Online Scheduling of Edge Multiple-Model Inference with DAG Structure and Retraining

Yifan Zeng*, Ruiting Zhou*†, Lei Jiao‡, Renli Zhang*,
*School of Cyber Science and Engineering, Wuhan University, China
†School of Computer Science and Engineering, Southeast University, China
‡Center for Cyber Security and Privacy, University of Oregon, USA
Email: yifanzeng@whu.edu.cn, ruitingzhou@seu.edu.cn, ljiao2@uoregon.edu, zhang_rl@whu.edu.cn

***Abstract*—Edge inference applications are becoming increasingly complex and composed of multiple models. The dependency of models is modeled by a Directed Acyclic Graph (DAG). The accuracy of the edge model is easily affected by data drift. Retraining is employed to sustain the inference accuracy of models. But the introduction of retraining complicates the inter-task dependency of the inference request. Moreover, model retraining prolongs the inference completion time. The accuracy improvement and the latency increment under different retraining configurations necessitate a trade-off between inference accuracy and request completion time. In this paper, we investigate multiple-model inference with retraining, aiming to maximize inference accuracy while minimizing request completion time. Through experimental analysis, we observed that retraining can enhance model inference accuracy in a short time. We represent the retraining tasks of models as nodes in the DAG of the inference request and then construct a unified DAG structure for both retraining and inference tasks. We first propose a Single Request Scheduling Algorithm (*SRS*) with a theoretical performance guarantee to select the optimal retraining configuration for each model under edge resource constraints and jointly schedule retraining and inference tasks. Subsequently, we extend *SRS* to a Multiple Requests Scheduling Algorithm (*MRS*) to address scheduling in a more general online multi-request scenario. The experiments on an edge system indicate that compared to existing methods, *MRS* can enhance the inference accuracy by 25% while reducing the request completion time by 45%.**

I. Introduction

The diversification of smart devices has greatly enriched the sources of inference data. To promptly handle data streams such as images, videos, and audio from various smart devices, multiple-model applications have been developed, including lifelogging [1], social media [2], video analysis [3], and more. These applications use multiple models organized in directed acyclic graphs (DAGs) to analyze data and achieve various inference objectives. For instance, the lifelogging [1] application integrates object detection models, face recognition models, vehicle classification models, and more. While identifying the objects in the image, it can further analyze the individual behavior and vehicle's property.

This work is supported in part by National Key Research and Development Program of China under Grant No. 2024YFB2907100, Shenzhen Science and Technology Program under Grant No. KJZD20240903100814018, National Natural Science Foundation of China under Grant No. 62232004, Central University Basic Research Fund of China under Grant No. 2242024K40021, Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Big Data Computing Center of Southeast University, as well as in part by the U.S. National Science Foundation under grants CNS-2047719 and CNS-2225949.

Corresponding author: Ruiting Zhou.

Requests from these applications demand quick completion to ensure the effectiveness of data analysis [4]. However, utilizing multiple models for inference is more time-consuming than using a single model, making it challenging to complete requests on edge devices with limited computing and storage resources. On the other hand, cloud computing introduces excessive communication latency. To guarantee low response time for multi-model applications, edge servers are typically employed to handle these requests.

Models deployed by applications are susceptible to the influence of data drift when inferring on edge servers. Data drift refers to significant disparities between the inference data and the data used to train the models [5]. In dynamic environments, the inference data is constantly changing. Factors such as lighting conditions, capture angles, and poses can all contribute to data drift. Due to limited resources (such as GPU), edge servers typically deploy small or compressed models with shallow structures and fewer parameters. These models are unable to properly recognize data drift, leading to a decrease in model inference accuracy.

To address the issue of data drift and enhance real-time inference accuracy, model retraining, also known as continuous learning, is an effective strategy [6]. However, handling the coordinated offloading of model retraining and model inference on edge servers presents new challenges for scheduling multiple-model application requests. **First**, model retraining makes the dependencies between tasks even more intricate. Apart from the dependencies between inference tasks, it is essential to consider the dependencies between retraining tasks and inference tasks for the same model. To maximize inference accuracy, it is desirable to utilize the latest retrained model for inference tasks. The model retraining must be completed before inference, indicating that inference tasks are dependent on the corresponding model's retraining tasks. Changes in the type and number of models being retrained will alter the dependencies between tasks. With varying dependency relationships, the scheduling order of requests needs to adapt accordingly. **Second**, selecting retraining configurations requires striking a balance between inference accuracy and request completion time. Retraining models involve diverse configurations, with resource consumption and end-accuracy varying under different setups. Allocating more resources for retraining may lead to insufficient resources for inference tasks, thus extending request completion times. Conversely, retraining models with limited resources may result in lower inference accuracy after training.

Current research on model retraining has mainly focused on training and updates, neglecting the joint scheduling of retraining tasks with inference tasks. Ekya [7] addresses single-model inference requests, allocating resources and configurations for each task but lacks scalability for multi-model DAG requests. While AdaInf [8] manages DAG with a focus on meeting the Service Level Objective (SLO) of inference requests. However, it provides limited resources for retraining, resulting in marginal accuracy improvements. Noted that both Ekya and AdaInf are tailored for offline requests and unable to manage online requests. We will discuss more details in Sec. VI.

To the best of our knowledge, we are **the first study** of online scheduling for multiple-model inference with retraining. Motivation experiments (in Sec. II) have demonstrated that the model accuracy can be significantly improved through short retraining periods. Therefore, retraining tasks can be inserted as nodes into the DAG of the request, while still ensuring the low completion time for the request. To ensure that retraining tasks are completed before their respective inference tasks, we expand the original DAG structure of requests, making each model's retraining task a predecessor node for its inference task. The dependencies between inference tasks are also preserved. Adding retraining tasks can enhance model inference accuracy but may delay the completion time of the request. To minimize request completion time while maximizing model inference accuracy, we model the request, which has multiple models' retraining tasks and inference tasks, as an extended DAG. Then we propose the single request scheduling algorithm (*SRS*), which selects the optimal retraining configuration for each model to balance request completion time and inference accuracy. The optimal configuration ensures that the improved accuracy after model retraining outweighs the additional request completion time; otherwise, the model is not retrained. We prove that the solution obtained by *SRS* has a theoretical upper bound. Furthermore, extending *SRS* to a multiple request scheduling algorithm (*MRS*) addresses multiple incoming online application requests. We summarize our contributions as follows:

- We demonstrate that inference accuracy can be improved in a short period of time by retraining. Then we add nodes for model retraining tasks to the request DAG, and formulate the scheduling problem to minimize the completion time and maximize accuracy for online requests.
- *SRS* is designed to address single request scheduling with model retraining. It estimates inference resource usage by scheduling the inference tasks. Then *SRS* selects optimal retraining configurations under resource constraints, and offloads retraining and inference tasks together. The upper bound of *SRS* can be proven. *MRS*, an extension of *SRS*, manages multi-request scheduling. It forms task lists, selects initial tasks to create candidate sets, and chooses the earliest task for completion iteratively until all requests are scheduled.
- We conduct extensive experiments on an edge system with real-world request traces. The results demonstrate that our algorithm achieves a 25% enhancement in inference accuracy and a 45% reduction in request completion time, compared to three state-of-the-art algorithms.

In the rest of the paper, we perform motivation analysis in Sec. II. Sec. III models the scheduling of multiple-model requests with retraining in edge computing. In Sec. IV, *SRS* and *MRS* are proposed to schedule single and multiple requests, respectively. Extensive experiments' results are presented in Sec. V. We review the related work in Sec. VI. Finally, we conclude the paper in Sec. VII.

II. Measurement-Based Motivation

To show the impact of data drift and the accuracy improvement of model retraining, we take the lifelogging application [1] shown in Fig. 1 as an example of the multi-model inference. More details of the experimental settings can be found in Section V.

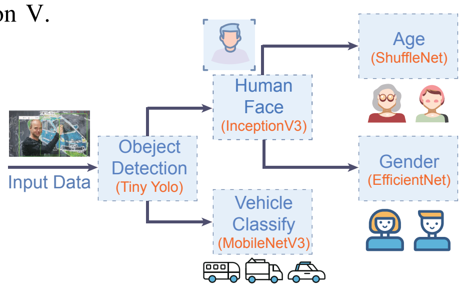

Fig. 1: DAG of the lifelogging application.

A. *Impact of Data Drift*

Fig. 2 illustrates the variation of model inference accuracy over time for different models in a dynamic environment. It can be observed that as time progresses, the inference accuracy of all models shows a decreasing trend. This is due to the significant variations in the data distribution of user input data in a dynamic environment [7]. Even for the same user, data drifts in user input data can occur due to multiple factors, including variations in device type, lighting conditions, posture, and other factors. Models pre-trained on initial datasets may struggle to accurately capture these variations, leading to a decrease in model inference accuracy, especially for tiny or compressed models deployed on edge servers.

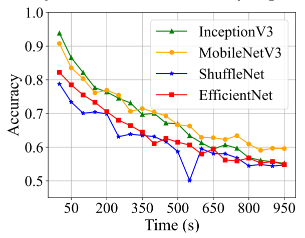

Fig. 2: Inference accuracy of different models (over time).

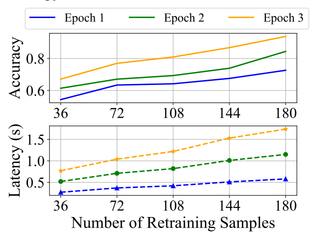

Fig. 3: Impact of the number of retraining samples.

Additionally, the data drift has different impacts on different models. For example, in Fig. 2, during the time intervals of

100-200 seconds and 250-400 seconds, there is a continuous decrease in the inference accuracy of EfficientNet, while the inference accuracy of ShuffleNet remains relatively stable. Therefore, it is necessary to selectively retrain the models used in the application. The timing of retraining should be determined considering the model's accuracy.

B. Improvement of Accuracy by Model Retraining

To address the decrease in model accuracy caused by data drift, we employ model retraining to improve the model's inference accuracy. Different retraining configurations yield varying effects on accuracy improvement. Using the InceptionV3 model as a case study, we explore the impact of retraining parameters, such as *training dataset size, number of training epochs, batch size, and resource allocation*, on the accuracy and end-latency of the model after retraining.

As shown in Fig. 3, with an increase in the number of drift retraining samples used during retraining (i.e., an increase in the size of the training dataset), the accuracy of the retrained model continues to improve. However, it is important to note that this improvement is accompanied by a proportional increase in the retraining latency required to complete the retraining process. The impact of retraining epochs is similar to that of the samples' number. By selecting different combinations of the epoch counts and sample quantities, it is possible to achieve a balance between model accuracy and retraining latency. For instance, when retraining with a dataset of 144 samples, it is possible to achieve a high level of accuracy with just 2 retraining epochs, while maintaining low latency.

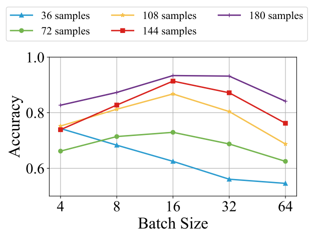

Fig. 4: Retraining accuracy of different batch size.

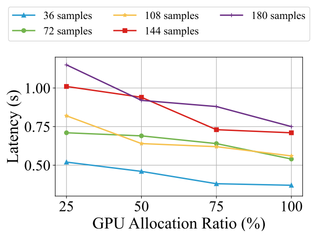

Fig. 5: Retraining latency of different resource.

The training batch size also has a considerable influence on the end-accuracy of retraining. Fig. 4 illustrates the variation in model accuracy after retraining as the batch size increases, considering different numbers of retraining samples. It can be observed that the optimal batch size for retraining varies for different numbers of samples. Typically, a batch size of 16 is a common optimal setting. However, for a small number of samples, such as 36, a smaller batch size may be required to achieve optimal results.

In addition, we need to pay attention to the retraining latency under different resource demands. Due to the time sensitivity of user requests, the retraining process needs to be completed within a short time frame. Fig. 5 displays the retraining latency under different GPU resource allocation ratios, with each retraining process involving two epochs. It is evident that most model retraining tasks can be completed within a short period of time, even with limited resource allocation. However, for retraining tasks with larger workloads, it is necessary to allocate more resources to reduce the training latency.

C. Observation

We observed that: *i)* Data drift causes a decline in model inference accuracy, with varying degrees of impact on different models. *ii)* Model inference accuracy can be improved in a short time by retraining. *iii)* The retraining configuration of models can be obtained through the experimental profile.

III. System Model

TABLE I: Notations and Descriptions

Notations	Description
$\mathcal{R}$	The set of requests
$\mathcal{E}$	The set of edge servers
$\mathcal{U}$	The set of retraining configurations
e_j	The j-th edge server of $\mathcal{E}$
C_j	The computation resources capacity of edge server e_j
S_j	The computational speed on edge server e_j
$D_{j',j}/D_j$	The data transfer rate from edge server $e_{j'}$/storage cluster to edge server e_j
r_k	The k-th request of $\mathcal{R}$
T_k^a	The arrival time of request r_k
$G_k = (\mathcal{V}_k, \mathcal{L}_k)$	The task DAG of the request r_k
$\lvert V \rvert$	the number of initial inference tasks in request r_k
v_k^i	The i-th task of request r_k
m_k^i	The model called by task v_i^k
a_k^i	The initial accuracy of the model called by task v_i^k
A_k^i	The inference accuracy of the model used by task v_i^k
c_k^i	The required computation resource of task v_i^k
w_k^i	The workload of task v_i^k
$d_k^{i,i'}$	The data size transferred from task v_i^k to task $v_{i'}^k$
u_h	The h-th retraining configuration of $\mathcal{U}$
$x_k^{i,j}$	Whether task v_i^k is assigned to edge server e_j
$y_k^{i,h}$	Whether retraining configuration u_h is used to retrain the model m_i^k

A. Edge Inference with Continuous Learning

System Overview. As shown in Fig. 6, we investigate a system comprising user devices, several edge servers, and a storage cluster. This system is designed to efficiently process user inference requests while ensuring high inference accuracy. Users transmit their inference requests to the edge servers, which may involve input data from various devices like mobile phones, smartwatches, and cameras. Initially, the input data from these devices is transmitted to a fixed edge server known as the controller. Upon receiving a user request, the controller aggregates the necessary data and schedules it for processing on the edge servers. To handle the inference requests, the edge servers leverage pre-stored models. These edge servers, denoted as $e_j \in \mathcal{E}$, show heterogeneity in terms of their computation resource capacities C_j, and computational speeds S_j, measured in teraflops.

Model Retraining. To maintain inference accuracy, model retraining is employed to mitigate the negative effects of data drift in dynamic scenarios. The system collects drift data samples from the inference requests and utilizes them for model retraining. Data drift detection is performed by the edge servers on the input data of each request. The detected drift data samples are received by a storage cluster, where the labels for these samples are derived from the golden models [7] stored in the cluster. Although the golden models exhibit

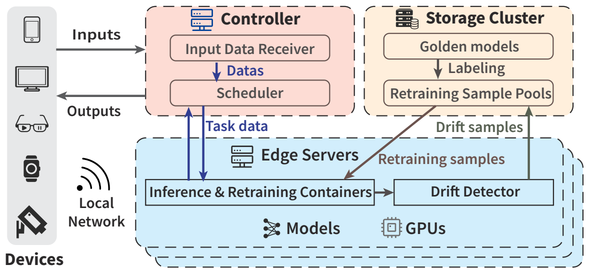

Fig. 6: System Overview.

exceptional accuracy, their high cost and prolonged inference time make them impractical for the direct use in inference requests. For each inference model, the storage cluster maintains a retraining sample pool, continuously capturing and labeling drift data samples. After completing a retraining iteration, the sample pool associated with the retrained model is cleared and begins accumulating fresh samples. Moreover, the updated model parameters obtained from the model retraining process are subsequently applied to every edge server.

Retraining Configuration. Various training hyperparameters are considered for model retraining, including the training dataset size, number of training epochs, batch sizes, frozen layers, resource utilization, etc. The retraining configuration $u_h \in \mathcal{U}$ (i.e., hyperparameters), is denoted as $u_h = (b_h, n_h, o_h)$. Here, b_h represents the data size of retraining samples, n_h denotes the retraining workload (i.e., GPU times), and o_h indicates the computation resource (e.g., GPUs) required for retraining.

Requests and Tasks. We consider a sequence of online inference requests arriving online in arbitrary time and order, denoted as $\mathcal{R}$. The arrival time of request r_k is denoted as T_k^a. We assume the request r_k can be represented as a Directed Acyclic Graph (DAG) $G_k = (\mathcal{V}_k, \mathcal{L}_k)$, where the vertex set $\mathcal{V}_k$ indicates the tasks of request r_k and the edge set $\mathcal{L}_k$ indicates the dependency between tasks. For each task $v_k^i \in \mathcal{V}_k$, the corresponding model called by the task is denoted as m_k^i and its initial accuracy is represented as a_k^i.

Decision Variables. For each task v_k^i in request r_k arriving at the system, the decisions made include: i) $x_k^{i,j} \in \{0, 1\}$, whether task v_i^k is served in edge server e_j; ii) $y_k^{i,h} \in \{0, 1\}$, whether retraining configuration u_h is used to retrain the model m_i^k. Table I summarized the important notations.

B. Extended DAG with Retraining Tasks

Retraining Nodes. For each model called in an inference request, we add a retraining task node of the corresponding model in the request DAG. The training configuration of the retraining task is determined by the decision variable $y_k^{i,h} \in \{0, 1\}$. When $y_k^{i,h} = 1$, the retraining task uses configuration u_h to complete training. When $\sum_h y_k^{i,h} = 0$, model m_k^i does not need to be retrained, that is, the retraining node does not run. We use $|V|$ to represent the number of initial inference tasks in request r_k, then the retraining task for model m_k^i can be expressed as $v_k^{|V|+i}$. For the retraining task $v_k^{|V|+i}$ with retraining configuration u_h, we have the required computation resource $c_k^{|V|+i} = \sum_{u_h \in \mathcal{U}} y_k^{i,h} o_h$, the workload $w_k^{|V|+i} = \sum_{u_h \in \mathcal{U}} y_k^{i,h} n_h$ and the retraining data size $d_k^{|V|+i} = \sum_{u_h \in \mathcal{U}} y_k^{i,h} b_h$. After model m_k^i is retrained with configuration u_h, the model inference accuracy is updated to

$$A_k^i = \sum_{u_h \in \mathcal{U}} y_k^{i,h} f(u_h, m_k^i) + (1 - \sum_{u_h \in \mathcal{U}} y_k^{i,h}) a_k^i, \tag{1}$$

where $f()$ models the relation between model retraining configuration and post-training accuracy. It is noted that $y_k^{i,h}$ indicates whether the model corresponding to task v_k^i is retrained. For the model corresponding to the retraining task (i.e., $i > |V|$), it will definitely not be retrained. Otherwise, the retraining task will be iteratively generated.

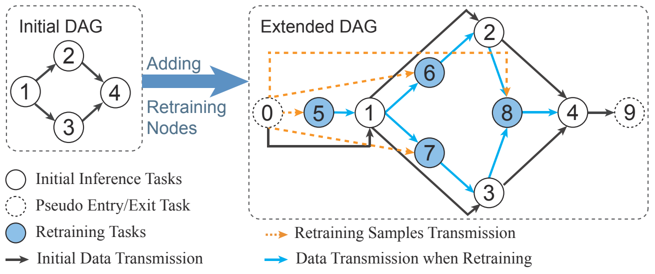

Fig. 7: Adding Retraining Task Nodes.

DAG Extension. As shown in Fig. 7, to add task $v_k^{|V|+i}$ node to the set $\mathcal{V}_k$ of request r_k's DAG, we add edges $< v_k^{i'}, v_k^{|V|+i} >$, $i' \in pre_k(i)$ and $< v_k^{|V|+i}, v_k^i >$ to the set $\mathcal{L}_k$, where $pre_k(i)$ denotes the predecessor tasks of task v_k^i. The transfer data size between predecessor tasks $v_k^{i'}$ and retraining task is $d_k^{i',|V|+i} = \sum_{u_h \in \mathcal{U}} y_k^{i,h} d_k^{i',i}$, and the transfer data size between retraining task $v_k^{|V|+i}$ and task v_k^i is $d_k^{|V|+i,i} = \sum_{u_h \in \mathcal{U}} y_k^{i,h} (\sum_{i' \in pre_k(i)} d_k^{i',i} + d(m_k^i)), i' \in pre_k(i)$. Here, $d(m_k^i)$ represents the data size of model m_k^i parameters, which should be updated to other edges after retraining. Note that the transfer data size between predecessor task $v_k^{i'}$ and task v_k^i is updated to $d_k^{i',i} = (1 - \sum_{u_h \in \mathcal{U}} y_k^{i,h}) d_k^{i',i}$. We call the task without any predecessor tasks as the entry task, and the task without any successor tasks as the exit task. For ease of expression, we assume the entry tasks all connect to a pseudo entry task, denoted as v_k^0, which does not need to be processed. Similarly, all the exit nodes are connected to a pseudo exit node $v_k^{2|V|+1}$. Therefore, there is one pseudo entry task in DAG for each request. Considering the additional training data required for retraining tasks, we include edge $< v_k^0, v_k^{|V|+i} >$ in the set $\mathcal{L}_k$ to represent the transmission of retraining data samples from the storage cluster, and $d_k^{0,|V|+i} = d_k^{|V|+i}$.

C. Request Completion Time

Execution Time. Due to the heterogeneity of edge servers, we assume that the computational speed of each unit resource in edge e_j is S_j. The execution time of the task v_k^i can be calculated as

$$t_k^{exe}(i) = \sum_{e_j \in \mathcal{E}} \frac{x_k^{i,j} w_k^i}{c_k^i S_j}, \tag{2}$$

where w_k^i stands for the workload of task and c_k^i denotes the computation resource required by the task.

Transmission Time. The data transfer rate between edge server $e_{j'}$ and e_j is denoted as $D_{j',j}$. We assume $D_{j',j} = D_{j,j'}$, and $D_{j',j} = +\infty$, if $j' = j$. The data transmission time between task v_k^i and its predecessor task $v_k^{i'}$ can be expressed as:

$$t_k^{trans}(i', i) = \sum_{e_{j'} \in \mathcal{E}} \sum_{e_j \in \mathcal{E}} \frac{x_k^{i',j'} x_k^{i,j} d_k^{i',i}}{D_{j',j}}, i' \in pre_k(i), \tag{3}$$

where $d_k^{i,i'}$ denotes the data size transferred from task v_k^i to $v_k^{i'}$. The retraining data sample transmission time is:

$$t_k^{trans}(0, i) = \sum_{e_j \in \mathcal{E}} \frac{x_k^{i,j} d_k^{0,i}}{D_j}, \tag{4}$$

where D_j denotes the data transfer rate between edge server e_j and the storage cluster.

Completion time. We define $T_k^s(i)$ and $T_k^f(i)$ as the start time and the finish time of the task v_k^i, respectively. We have

$$T_k^s(0) = T_k^f(0) = T_k^a. \tag{5}$$

$$T_k^f(i) = T_k^s(i) + t_k^{exe}(i). \tag{6}$$

Considering the dependencies between tasks, the start time of task v_k^i should satisfy

$$T_k^s(i) \geq T_k^f(i') + t_k^{trans}(i', i), \forall i' \in pre_k(i). \tag{7}$$

Resource Constraint. For tasks scheduled to the same edge server, resource limit constraints need to be ensured. We use $g_j(t)$ to record the set of tasks that are processed in of edge server e_j at each time t. When $x_k^{i,j} = 1$ and $T_k^s(i) \leq t \leq T_k^f(i)$, $v_k^i \in g_j(t)$. Hence, we have the following constraint:

$$\sum_{v_k^i \in g_j(t)} c_k^i \leq C_j. \tag{8}$$

D. Problem Formulation

Our objective is to minimize the completion time of inference requests and maximize the inference accuracy while satisfying the DAG dependencies and edge resource constraints. The online problem for DAG request scheduling with retraining can be formulated as follows: [1]

$$\min \quad P(X, Y) = \sum_k \big(\max_{v_k^i \in \mathcal{V}_k} (T_k^f(i)) + \theta \sum_{v_k^i \in \mathcal{V}_k} (1 - A_k^i) \big) \tag{9}$$

$$\text{s.t.} \quad \sum_{e_j \in \mathcal{E}} x_k^{i,j} = 1, \ \forall i, \forall k, \tag{9a}$$

$$\sum_{u_h \in \mathcal{U}} y_k^{i,h} \leq 1, \ \forall i \leq |V|, \forall k, \tag{9b}$$

$$y_k^{i,h} = 0, \ \forall i > |V|, \forall k, \forall h, \tag{9c}$$

$$A_k^i \geq A_{\min}, \ \forall i, \forall k, \tag{9d}$$

$$x_k^{i,j} \in \{0, 1\}, \ \forall i, \forall j, \forall k, \tag{9e}$$

$$y_k^{i,h} \in \{0, 1\}, \ \forall i, \forall h, \forall k, \tag{9f}$$

$$(6), (7), (8)$$

Constraint (9a) guarantees that each task of each request should be offloaded on only one edge server. Constraint (9b) means that at most one configuration is selected for retraining the model called by each task. Constraint (9c) guarantees that nested retraining tasks are prohibited. Constraint (9d) requires that the accuracy of models used for inference in each request must be higher than the threshold $A_{\min}$.

Challenge. [9] has already proven that scheduling a DAG request on heterogeneous servers while considering inter-server communication time is NP-hard, which is a special case of our problem. Therefore, Problem (9) is NP-hard. Furthermore, Problem (9) is more complex. It not only involves online scheduling of multiple DAG requests but also considers whether to add retraining tasks within the DAG, along with their configurations.

IV. Design of Online Scheduling Algorithm

To start, we consider the case where there is a single request. We propose the single request scheduling algorithm to tackle the problem. Then, expanding on the single request algorithm, we have devised an algorithm that can effectively handle multiple requests.

A. List Scheduling

We first design a list scheduling algorithm to determine the assignment of each task in the request for offloading. This algorithm selects the highest-priority task and assigns it to the "best" edge server, repeating this process until all tasks in the request are offloaded.

We define the priority of the task v^i as

$$\phi(i) = \max_{i' \in suc(i)} \{ \bar{t}^{exe}(i) + t_{\max}^{trans}(i, i') + \phi(i') \}, \tag{10}$$

where $suc(i)$ represents the successor tasks connected to task v^i. $\bar{t}^{exe}(i)$ is the average execution time of the task v^i on all edge servers, and $t_{\max}^{trans}(i, i')$ is the maximum data transmission time between task v^i and its successor task $v^{i'}$, which is calculated by the lowest transfer rate on all edge servers.

Lemma 1. *The decreasing order of priority provides a topological ordering of tasks in the task DAG.*

Proof. See details in Appendix A. □

Algorithm Details. The list scheduling algorithm is presented in Alg. 1. Alg. 1 first calculates the priority of each task according to the Eq. (10) (line 1). Then, it assigns tasks to their "best" edge server based on the decreasing order of priorities (lines 2-9). For the task with the highest priority (line 4), it calculates the finish time of the task on every edge server (line 5) and selects the edge server with minimum finish time as the "best" edge server to offload (line 6-8).

Algorithm 1 List Scheduling Algorithm

```
1: Calculate the priorities of all tasks in $\mathcal{V}$ with Eq.(10);
2: $q \leftarrow$ scheduling list of tasks following decreasing order of priorities;
3: while $q \neq \emptyset$ do
4:     Select the first task $v^i$ in the list $q$;
5:     Calculate $T^f(i)$ on each edge server by Eq.(5)-(7);
6:     Find the edge server $e_j \in \mathcal{E}$ that satisfies $\min_X T^f(i)$;
7:     Assign task $v^i$ to edge server $e_j$, $x^{i,j} = 1$;
8:     Delete task $v^i$ from list $q$;
9: end while;
10: return $X$;
```

[1]We use parameter θ to model the time sensitivity of users. Specifically, for every additional θ unit of completion time extended, there is an expected increase of one unit in the inference accuracy from users.

B. Algorithm for Single Request

While the list scheduling algorithm is effective in handling DAG request scheduling problems, it is insufficient for making decisions related to model retraining. We design the single request scheduling algorithm (*SRS*) in Alg. 2, which balances the trade-off between the latency introduced by model retraining and the improvement in accuracy, and selects the optimal retraining configuration for each model.

Algorithm Details. Alg. 2 first schedules the initial task DAG, which only handles inference tasks (line 1). In this way, it can pre-evaluate the resource utilization and latency of inference tasks in the request, thereby avoiding the overload of retraining tasks. For easy calculation, the initial DAG structure is converted into a single-chain DAG (line 2). Given the existing resource constraints, the algorithm selects the optimal retraining configuration for each inference task's model, aiming to minimize the change value ΔP in the objective function after retraining (lines 3-6). The change value ΔP with configuration $u_h \in \mathcal{U}$ can be calculated as:

$$\begin{aligned}\Delta P(i,h) &= \frac{n_h}{o_h S_{j*}} + \frac{d^{i',i} + d(m^i)}{D_{j'*,j*}} + \frac{b_h}{D_{j*}} \\ &\quad + \theta(1 - A^i) - \theta(1 - a^i) \\ &= \frac{n_h}{o_h S_{j*}} + \frac{d^{i',i} + d(m^i)}{D_{j'*,j*}} + \frac{b_h}{D_{j*}} \\ &\quad + \theta(a^i - f(u_h, m^i)), \end{aligned} \tag{11}$$

where edge server e_{j*} is the available edge server with the average computing resources, b_h represents the data size of retraining samples, and n_h denotes the retraining workload. Finally, the algorithm invokes Alg. 1 once again to make scheduling decisions for the extended DAG with retraining task nodes (line 7).

Algorithm 2 Single Request Scheduling Algorithm (*SRS*)

```
1: Invoke Algorithm 1 to schedule initial inference tasks;
2: Convert the initial DAG into a single chain DAG in topological order;
3: for each $v^i \in \mathcal{V}$ do
4:     Find the retraining configuration $u_h \in \mathcal{U}$ that satisfies $\min \Delta P(i)$ with Eq. (11)
5:     Add the retraining node to the DAG with configuration $u_h$, $y^{i,h} = 1$
6: end for
7: Invoke Algorithm 1 to schedule extended DAG to get $X$;
8: return $X, Y$;
```

Theoretical Analysis. Here, we analyze the gap between the solution obtained by our algorithm and the optimal solution for the single request scheduling with model retraining problem, which are denoted as M_{alg} and M_{opt}, respectively.

Theorem 1. *The solution M_{alg} achieved by our algorithm satisfies*

$$M_{alg} \leq \frac{s_{\max}}{s_{\min}} M_{opt} + \sum_{i=1}^{|V|-1} t_{\max}^{trans} + \theta \sum_{i=1}^{|V|} (1 - a^i), \tag{12}$$

where $s_{\max}$ and $s_{\min}$ denote the maximum and minimum computational speed by per computation unit on edge servers, respectively.

Proof. See details in Appendix B. □

C. Algorithm for Multiple Requests

SRS is not efficient when handling online multiple requests, particularly regarding task scheduling. The main reason is that multiple tasks should be processed in parallel while satisfying resource constraints, while *SRS* only schedules one request at a time. In this case, we propose the multiple request scheduling algorithm (*MRS*) for a series of requests arriving online. Similar to the list scheduling approach, *MRS* determines the scheduling order by considering the priority of tasks and selects the currently optimal edge server to offload each task.

Algorithm Details. For each request r_k, the model retraining decisions Y_k are obtained by invoking Alg. 2 (line 3). The algorithm maintains a task list q_k where tasks are sorted in descending order of priority (lines 4-5). In the set of task lists, the algorithm first chooses the first task of each list (line 9). Next, the algorithm selects the task from the candidate tasks that can start the earliest for scheduling (line 10). Lines 11-13 select the best edge server with minimum finish time of the task to assign. Once the task is assigned, it is removed from the list (line 14). The scheduling for that request is considered complete when the list is empty (lines 15-16).

Algorithm 3 Multiple Requests Scheduling Algorithm (*MRS*)

```
1: Initialize set $Q \leftarrow \emptyset$;
2: if new request $r_k$ arrives then
3:     Invoke Algorithm 2 to get $Y_k$;
4:     Calculate the priorities of all tasks in $\mathcal{V}_k$ with Eq.(10);
5:     $q_k \leftarrow$ scheduling list of tasks following decreasing order of priorities;
6:     $Q \leftarrow Q \cup \{q_k\}$;
7: end if
8: while $Q \neq \emptyset$ do
9:     Choose the first task of each list $q_k$ to construct set $H$;
10:    Select the earliest task $v_k^i$ that can be processed from set $H$;
11:    Calculate $T_k^f(i)$ on each edge server by Eq.(5)-(7);
12:    Find the edge server $e_j \in \mathcal{E}$ that satisfies $\min_X T_k^f(i)$;
13:    Assign task $v_k^i$ to edge server $e_j$, $x_k^{i,j} = 1$;
14:    Delete task $v_k^i$ from list $q_k$;
15:    if $\exists q_k \in Q$ and $q_k = \emptyset$ then
16:        Delete list $q_k$ from set $Q$;
17:    end if
18: end while;
19: return $X, Y$;
```

V. Performance Evaluation

A. Evaluation Settings

Testbed Setup. We constructed an edge computing environment using containers, comprising five edge servers, a storage cluster, and a controller. Each edge server is equipped with two NVIDIA A100 GPUs, each with 80GB of memory. The GPU computational speed of the edge servers ranges between [9.7, 19.5] teraflops [10]. Similar to [11], we use the Multi-Process Service (MPS) [12] to partition GPU resources for tasks in requests. The controller and the edge servers have the same GPU configuration. The storage cluster consists of two object storage servers (OSS), each with 256GB of storage space and one Nvidia GTX 2060 GPU [13]. The data transfer rates between edge servers, between edge servers and the controller are $[3, 5]$ Gbps, and between edge servers and the storage cluster are $[7, 10]$ Gbps [14].

Request Arriving. Similar to [9], the application requests arrive following the Alibaba trace [15], which indicates the real-world inference request workload. We consider requests of the same type in the trace as requests from the same application. These requests are then scaled in arrival time according to appropriate proportions to align with the total runtime. The data size of inference tasks is within $[0.8, 1.6]$ MB, while the workload is set from 0.1 to 0.5 GPU times.

Applications and Models. By default in the experiment, we evaluated five applications: life logging, image processing, video monitoring, social media, and TF cascade. For the lifelogging application, we utilize TinyYolo [16], InceptionV3 [17], MobileNetV3 [18], ShuffleNet [19], and EfficientNet [20] models to accomplish tasks in this DAG application. The dataset is the Adience image dataset [21], which contains 26580 photos and 2284 subjects. The 40% of the dataset was used to pre-train the models. The models and datasets of the other four applications are from [22]. For the experiment with different numbers of applications, we obtained the additional applications and datasets from [8]. In the experiment, requests from the same application share the same DAG structure.

Offline Profiling. To establish the correspondence between different retraining configurations and the post-accuracy achieved by model training, denoted by the function $f()$, we conducted offline profiling of the models as shown in Sec. (II-B). We randomly sampled from the test dataset to obtain data samples for retraining. Then, we retrained the models under various training hyperparameters, including epochs, batch size, number of samples, frozen layers, resources, etc., and obtained the post-accuracy of models. Given that $f()$ may vary in a dynamic environment, it is advisable to periodically run offline profiling to update it to mitigate errors resulting from environmental changes.

Baselines. To evaluate the performance of our algorithm, we compare it with the following three baselines.

- *AdaInf* [8]: *AdaInf* employs incremental retraining and considers the SLO of requests. It divides GPU time between retraining and inference to meet the SLO and then assigns GPU time to retraining tasks based on their impact levels for each request.
- *Ekya* [7]: *Ekya* is a heuristic algorithm that makes inference and retraining scheduling jointly. It employs a greedy strategy to select the retraining configuration and corresponding resource allocation scheme that maximizes inference accuracy.
- *PASS* [23]: *PASS* is a priority-based DAG scheduling algorithm that computes priorities for each task and selects edge servers for tasks in priority order. As *PASS* does not involve model retraining, we have configured it to periodically retrain all models to achieve the highest accuracy.

B. Evaluation Results

As *MRS* is an extension of *SRS* tailored for multi-request scenarios, *MRS* leverages *SRS* to make decisions on retraining configurations, with both algorithms sharing a common core scheduling philosophy. The primary objective of our experiments is to evaluate the effectiveness of *MRS* in online multi-request environments.

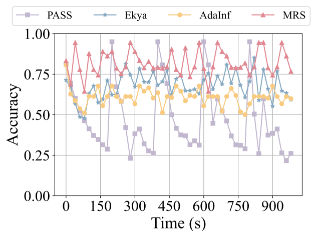

Fig. 8: Average inference accuracy over time.

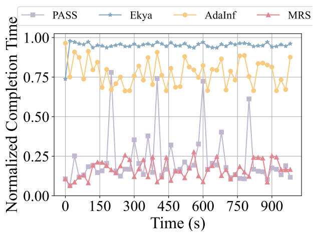

Fig. 9: Normalized completion time over time.

Accuracy Over Time. Fig. 8 illustrates the average inference accuracy of requests under different algorithms. It is evident from the graph that our algorithm can sustain an accuracy of around 85%, surpassing the other three algorithms. *AdaInf* achieves an average inference accuracy of around 60%. This is attributed to its utilization of idle time, apart from inference time, to incrementally retrain models, resulting in shorter training duration and limited improvements in model inference accuracy. *Ekya* performs slightly better than *AdaInf* but still lags behind our algorithm by approximately 15%. This is because *Ekya* sequentially selects the optimal retraining configurations and resource allocations for each inference model, even those that do not require accuracy improvements. This approach reduces available resources for models that truly need retraining, preventing the selection of higher end-accuracy retraining configurations. As a result, this diminishes overall inference accuracy and prolongs request completion times. As for *PASS*, it conducts regular retraining for all models. Consequently, model accuracy gradually declines until the models are retrained.

Completion Time Over Time. Fig. 9 displays the completion time of requests under different algorithms. For ease of representation, we have normalized the time data. It shows that our algorithm consistently achieves the fastest request completion time for the majority of cases. *Ekya*, due to its excessive retraining tasks, exhibits the longest completion times, with *AdaInf* slightly outperforming it. Our algorithm is more than 45% faster than these two algorithms. For *PASS* with periodic retraining, during non-retraining periods, the completion time for requests may be faster than our algorithm, which schedules retraining tasks as well. However, during retraining periods, the completion time for requests is significantly delayed.

Impact of Application Numbers. As the number of applications increases, Fig. 10 displays a decrease in inference accuracy, and Fig. 11 demonstrates a continuous increase in completion time. This is because as the number of applications grows, the volume of requests within the system also increases, leading to a gradual reduction in idle resources on the edge servers. When GPU resources become insufficient, the wait

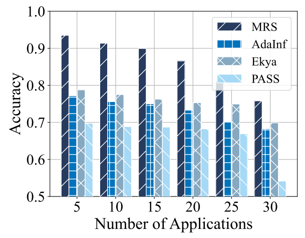

Fig. 10: Average inference accuracy of different number of applications.

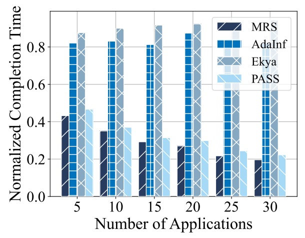

Fig. 11: Normalized completion time of different number of applications.

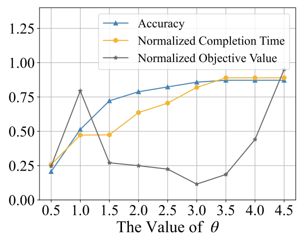

Fig. 14: Accuracy, completion time and objective value of different value of θ.

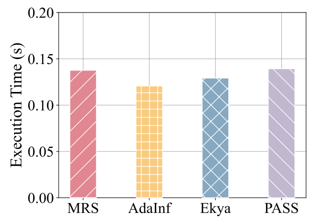

Fig. 15: Execution time of different algorithms.

time for tasks in requests increases, consequently prolonging the requests completion time. Insufficient resources for retraining can result in a reduced frequency of retraining or the selection of configurations that require fewer resources, leading to a decrease in the model's inference accuracy. Compared to the other three algorithms, our algorithm excels at achieving the highest accuracy in the shortest time when fulfilling application requests, even when the number of applications is high.

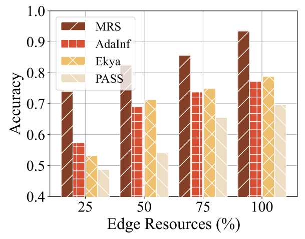

Fig. 12: Average inference accuracy of different edge resources.

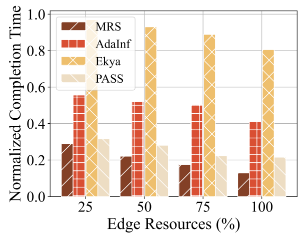

Fig. 13: Normalized completion time of different edge resources.

Impact of Edge Resources. We conducted experiments on algorithms using different computing resource capacities of edge servers. Fig. 12 indicates that as edge computing resources increase, the model's inference accuracy also improves, with *MRS* consistently achieving the highest accuracy. This is because with a fixed inference workload, the greater the computing resource capacity of the edge servers, the more resources are available for retraining, leading to an improvement in model accuracy. *MRS* can flexibly choose retraining configurations, allowing it to maintain high model accuracy even with smaller resource capacities. Fig. 13 illustrates the impact of edge computing resources on request completion time. When edge servers have larger resource capacities, requests scheduled by the algorithms can be completed earlier, with *MRS* achieving the shortest completion time.

Impact of The Value of θ. Here we investigate the impact of the value of θ on accuracy and completion time. θ is used to model user time sensitivity, indicating that users prefer to trade θ times the unit completion time for one unit of accuracy. In other words, the larger the value of θ, the more inclined users are to sacrifice completion time for accuracy. From the Fig. 14, we can observe that as the value of θ increases, the model's accuracy rises. Simultaneously, the completion time also increases. This is because, to enhance accuracy, the algorithm conducts more frequent model retraining sessions and utilizes more time-consuming configurations, thereby delaying the completion of requests. Since our goal is to minimize request completion time and maximize inference accuracy, for our system, the more suitable theta parameter value is 3. It can strike a balance between accuracy and completion time, ultimately minimizing the objective value. Certainly, the value of θ can also be adjusted based on user requirements.

Execution Time. Fig. 15 illustrates the execution time of different algorithms within 100 time slots, where all four algorithms exhibit similar time. Taking our algorithm as an example, the execution time of the algorithm within each time slot is 0.013 seconds. With each time slot set at 2 seconds, the algorithm's runtime occupies 0.65% of the entire time slot. This indicates that our algorithm can swiftly schedule user requests and achieve the goal of completing requests with high accuracy in a short amount of time.

VI. Related Work

A. Scheduling of Dependent Tasks

Many studies focus on the scheduling of tasks with dependencies in edge computing [9], [23]–[29]. Chen *et al.* [24] proposed the LPSched algorithm, which took a linear programming approach to schedule the jobs with precedence and resource constraints. Zeng *et al.* [26] structured the deployment of linked microservices as a quadratic integer problem, and then proposed a randomized rounding algorithm to solve it. Zhao *et al.* [25] investigated the issue of offloading interdependent tasks with service caching, introducing the CP algorithm based on convex programming to address it. Li *et al.* [23] explored the scheduling of dependency-aware serverless functions on edge servers, considering the priority of each function. Li *et al.* [27] investigated the offloading of dependency tasks involving encryption and decryption operations on edge servers, and proposed a customized list scheduling algorithm to minimize the makespan.

Nevertheless, these studies are founded on predetermined task dependencies. To mitigate accuracy loss caused by data drift, model retraining becomes crucial. Yet, the addition of retraining tasks may alter the DAG structure, a situation for which these algorithms are inadequately prepared.

B. Model Retraining in Edges

Model retraining stands out as a promising strategy to tackle data drift. Zhang *et al.* [30] sampled drift data for retraining to maintain the model accuracy in edges and reduce costs. Tian *et al.* [6] provided two policies to decide when to update models to cope with drift data. Aleksandrova *et al.* [31] updated models based on the Optimal Stopping Theory (OST) principles to minimize the network overhead. Chen *et al.* [32] updated models from cloud server to edges aimed to minimize the transferring of data. Additionally, some papers focus on the scheduling of retraining, especially in conjunction with inference scheduling. Bhattacharjee *et al.* [33] explored the scheduling of model retraining and inference colocation. But the inference tasks were only treated as known background processes without resource allocation in [33]. Ekya [7] utilized heuristic algorithms to balance the accuracy of inference and retraining, allocating limited edge computing resources. AdaInf [8] employed incremental retraining on edge servers to enhance model accuracy, and allocated GPU time for retraining based on the model's drift impact while meeting the SLO of inference.

However, in these studies, the timing and scheduling of model retraining are suboptimal. Ekya conducts model retraining for every inference task, even for models unaffected by data drift, leading to resource wastage. Conversely, AdaInf utilizes the remaining time of the inference SLO for retraining, providing only a limited accuracy boost. And their offline algorithms are not suitable for the online multi-requests edge environment.

VII. CONCLUSION

This paper studies the online scheduling of multiple-model inference with DAG structure and retraining in edge computing. To minimize the completion time of inference requests while maximizing the inference accuracy, we propose to add the retraining nodes to the original DAG structure of the request. We then design a single request scheduling algorithm with a performance guarantee, which selects optimal retraining configurations under resource constraints, and offloads retraining and inference tasks together. We further design a multiple requests scheduling algorithm to address incoming online application requests. Experiments on the edge system show that our algorithm outperforms three baselines in inference accuracy and request completion time.

APPENDIX

A. Proof of Lemma 1

From the definition of task priority in Eq. (10), for each edge $(i,i') \in \mathcal{L}_k$ in the request DAG $G_k = (\mathcal{V}_k, \mathcal{L}_k)$, it always satisfies $\phi(i) \geq \phi(i')$. This indicates that task i will be scheduled before its successor task i', ensuring the topological order of the DAG. $\square$

B. Proof of Theorem 1

Lemma 2. *In single request scheduling algorithm, the solution M_{alg} satisfies*

$$M_{alg} \leq \sum_{i=1}^{|V|}(\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i)) + \sum_{i=1}^{|V|-1} t_{\max}^{trans} + \theta \sum_{i=1}^{|V|}(1-a^i), \quad (13)$$

where $t_{\max}^{trans}$ is the maximum data transmission time in graph G, $t_{\min}^{exe}(i)$ is the minimum execution time of task v^i on the edge servers. $s_{\max}$ and $s_{\min}$ denote the maximum and minimum computational speed by per computation unit on edge servers, respectively. a^i is the initial inference accuracy of task v^i .

Proof. T_{alg} denotes the request completion time of our algorithm. For the extended request DAG with retraining nodes, it has been proved that we can always extract a task chain $\mathcal{V}: v_1 \to v_2 \to ... \to v_{2|V|}$, whose completion time is equal to T_{alg} [27].

The completion time of chain $\mathcal{V}$ consists of two parts: task execution time $t_{exe}(\mathcal{V})$ and data transmission time $t_{trans}(\mathcal{V})$. For the data transmission time $t_{trans}(\mathcal{V})$, it is obvious that

$$\begin{aligned} t_{trans}(\mathcal{V}) &= t^{trans}(1,2) + ... + t^{trans}(2|V|-1, 2|V|) \\ &\leq \sum_{i=1}^{2|V|-1} t_{\max}^{trans}. \end{aligned} \quad (14)$$

And the task execution time $t_{exe}(\mathcal{V})$ satisfies

$$\begin{aligned} t_{exe}(\mathcal{V}) &= t^{exe}(1) + t^{exe}(2) + ... + t^{exe}(|V|) \\ &\leq \sum_{i=1}^{2|V|}(\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i)). \end{aligned} \quad (15)$$

By (14) and (15), we can prove that

$$T_{alg} \leq \sum_{i=1}^{2|V|}(\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i)) + \sum_{i=1}^{2|V|-1} t_{\max}^{trans}, \quad (16)$$

Considering the change value ΔP in Eq. (11), each added retraining node $v^{i+|V|}$ and its retrained accuracy A_{alg}^i in the request DAG satisfy that

$$\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i+|V|) + t_{\max}^{trans} + \theta(a^i - A_{alg}^i) \leq 0, \quad (17)$$

By (16) and (17), we can prove that

$$\begin{aligned} M_{alg} &= T_{alg} + \theta \sum_{v^i \in \mathcal{V}} (1 - A_{alg}^i) \\ &\leq \sum_{i=1}^{|V|}(\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i)) + \sum_{i=1}^{|V|-1} t_{\max}^{trans} + \theta \sum_{i=1}^{|V|}(1-a^i) \end{aligned} \quad (18)$$

$\square$

Previous study [34] proved that the optimal completion time for any DAG is longer than the completion time of any chain extracted from the DAG. Therefore, we have

$$T_{opt} \geq \sum_{i=1}^{|V|} t_{\min}^{exe}(i). \quad (19)$$

Then, based on the inference accuracy updating in Eq. (1) and (19), we have

$$M_{opt} = T_{opt} + \theta \sum_{v^i \in \mathcal{V}} (1 - A_{opt}^i) \geq \sum_{i=1}^{|V|} t_{\min}^{exe}(i) \quad (20)$$

According to (20) and Lemma 2, we can derive that

$$\begin{aligned} M_{alg} &\leq \sum_{i=1}^{|V|}(\frac{s_{\max}}{s_{\min}} t_{\min}^{exe}(i)) + \sum_{i=1}^{|V|-1} t_{\max}^{trans} + \theta \sum_{i=1}^{|V|}(1-a^i) \\ &\leq \frac{s_{\max}}{s_{\min}} M_{opt} + \sum_{i=1}^{|V|-1} t_{\max}^{trans} + \theta \sum_{i=1}^{|V|}(1-a^i). \end{aligned} \quad (21)$$

$\square$

References

[1] A. Mathur, N. D. Lane, S. Bhattacharya, A. Boran, C. Forlivesi, and F. Kawsar, "Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware," in *Proc. of MobiSys*, 2017, p. 68–81.

[2] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *Proc. of HPCA*, 2019, pp. 331–344.

[3] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a gpu cluster engine for accelerating dnn-based video analysis," in *Proc. of SOSP*, 2019, p. 322–337.

[4] X. Dai, P. Yang, X. Zhang, Z. Dai, and L. Yu, "Respire: Reducing spatial–temporal redundancy for efficient edge-based industrial video analytics," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9324–9334, 2022.

[5] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Networks*, vol. 116, no. C, p. 56–73, 2019.

[6] H. Tian, M. Yu, and W. Wang, "Continuum: A platform for cost-aware, low-latency continual learning," in *Proc. of SoCC*, 2018.

[7] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *Proc. of NSDI*, 2022, pp. 119–135.

[8] S. S. Shubha and H. Shen, "Adainf: Data drift adaptive scheduling for accurate and slo-guaranteed multiple-model inference serving at edge servers," in *Proc. of SIGCOMM*, 2023, p. 473–485.

[9] G. Li, H. Tan, L. Liu, H. Zhou, S. H.-C. Jiang, Z. Han, X.-Y. Li, and G. Chen, "Dag scheduling in mobile edge computing," *ACM Transactions on Sensor Networks*, vol. 20, no. 1, 2023.

[10] NVIDIA, *NVIDIA A100 TENSOR CORE GPU*, 2021, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf.

[11] Z. Han, R. Zhou, C. Xu, Y. Zeng, and R. Zhang, "Inss: An intelligent scheduling orchestrator for multi-gpu inference with spatio-temporal sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, pp. 1–13, 2024.

[12] NVIDIA, *NVIDIA Multi-Process Service (MPS)*, 2020, https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf.

[13] P. Hamandawana, A. Khan, J. Kim, and T.-S. Chung, "Accelerating ml/dl applications with hierarchical caching on deduplication storage clusters," *IEEE Transactions on Big Data*, vol. 8, no. 6, pp. 1622–1636, 2022.

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

[15] Alibaba production cluster trace data. [Online]. Available: Available:https://github.com/alibaba/clusterdata

[16] P. Adarsh, P. Rathi, and M. Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *Proc. of ICACCS*, 2020, pp. 687–694.

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. of CVPR*, 2016, pp. 2818–2826.

[18] A. G. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[19] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. of CVPR*, 2018, pp. 6848–6856.

[20] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proc. of ICML*, 2019, pp. 6105–6114.

[21] E. Eidinger, R. Enbar, and T. Hassner, "Age and gender estimation of unfiltered faces," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 12, pp. 2170–2179, 2014.

[22] D. Crankshaw, G.-E. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, and A. Tumanov, "Inferline: latency-aware provisioning and scaling for prediction serving pipelines," in *Proc. of SoCC*, 2020, p. 477–491.

[23] Y. Li, D. Zeng, L. Gu, K. Wang, and S. Guo, "On the joint optimization of function assignment and communication scheduling toward performance efficient serverless edge computing," in *Proc. of IWQoS*, 2022, pp. 1–9.

[24] C. Chen, X. Ke, T. Zeyl, K. Du, S. Sanjabi, S. Bergsma, R. Pournaghi, and C. Chen, "Minimum makespan workflow scheduling for malleable jobs with precedence constraints and lifetime resource demands," in *Proc. of ICDCS*, 2019, pp. 2068–2078.

[25] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. of INFOCOM*, 2020, pp. 1997–2006.

[26] D. Zeng, H. Geng, L. Gu, and Z. Li, "Layered structure aware dependent microservice placement toward cost efficient edge clouds," in *Proc. of INFOCOM*, 2023, pp. 1–9.

[27] Y. Li and D. Zeng, "Dependency-aware task scheduling in trustzone empowered edge clouds for makespan minimization," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, pp. 423–434, 2023.

[28] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. of INFOCOM*, 2018, pp. 46–54.

[29] J. Liang, K. Li, C. Liu, and K. Li, "Joint offloading and scheduling decisions for dag applications in mobile edge computing," *Neurocomputing*, vol. 424, pp. 160–171, 2021.

[30] L. Zhang, G. Gao, and H. Zhang, "Towards data-efficient continuous learning for edge video analytics via smart caching," in *Proc. of SenSys*, 2023.

[31] E. Aleksandrova, C. Anagnostopoulos, and K. Kolomvatsos, "Machine learning model updates in edge computing: An optimal stopping theory approach," in *Proc. of ISPDC*, 2019.

[32] B. Chen, A. Bakhshi, G. E. A. P. A. Batista, B. Ng, and T.-J. Chin, "Update compression for deep neural networks on the edge," *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3075–3085, 2022.

[33] A. Bhattacharjee, A. D. Chhokra, H. Sun, S. Shekhar, A. Gokhale, G. Karsai, and A. Dubey, "Deep-edge: An efficient framework for deep learning model update on heterogeneous edge," in *Proc. of ICFEC*, 2020.

[34] E. Bampis and A. Kononov, "Bicriteria approximation algorithms for scheduling problems with communications," in *Proc. of SPAA*, 2003, p. 252–253.