# Orchestrating Communication and Computation Efficiency in Edge-Based Knowledge Distillation

Gaoyun Lin[†], Wanglei Feng[†], Shenglan Luo[†], Bin Qian[‡], Zhenyu Wen[†], Cong Wang[*]
[†]Zhejiang University of Technology, Hangzhou, China
[‡]Newcastle University, UK
[*]Zhejiang University, Hangzhou, China
E-mail: {lingaoyun_zj, zjutlsl}@163.com, {211122030077, zhenyuwen}@zjut.edu.cn, b.qian3@ncl.ac.uk, cwang85@zju.edu.cn

***Abstract*—With the rapid development of edge computing, knowledge distillation is an effective method to overcome the challenges of training high-performance models with limited computational resources on the edge side. However, a naive implementation of knowledge distillation could cause resource contention on edge devices, specifically in the communication process of receiving distilled "dark knowledge" while simultaneously performing knowledge transfer, both competing for the limited resources. This paper delves into the system-level challenges of resource contention in edge-based knowledge distillation. In particular, we focus on the resource competition caused by the communication and computation processes. To address this issue, we first formulate the problem of edge-based knowledge distillation. Then we propose a method to orchestrate resources between computation and communication in an approximately optimal manner by employing a dynamic resource allocation strategy. Through simulation experiments, we demonstrate the effectiveness of the proposed scheme via evaluating against random and priority-based resource allocation strategies.**

***Index Terms*—Edge Computing, Knowledge Distillation, Resource Allocation, Communication Efficiency.**

## I. INTRODUCTION

Model personalization is important customization for heterogeneous edge devices to solve local tasks [1]. However, it comes with its own challenges such as model drift, when model performance degrades due to variations in the local environment [2].

Although large-scale models adapt better to environmental shifts, their computational demands often exceed the resources available on edge devices [3]. Federated learning is a collaborative approach that mitigates model drift by training on local data and aggregating updates across multiple devices [4, 5]. However, it requires extensive communication resources and consistent client engagement, which can be problematic in edge settings with resource constraints, pose challenges for model quality and security [6, 7]. Knowledge distillation (KD) [8], as an effective method for model compression, presents a viable solution to these problems. KD involves transferring knowledge from comprehensive, complex models into streamlined, edge-adapted models, thus retaining the sophisticated model performance within device computational capacity, which offers a competitive solution to model personalization and potential model drift.

However, KD also comes with a new set of challenges, particularly in the form of resource contention between computation and communication in resource-limited edge environments [9–11]. In edge environments where resources are limited, this contention becomes more prominent as edge devices with compact models continuously distill knowledge from over-parameterized models on the server, which necessitates a fraction of computational resources to be allocated while the local training is in progress. This contention leads to resource bottlenecks with adverse impact on KD: slower communication speeds due to competition for data transmission, or delayed computation as processing power is overstretched.

In this paper, we first conduct a detailed problem formulation for KD in the context of edge computing. We particularly highlight the unique KD dependencies and resource contention challenges. To this end, we introduce a novel methodology to resolve the contention between communication and computation which specifically tailored for edge side. Our approach focuses on optimizing resource allocation, particularly between communication and computation during distillation. Through dynamic resource management, we aim to enhance the operational efficiency of edge models.

The rest of this paper is structured as follows. In Section II, we provide a background description of the process of edge distillation. In Section III, we model the resource competition situation and formulate the problem for the training process, and in Section IV, we propose an approximate optimal solution based on dynamic resource allocation framework to schedule the competition. In Section V, we show the simulation experiment and evaluation results, followed by conclusion in Section VI.

## II. BACKGROUND

### A. Logits-based Knowledge Distillation

The general knowledge distillation process involves both a teacher model ($\mathcal{T}$) and a student model ($\mathcal{S}$). During training, the student model $\mathcal{S}$ receives logits from the teacher model $\mathcal{T}$, which encapsulates the knowledge of the teacher model, aiding in accelerating the local training progress of $\mathcal{S}$.

The training of the student model encompasses a forward propagation phase for computing loss and a backward propa-

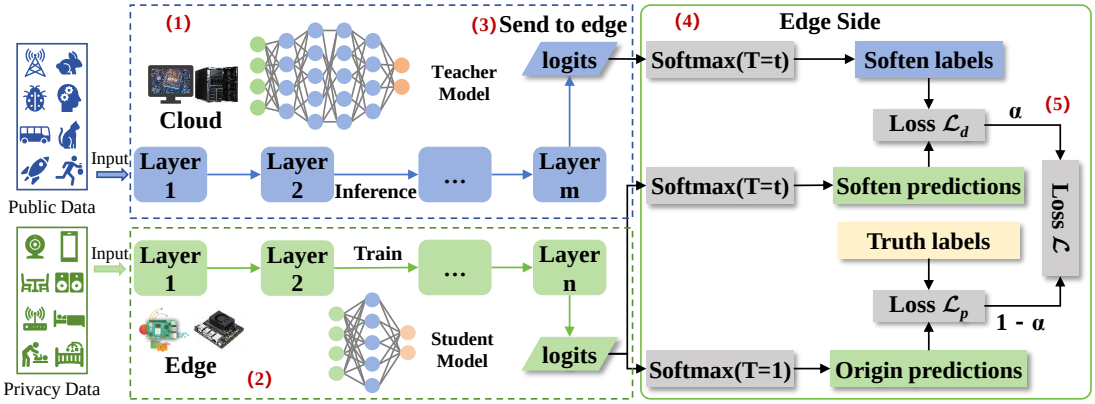



Fig. 1. Edge-based knowledge distillation process. The teacher model, deployed on the `Cloud`, performs model inference using public data and transmits the resulting logits to the `Edge`. Meanwhile, the student model, residing on the `Edge`, undergoes training using private data. At the `Edge`, both the logits from the teacher model and the student model are softened using a temperature parameter $T_{em}$, leading to the computation of the distillation loss $\mathcal{L}_d$. Additionally, the original predictions of the student model, along with the ground truth labels of the private data, contribute to the calculation of the prediction loss $\mathcal{L}_p$. The final loss, utilized for backward gradient calculation, is obtained by combining $\mathcal{L}_d$ and $\mathcal{L}_p$.

gation phase for gradient computation. The loss function $\mathcal{L}$ of the student model $\mathcal{S}$ is a combination of the weighted sum of its prediction loss $\mathcal{L}_p$ and its distillation loss $\mathcal{L}_d$:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_p + (1 - \alpha) \cdot \mathcal{L}_d, \tag{1}$$

where $\alpha$ is a hyperparameter that balances these two loss components.

During the backward propagation, $\mathcal{S}$ updates its parameters, denoted as $\theta_{\mathcal{S}}$, to minimize the loss $\mathcal{L}$. This involves computing the gradient of $\mathcal{L}$ with respect to $\theta_{\mathcal{S}}$ and adjusting $\theta_{\mathcal{S}}$ accordingly:

$$\theta_{\mathcal{S}} \leftarrow \theta_{\mathcal{S}} - \eta \cdot \nabla_{\theta_{\mathcal{S}}} \mathcal{L}, \tag{2}$$

where $\eta$ is the learning rate.

### B. Edge-based Knowledge Distillation System

Given the challenges associated with deploying large teacher models on edge devices and the demands for student models to train from sensitive data, we propose a collaborative approach for knowledge distillation across the edge and cloud environments.

The edge-based knowledge distillation system consists of two main entities: `Cloud` and `Edge`. Within this framework, the `Cloud` hosts a pre-trained comprehensive teacher model, which facilitates model inference using public data. Conversely, the `Edge` is equipped with an initialized student model, similar to the teacher model but streamlined, and is responsible for executing model training operations using private data at the edge. Figure 1 presents the overview of the system: (1) The teacher model utilizes public data to perform model inference and generate the final layer's outputs (i.e., logits) on the cloud side. (2) Simultaneously, the student model employs private data for training on the edge side, generating its own training logits. (3) The cloud side then transmits the teacher model's logits to the edge side. (4) On the edge side, the logits from the cloud are distilled through a temperature $T_{em}$, yielding the distilled soft label $L_{logits}$. Simultaneously, the logits from the student model undergo two processes. One process generates soft predictions $L_{dis}$ through $T_{em}$ distillation, while the other yields original predictions $p(x_{\mathcal{S}})$ via the standard *softmax* function. (5) Finally, the edge side calculates the distillation loss $\mathcal{L}_d$ based on distilled label $L_{dis}$ from the student model and distilled label $L_{logits}$ from the teacher model, while the prediction loss $\mathcal{L}_p$ is computed using $p(x_{\mathcal{S}})$ and the ground truth labels $y$. These losses are then linearly combined to obtain the final loss $\mathcal{L}$.

The transfer of the logits from $\mathcal{T}$ to $\mathcal{S}$ is influenced by several factors, particularly the CPU resource in $\mathcal{S}$ which is closely correlated with the efficiency of knowledge transfer. Thus, the CPU capabilities of $\mathcal{S}$ are a critical factor in edge knowledge distillation, to ensure an effective and efficient distillation process.

## III. Problem Formulation

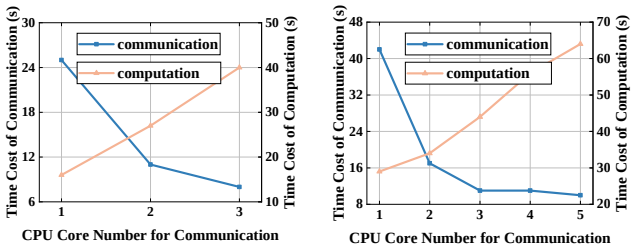
In this section, we explore resource contention in edge-based KD, focusing on the intricate balance between communication and computation processes. We further provide mathematical models to represent their time dependencies and propose strategies for optimized time management within the KD cycle.

### A. Resource Contention

Resource contention is a critical bottleneck in resource-constrained edge devices, particularly those lacking advanced processing capabilities like GPUs. Commonly used in IoT applications, these devices must manage simultaneous communication and computation demands, which often leads to resource competition and resultant inefficiencies such as delayed task processing and diminished operational efficacy [12, 13].

To better understand and address these challenges, our study focuses on the impact of CPU resource scheduling on KD tasks, which involve significant data transfer and computation between $\mathcal{T}$ and $\mathcal{S}$ models in edge environments. Our methodology includes a series of controlled experiments

conducted on Raspberry Pi and Jetson Xavier NX platforms to quantitatively assess variations in task completion time under different CPU core allocations, as depicted in Figure 2.




(a) Raspberry Pi (b) Jetson Xavier NX

Fig. 2. The Time Taken for Communication and Computation under Different CPU Core Allocations. (a) The trends of computational time for bathsize=16 training and communication time for 1MB data transfer when 1,2,3 CPU cores are assigned to communication tasks in Pi. (b) The trends of computational time for bathsize=128 training and communication time for 5MB data transfer when 1,2,3 CPU cores are assigned to communication tasks in NX.

*1) Computation:* The time required for computation tasks in edge devices depends on the allocation of CPU cores. The relationship between the required computation time and the number of CPU cores allocated for communication can be approximately fitted by the following function:

$$f(x,c) = \frac{a_1 \cdot x}{1 - a_2 \cdot c}, \tag{3}$$

Here, $f(x,c)$ is the computation time required for computational workload $x$ with $c$ CPU cores allocated to the communication task. The constants $a_1$ and $a_2$ reflect the impact of CPU allocation on computation time. In this equation, the computation time increases with the increase of the CPU cores available for the communication task. This highlights the linear decrease in computational resources as more CPU cores are allocated away from computation tasks.

*2) Communication:* The time required for communication tasks in edge devices can also be represented by an alternative formula that captures the diminishing marginal returns in terms of CPU core allocation for these tasks:

$$g(y,c) = \frac{y}{b_1 \cdot c + b_2 \cdot \log(b_3 \cdot c)} \tag{4}$$

Here, $g(y,c)$ represents the time needed for communication tasks with a data size $y$ and $c$ CPU cores allocated for communication. The term $b_1 \cdot c$ represents a linear improvement in communication speed with more cores, whereas $b_2 \cdot \log(b_3 \cdot c)$ indicates the logarithmic reduction in the rate of improvement as more cores are allocated. The constants $b_1$, $b_2$, and $b_3$ define the specific characteristics of this relationship. This formula suggests that initially, increasing the number of cores $c$ dedicated to communication significantly reduces the communication time, but after a certain point, the gains become less pronounced.

### B. Time Optimization

The iterative process of edge-based KD, as visualized in Fig 3, involves $N$ cycles of `Comm` and `Comp` tasks, each `Comp` task consists of a `Forward` and a `Backward` processes. The parallel execution of `Comm` and `Comp` tasks highlights the inherent resource contention and its impact on the speed of both tasks. The optimization of time, in this context, revolves around strategically allocating resources to minimize the total duration of these $N$ cycles.

*1) Constraints:* Given that `Comm` tasks are a continuous process, with the $\mathcal{T}$ transmitting logit data from the cloud, and `Comp` task involves both `Forward` and `Backward` process, where the `Backward` is dependent on the logits, we can formalize the time optimization problem with the following constraints:

1) `Comm` tasks are uninterrupted, with logits being transmitted from the $\mathcal{T}$ in the cloud, which requires persistent allocation of resources for data transfer until all `Comm` tasks are completed.
2) Each `Backward` in the `Comp` sequence depends on the logits from the corresponding `Comm` cycle. If $\texttt{Comm}_i$ has not completed, $\texttt{Backward}_i$ must wait, causing delays, for example, $t''$.
3) When the time exceeds $t^{end}$, it means that all `Comm` tasks have finished, and at this time all CPU resources will be allocated to the `Comp` tasks, that is, there will be only `Comp` tasks in the $t^*$ phase.

*2) Calculation of Time:* In the intricate orchestration of `Comm` and `Comp` within our system, the calculation of time emerges as a pivotal element, especially when considering the interdependencies between `Comm` and `Comp` tasks. To facilitate a thorough analysis, we adopt a phase-based approach, segmenting the timeline before each `Backward`$i$. Given that there are $N$ instances of `Backward`, we consequently identify $N+1$ distinct time segments for analysis.

Within this framework, consider the first segment, denoted as $t_1$. We let $s_1^f$ represent the computational workload of `Forward1`, and $s_1^c$ denote the load of `Comm1`. The duration of $t_1$ can be formulated as follows:

$$t_1 = \begin{cases} g(s_1^c, c) + g(s_1^{\hat{c}}, \hat{c}), & f(s_1^f, c) < g(s_1^c, c) \\ f(s_1^f, c) & \text{otherwise} \end{cases} \tag{5}$$

In this expression, $f(s_1^f, c)$ and $g(s_1^c, c)$ respectively represent the completion times for the `Forward1` and the `Comm1` processes. The term $g(s_1^{\hat{c}}, \hat{c})$ denotes the time required to complete the remaining $s_1^{\hat{c}}$ with a reallocated number of cores $\hat{c}$, following the completion of $f(s_1^f, c)$. The decision criterion for the duration of $t_1$ is based on whether the `Forward1` time is less than the `Comm1` time. Notably, if $f(s_1^f, c)$ is greater, it implies that subsequent $\texttt{Comm}_i$ (for $i > 1$) will overlap with `Forward1`.

For the segments from $t_2$ to $t_N$, we consider three scenarios. The first scenario, as exemplified by $t_2$, consists of an overlap time $t'$ and a communication time $t''$. The second scenario, illustrated by $t_3$, shows that `Comm` and `Comp` tasks are completed in parallel. The third scenario, as seen in $t_N$, occurs when there are no `Comm` tasks remaining, leading to a reallocation of core resources favoring `Comp` tasks.

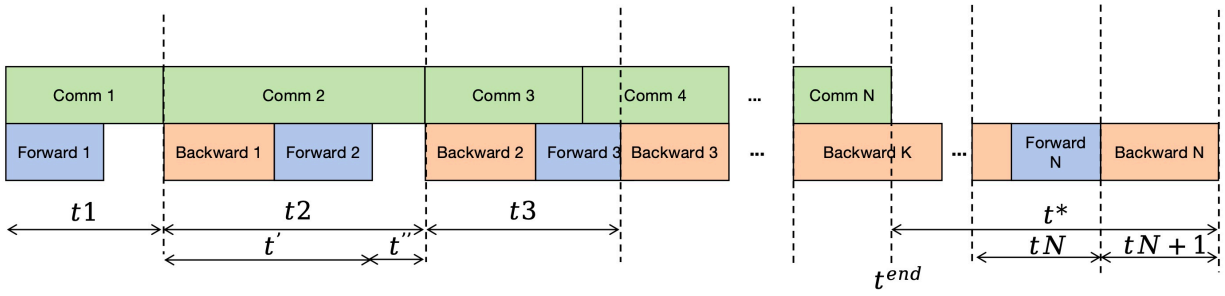



Fig. 3. `Comp` and `Comm` for edge-based KD. $N$ cycles of `Comm` and `Comp` tasks, each `Comp` task consists of a `Forward` and a `Backward` process.

**Scenario 1: Initial Overlapping.** In this scenario, the tasks `Comm` and `Comp` are initially overlapped. This is represented for a general time segment $t_i$ as follows:

$$\begin{aligned} t_i &= f(s_i^f, c) + f(s_i^b, c) + g(s_1^{\hat{c}}, \hat{c}), \\ s.t. \sum_{j=1}^{i-1} t_j + f(s_i^f, c) + f(s_i^b, c) &< \sum_{j=1}^{i-1} t_j + \sum_{k=1}^{i} g(s_k^c, c) \end{aligned} \tag{6}$$

**Scenario 2: Parallel Completion.** In this scenario, `Comm` and `Comp` are completed in parallel for segment $t_i$:

$$\begin{aligned} t_i &= f(s_i^f, c) + f(s_i^b, c), \\ s.t. \sum_{k=1}^{i} g(s_k^c, c) &< \sum_{j=1}^{i-1} t_j + f(s_i^f, c) + f(s_i^b, c) \end{aligned} \tag{7}$$

**Scenario 3: Exclusive `Comp`.** This scenario occurs when all `Comm` tasks are completed, allowing reallocation of resources solely to `Comp` for segment $t_i$:

$$t_i = f(s_i^f, c) + f(s_i^b, c), \quad s.t. \prod_{k=1}^{N} \mathbb{I}(\texttt{Comm}k \text{ completed}) \tag{8}$$

For the final time segment $t_{N+1}$, as there are no additional `Comm` tasks, the time required is exclusively dedicated to the `Backward`$N$. This can be mathematically represented as follows:

$$t_{N+1} = f(s_{N+1}^b, c) \tag{9}$$

The total computational time, $T_{total}$, representing the entire duration of the KD process across $N$ `Comm` and `Comp` tasks respectively. The optimization of $T_{total}$ hinges on an effective strategy for allocating CPU core resources. The primary goal is to minimize $T_{total}$ by optimizing core utilization for each segment, ensuring that the sum of the times from $t_1$ to $t_{N+1}$ is as short as possible. This concept can be expressed mathematically as:

$$T_{\text{total}} = \underset{c_1, c_2, \ldots, c_{N+1}}{\arg\min} \sum_{i=1}^{N+1} t_i(c_i) \tag{10}$$

In this equation, $t_i(c_i)$ denotes the time taken for segment $i$ with a given allocation of CPU cores $c_i$.

## IV. Solution

In this section, we detail our approach to solving the edge-based KD interplay between `Comm` and `Comp` tasks. Our solution employs a dynamic resource allocation-based framework to achieve an approximately optimal solution. We introduce state representation and state transition to better describe the algorithm.

### A. State Definition

We define $S_{i,j,k}$ to denote the state of having completed the $i, j, k$ rounds of `Comm`, `Forward` and `Backward` cycles, and the time of the state $S_{i,j,k}$ can be denoted as $T(S_{i,j,k})$.

### B. Goal and Constraint

The objective is formulated as:

$$\min T(S_{N,N,N}) \tag{11}$$

This objective aims to find the strategy that minimizes the final value $T(S_{N,N,N})$, thus optimizing the overall process across all $N$ cycles of `Comm`, `Forward`, and `Backward`.

In the process of model training, it is crucial to select an optimal CPU allocation strategy that minimizes the overall training time while satisfying the unique dependencies of the training process inherent to edge-based KD. This involves constraints from two core allocation models, defined by the functions $\Phi(S_{i,j,k})$ and the associated conditions:

$$c_{\texttt{Comm}}(S) + c_{\texttt{Comp}}(S) \leq c_{total}, \quad \forall S_{i,j,k} \tag{12}$$

$$\Phi(S_{i,j,k}) = \begin{cases} c_{\texttt{Comm}}(S) = c_{total},\ c_{\texttt{Comp}}(S) = 0, & i < j \\ c_{\texttt{Comm}}(S) = 0,\ c_{\texttt{Comp}}(S) = c_{total}, & i = N \\ c_{\texttt{Comm}}(S),\ c_{\texttt{Comp}}(S) = \textbf{Alloc}(S), & \text{otherwise} \end{cases} \tag{13}$$

In this function, the total number of cores allocated at any state $S_{i,j,k}$ does not exceed $c_{\text{total}}$, the total available cores. Besides, $c_{\texttt{Comm}}(S)$ and $c_{\texttt{Comp}}(S)$ represent the number of cores allocated to `Comm` and `Comp` at state $S_{i,j,k}$. The function **Alloc**$(S)$ dynamically determines the allocation based on the current state $S_{i,j,k}$, shown in Algorithm 1. The CPU allocation is strategically managed through three specific conditions to optimize system performance: **Case 1:** $i < j$ ensures all available cores are dedicated to `Comm` when `Comm` tasks

**Algorithm 1** Optimize Core Allocation

**input :** Current state $(i, j, k)$,
Total cores $c_{total}$.
**output:** Core allocation $(c_{\texttt{Comm}}(S_{i,j,k}), c_{\texttt{Comp}}(S_{i,j,k}))$
`Function` (***Alloc***$(i, j, k)$) **begin**
  Initialize minTime $\leftarrow \infty$
  **for** $c_{Comm} = 0$ **to** $c_{total}$ **do**
    $c_{\texttt{Comp}} \leftarrow c_{total} - c_{\texttt{Comm}}$
    currentTime $\leftarrow \max(g(x, c_{\texttt{Comp}}), f(y, c_{\texttt{Comm}}))$
    **if** *currentTime* $<$ *minTime* **then**
      minTime $\leftarrow$ currentTime
      optimalCores $\leftarrow (c_{\texttt{Comm}}, c_{\texttt{Comp}})$
    **end**
  **end**
  **return** *optimalCores*
**end**

**Algorithm 2** Core Allocation and Time Calculation

**Input :** Number of cycles $N$, Total cores $c_{total}$,
Workloads $x, y$, Parameters $a_1, a_2, b_1, b_2, b_3$,
Functions $f(x, c)$, $g(y, c)$.
**Output:** $T(S_{N,N,N})$.
`Function` (***Compute***$(N, c_{total}, x, y, a_1, a_2, b_1, b_2, b_3)$) **begin**
  $T(S_{0,0,0}) \leftarrow 0$,
  **while** $j < N$ **do**
    $(c_{\texttt{Comm}}, c_{\texttt{Comp}}) \leftarrow \Phi(i, j, k)$
    $t_{\texttt{Comm}} \leftarrow g(y, c_{\texttt{Comp}})$
    $t_{\texttt{Forward}} \leftarrow f(x, c_{\texttt{Comm}})$
    $t_{\texttt{Backward}} \leftarrow f(x, c_{\texttt{Comm}})$
    $\Delta \leftarrow \min(t_{\texttt{Comm}}, t_{\texttt{Forward}}, t_{\texttt{Backward}})$
    Update $\mathbf{S}(i', j', k') \leftarrow \mathbf{S}(i, j, k) + \Delta$.
    Update workloads based on $\Delta$, $c_{\texttt{Comm}}$, $c_{\texttt{Comp}}$.
  **end**
  **return** $T(S_{N,N,N})$
**end**

precede `Comp`, for example Section III-B2 Scenario 1. **Case 2:** $i = N$ redirects all resources to `Comp` at the final stage of the process once all `Comm` tasks are completed as demonstrated by Section III-B2 Scenario 3. **Case 3: Otherwise**, core allocations are adjusted dynamically to adapt to real-time changes in task demands and system performance, for instance Section III-B2 Scenario 2.

### C. Initialization and State Transition Equation

The process is initialized as:

$$T(S_{0,0,0}) = 0 \tag{14}$$

The state transition at $S_{i,j,k}$ is contingent upon the core allocation and is mathematically expressed as follows:

$$T(S_{i',j',k'}) = \min_{c_{\texttt{Comm}}(S_{i,j,k}), c_{\texttt{Comp}}(S_{i,j,k})} \{T(S_{i,j,k}) + \Delta\} \tag{15}$$

where $\Delta$ represents the quantum of time transfer, defined as the shortest interval amongst three distinct temporal segments: the minimal completion times of the `Comm`, `Forward`, `Backward` phases within a given cycle. The variables $i', j', k'$ signify the respective completion counts of `Comm`, `Forward`, `Backward` cycles at time $T(S_{i,j,k}) + \Delta$. The whole process is illustrated in Algorithm2.

## V. Evaluation

### A. Experiment Setting

In this section, we utilize simulation experiments as a primary method to evaluate the efficacy of our approach in edge-based KD.

Firstly, we utilize datasets with different batch sizes as inputs for the $\mathcal{T}$ inference and $\mathcal{S}$ training, obtaining logits of various sizes and corresponding communication and computation times (an example is given in Section III-A). These empirically sampled data are then used as inputs (workloads of `Comm` and `Comp`) for our simulation framework and function ($f$ and $g$) fitting, allowing us to simulate the actual edge-based KD process.

Secondly, we compare our solution against commonly used methods: *random allocation* and *priority-based allocation*.

- *random allocation*: Considering the dependence of edge-based KD, the CPU resources are randomly allocated in the case of parallel `Comm` and `Comp` tasks.
- *priority-based allocation*: Regardless of parallel competition, CPU resources are allocated exclusively to either `Comm` or `Comp` tasks at any given time. Notably in our experiments, the time taken by `Comp`-priority and `Comm`-priority algorithms is identical. This outcome arises because the exclusive allocation of all cores either to `Comm` or `Comp` makes the superposition of time slices invariant to the priority assigned to `Comp` and `Comm`.

We conduct a series of experiments to evaluate the performance of our solution under varied simulated scenarios. The workload ratios of `Comm` to `Comp` are set to 1:1 and 1:2, respectively, to closely simulate real-world operations. Additionally, the workloads for `Comm` and `Comp` are configured to reflect true operational data. To methodically assess the impact of system resources on our solution, we vary critical parameters such as the total number of CPU cores, denoted as $c_{\text{total}}$. The experiments are performed with $c_{\text{total}}$ set to 6, 8, 10, and 12, allowing us to examine how changes in CPU core availability influence the effectiveness of our resource allocation strategy.

### B. Result and Analysis

By fitting a large number of actual measured data, we set the parameters of $f$ and $g$ such that the two functions satisfy the changes in the actual edge-based KD (detailed in Figure 4). As depicted, the `Comp` time decreases monotonically as more CPU cores are allocated to `Comp` tasks. This is an expected outcome, underpinned by the efficiency gains from parallel processing capabilities inherent in multi-core systems. Conversely, `Comm` time exhibits a non-linear increase as the allocation for `Comp` intensifies. Initially, when the `Comp` allocation is low, `Comm` benefits from the availability of more cores, but as more cores are diverted to `Comp`, the increase in `Comm` time becomes pronounced. This suggests a critical inflection point beyond which additional cores assigned to `Comp` disproportionately extend `Comm` time, indicating the diminishing marginal utility of cores for `Comm` tasks.

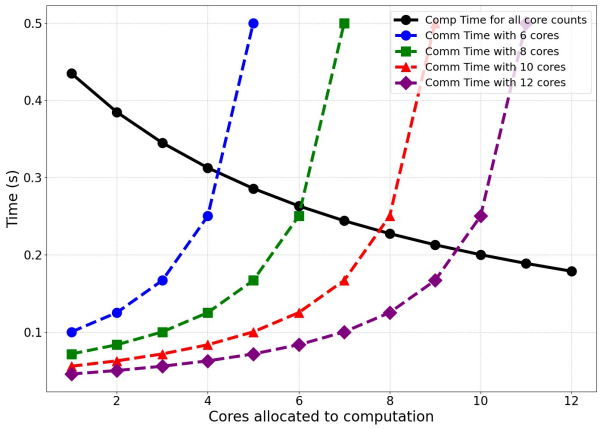
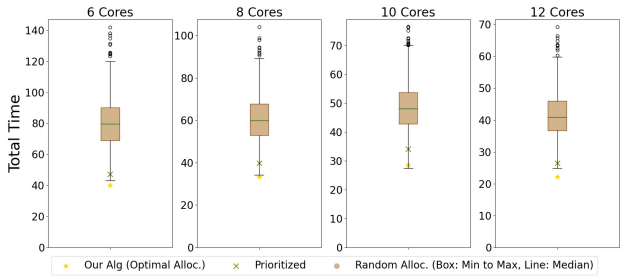
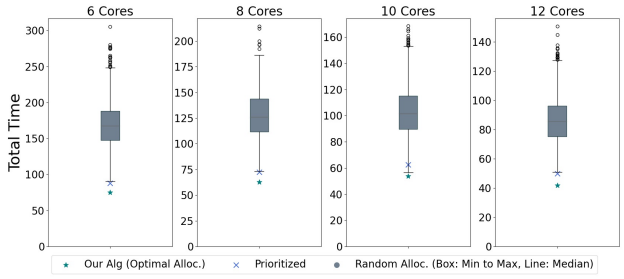



Fig. 4. $f$ and $g$ (Comp and Comm Times) with Different Core Allocations (Parameter settings: $x = 1$, $y = 1$. Core counts: 6, 8, 10, 12)




(a) $x$ (Comp) : $y$ (Comm) = 1 : 1




(b) $x$ (Comp) : $y$ (Comm) = 2 : 1

Fig. 5. Comparison with Prioritized and Random Allocation

The performance of our solution is shown in Fig 5. The ratio between Comp load and Comm load is set to 1:1 and 2:1 respectively. The random allocation algorithm is experimented with 1000 times for each configuration. The time taken by our solution can achieve approximately optimal performance under different workload distributions and different total numbers of CPUs available. The dynamic resource allocation strategy utilized in our solution enhances performance by adapting resource distribution in real time based on current task demands and system status. This adaptability allows for the precise tuning of CPU cores between Comp and Comm tasks, minimizing delays caused by resource contention.

## VI. Conclusion

In conclusion, we have delved into the complex and challenging world of KD within the context of edge computing. Our research has provided key insights into the speed impacts of CPU core allocation to communication and computation on edge devices with limited resources. We've highlighted ideal dependencies and constraints within edge-based KD and introduced a dynamic resource allocation solution. The simulation experiments, which varied key parameters like CPU core count and data volume, underscored the robustness of our approach compared to traditional strategies. This framework can provide for further exploration and innovation in optimizing edge computing operations.

## References


[1] H. Jin, D. Bai, D. Yao, Y. Dai, L. Gu, C. Yu, and L. Sun, "Personalized edge intelligence via federated self-knowledge distillation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 567–580, 2022.

[2] K. Nelson, G. Corbin, M. Anania, M. Kovacs, J. Tobias, and M. Blowers, "Evaluating model drift in machine learning algorithms," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. IEEE, 2015, pp. 1–8.

[3] Z. Sharif, L. T. Jung, I. Razzak, and M. Alazab, "Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3079–3093, 2021.

[4] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[5] C. T Dinh, N. Tran, and J. Nguyen, "Personalized federated learning with moreau envelopes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 394–21 405, 2020.

[6] J. Liu, H. Xu, L. Wang, Y. Xu, C. Qian, J. Huang, and H. Huang, "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 674–690, 2021.

[7] H. G. Abreha, M. Hayajneh, and M. A. Serhani, "Federated learning in edge computing: a systematic survey," *Sensors*, vol. 22, no. 2, p. 450, 2022.

[8] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[9] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[10] Y. Yang, C. Long, J. Wu, S. Peng, and B. Li, "D2d-enabled mobile-edge computation offloading for multiuser iot network," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 12 490–12 504, 2021.

[11] Y. Liu, B. Jiang, S. Zhao, T. Lin, X. Wang, and C. Zhou, "Libra: Contention-aware gpu thread allocation for data parallel training in high speed networks," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023, pp. 1–10.

[12] I. Murturi and S. Dustdar, "A decentralized approach for resource discovery using metadata replication in edge networks," *IEEE Transactions on Services Computing*, vol. 15, no. 5, pp. 2526–2537, 2021.

[13] B. Qian, Z. Wen, J. Tang, Y. Yuan, A. Y. Zomaya, and R. Ranjan, "Osmoticgate: Adaptive edge-based real-time video analytics for the internet of things," *IEEE Transactions on Computers*, vol. 72, no. 4, pp. 1178–1193, 2022.