function, indicating that increasing the batch size has little effect on the TIR, and as shown in Eq. 6, increasing the batch size will increase memory usage, which instead causes the inference loss to increase under the low inference workload.

In addition, at the threshold value $\beta_{j_i}^k$, the result of the linear function is very close to that produced by the first segment as Figure. 2 shows. This indicates that when the batch is at the threshold

value, the corresponding decision is better than when the batch is greater than the threshold value. Therefore, for this piecewise function, we only need the threshold point and the first segment.

In this way, TIR function can be transformed into:

$$TIR_{j_i}^{k,t}(b_{ijk}^t) = (b_{ijk}^t)^{\eta_{j_i}^{k,t}}, b_{ijk}^t \le \beta_{j_i}^{k,t} \tag{11}$$

Then, the computational limitations described by Eq. 7 and Eq. 8 can be transformed as two constraints follows:

$$\forall t,k, \sum_i \sum_j x_{ijk}^t y_{j_i}^k (b_{ijk}^t)^{1-\eta_{j_i}^{k,t}} \le \tau,\ b_{ijk}^t \le \beta_{j_i}^{k,t}. \tag{12}$$

Here, $\beta_{j_i}^k$ and $\eta_{j_i}^k$ are both hyperparameters, and they will change the function value. We use a Multi-Armed Bandit (MAB) based method to estimate them, and the hyperparameter change in every time slot, denoted by the superscript t. Besides, $C_{j_i}^k$ is also a hyperparameter. Although it does not appear in Eq. 12, it affects the selection of the other two hyperparameters. We will provide a detailed introduction to this method in the next subsection.

Time decoupling. Eq. 9 indicates that the problem solution in the current time slot depends on the solution of the previous time slot. As model weights are often transmitted after compression, compared with the inference workload redistribution, the consumption of network resources is less and is not the determining factor. Therefore, we simplify the process and do not seek to make decisions on the model deployment method in the entire time domain. Instead, we seek the optimal solution under the current workload, and then treat it as a known quantity and pass it to the next time slot, thus decoupling the problem in the time domain.

In other words, when $x_{ijk}^{t-1} = 0$, we have

$$\forall k,t \sum_i \zeta_i \sum_{k',k'\neq k} (y_{ikk'}^t + y_{ik'k}^t) \le N_k^t. \tag{13}$$

then the problem $\mathcal{P}_0$ can be transformed into $\mathcal{P}_1^t$ at each time slot t. The formulation of $\mathcal{P}_1^t$ is as follows:

$$\mathcal{P}_1^t: \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{b}} \sum_k \sum_i \sum_j loss_{ij} x_{ijk}^t b_{ijk}^t,$$
$$s.t.\quad Eq.\ 3 - Eq.\ 6,\quad Eq.\ 12, Eq.\ 13.$$

When $x_{ijk}^{t-1} = 1$, Eq. 9 is transformed as follows:

$$\forall t,k, \sum_i \sum_j \xi_{j_i} x_{ijk}^t + \sum_i \zeta_i \sum_{k',k'\neq k} (y_{ikk'}^t + y_{ik'k}^t) \le N_k^t. \tag{14}$$

Then the $\mathcal{P}_0$ can be transformed into $\mathcal{P}_2^t$ at each time slot t. The formulation of $\mathcal{P}_2^t$ is as follows:

$$\mathcal{P}_2^t: \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{b}} \sum_k \sum_i \sum_j loss_{ij} x_{ijk}^t b_{ijk}^t,$$
$$s.t.\quad Eq.\ 3 - Eq.\ 6,\quad Eq.\ 12, Eq.\ 14.$$

4.2 Online Hyperparameters Tuning

As Eq. 7 shows and discussed in the motivation part and previous section, there are three hyperparameters $\beta_{j_i}^k$, $\eta_{j_i}^k$, and $C_{j_i}^k$.

Historical estimates and observed values. At each time slot t, we solve either problem $\mathcal{P}_1^t$ or $\mathcal{P}_2^t$ based on the solution obtained in the previous time slot (the solution process of these two problems will be discussed in the following subsection). When solving $\mathcal{P}_1^t$ or $\mathcal{P}_2^t$, hyperparameters are used with historical estimates.

We use '$\bar{\cdot}$' to denote historical estimates of the hyperparameters, i.e. $\bar{\eta}_{j_i}^{k,t}$, $\bar{C}_{j_i}^{k,t}$ and $\bar{\beta}_{j_i}^{k,t}$. According to Eq. 2, the observed values of these three hyperparameters can be calculated based on the observed value of TIR. We use '$\widehat{\cdot}$' to represent the observed value i.e. $\widehat{TIR}_{j_i}^{k,t}$, $\widehat{\eta}_{j_i}^{k,t}$, $\widehat{C}_{j_i}^{k,t}$ and $\widehat{\beta}_{j_i}^{k,t}$.

We divide the process of tuning hyperparameters into two parts: within thresholds and beyond thresholds, based on the difference between the observed value of $TIR_{j_i}^{k,t}$ and the historical estimate of $C_{j_i}^{k,t}$. $n_{j_i,1}^{k,t}$ and $n_{j_i,2}^{k,t}$ is used to represent the number of total times $b_{ijk}^{t'}(t' < t)$ falls within the thresholds and beyond thresholds in Eq. 2, respectively.

If the b_{ijk}^t is within the threshold, only the $\eta_{j_i}^{k,t}$ need to be adjusted based on Eq. 11. However, if the value is beyond the threshold, the threshold size needs to be adjusted based on Eq. 2. We will discuss two cases based on the values of $\bar{C}_{j_i}^{k,t}$ and $\widehat{TIR}_{j_i}^{k,t}$, in order to gradually approach the observed value through historical estimates.

$\beta_{j_i}^{k,t}$ and $C_{j_i}^{k,t}$ Tuning. At time slot t, if the following conditions are satisfied, which means b_{ijk}^t is beyond the threshold,

$$\widehat{TIR}_{j_i}^{k,t} \ge (1+\epsilon_1) * \bar{C}_{j_i}^{k,t}, \tag{15}$$

where ϵ_1 is a predetermined parameter, $\widehat{\beta}_{j_i}^{k,t}$ and $\widehat{C}_{j_i}^{k,t}$ will be updated as follows:

$$\begin{aligned}\bar{\beta}_{j_i}^{k,t+1} &= \frac{1}{n_{j_i,2}^{k,t}+1}\left(b_{ijk}^t - \bar{\beta}_{j_i}^{k,t}\right) + \bar{\beta}_{j_i}^{k,t},\\ \bar{C}_{j_i}^{k,t+1} &= \frac{1}{n_{j_i,2}^{k,t}+1}\left(\widehat{TIR}_{j_i}^{k,t} - \bar{C}_{j_i}^{k,t}\right) + \bar{C}_{j_i}^{k,t}.\end{aligned} \tag{16}$$

Eq. 15 indicates that due to factors such as imprecise initialization and computation random changes, the historical estimated threshold $\beta_{j_i}^{k,t}$ of TIR and $C_{j_i}^{k,t}$ at the previous time slot t are inaccurate and require readjustment. To account for these factors, we use a predefined ϵ_1 to characterize the allowable range of changes that the algorithm can tolerate.

Eq. 16 provides an unbiased estimation of $\beta_{j_i}^{k,t}$ and $C_{j_i}^{k,t}$ based on their historical estimates. However, when the inference workload changes gradually, this will lead to a locally optimal solution. Therefore, following the Multi-Armed Bandit (MAB) theory [37], we adopt a balanced exploration-exploitation approach by using the lower bound of the hyperparameter's confidence interval.

We replace the historical estimates $\widehat{\beta}_{j_i}^{k,t}$ and $\widehat{C}_{j_i}^{k,t}$ with its lower confidence bound without violating the original computing constraints. This approach allows for a balance between exploration and exploitation. Specifically, we add a padding term that depends on the historical estimation, as shown in the following equation:

$$\begin{aligned}\beta_{j_i}^{k,t+1} &= \left\lceil \bar{\beta}_{j_i}^{k,t+1}\left\{1 - \sqrt{\epsilon_2 \ln(t+1)/(n_{j_i,2}^{k,t}+1)}\right\}\right\rceil,\\ C_{j_i}^{k,t+1} &= \bar{C}_{j_i}^{k,t+1}\left\{1 - \sqrt{\epsilon_2 \ln(t+1)/(n_{j_i,2}^{k,t}+1)}\right\},\end{aligned} \tag{17}$$

where $\sqrt{\epsilon_2 \ln(t+1)/(n_{ji,2}^{k,t}+1)}$ is the confidence interval ratio to different hyperparameters [37], and $\beta_{ji}^{k,t+1}$ is ceilinged due to the threshold must be integers.

After the updates of $\beta_{ji}^{k,t}$ and $C_{ji}^{k,t}$, the algorithm updates the number of total times b_{ijk}^{t} falls into the first and second stage functions in Eq. 2 as follows:

$$n_{ji,1}^{k,t+1} = n_{ji,1}^{k,t}, \quad n_{ji,2}^{k,t+1} = n_{ji,2}^{k,t} + 1. \tag{18}$$

$\eta_{ji}^{k,t}$ Tuning. Similarly, at time slot t, if the conditions Eq. 15 are not satisfied, which means b_{ijk}^{t} is within the thresholds, $\widehat{\eta}_{ji}^{k,t}$ and $n_{ji,1}^{k,t}$ will be updated as follows:

$$\bar{\eta}_{ji}^{k,t+1} = \frac{1}{n^t+1}\left(\widehat{\eta}_{ji}^{k,t} - \bar{\eta}_{ji}^{k,t}\right) + \bar{\eta}_{ji}^{k,t}, \tag{19}$$

$$n_{ji,1}^{k,t+1} = n_{ji,1}^{k,t} + 1, \quad n_{ji,2}^{k,t+1} = n_{ji,2}^{k,t}, \tag{20}$$

where

$$\widehat{\eta}_{ji}^{k,t} = \frac{\ln \widehat{TIR}_{ji}^{k,t}}{\ln b_{ijk}^{t}}, 1 < b \le \bar{\beta}_{ji}^{k,t}. \tag{21}$$

After the update of historical estimate $\eta_{ji}^{k,t}$, We replace it with its lower confidence bound as follows:

$$\eta_{ji}^{k,t+1} = \bar{\eta}_{ji}^{k,t+1}\left\{1 - \sqrt{\epsilon_2 \ln(t+1)/(n_{ji,2}^{k,t}+1)}\right\}. \tag{22}$$

Hyperparameter initialization. According to our experimental motivation observations, when executing batch-aware multi-workload inference on different devices, we observed that their η_{ji}^{k} is generally greater than 0.1 and β_{ji}^{k} is less than 16. Thus, based on the definition of TIR in Eq. 2, we assign initial values to these three variables using a conservative approach.

$$\eta_{ji}^{k,0} = 0.1, \beta_{ji}^{k,0} = 16, C_{ji}^{k,0} = 16^{0.1} \approx 1.31 \tag{23}$$

4.3 Linear Fitting Method

Despite the entire problem $\mathcal{P}_0$ decoupled into each time slot and the values of each hyperparameter can be dynamically estimated, the problem still lacks a good solution due to the inherent non-linear function in Eq. 12.

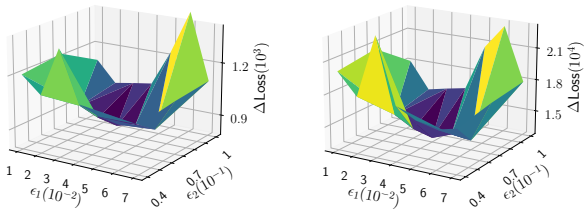
To address this problem, we use the Taylor series to expand Eq. 12 at the point (1,1) as linear constraints during solving $\mathcal{P}_1^t$ or $\mathcal{P}_2^t$. Specifically, we set

$$\gamma_{ji}^{k}(b_{ijk}^{t})^{1-\eta_{ji}^{k,t}} \approx \gamma_{ji}^{k}\left[(1-\eta_{ji}^{k,t})b_{ijk}^{t} + \eta_{ji}^{k,t}\right] = h_{ji}^{k,t}(b_{ijk}^{t}) \tag{24}$$

Then, we use Eq. 24 to replace the non-linear term in Eq. 12 and obtain the following constraints:

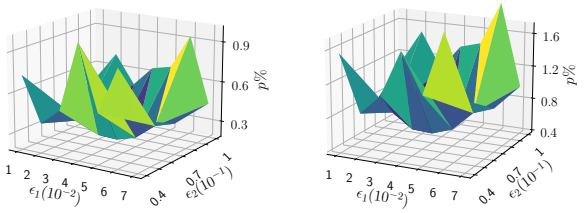
$$\forall t, k, \sum_i \sum_j x_{ijk}^{t} * h_{ji}^{k,t}(b_{ijk}^{t}) \le \tau, \; b_{ijk}^{t} \le \beta_{ji}^{k,t}. \tag{25}$$

We ultimately formulate the problem $\mathcal{P}_1^t$ or $\mathcal{P}_2^t$ for each time slot t as an integer quadratic programming problem, for which there exist mature algorithms that can solve it quickly. We employ Gurobi [14] for solving this problem. The overall framework of this **B**atch-aware **I**nference workload **R**edistribution and **P**arallel execution scheme, called **BIRP**, is illustrated in Fig. 3.



(a) When $t = 10$. **(b) When $t = 100$.**

Figure 4: The impact of ϵ_1 and ϵ_2 selections on the $\Delta Loss$.



(a) When $t = 100$. **(b) When $t = 300$.**

Figure 5: The impact of ϵ_1 and ϵ_2 selections on the p%.

5 EXPERIMENTS AND EVALUATIONS

5.1 Experiments settings.

We prototype BIRP on an edge collaborative system composed of three heterogeneous edges (Jetson NX, Jetson Nano, Atlas200DK), with each type of edge having two instances. Each edge is responsible for receiving the inference workloads generated by the terminals within its own region. The cloud-edge interface, which is also the workload redistribution scheduler, can perceive the inference workload received by each edge in each time slot.

According to the literature [19], we have adopted the following settings: each time slot is 15 minutes, a total duration of three days. The inference workload trace comes from [34]. We employ five common intelligent applications in the industrial internet, including object detection, face recognition, image recognition, neural language understanding, and semantic segmentation. Each application corresponds to five different DNN inference models, from Resnet-18 to BERT. The inference loss of these models varies in $[0.15, 0.49]$. Their inference time with one request on heterogeneous edges varies in $[18, 770]$ ms. The size of these inference weights varies in $[33, 550]$ MB. The compressed model weights consume network resources vary in $[7, 98]$ MB during transmission in the network. When only one request is being executed for these DNN inference models, the size of the intermediate results varies in $[55, 480]$ MB. The size of different inference requests varies in $[0.2, 3]$ MB. The network bandwidth of each edge per time slot varies in $[50, 100]$ Mbps. Excluding system overhead, the memory resources of each edge vary in $[4500, 6500]$ MB.

5.2 Comparative algorithms.

BIRP: our proposed **B**atch-aware **I**nference workload **r**edistribution and **P**arallel execution scheme, which will be analyzed in the third subsection for the impact of preset parameters, and compared with other algorithms in the fourth subsection.

OAEI: the state-of-the-art model selection-based inference workload redistribution algorithm [19], using online learning and random rounding to solve the optimal inference load balancing for edge collaborative systems.

BIRP-OFF: Offline analysis of the relationship between batch size and TIR, without performing online hyperparameters tuning, directly solves for $\mathcal{P}_1^t$ or $\mathcal{P}_2^t$ to obtain the optimal distribution of inference workload.

MAX: set a large batch size B_0 which can optimize resource utilization, and when performing workload redistribution, the inference batch transfer must be followed according to the B_0.

We use two types of metrics to evaluate **BIRP**: overall inference loss and the failure rate $p\%$ that inference requests violate the response-time SLO.

Note that we do not compare simple algorithms such as selecting only the best model to improve accuracy, because these methods are not better than OAEI [19]. In addition, we only tested BIRP-OFF on 5 models, as offline profiling the TIR function on different devices takes a long time.

5.3 Preset parameters analysis.

We first evaluate the impact of the preset parameters' values on the two metrics, as shown in Fig. 4 and Fig. 5.

In **BIRP**, there are preset parameters: ϵ_1 and ϵ_2. ϵ_1 affects the exploration stochastic changes of the TIR function over time, while ϵ_2 mainly affects the exploration efficiency.

The impact of ΔLoss. Regarding the overall inference loss, we describe it using the cumulative error of **BIRP** and algorithm **BIRP-OFF** as $\Delta Loss$, where $\Delta Loss = \sum_t (loss_{BIRP} - loss_{off})$, in order to characterize the impact of different preset parameters selections on the inference loss. As Fig. 4a and Fig. 4b show, when ϵ_2 is large, the padding of the MAB is also large, resulting in higher volatility and larger cost for exploration. Therefore, when the ϵ_2 is large at the initial stage, the $\Delta Loss$ increases significantly. As time goes on, the padding decreases compared to the historical estimates, resulting in a relatively small increase compared to other smaller ϵ_2. For ϵ_1, a smaller value can ensure a more accurate estimation of the inference workload redistribution at the initial stage. However, as time goes on, ϵ_1 also prevents **BIRP** from finding better values for inference workload redistribution. That is, as shown in the comparison between Fig. 4a and Fig. 4b, the lower part with smaller ϵ_1 quickly rises as time increases.

The impact of $p\%$. Regarding the failure rate of SLO, we describe it as the percentage $p\%$ of inference workloads that were not completed within SLO to the total inference workload, in order to characterize the impact of ϵ_1 and ϵ_2 on the exploration cost. As Fig. 5 shows, when ϵ_2 is small, the smaller padding of MAB leads to a decrease in overall exploration and limits it to local optima. In cases of high workload, the batch-aware parallel processing feature is not fully utilized, resulting in an increase in $p\%$. As ϵ_1 increases, the high-tolerance threshold $\beta_{j_i}^{k,t}$ of TIR function causes **BIRP** to favor exploration of larger batch sizes with optimism, resulting in more resource consumption of larger models at the expense of smaller models, leading to an increase in $p\%$.

Therefore, we should choose appropriate values for ϵ_1 and ϵ_2, neither too large nor too small. Based on Fig. 4 and Fig. 5, we choose $\epsilon_1 = 0.04$ and $\epsilon_2 = 0.07$. In the following subsection, **BIRP** will use these presets for experiments.

5.4 Performance evaluation.

Firstly, we deployed BIRP on a small-scale heterogeneous edge collaborative system with one application and 3 inference models as shown in Fig. 6. The TIR function of these 3 models has been measured offline. Then, we conducted a large-scale deployment of BIRP with five applications and 25 models, and the results are shown in Fig. 7.

CDF. We first compared the inference completion time distributions of different algorithms, as shown in Fig. 6a and Fig. 7a. When $t = T+1$, we provided the Cumulative Distribution Functions (CDF) of inference completion time. As Fig. 6a shows, During the execution of the **BIRP** and **BIRP-OFF** algorithms, the failure rate of SLO is 1.9%, that is, the proportion of inference workloads with $\tau \geq 1.0$ did not exceed 1.9% of the total. While the failure rate of OAEI is 10.0%.

The CDF of **BIRP** is slightly skewed to the right compared to **BIRP-OFF**, but it does not affect the overall execution time. Instead, this demonstrates that the tuning module of **BIRP** is effective because their CDFs are close. The distribution of **OAEI** is dense when $\tau \leq 0.3$, but becomes sparse when $0.3 < \tau < 1$. This is because the single inference is completed quickly during serial execution, but additional waiting time causes some inference completion time to decrease when the workload is high. The pattern of **MAX**'s CDF is exactly opposite to that of **OAEI**, because it tends to execute in batches, resulting in an increase in the execution time of individual inference.

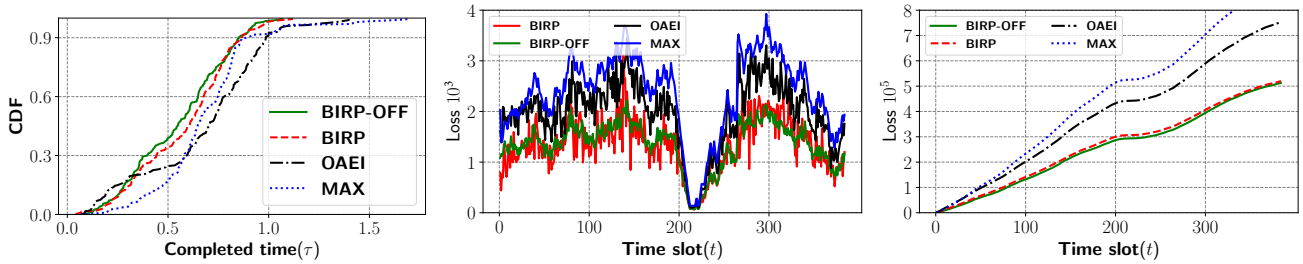
As shown in Fig. 7a, when executed at a large scale, **BIRP** still has significant advantages over algorithms **OAEI** and **MAX**. This is because **BIRP** can dynamically select the batch size and inference model to achieve a decrease in overall completion time and a lower failure rate. In Figure Fig. 7a, **BIRP** has a failure rate of 0.21%, while **OAEI** has a failure rate of 4.1%. Compared to **OAEI**, **BIRP**'s failure rate of SLO has decreased by to 19.8% of **OAEI**.

Inference loss. We evaluate the improvement of **BIRP** on inference accuracy from two aspects: inference loss $loss_t$ at time slot t and accumulated inference loss $\sum_t loss_t$.

Due to the adoption of dynamic batch selection technology, **BIRP** has more computational resources available for intelligent applications to choose better models for computation. Therefore, compared to **MAX** and **OAEI**, **BIRP** achieved better results at each time slot t as shown in Fig. 6b and Fig. 7b.

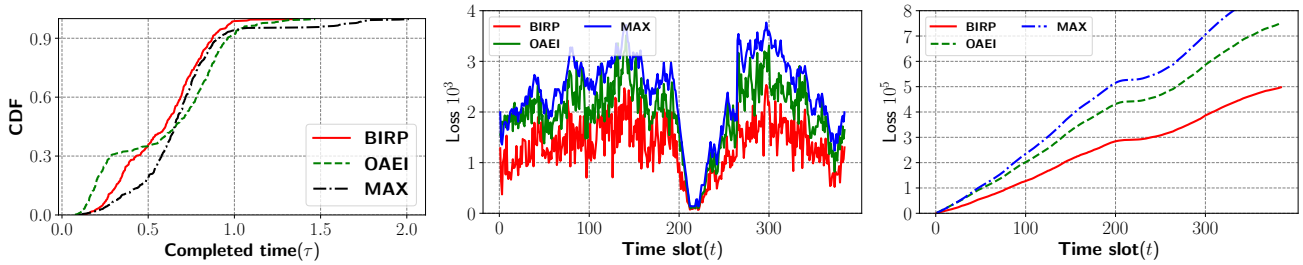
When the inference workload is low, **OAEI** and **BIRP** achieved similar results, as serial execution of large models can still meet the requirements of intelligent applications under low workloads. **MAX** still had high loss under high workload, because **MAX** tends to maximize utilization, but resource constraints make it difficult to apply large model parallelism. Although **MAX** has a higher resource utilization rate, it did not fully utilize the capabilities of the models.

As shown in Fig. 6c, there is a slight difference between **BIRP** and **BIRP-OFF**, which is due to the additional cost brought by **BIRP**'s exploration phase. But as the inference workload redistribution continues, this difference gradually approaches zero. After



(a) The CDF of inference completion time. **(b) The overall inference loss at time slot t.** **(c) Accumulative inference loss at time slot t.**

Figure 6: Inference workload redistribution results in small-scale evaluations.



(a) The CDF of inference completion time. **(b) The overall inference loss at time slot t.** **(c) Accumulative inference loss at time slot t.**

Figure 7: Inference workload redistribution results in large-scale evaluations.

the inference workload redistribution execution is completed, as shown in Figure. 7c, the inference loss of **BIRP** decreased by 32.3% compared to **OAEI**.

6 RELATED WORK

Edge DNN inference accelerating. Despite the powerful pattern recognition capabilities of DNN inference models, they often require complex tensor operations, and traditional CPU architectures tend to be time-consuming in processing DNN inference [26–28]. Hardware accelerators with parallel acceleration capabilities are needed for processing DNN complex tensor operations, often with multiple Streaming Multiprocessors (SMs). Typical edge accelerators include Jetson Nano [23], Huawei Atlas 200DK [18], Google Edge TPU [13], and other platforms. After an inference request is initiated, it will be executed on these edge accelerators. Some DNN inference methods are specifically designed for accelerating on particular platforms. For example, the ATC [17] tool is only effective for Huawei Ascend [18], while TensorRT [23] is only effective for GPUs. There have been approaches that propose kernel-level inference workload parallelization techniques [15], as well as methods for GPU virtualization [6]. However, these methods require the hardware architecture to have certain special configurations, such as NV-Link [24] or preemptive scheduling [6]. Compared to these parallel methods, batch-aware parallel methods only defer certain inference tasks within their SLO to execute similar types of inference workloads together, achieving maximum resource utilization and better generality.

Inference workload redistribution. Inference workload redistribution techniques are typically designed to address the issue of workload imbalance for uncertain edge collaborative systems. Thai *et. al* proposed a cloud-edge computing architecture, optimized through a workload and capacity optimization problem [30]. Wang *et. al* adopts a reinforcement learning approach from the perspective of each user to decide the inference workload allocation [33]. Ding *et. al* proposes potential game-based algorithms with end-edge-cloud systems for workload redistribution [10]. These works optimize overall resource utilization from the perspective of cloud, edge, or end, but do not jointly consider the combined impact of inference accuracy and inference latency. Jin *et. al* balance the tradeoff between inference accuracy and inference latency by employing different versions of inference models [19]. In addition, some research [7] focuses on predicting the distribution of workloads to provide a decision-making basis. But these inference workload redistribution methods only consider scenarios where inference is performed serially at the edges.

Online system parameters configuration. In distributed systems, it is common to treat the system as a black box and configure various system parameters online. Ernest [31] uses collected historical hardware parameters on its cloud computing infrastructure to predict the performance of various tasks running on it. Vizier [12], on the other hand, targets Google's cloud service infrastructure and uses machine learning models to adjust various hardware parameters. Morphling [32] attempts to use meta-learning methods to find the optimal configuration for inference performed on multiple GPUs. These methods often require pre-setting machine learning models and preparing large amounts of training data, which is difficult to achieve in edge collaborative environments. There are also works [37] that propose using MAB to obtain accurate computation latency of inference models, but they focus on the execution latency of a single inference task model.

7 CONCLUSION

To address the issue of inference workload imbalance, **BIRP** was the first to introduce a batch-aware parallel inference approach

that improves system utilization by merging the same inference requests within the allowed range of the SLO. Based on the real inference workload trace, we validate the superiority of **BIRP** with other SOTA alternatives.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation of China, under grant No. 61832005; China University Industry Research Innovation Foundation, under grant No. 2021FNA04005. And this work is also partially supported by the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

[1] Hyame Assem Alameddine, Sanaa Sharafeddine, Samir Sebbah, Sara Ayoubi, and Chadi Assi. 2019. Dynamic Task Offloading and Scheduling for Low-Latency IoT Services in Multi-Access Edge Computing. *IEEE Journal on Selected Areas in Communications* 37, 3 (March 2019), 668–682. https://doi.org/10.1109/JSAC.2019.2894306

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934 [cs, eess]

[3] Chen Chen, Jiange Jiang, Yang Zhou, Ning Lv, Xiaoxu Liang, and Shaohua Wan. 2022. An Edge Intelligence Empowered Flooding Process Prediction Using Internet of Things in Smart City. *J. Parallel and Distrib. Comput.* 165 (July 2022), 66–78. https://doi.org/10.1016/j.jpdc.2022.03.010

[4] Jiasi Chen and Xukan Ran. 2019. Deep Learning With Edge Computing: A Review. *Proc. IEEE* 107, 8 (Aug. 2019), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977

[5] Miaojiang Chen, Wei Liu, Tian Wang, Anfeng Liu, and Zhiwen Zeng. 2021. Edge Intelligence Computing for Mobile Augmented Reality with Deep Reinforcement Learning Approach. *Computer Networks* 195 (Aug. 2021), 108186. https://doi.org/10.1016/j.comnet.2021.108186

[6] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 199–216.

[7] Rong Cong, Zhiwei Zhao, Linyuanqi Zhang, and Geyong Min. 2022. CoopEdge: Cost-Effective Server Deployment for Cooperative Multi-Access Edge Computing. In *2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, Stockholm, Sweden, 208–216. https://doi.org/10.1109/SECON55815.2022.9918592

[8] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. 2020. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet of Things Journal* 7, 8 (Aug. 2020), 7457–7469. https://doi.org/10.1109/JIOT.2020.2984887

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs]

[10] Yan Ding, Kenli Li, Chubo Liu, and Keqin Li. 2022. A Potential Game Theoretic Approach to Computation Offloading Strategy Optimization in End-Edge-Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems* 33, 6 (June 2022), 1503–1519. https://doi.org/10.1109/TPDS.2021.3112604

[11] Fotis Foukalas and Athanasios Tziouvaras. 2021. Edge Artificial Intelligence for Industrial Internet of Things Applications: An Industrial Edge Intelligence Solution. *IEEE Industrial Electronics Magazine* 15, 2 (June 2021), 28–36. https://doi.org/10.1109/MIE.2020.3026837

[12] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax NS Canada, 1487–1495. https://doi.org/10.1145/3097983.3098043

[13] Google. 2019. Edge TPU. https://cloud.google.com/edge-tpu

[14] Gurobi. 2023. Gurobi Optimization. https://www.gurobi.com/

[15] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale Preemption for Concurrent GPU-accelerated DNN Inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 539–558.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs]

[17] Huawei. 2019. ATC Tool - Ascend Data Center Solution V100R020C30 Center Inference Solution Description 01 - Huawei. https://e.huawei.com/hk/material/

[18] Huawei. 2019. Huawei Ascend. https://www.hiascend.com/

[19] Yibo Jin, Lei Jiao, Zhuzhong Qian, Sheng Zhang, Ning Chen, Sanglu Lu, and Xiaoliang Wang. 2020. Provisioning Edge Inference as a Service via Online Learning. In *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, Como, Italy, 1–9. https://doi.org/10.1109/SECON48991.2020.9158425

[20] Yann LeCun, Yoshua Bengio, and T Bell Laboratories. 1995. Convolutional Networks for Images, Speech, and Time-Series. *The handbook of brain theory and neural networks* 3361, 10 (1995), 255–258.

[21] Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. 2021. Resource Scheduling in Edge Computing: A Survey. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2131–2165. https://doi.org/10.1109/COMST.2021.3106401

[22] Nivdia. 2019. TensorRT SDK | NVIDIA Developer. https://developer.nvidia.com/tensorrt

[23] Nvidia. 2019. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[24] Nvidia. 2022. Design & Professional Visualization Solutions | NVIDIA. https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/

[25] George Plastiras, Maria Terzi, Christos Kyrkou, and Theocharis Theocharidcs. 2018. Edge Intelligence: Challenges and Opportunities of Near-Sensor Machine Learning Applications. In *2018 IEEE 29th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. 1–7. https://doi.org/10.1109/ASAP.2018.8445118

[26] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2020. Survey of Machine Learning Accelerators. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–12. https://doi.org/10.1109/HPEC43674.2020.9286149

[27] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2021. AI Accelerator Survey and Trends. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–9. https://doi.org/10.1109/HPEC49654.2021.9622867

[28] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2022. AI and ML Accelerator Survey and Trends. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–10. https://doi.org/10.1109/HPEC55821.2022.9926331

[29] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411.

[30] Minh-Tuan Thai, Ying-Dar Lin, Yuan-Cheng Lai, and Hsu-Tung Chien. 2020. Workload and Capacity Optimization for Cloud-Edge Computing Systems with Vertical and Horizontal Offloading. *IEEE Transactions on Network and Service Management* 17, 1 (March 2020), 227–238. https://doi.org/10.1109/TNSM.2019.2937342

[31] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA) *(NSDI'16)*. USENIX Association, USA, 363–378.

[32] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, Seattle WA USA, 639–653. https://doi.org/10.1145/3472883.3486987

[33] Xiong Wang, Jiancheng Ye, and John C.S. Lui. 2022. Decentralized Task Offloading in Edge Computing: A Multi-User Multi-Armed Bandit Approach. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. IEEE, London, United Kingdom, 1199–1208. https://doi.org/10.1109/INFOCOM48880.2022.9796961

[34] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 945–960.

[35] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. 2021. Edge Intelligence: Empowering Intelligence to the Edge of Network. *Proc. IEEE* 109, 11 (Nov. 2021), 1778–1837. https://doi.org/10.1109/JPROC.2021.3119950

[36] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. 2021. Nn-Meter: Towards Accurate Latency Prediction of Deep-Learning Model Inference on Diverse Edge Devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (Virtual Event, Wisconsin) *(MobiSys '21)*. Association for Computing Machinery, New York, NY, USA, 81–93. https://doi.org/10.1145/3458864.3467882

[37] Yitong Zhou, Hesheng Sun, Yibo Jin, Yanfang Zhu, Yuan Li, Zhuzhong Qian, Sheng Zhang, and Sanglu Lu. 2022. Inference Replication at Edges via Combinatorial Multi-Armed Bandit. *Journal of Systems Architecture* 129 (2022), 102636. https://doi.org/10.1016/j.sysarc.2022.102636

[38] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* 107, 8 (Aug. 2019), 1738–1762. https://doi.org/10.1109/JPROC.2019.2918951