ROG: A High Performance and Robust Distributed Training System for Robotic IoT

Xiuxian Guan[1,#], Zekai Sun[1,5,#], Shengliang Deng[1], Xusheng Chen[1], Shixiong Zhao[1,*],
Zongyuan Zhang[1], Tianyang Duan[1], Yuexuan Wang[1], Chenshu Wu[1],
Yong Cui[2], Libo Zhang[3], Yanjun Wu[3], Rui Wang[4], and Heming Cui[1,5]
[1]*Department of Computer Science, The University of Hong Kong, Hong Kong, China*
Email: {xxguan, zksun, sldeng, xschen, sxzhao, zyzhang2, tyduan, amywang, chenshu, heming}@cs.hku.hk
[2]*Tsinghua University, Beijing, China, Email: cuiyong@tsinghua.edu.cn*
[3]*Institute of Software, Chinese Academy of Sciences, Beijing, China, Email: {libo, yanjun}@iscas.ac.cn*
[4]*EEE, Southern University of Science and Technology, Email: wang.r@sustech.edu.cn*
[5]*Pujiang Lab, Shanghai, China*

***Abstract*—Critical robotic tasks such as rescue and disaster response are more prevalently leveraging ML (Machine Learning) models deployed on a team of wireless robots, on which data parallel (DP) training over Internet of Things of these robots (robotic IoT) can harness the distributed hardware resources to adapt their models to changing environments as soon as possible. Unfortunately, due to the need for DP synchronization across all robots, the instability in wireless networks (i.e., fluctuating bandwidth due to occlusion and varying communication distance) often leads to severe stall of robots, which affects the training accuracy within a tight time budget and wastes energy stalling. Existing methods to cope with instability of datacenter networks are incapable of handling such straggler effect. That is because they are conducting model-granulated transmission scheduling, which is much more coarse-grained than the granularity of transient network instability in real-world robotic IoT networks, making a previously reached schedule mismatch with the varying bandwidth during transmission.**

We present ROG, the first ROw-Granulated distributed training system optimized for ML training over unstable wireless networks. ROG confines the granularity of transmission and synchronization to each row of a layer's parameters and schedules the transmission of each row adaptively to the fluctuating bandwidth. In this way the ML training process can update partial and the most important gradients of a stale robot to avoid triggering stalls, while provably guaranteeing convergence. The evaluation shows that, given the same training time, ROG achieved about 4.9%~6.5% training accuracy gain compared with the baselines and saved 20.4%~50.7% of the energy to achieve the same training accuracy.

***Keywords*-distributed training, wireless networks, training throughput, robust, energy efficient**

I. Introduction

Critical robotic tasks such as rescue [1] and disaster response [2] are more prevalently leveraging machine learning models (e.g., objective recognition models [3] or action control models [4], [5]) deployed over a team of mobile robots. These models typically require real-time training to adapt pre-trained parameters to changing environments [6], [7] (e.g., from sunny to foggy), but it is often expensive for the robots to access a cloud data center for model training due to the lack of stable internet access. Therefore, such training for critical robotic tasks is often distributedly deployed among a team of robots over the robotic IoT networks [8], [9].

Such distributed training typically adopts the parameter server paradigm [10], [11]: each device keeps a copy of the model and computes the model's parameter updates (gradients) on its own data iteratively; between iterations, a synchronization barrier (BSP [12]) is inserted, where each device pauses its computation, pushes the computed gradients of the whole model to and pulls the averaged gradients from a parameter server (located on one of the devices) over wireless networks. The process of training iterates until the shared model converges (i.e., reaches a desired accuracy).

To ensure high performance of the critical robotic tasks with the typically limited computation power and battery energy on robots, we identify that such distributed training should meet the following requirements (**3Rs**):

- Robust (**R1**): The critical robotic tasks are often confronted with complex environments (e.g., crowds, damaged areas). The performance of the distributed training should be resilient to these environments for the robots to adapt to various changing environments and fulfill the critical tasks.
- High training throughput (i.e., the number of training iterations in unit time) (**R2**): Given a tight time budget, high training throughput is crucial for high training accuracy to better adapt to the changing environments.
- High statistical efficiency (i.e., the training accuracy gain per training iteration) (**R3**): With higher statistical efficiency, the training model can reach higher accuracy with the same number of training iterations.

3Rs are important for the training model to reach high accuracy given a tight training time budget, while preserving battery energy to reach a desired accuracy.

Unfortunately, although such distributed training with BSP

[#]Xiuxian Guan and Zekai Sun equally contributed. [*]Shixiong Zhao is the corresponding author.

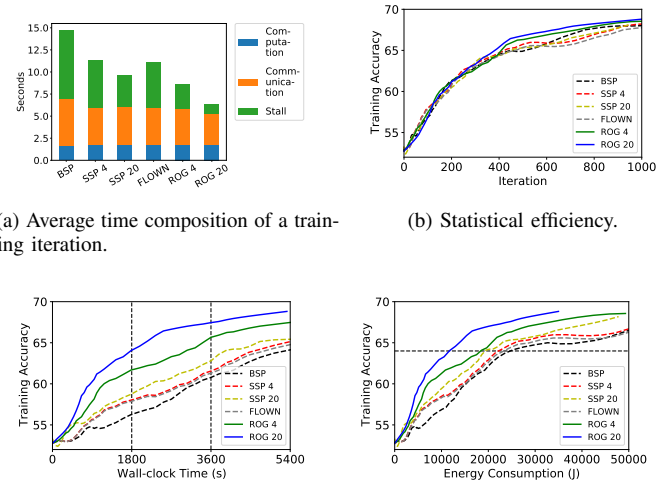
empirically achieves high statistical efficiency (**R3**) [13], the instability of real-world robotic IoT networks hinders it from meeting **R1** and **R2** by causing the ***straggler effect***: the transmission of gradients from some devices (i.e., stragglers) can be dramatically delayed (e.g., transmission time prolonged from 1.43s to 12.9s recorded in an unstable environment, see Sec. II-B) by sharp bandwidth degradation due to movement of the devices [14], [15], occlusion from obstacles [16], [17], etc; the devices that finish transmission (i.e., non-stragglers) have to stall until the delayed gradients from stragglers are transmitted in severely downgraded bandwidth, prolonging training iterations (violating **R1** and **R2**) and wasting energy stalling.

Although mainly designed for datacenter networks, Stale Synchronous Parallel (SSP) [13], [18] has the potential to mitigate such straggler effect. SSP allows non-stragglers to continue computing without the latest gradients from stragglers and only stall when the gradients from stragglers fall behind (stale) for a preset number of iterations (staleness threshold).

However, when coping with the instability of real-world robotic IoT networks, **R2** and **R3** are contradictory in SSP: high statistical efficiency requires a small staleness threshold [13], while high training throughput requires a large staleness threshold. In our evaluation (Fig. 1), SSP with a small threshold (4) achieved similar statistical efficiency as BSP but suffered severe straggler effect (stall time on average takes up 44.1% of the duration of a training iteration); a larger threshold (20) slightly reduced the stall time to 42.5% of a training iteration, at the cost of lower statistical efficiency.

Recent studies [19], [20] extend SSP by dynamically assigning (scheduling) the staleness threshold to simultaneously fulfill **R2** and **R3**: higher staleness threshold for devices that are estimated to have low bandwidth and less contribution to training accuracy; smaller threshold for the opposite. However, they are designed for datacenter networks and wired edge networks and cannot fulfill **R1**, because the random and rapid nature (see Sec. II-B) of bandwidth degradation in wireless networks can transform the non-stragglers estimated during scheduling into stragglers during the actual transmission, making the scheduling mismatch with the actual bandwidth. In our evaluation (Fig. 1), such methods still suffered straggler effect, which caused stall time to on average take up 45.2% of a training iteration, violating **R1**.

The key reason for the problem of the above methods is that they are synchronizing the model gradients on the granularity of a whole model, whose transmission time is typically coarser (longer) than the granularity (or frequency) of bandwidth fluctuation in real-world robotic IoT networks. From the view of robustness (**R1**) and training throughput (**R2**), the scheduling based on the granularity of a whole model can not adapt to the real-time fluctuation of bandwidth and will be frequently invalidated, causing more stall and reduced training throughput. From the view of statistical efficiency (**R3**), the scheduling treats all computed gradients from a device as a whole and neglects that gradients from a device have different contributions to training accuracy (e.g., gradients with small absolute value contribute little). Thus, it is a must to break up the gradient transmission and schedule the transmission of the gradients with a finer granularity.



(a) Average time composition of a training iteration.

(b) Statistical efficiency.

(c) Training accuracy against wall-clock time.

(d) Energy consumption against training accuracy.

Figure 1: Comparison between ROG and the baselines on the unsupervised domain adaptation application paradigm in the outdoor environments.

In this paper, we present ROG, a ROw-Granulated, high-performance and robust wireless distributed training system optimized for real-world robotic IoT networks. We choose the granularity of rows after comparing three typical levels of granularity to break up the parameters of an ML model: layers (matrixes), rows (matrix rows), and elements (individual parameters). Specially, element granularity requires indexing each element of the whole model for management, taking up data volume comparable to the whole model (high management cost); layer granularity is large in size and is still comparable with the granularity of bandwidth fluctuation (low transmission flexibility). Row granularity best trades off between management cost and transmission flexibility and enables that whenever bandwidth fluctuation happens, ROG can in real-time adapt to it by adjusting the scheduling of rows to be transmitted, at a negligible cost of transmitting only one row in degraded bandwidth.

The design of ROG is confronted with two major challenges. The first one is how to guarantee convergence in ROG when synchronizing gradients on row granularity. We propose Row Synchronous Parallel (RSP) that breaks up and enforces the staleness control of SSP to each row of a model across different devices and different rows within

a same device. RSP guarantees convergence by confining the divergence of rows within the staleness threshold and thus confining the divergence of the whole training model on different devices. We formally prove that RSP achieves the same convergence guarantee as SSP (see Sec. IV-C).

The second challenge is under RSP, how to properly schedule the transmission of each row from different devices to fulfill **3Rs**. ROG adaptively aligns the transmission time of each device by speculatively transmitting each row with a novel *Adaptive Transmission Protocol* (ATP). In a training iteration, ATP monitors the transmission time taken by the transmitted rows and in real-time updates the scheduling of the pending rows to be transmitted, to ensure that all devices roughly spend equal time transmitting gradients under random and sharp bandwidth fluctuation (**R1**), avoiding straggler effect (**R2**). ATP further prioritizes the transmission of different rows based on their staled versions and contribution to model convergence (e.g., the absolute values of the gradients), reducing stall and accelerating convergence (**R3**).

We implemented ROG in PyTorch [21] and evaluated ROG on a team of mobile robots under two representative real-world online training application paradigms (unsupervised domain adaptation and implicit mapping and positioning, see II). We compared ROG with BSP [12], SSP [13] and a SOTA dynamic threshold method [19] (referred to as FLOWN) under different real-world robotic IoT networks environments (namely indoor with moderate instability and outdoor with more severe instability). We also minimized the communication volume with gradient compression [22] (the compressed gradients were only sized at 2.1 MByte and 0.75 MByte in the two paradigms) to conduct the tightest comparison between ROG and the baselines. Evaluation shows that:

- ROG achieves high accuracy. ROG achieved a 4.9% ~6.5% accuracy gain over the baselines after training for 60 minutes, due to 25.2%~80.4% higher training throughput and non-degraded statistical efficiency under outdoor and indoor environments.
- ROG is energy-efficient. With the above advantage of training throughput and statistical efficiency, ROG reduced battery energy consumption by 20.4%~50.7% compared with the baselines when the training model reached a same high accuracy,
- ROG is scalable. When increasing the number of robots involved or increasing the batch size of training, ROG still achieved 3.0%~5.3% accuracy gain and a 30.3% ~55.1% energy consumption reduction over the baselines.
- ROG is easy to use. It took only tens of lines of code to apply ROG to existing ML applications.

Our main contribution is RSP, a new row-granulated synchronization model and ATP, a fine-grained scheduling strategy optimized for distributed training over real-world robotic IoT networks. ROG fulfills **3Rs**: while conducting row-granulated staleness control to guarantee convergence with RSP, ATP schedules the transmission of each row adaptively to the fluctuating bandwidth (**R1**), so as to avoid the straggler effect (**R2**) and make gradients with more contribution to training accuracy be transmitted first (**R3**). We envision that ROG will nurture diverse ML applications deployed on mobile robots in the field, such as robot rescue [1], disaster response [2], and robot surveillance [23], [24], making them fast and energy-efficiently adapt to changing environments under an extremely unstable local wireless network without being affected by straggler effect. ROG's code is released on https://github.com/hku-systems/ROG.

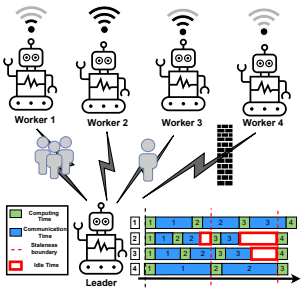


Figure 2: The instability of real-world robotic IoT networks.

In the rest of this paper, we introduce the background of this paper in Sec. II, give an overview of ROG in Sec. III, present the detailed design of ROG in Sec. IV, evaluate ROG in Sec. VI, and finally conclude in Sec. VII

II. BACKGROUND

A. Online training on Robotic IoT

While machine learning methods heavily rely on labeled training dataset (supervised training), it is costly to label datasets in every possible environment. As a result, various unsupervised training algorithms are developed to learn knowledge from unlabeled data. For example, adversarial unsupervised domain adaptation methods [25], [26] typically adapt a pretrained model to a new environment by training it with both shifted (noised) unlabeled data from the new environment and labels predicted with generative methods. With such methods, robots can adapt their pretrained models to new environments after training with online collected data to retain high accuracy of the models. As another example, implicit mapping and positioning [27], [28] construct a machine learning model representation of a 3D dense map by training the model with online collected unlabeled image sequences. We envision that the prosperity of these unsupervised training algorithms are making online training on online collected data on robots feasible and practical.

B. Characteristics of Robotic IoT Networks

In real-world robotic IoT applications (Fig. 2), devices typically need to move around for rescue, search, etc. Although wireless networks suffice for high mobility, the occlusion

of obstacles and the change of distances among devices cause *instability* in the bandwidth capacity: sharp bandwidth fluctuation with random duration happens frequently and randomly. This causes divergence in gradient transmission time from different robots and the straggler effect.

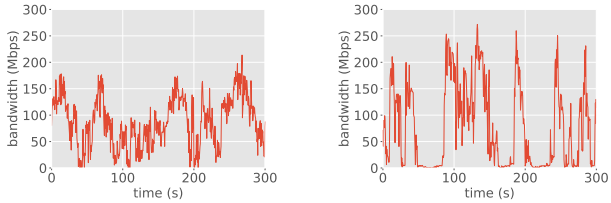
To demonstrate the instability, we set up a robot surveillance task: two four-wheel robots navigate around several given points at 5˜40cm/s speed in our lab (indoors) and campus garden (outdoors). The hardware and wireless network settings are as described in Sec. VI. We believe our setup represents the state-of-the-art (SOTA) computation and communication capabilities of robotic IoT devices.

We saturated the wireless network connection with iperf [29] and recorded the average bandwidth capacity between these two robots every 0.1s for 5 minutes, shown in Fig. 3. Both indoor and outdoor records show frequent and sharp bandwidth fluctuation. Statistically, on average a 20% fluctuation of bandwidth capacity happened every 0.4s, and a 40% fluctuation typically happened every 1.2s. Such times are comparable to the time of transmitting compressed gradients recorded with ideal wireless networks (e.g., 1.47s), causing high variability of transmission time. Besides, the outdoors bandwidth more frequently dropped to extremely low values around 0Mbit/s, exhibiting higher instability than indoors. The reason is the outdoor open area lacks walls to reflect wireless signals. When there are obstacles (e.g., trees) between communicating robots, fewer signals could be received in the outdoor area than the indoor.

Comparison with Datacenter Networks and Edge Networks. Compared with robotic IoT networks, datacenter networks (for distributed training in datacenter) and edge networks (for federated learning) are wired and often exhibit much lower bandwidth fluctuation. In datacenter networks, bandwidth fluctuation is typically caused by congestion on intermediate switches, and could be mitigated by scheduling traffic on switches [30]. In edge networks, bandwidth fluctuation is often caused by the variation of overall traffic volume, and typically happens at the scale of hours [31]. Existing methods target these two types of networks, and are not designed for handling instability in robotic IoT networks.

C. Impact of Straggler Effect on Power Consumption

People may think a stalling robot can be consuming little energy. However, we recorded the energy consumption when a robot is stalling due to straggler effect and found that a stalling robot still consumed almost one third of the energy consumption when the robot was computing (see Sec. VI). That is because the device cannot be put into low power sleep mode even when stalling, as it has to wait for messages from the parameter server and promptly continue working when stragglers catch up, and chips like CPU, GPU, and memory consume non-negligible power even when not computing, due to the static power consumption rooted in transistors' leakage current [32]. Consequently, besides damaging training throughput, stall caused by straggler effect also has a major impact on power consumption of the training process.



(a) Indoors. (b) Outdoors.

Figure 3: The instability of robotic IoT networks. A 40% fluctuation of bandwidth typically happens every 1.2s, comparable to the time of transmitting compressed model gradients.

D. Related Work

BSP, SSP and their Variants. Bulk Synchronous Parallel (BSP) methods [13] enforces synchronization between each iteration. Therefore, it could easily get blocked by stragglers. To mitigate the straggler effect in datacenter networks while guaranteeing model convergence, SSP is usually adopted [13] in practice. By loosening the synchronization barrier, SSP allows fast workers to continue their iterations when the updates from slow workers are staled until the staled version reaches a *staleness threshold*. With a small staleness threshold, SSP ensures that all gradients extracted from each device's dataset equally contribute to the SGD convergence (same as BSP), which is widely reported to be necessary for an SGD process to achieve high statistical efficiency and high final accuracy [33], [34], [35], [36]. However, a small threshold cannot contain the instability in real-world robotic IoT networks while a large threshold sacrifices high statistical efficiency.

Inheriting SSP's more flexible synchronization model compared with BSP, subsequent studies (including federated learning) [19], [20], [37] extensively explored the scheduling of synchronization among workers according to network conditions and the contribution to training accuracy. Scheduling strategies work well in datacenter networks and edge networks with slow and moderate bandwidth fluctuation. However, these scheduling strategies are not robust to the rapid and random bandwidth fluctuation in robotic IoT networks, because their model-granulated scheduling and transmission are often coarser-grained than the transient instability of real-world robotic IoT networks.

Gradient Compression. Gradient compression greatly reduces the communication traffic volume and is indeed essential for practical distributed training over wireless networks. Some lossy gradient compression methods [38]

(information is lost during compression and cannot be recovered) achieve up to 0.1% compression rate (i.e., size after compression divided by original size), but they cannot provide convergence guarantee [38]. In this paper, we only consider lossless compression methods (e.g., the lost information during compression is compensated with error compensation [22]) which have a typical compression rate of around 3% [22].

Even with gradient compression, communication still takes a major time portion in distributed training on robotic IoT devices for two reasons. First, the devices typically share the same wireless channel, incurring traffic volume proportional to the number of devices involved in the distributed training process. Second, with the rapid advancement of SOTA robotic IoT devices [39], the computation time on each device is also decreasing. As a result, the communication time is typically comparable to the computation time.

Consequently, even with gradient compression, the straggler effect, which severely prolongs the communication time, still has a major impact on the distributed training process. In our experiments, a Jetson Xavier NX [40] device out of a four-device team computed gradients in 2.18s and ideally needed to wait for 1.47s upon the synchronization barrier in BSP (four devices push and pull the compressed gradients sized 2.1MByte, summing up to 134.4Mbit), which is comparable to (equal to 67.4% of) the computation time. Meanwhile, the straggler effect in the above indoor scenario caused each device to on average stall for 2.23 s in each iteration, equal to 102.2% of the computation time, severely degrading the training throughput.

III. Overview

A. Workflow

Fig. 4 presents the workflow of ROG and compares it with BSP and SSP in unstable networks. The random and sharp wireless bandwidth fluctuation causes the transmission time of each model among devices in BSP and SSP to diverge and causes straggler effect. Since the transmission time of each row among the devices also diverge in ROG, to avoid straggler effect, the main idea of ROG is to align the transmission time among all devices in real-time by dynamically and adaptively scheduling the transmission of rows, as shown in Fig. 4. The design of ROG tackles three problems: how to properly break up the gradient synchronization granularity, how to guarantee convergence, and how to schedule the gradient transmission in real-time.

The choice of granularity. Out of three possible granularity choices: elements, rows and layers, we choose rows to best tradeoff between the management overhead and flexibility in transmission (duration of transmission of the smallest unit). While ROG is adaptively transmitting the smallest units, it causes a management overhead that we need to at least maintain a list of indexes of all the smallest managed units on the whole model and transmit the list during every model synchronization, so that the adaptively transmitted units can be correctly indexed to its position on the model.

On the one hand, element granularity will apparently cause an index list as large as the number of elements of the whole model; as an integer (an index) and a floating-point number (an element) typically take up the same amount of data volume when being transmitted (i.e., the default int32 and float32 encoding configuration of PyTorch [21]), the transmission data volume will be doubled during every model synchronization. On the other hand, layer granularity typically causes a small index list (e.g., 226 layers in the model [41] with 16.95M elements we evaluated for the first application paradigm); however, a layer of a model can be still large at size (e.g., the largest layer of the aforementioned model has 1.18M elements) and bandwidth degradation during its transmission will still evidently prolong the training iteration. Overall, as a row of a model is neither too small (33307 rows on the aforementioned model that take up data volume only sized 0.24% of the model size for indexing in our evaluation) nor too big (a row typically contains several to hundreds of elements), row granularity is the best choice for ROG.

Row Stale Parallel (RSP). Since not all rows are synchronized in an iteration in ROG, gradients of different rows of the training model on a device can have different versions. Uncontrolled version differences could slow down the training and even fail to guarantee convergence. We find that breaking up and applying the staleness control of SSP to each row of the training model would confine the divergence of the same row across different devices and thus confine the divergence of the training model across different devices, which is key to the convergence of the distributed training process [13]. Consequently, we design RSP that adopts a two-level row-granulated staleness control: for the same row on the training model across different workers, the staled version should be within a preset staleness threshold; for different rows within the same worker, the staled version should also be within the same staleness threshold. Workers are forced to wait when these two requirements are not met, as shown in Fig. 4. In this way, RSP provably achieves the same level of convergence guarantee as SSP (see Sec. IV-C).

Adaptive Transmission Protocol (ATP). To align the transmission time among all devices, for a straggler in an iteration, ATP controls it to transmit MTA (minimum transmission amount, an empirical lower bound of the number of rows to be transmitted by stragglers to avoid stall) of the total rows and reports its transmission time of MTA (MTA time) to other devices. A non-straggler then keeps transmitting rows for MTA time (or all of their rows if the transmission finishes before MTA time), so that the transmission time among stragglers and non-stragglers is balanced and the straggler effect is avoided. Among rows within a device, ATP maintains the importance (depth of the

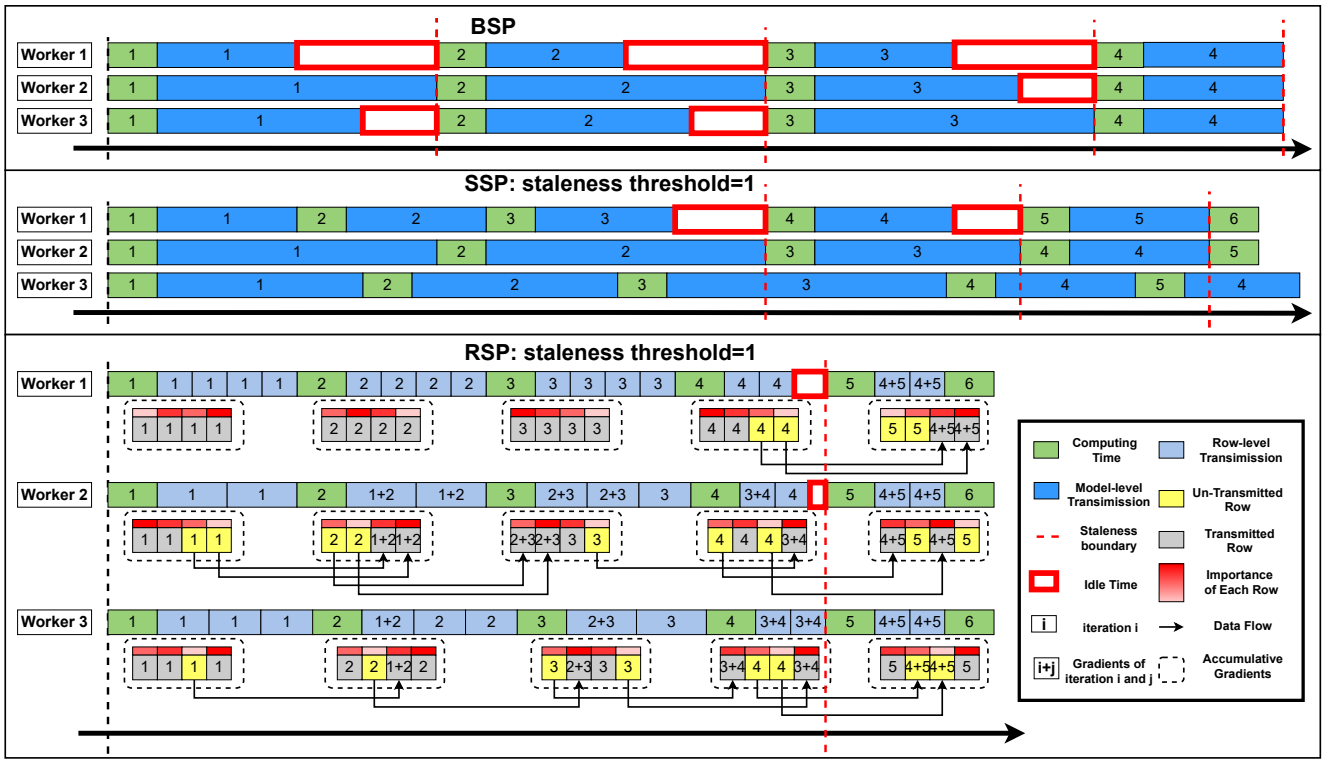


Figure 4: Workflow of ROG. The training model in RSP is divided into four rows and each row is synchronized in one row-level transmission. The bandwidth on these three devices are identical at the same time point in the three cases and the bandwidth is interfered by distance, occlusion, etc.

red color in Fig. 4) of each row based on its possibility to cause stall (e.g., the staled version) and the contribution of gradients of each row to training accuracy (e.g., the absolute value of the gradients); the rows with the highest importance will be transmitted first to minimize stall time and optimize statistical efficiency.

Technically, besides the management overhead, smaller granularity also brings extra transmission overhead. To ensure non-stragglers to keep transmitting rows only for MTA time, a straight forward approach is inserting judgement about whether MTA time is reached between the transmission of each two successive rows. However, such an approach is infeasible in ROG because empirically the transmission time of a row is comparable to the time cost of the inserted judgement, leading to severe under-utilization of the bandwidth capacity. Instead, we co-design ATP with the underlying transmission protocol and enable *speculative transmission*: the device continuously transmits rows in the priority determined by their importance without inserting judgement and discards the ongoing transmitting row once the MTA time is reached (see Sec. IV). In this way, the transmission overhead is reduced to possibly discarding the last row transmitted if its transmission is incomplete, which is also negligible.

B. Architecture of ROG

Fig. 5 shows the architecture of ROG. On each worker and the parameter server, ROG divides the parameters of the shared model into rows and maintains the gradients and staled version of each row individually. In an iteration, each worker computes gradients of the model based on its own share of the dataset, and Importance Metric sorts the order of transmission of each row according to the row's staled version and the average absolute values of the gradients. Speculative Transmission keeps transmitting for the aforementioned MTA time on non-straggler or transmits MTA on stragglers, balancing the transmission time of each worker. ROG increases the staled version of un-transmitted rows by one and set the staled version and gradients of transmitted rows to zero, so that only gradients of un-transmitted rows will be accumulated.

The parameter server aggregates and averages the received gradients, and updates Version Storage of the corresponding rows. If the requirements of RSP are met, ROG will similarly determine the importance of the rows in Importance Metric and transmit the most important rows' gradients in Speculative Transmission for MTA time or transmit MTA; otherwise, idle time (stall) will be inserted until RSP is met. Note that ROG maintains a copy of the gradients for each

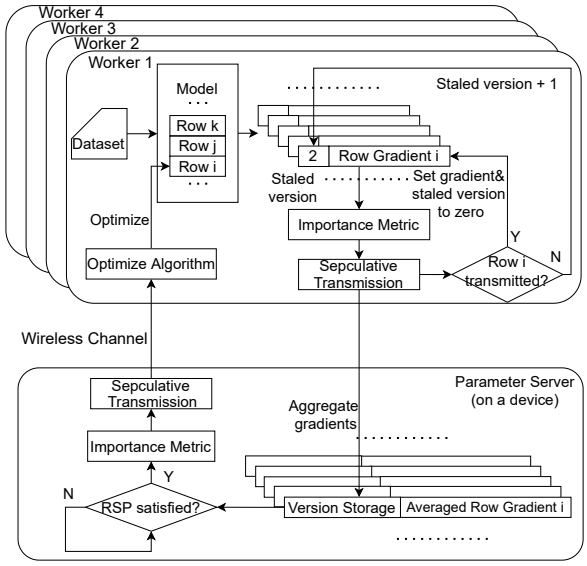


Figure 5: Architecture of ROG. Note that on the parameter server, similar to the worker side, ROG checks whether a row is transmitted and manages its accumulated gradients and the version storage accordingly. We leave out this part of the figure for simplicity.

worker, because the importance of each row can differ for different workers and thus different rows can be transmitted for different workers. If ROG sends the gradients of a row to a specific worker, ROG will only set the gradients on the copy for this worker to zero and the gradient copies for other workers are not affected.

On reception of gradients of certain rows, the worker optimizes the parameters of these rows with the received gradients. It is worth noting that, since the produced gradients of each worker will either be accumulated at the worker side or the parameter server side and eventually be sent to each worker, the model on each worker will be optimized with exactly the same gradients. Thus convergence of the shared model will not be affected.

IV. Detailed Design

A. Algorithms of ROG

Here we present how ROG integrates RSP and ATP together to achieve finer-grained staleness control and adaptive scheduling. The local worker part is given in Algo. 1 and the parameter server part is given in Algo. 2. Details of ATP are mainly described in Algo. 3 and Algo. 4 in the next subsection.

On the worker side in Algo. 1, when gradients are computed in a training iteration, they are added to the accumulated gradients (line 2, 3) and we then transmit the accumulated gradients to the parameter server in PushGradients(). PushGradients() sorts the transmission order of the accumulated gradients of each row and then speculatively

Algorithm 1 Local Worker

Function *LocalWorker_*ROG*()*: `// On workers`
  Data: w: local model parameters; η: learning rate; N: total training iterations; $iters$: training iterations that each row is pushed; $g\prime$: accumulated gradients; t: staleness threshold
1 **for** *each iteration n:* $1...N$ **do**
2 $g \leftarrow$ Training(w)
3 $g\prime \leftarrow g\prime + g$
4 $iters \leftarrow$ PushGradients($g\prime$, n, $iters$)
5 PullAveragedGradients(w, eta)
6 **Function** *PushGradients($g\prime$, n, iters, t)*:
  `// Worker mode of ImportanceMetric`
7 ImportanceMetric($g\prime$, $iters$, $'worker'$)
8 $Transmitted \leftarrow$ SpeculativeTransmission($g\prime$, n, t)
9 **for** *each row i in Transmitted* **do**
10 $g\prime_i \leftarrow 0$
11 $iters_i \leftarrow n$;
12 **end**
13 **Function** *PullAveragedGradients(w, eta)*:
14 $\bar{g} \leftarrow$ RecvGradients()
15 **for** *each* $\bar{g}_i$ *received from server* **do**
16 $w_i \leftarrow w_i - \eta \bar{g}_i$
17 **end**

Algorithm 2 Parameter Server

Function *ParameterServer_*ROG*()*:
  Data: $\bar{g}^r$: averaged gradient for worker r; $\bar{g}^r_i$: i-th row of $\bar{g}^r$; v^r_i: the latest training iteration on worker r that updates row i; V: $\{v^r_i\}$; num: the number of workers; P: rows' priority; t: staleness threshold
1 **upon** *receive gradients $g\prime$ from worker r* **do**
  `// Worker r push gradients`
2 $g\prime, n \leftarrow$ RecvGradients(r)
3 **for** *each row i in $g\prime$* **do**
4 $v^r_i \leftarrow n$
5 **for** *each worker s* **do**
6 $\bar{g}^s_i \leftarrow \bar{g}^s_i + \frac{g\prime_i}{num}$
7 **for** *each row i in $g\prime$* **do**
  `// $v^r_i$ triggers the finer-grained threshold`
8 **while** $v^r_i - min(V) \geq t$ **do**
9 wait for other worker update $\bar{g}_i$
  `// Worker r pull gradients`
  `// Server mode of ImportanceMetric`
10 ImportanceMetric($\bar{g}$, V, $'server'$)
11 $Transmitted \leftarrow$ SpeculativeTransmission($\bar{g}$, t)
12 **for** *each row i in Transmitted* **do**
13 $\bar{g}^r_i \leftarrow 0$

transmits these rows such that the transmission time among different workers is balanced in SpeculativeTransmission() (line 7 to 8). SpeculativeTransmission() also reports the latest training iteration that produced these gradients to the parameter server for it to maintain its Version Storage. Accumulated gradients of the transmitted rows will be assigned to zero and their latest training iteration that is pushed to the parameter server is recorded in line 9 to 11. In PullAveragedGradients(), pulled averaged gradients of certain rows would be used to update the parameters of the corresponding rows in line 15 to 16.

On the parameter side in Algo. 2, upon reception of gradients of certain rows from a worker r, ROG on the parameter side first finds the corresponding rows on the shared model and then accumulates the averaged gradients as shown in line 2 to 6. From line 7 to line 9, we only consider the situation that the times (training iterations) that gradients of row i ($\bar{g}_i$) is updated by worker r (v_i^r) should not be ahead of the updated times of any rows by any workers ($min(V)$) more than the threshold. That's because when the threshold is triggered, we only need to stall the non-stragglers and wait for stragglers to catch up to satisfy RSP. In line 10 to 13, ROG determines the transmission priority of rows, speculatively transmits these rows, and manages the accumulated gradients of transmitted rows similar to the worker side.

B. Adaptive Transmission Protocol

Algorithm 3 Importance Metric

Function *ImportanceMetric*:
  Data: $g\prime$: gradients of all rows; *iters*: training iterations that each row is updated; *mode*: worker or parameter server mode
1 $importance \leftarrow \{\}$
2 **for** *each row i in* $g\prime$ **do**
3 **if** *mode*=='*worker*' **then**
4 $j \leftarrow f_1 \times$mean(abs($g_i\prime$)) + $f_2 \times$(max($iter$) - $iter_i$)
5 **else**
6 $j \leftarrow f_1 \times$mean(abs($g_i\prime$)) + $f_2 \times$($iter_i$ - min($iter$))
7 **end**
8 $importance$.append(j)
9 Sort($g\prime$, $importance$)

The ATP protocol consists primarily of two functions: *ImportanceMetric* (Algo. 3) that prioritizes the transmission of different rows, and *SpeculativeTransmission* (Algo. 4) that records and aligns the gradient transmission time among different workers.

Importance Metric in Algo. 3 shows our scheme to prioritize the gradient rows. Notably, we treat workers and the parameter server differently (line 3 to 6). Besides absolute values of gradients ($mean(abs(g_i\prime))$), since the staleness threshold is triggered and handled at the parameter server side, workers pushing gradients to the parameter server need

Algorithm 4 Speculative Transmission

Function *SpeculativeTransmission*:
  Data: $g\prime$: sorted gradients of all rows; n: current training iteration training; t: staleness threshold
1 $MTA \leftarrow$ MTATable(t) $\times$ len($g\prime$)
2 $t_{MTA} \leftarrow$ GetMTATime()
3 $Transmitted \leftarrow$ SendWithTimeout($g\prime$, t_{MTA})
4 **if** *len(Transmitted)* $< MTA$ **then**
5 Send($g\prime$[len($Transmitted$): MTA])
6 $Transmitted \leftarrow MTA$
7 **end**
8 UpdateMTATime()
9 **return** $Transmitted$

to especially give priority (bigger j) to the staled rows, so as to reduce the possibility to trigger the staleness threshold and cause stall. Thus we add a term $max(iter) - iter_i$ to the importance of each row on workers to estimate the number of iterations that the row has not been pushed (staled) to the parameter server (line 4, f_1 and f_2 are empirical coefficients). On the contrary, pulling gradients from the parameter server will not affect the triggering of the staleness threshold, and thus we give extra priority to fresher rows (estimated with $iter_i - min(iter)$ in line 6) that typically have higher contribution to training accuracy. These rows are then sorted in descending order according to their assigned importance and will be transmitted in the sorted order.

After sorting these rows, Speculative Transmission (Algo. 4) retrieves the scheduled transmission time (t_{MTA}) and enforces the transmission time limit by setting the timeout of the ongoing transmission to t_{MTA} (line 3). Upon timeout, the ongoing transmission will be immediately stopped and the transmitted gradients will be recorded in $Transmitted$. If at least P percent of rows is transmitted each time, at most $(1-P)^s$ percent of the row will remain un-transmitted after s steps, because all rows are transmitted and updated independently of each other. In order to ensure all rows are transmitted before triggering the threshold, there should be $(1-P)^{S-1} < P$, where the staleness threshold is S. We set a minimum transmission amount (MTA), which the percentage of rows per transmission cannot be lower than, to be the solution to the above inequality in Table I. If the amount of transmitted rows has not reached MTA, Speculative Transmission would go on transmitting the remaining rows of MTA in line 4 to 7. The possible cost of such speculative transmission is that the transmission of the last row transmitted could be incomplete and needs to be discarded, which is a negligible cost thanks to the small size of a row.

C. Proof of guaranteed convergence

Following the convention in [13], we refer to x as the "system state", and the operation $x \leftarrow x + u$ as "writing an

Threshold	2	3	4	5	6	7	8
MTA	0.5	0.38	0.32	0.28	0.25	0.22	0.2

Table I: MTA values under different thresholds

update", where u is a "model update". We define $D(x||x\prime) = \frac{1}{2}\|x-x\prime\|^2$ and assume that P workers write model updates to parameter server independently. Let u_t be the tth update written by workers and $\tilde{x}_t$ be the tth model parameters read by workers, and we divide the whole model parameters into M parts by row, which means $u_t = [u_t^1, u_t^2, ..., u_t^M]^T$ where u_t^i is the ith row of u_t and $\tilde{x}_t^i$ is the ith row of $\tilde{x}_t$.

In this paper, we focus on SGD [42] and prove convergence of each row and further convergence ot the entire model. Since ROG either synchronizes or aggregates each row of parameter updates, no parameter update in a row is lost; thus the final convergence of the whole training model can provably have the same convergence guarantee as SSP.

Theorem 1 (SGD under RSP*):* Suppose we want to find the minimizer $x^\star$ of a convex function $f(x) = \sum_{t=1}^{T} f_t(x)$, via gradient descent on one component ∇f_t at a time. We assume the components f_t are also convex. Let $u_t = -\eta_t \nabla f_t(\tilde{x}_t)$, where $\eta_t = \frac{\sigma}{\sqrt{t}}$ with $\sigma = \frac{F}{L\sqrt{2(S+1)P}}$ for certain constants F, L, and $S_{max} = MAX_{i=1,2,...,M}(S_i)$. Then, assuming that $||\nabla f_t(x)|| \leq L$ for all t (i.e. f_t are L-Lipschitz), and that $MAX_{x,x\prime \in x} D(x||x\prime) \leq F^2$ (the optimization problem has bounded diameter), we claim that $R[x] = \sum_{t=1}^{T}(f_t(\tilde{x}_t) - f(x^\star)) \leq o(T)$, which implies $E_t[f_t(\tilde{x}_t) - f_t(x^\star)] \to 0$ and thus convergence.

Proof: We define $\tilde{g}_t = \nabla f_t(\tilde{x}_t) = [\tilde{g}_t^1, \tilde{g}_t^2, ..., \tilde{g}_t^M]^T$ and there is

$$\begin{aligned} R[x] &= \sum_{t=1}^{T}(f_t(\tilde{x}_t) - f(x^\star)) \leq \sum_{t=1}^{T}\langle \nabla f_t(\tilde{x}_t), \tilde{x}_t - x^\star\rangle \\ &\quad \text{(the property of convex functions)} \\ &= \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{t=1}^{T}\sum_{i=1}^{M}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\right\rangle \\ &\quad \text{(the property of inner product)} \\ &= \sum_{i=1}^{M}\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\right\rangle \quad \text{(since } i \text{ and } t \text{ are independent)} \end{aligned} \tag{1}$$

■

Then, we are going to prove the convergence of each row by finding the upper boundary of $\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\right\rangle$ for each row. Thanks to SSP's pioneering work [13], we can learn the following Lemma 1.

Lemma 1: For P workers with staleness threshold $S\prime$, we assume that $||\nabla f\prime_t(x)|| \leq L\prime$ and $max_{x^i, x\prime^i \in X^i} D(x^i||x\prime^i) \leq F\prime^2$. If we set the initial step size $\sigma = \frac{F}{L\sqrt{2\kappa}}$, where $\kappa = (s+1)P$ and assume T large enough that $\frac{1}{2\kappa} + \frac{\kappa}{\sqrt{T}} \leq 1$, then

$$\begin{aligned} R[X] &\leq F\prime L\prime\sqrt{2\kappa T}\left[3 + \frac{1}{2\kappa} + \frac{\kappa}{\sqrt{T}}\right] \\ &\leq 4F\prime L\prime\sqrt{2(S\prime+1)PT} \end{aligned} \tag{2}$$

Because the parameters of each row are independent of each other, RSP keeps the same staleness threshold control for each row as SSP does. In this way, RSP breaks the granularity of the synchronization model from the whole model to rows and still achieves the same convergence guarantee for any single row as SSP from Lemma 1, which means each RSP's row still satisfies Inequality 2:

$$\begin{aligned} R[X^i] &= \sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\right\rangle \leq 4F\prime L\prime\sqrt{2(S_i+1)PT} \\ &\leq 4F\prime L\prime\sqrt{2(S_{max}+1)PT} \quad (S_{max} = MAX_{i=1,2,...,M}(S_i)) \end{aligned} \tag{3}$$

Based on Inequality 1 and Inequality 3, we further have

$$\begin{aligned} R[X] &\leq \sum_{i=1}^{M}\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\right\rangle \leq M * MAX(R[X^i]) \\ &= M * 4F\prime L\prime\sqrt{2(S_{max}+1)PT} \end{aligned} \tag{4}$$

As $x = [x^1, x^2, ..., x^M]^T$ and the parameters of each row are independent of each other, there is $\|x-x\prime\|^2 = \sum_{i=1}^{M}\left\|x^i - x\prime^i\right\|^2$, and we can further have

$$D(x||x\prime) = \frac{1}{2}\|x-x\prime\|^2 = \frac{1}{2}\sum_{i=1}^{M}\left\|x^i - x\prime^i\right\|^2 = \sum_{i=1}^{M} D(x^i||x\prime^i) \tag{5}$$

Since $D(x^i||x\prime^i)$ is independent of each other with various i, in order for $D(x||x\prime)$ to be maximized, all $D(x^i||x\prime^i)$ need to be maximized, which means

$$MAX(D(x||x\prime)) = \sum_{i=1}^{M} MAX(D(x^i||x\prime^i)) = M * MAX(D(x^i||x\prime^i)) \tag{6}$$

In other words, there is $F^2 = M * F\prime^2$ and $F\prime = \frac{F}{\sqrt{M}}$can be obtained after deformation. In the same way, we can have $L\prime = \frac{L}{\sqrt{M}}$.

Returning to the proof of Theorem 1, we substitute $F\prime$,$L\prime$ into Inequality 4 and

$$\begin{aligned} R[X] &\leq M * 4F\prime L\prime\sqrt{2(S_{max}+1)PT} \\ &= M * 4\frac{F}{\sqrt{M}}\frac{L}{\sqrt{M}}\sqrt{2(S_{max}+1)PT} \\ &= 4FL\sqrt{2(S_{max}+1)PT} \leq o(T) \end{aligned} \tag{7}$$

Until now, we have completed the proof of Theorem 1, which means $\tilde{x}_t$ converges in expectation to the minimizer $x^\star$.

V. Implementation

ROG is implemented as an optimizer in PyTorch [21] with nearly 1200 lines of code. ROG exposes similar APIs as existing PyTorch optimizers (`torch.optim` [43]), so that it can be integrated by simply replacing the application's original optimizer with ROG's optimizer. Under the hood, ROG launches a parameter server on one of the devices to keep track of the training process among all the devices. On each device, ROG transparently inspects the underlying tensors storing parameters of the model and tracks each row's versions. Each time `optimizer.step()` is called, ROG updates the parameters and row versions of the local model, and synchronizes with the parameter server according to the ATP protocol.

During the synchronization process, the communicated gradients are compressed before transmission and decompressed after reception using the lossless one-bit compression algorithm described in [22] which typically reduces the transmission volume to 3.2% of the uncompressed counterpart. Our implemented compression process is as follows: the gradients are first compressed to one-bit tensors as defined in [22] on GPU (or CPU on devices without a GPU) and these tensors are then serialized and moved to CPU using cupy.packbits() [44] (or numpy.packbits() [45] on devices without a GPU) for transmission. The decompression process is exactly the reversion of the compression process. For the underlying optimizer, we implemented the block-wise distributed SGD-momentum algorithm in [22] and integrated it with [46] which supports staleness and local updates in SGD-momentum algorithms without damaging convergence.

In Speculative Transmission of ATP, we implemented SendWithTimeout() that enforces a time limit for the transmission and discards the ongoing transmission if the time limit is reached. The enforcement of the time limit is simply accomplished with *socket*: setting a timeout with $socket.settimeout()$ and then transmitting with $socket.sendall()$. One issue is that once the ongoing transmission is discarded, it is difficult for the receiver to be aware of the ending of the transmission and the discarded transmission can bring many fragments of incomplete information in the buffer of the receiver. To cope with it, we wrap such transmission with several unique bytes at both the beginning and the ending of the transmission, so that the receiver can be aware of the start and ending of the transmission once it retrieves these unique bytes and the fragments are skipped.

VI. Evaluation

Testbed. The evaluation was performed on five devices consisting of three four-wheel robots and two laptops. Each robot was equipped with an NVIDIA Jetson Xavier NX [47] and each laptop was equipped with an Intel Core i7-8565U CPU@1.80GHz CPU and a 940mx GPU (weaker than the NVIDIA Jetson Xavier NX in computation power). One laptop was chosen as the parameter server and directly connected to all other devices by enabling an IEEE802.11ac [48] hotspot over 80MHz channel at 5GHz frequency. Since the heterogeneity in computation power among the devices is out of our scope, we adopted dynamic batching in [49] to make all the involved devices spend equal time computing gradients in each iteration (see Table II).

Experiment Scenarios. Our first evaluated online training application paradigm is referred to as coordinated robotic unsupervised domain adaptation (CRUDA): a team of robots are recognizing the images of objectives captured by their cameras with a shared objective recognition model, and the recognition accuracy is accidentally degraded by environmental noises (domain shift) such as fog. Such a paradigm is fundamental and representative [50], [51], [52] in typical robotic application scenarios including search, rescue, surveillance, and field exploration. In these scenarios, powerful datacenter servers are typically unavailable due to the lack of internet access in damaged buildings and outdoor fields. To recover the recognition accuracy as soon as possible, the team of robots adapt the shared model to the noises via wireless distributed training on collected noised images. We assume the dataset with noise for online adaptation can be generated following the unsupervised domain adaptation methods [25], [26], which typically train the model on generated adversarial (noised) examples for adaptation; we generated these adversarial examples by adding noise to the original datasets for simplicity.

The second paradigm is referred to as coordinated robotic implicit mapping and positioning (CRIMP): a team of robots continuously collect images of their surroundings through their cameras; meanwhile, the team of robots cooperatively constructs a shared implicit map (a machine learning model representing the 3D map of an area) over the collected images and positions themselves in the shared map. This is also an important task for robotics, as it provides not only 3D reconstruction of the area of interest, but also the real-time positioning information of the robots which is essential for many robotic tasks such as navigation and exploration. The major metric we use for CRIMP is the trajectory error (i.e., the error between the ground-truth positions of the robots and their predicted positions).

Experiment Environments. We setup two real-world environments for our evaluation, namely *indoors* and *outdoors*. In the *indoors* scenario, robots move around in our laboratory with desks and separators interfering with wireless signals. In the *outdoors* scenario, robots move around in our campus garden with trees and bushes interfering with wireless signals and this scenario imposes higher level of instability as discussed in Sec. II-B.

Baselines. We compared ROG with BSP [12], SSP [13] and the framework proposed in [19] (referred to as FLOWN). FLOWN is one of the most SOTA scheduling-based methods

specified for distributed training over unstable wireless networks.

Datasets. For CRUDA, we use the well-studied Fed-CIFAR100 [53], [54] as the image dataset of objectives, with 50000 samples (100 types and each has 500 images) for training and 10000 samples (100 images for each type) for testing. This dataset is also plausible for simulating the real-world unbalanced data distribution by partitioning the images into 500 shards using the Pachinko Allocation Method [55] and we equally divided the dataset to four parts without overlap, each for one of the workers. We follow the methods of DeepTest [56], a DNN testing framework, to add noises to the Fed-CIFAR100 dataset to simulate fog and brightness changes. For CRIMP, we use a short sequence of 500 continuous images captured inside an apartment from the ScanNet dataset [57] and separate the sequence into several continuous sequences for each robot. One of the images is fixed and shared among all the robots as the shared starting point of mapping and positioning.

Models. We choose different model sizes for the two applications respectively to evaluate how ROG performs under different communication data volume. For CRUDA, we choose ConvMLP [41] as the objective recognition model as it achieves both lightweight (total gradients are sized 65 MB before compression and 2.1MB after compression) and high recognition accuracy (89.13%) on the Fed-CIFAR100 dataset. Our added noise leads to a lowered recognition accuracy (52.88%) of ConvMLP and we need to online train the ConvMLP model to recover its accuracy. For CRIMP, we choose nice-slam [58], one of the SOTA implicit mapping and positioning methods. The nice-slam model we used is sized 24.2MB before compression and 0.76MB after compression, which is much smaller than the size of ConvMLP.

The default training configuration of ConvMLP [41] and statistics are listed in Table II and we used the default training configuration of the demo of nice-slam [28]. Note that we include time cost for compression and decompression in the computation time.

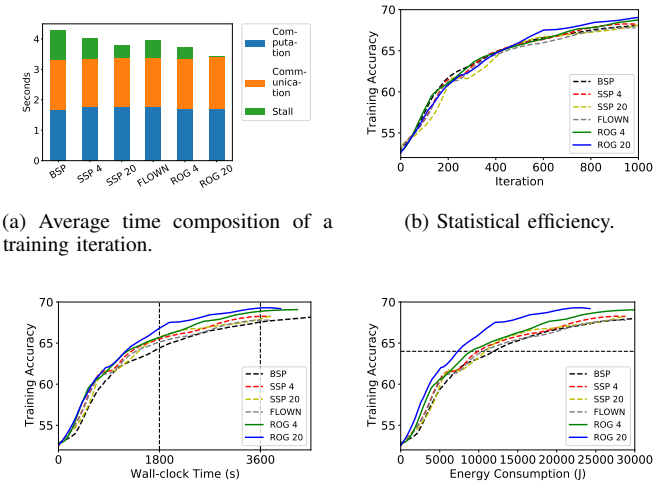
batch size (robot)	batch size (laptop)	learning rate	compress + decompress time cost
24	16	1e-6	0.42s to 0.51s

Table II: Default Setup

The evaluation questions are as follows:

- RQ1: How does ROG benefit real-world robotic applications compared to baseline systems in terms of training accuracy and power consumption by fulfilling **3Rs**?
- RQ2: How does ROG handle the unstable wireless networks?
- RQ3: How sensitive is ROG to different batch sizes, different numbers of devices, and different thresholds?
- RQ4: What are the limitations and potentials of ROG?

A. End-to-end Performance



(a) Average time composition of a training iteration.

(b) Statistical efficiency.

(c) Training accuracy against wall-clock time.

(d) Energy consumption against training accuracy.

Figure 6: Comparison between ROG and the baselines with CRUDA in *indoors*.

We first compare the training accuracy/trajectory error of the two applications and energy consumption of their training processes over time under different training systems. The training accuracy of CRUDA and trajectory error of CRIMP were both obtained by checkpointing and validating the training model on each worker every 50 training iterations and then averaging the validated accuracy/trajectory error among the workers. We measured and averaged the energy consumption of the whole development board including CPU, GPU, memory and wireless card on all robots with jtop [59], a well-recognized monitoring tool for NVIDIA Jetson boards. Since jtop only reports transient power consumption, we approximated the energy consumption of each run by recording the power consumption at 10 Hz and calculating the total power consumption with numerical integraion.

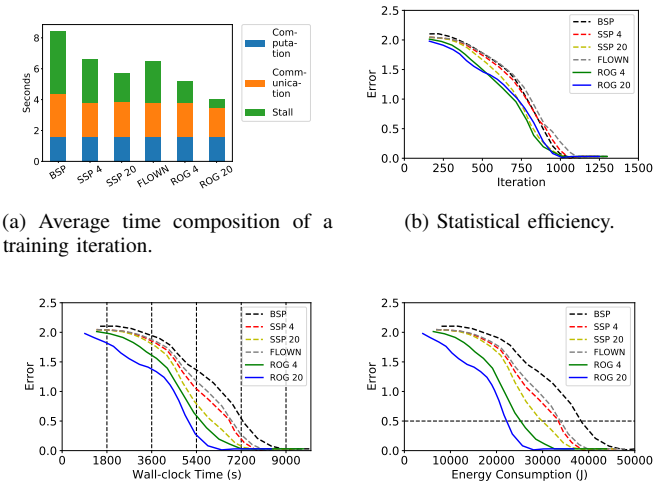
CRUDA. Our evaluation results of CRUDA in Fig. 1 and Fig. 6 show that ROG achieved both high training accuracy and high energy efficiency. When training for 30 minutes, ROG achieved 3.3%~6.8% higher accuracy than the baselines in *outdoors* in Fig. 1c, and up to 1.8% accuracy gain in *indoors* in Fig. 6c due to reduced instability. When training for 60 minutes, ROG achieved 4.9%~6.5% higher accuracy than the baselines in *outdoors*. Such accuracy gains have been reported as critical in real-world robotic applications [50], [60]. In terms of energy consumption, when the training model reached an accuracy of 64.0%, ROG saved 20.4%~50.7% of the battery energy in *outdoors* (Fig. 1d). The reduction of energy consumption was up to 41.3% in *indoors* still due to reduced instability.

The key reason for ROG's high performance is its mitigation of stall time (high training throughput, **R2**) without sacrificing statistical efficiency (high statistical efficiency, **R3**) in various environments with different levels of instability (robustness, **R1**). Fig. 1a and Fig. 6a show the recorded average time composition of a training iteration where a shorter total time duration of a training iteration implies higher training throughput. In a training iteration while all systems took almost the same time computing gradients and communicating the compressed gradients, BSP, SSP-4, SSP-20 and FLOWN suffered at least 4.8s stall in *outdoors* and 0.4s stall in *indoors*. ROG reduced the stall time by 49.1% to 86.5% in *outdoors* and by 42.4% to 97.6% in *indoors*. ROG achieved less stall time reduction in *indoors* because the wireless networks are less unstable in *indoors*. By breaking down the whole model into rows and transmitting at the row granularity, ROG prevents transiently degraded bandwidth from blocking the overall training process.

Fig. 1b and Fig. 6b show that ROG achieved similar statistical efficiency as BSP. In order to avoid being blocked by degraded bandwidth during synchronization, it is inevitable to reduce the transmission traffic volume and postpone the synchronization of some rows, which causes staleness in untransmitted gradients. To minimize its impact on statistical efficiency, ROG's ATP identifies rows with large gradients and prioritizes them. Therefore, even if stragglers transmit fewer gradients than non-stragglers, important changes to the model are always synced, resulting in a comparable statistical efficiency as BSP.

To further understand the energy consumption statistics, we identify three major states, namely *computation*, *communication*, and *stall* of a system during training, and measure the power consumption of each state. We obtained the power consumption of different states in Table III by matching power consumption records with the training system status log. There was minor (below 5%) difference across all the evaluated systems, since all the systems do not change how computation and stall states behave, while the overhead of scheduling during communication is negligible. The stall state power was nearly 30% of the computation state power, since chips like CPU, GPU, and memory consume non-negligible power even when not computing (i.e., in stall state) due to the static power consumption rooted in transistors' leakage current [32]. Note that communication and stall have similar power consumption, this may be due to the relatively low wireless transmission rate (compared to high-speed datacenter networks), involving little energy-consuming operations. Since ROG reduced stall time, the corresponding power consumed during stall was reduced accordingly, accounting for ROG's high energy efficiency.

CRIMP. We mainly evaluated CRIMP in *outdoors*, as shown in Fig. 7. As shown in Fig. 7c and Fig. 7d, ROG also achieved similar high training accuracy (less trajectory error)



(a) Average time composition of a training iteration.

(b) Statistical efficiency.

(c) Training accuracy against wall-clock time.

(d) Energy consumption against training accuracy.

Figure 7: Comparison between ROG and the baselines with CRIMP in *outdoors*.

	computation	communication	stall
Power (W)	13.35	4.25	4.04

Table III: Power (Watt) in different states.

and high energy efficiency in CRIMP. When training for 30 minutes, ROG reduced 6%~13% trajectory error compared with the baselines in Fig. 7c. After training for 60 minutes, the reduction of trajectory error of ROG over the baselines increased to 16%~30%. Also, the energy consumption reduction of ROG over the baselines is outstanding: 32%~41% less energy to reach the trajectory error of 0.5 in Fig. 7d. Note that while the model size of CRIMP amounts to only one third of CRUDA and its average communication time is reduced, the straggler effect of CRIMP is still severe with stall time taking up 60% of the communication time in BSP in Fig. 7a. That is because with a smaller model, the computation time in a training iteration is also reduced and communication remains the bottleneck of the training process.

B. Micro-Event Analysis

To further understand the performance gain of ROG, we recorded the real-time bandwidth and how ROG responded to it by adjusting the percentage of rows to be transmitted out of all rows in each iteration (referred to as transmission rate) on one robot, as shown in Fig. 8. How many training iterations that this robot fell behind the fastest worker is also recorded (referred to as staleness). Since proactive methods (e.g., measuring with `iperf`) would affect the application traffic and bandwidth, we passively measured the real-time bandwidth with the expected throughput reported by `iw` [61]. Note that `iw`'s output is an estimation of the physical

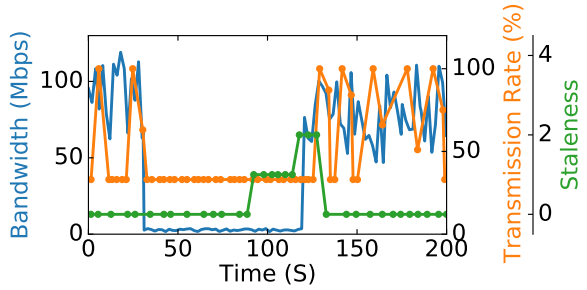


Figure 8: Real-time bandwidth and the percentage of rows transmitted by ROG

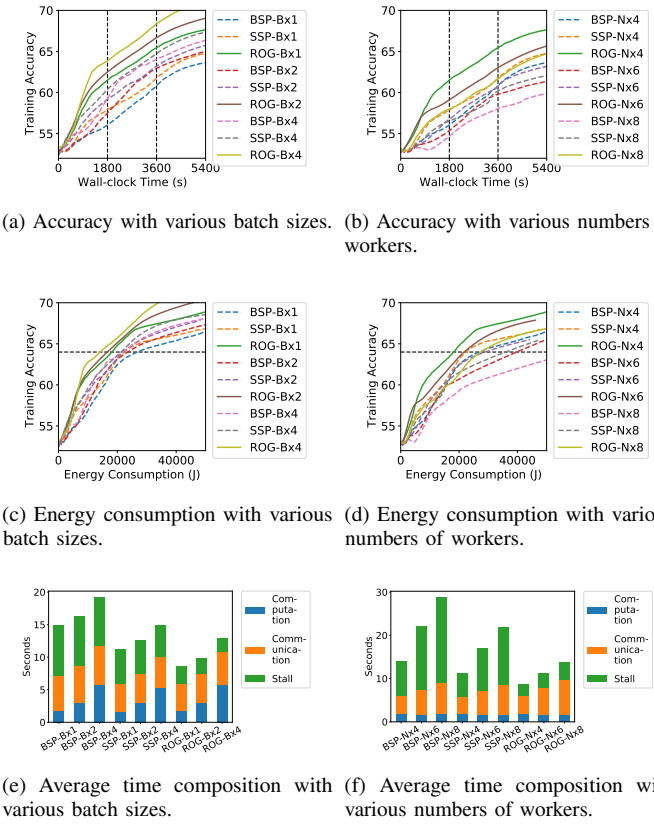
layer bitrate which deviates from the actual bandwidth the application could exploit, we normalize the output with its average.

When bandwidth was fluctuating in the former part of Fig. 8, ROG responded immediately and adjusted the transmission rate on a robot accordingly. This aligned the transmission time between this robot and the fastest, avoiding possible straggler effect and the staleness was in a low level (0 to 1). In this way, ROG prevented a robot from straggling and stalling the training process. When bandwidth degraded to an extremely low level and lasted for a long time in the middle part of Fig. 8, it was impossible to perform even minimal necessary synchronization under such conditions, and no system could keep in sync. Thus staleness slowly accumulated on this robot. When bandwidth recovered in the latter part of Fig. 8, this robot caught up quickly (staleness decreased) because it was allowed to transmit partial of its rows.

C. Sensitivity Studies

batch size. We varied the batch size (x2, x4) of training in CRUDA in *outdoors* to examine how ROG performs with different ratios of computation and communication, as shown in the left column of Fig. 9. As FLOWN typically achieved performance between SSP and BSP, we omit it in the following sections for simplicity. When the batch size increased, the computation time proportionally increased and thus communication time would take a smaller portion in a training iteration. In this case, the straggler effect will be less severe (stall time decreased in the baselines) and ROG's potential gain over the baselines is limited. When training for 30 minutes, ROG achieved 5.3% accuracy gain over the baselines and 30.3% energy consumption reduction when training accuracy reached 64% in the doubled batch size case; When the batch size was increased to four times, ROG achieved 3.5% accuracy gain over the baselines and 33.7% energy consumption reduction when training accuracy reached 64%.

Number of workers. Increasing the number of training workers (4, 6, 8 workers) in CRUDA caused more severe



(a) Accuracy with various batch sizes. (b) Accuracy with various numbers of workers.

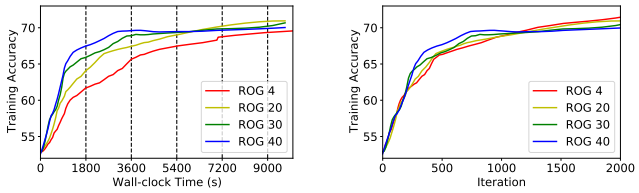
(c) Energy consumption with various batch sizes. (d) Energy consumption with various numbers of workers.

(e) Average time composition with various batch sizes. (f) Average time composition with various numbers of workers.

Figure 9: Sensitivity Studies about different batch sizes (left column) and worker numbers (right column)

straggler effect, as shown in the right column of Fig. 9. First, as the workers all share a same wireless channel, varied number of workers involved will incur traffic volume proportional to the number of workers, causing communication time to take a larger portion in a training iteration. Second, the contention for wireless channel among workers is an extra source of instability, deteriorating the straggler effect. In this case in Fig. 9, when training for an hour, ROG achieved 3.0% accuracy gain over the baselines and 48.1% energy consumption reduction when training accuracy reached 64.2% in the 6 workers case; When the number of workers was increased to 8, ROG achieved 3.7% accuracy gain over the baselines and 55.1% energy consumption reduction when training accuracy reached 64%.

Threshold. We empirically evaluated ROG's performance under a wider variety of staleness thresholds with the default training configuration with CRUDA, as shown in Fig. 10. From Fig. 10 we can learn that there is a tradeoff between training speed and final training accuracy when using different thresholds in ROG: while a large threshold (30 or 40) brings higher training throughput and potentially even higher statistical efficiency in the early stage of training, a too



(a) Training accuracy against wall-clock time.

(b) Statistical efficiency.

Figure 10: Sensitivity Studies about different thresholds

large threshold will degrade the statistical efficiency in the late stage of training and lead to slightly degraded final training accuracy (similarly reported by SSP with different staleness thresholds [13]). The reason could be although ROG limits divergence of training models on different workers and guarantees convergence, a large threshold inevitably leads to larger level of divergence among the models and leads to suboptimal final training accuracy. Depending on whether the training task requires extra fast training speed or high training quality, there would be an optimal threshold selection for it and we leave automatic finding the optimal threshold as future work.

D. Lessons learned

Wireless distributed training over robots still faces many challenges. During the implementation and evaluation of ROG, we find that wireless distributed training over robots is challenging both systematically and algorithmically. Systematically, unlike GPU clusters equipped with fast interconnects such as InfiniBand [62], robots lack fast and stable network connection between each other for model synchronization and lack enough power for long term training, which are partially mitigated by ROG. Algorithmically, the collected data of different robots are typically non-IID (e.g., different robots are surveilling and capturing data from different parts of an area), while there is not yet a well-recognized method for distributed training over non-IID datasets.

Generalizability of ROG. Due to limitations of our hardware, we did not evaluate ROG's performance on a wider variety of wireless networks such as 5G [63] or WiMAX [64], but only the most common and easily accessible Wi-Fi networks on robots under different environments. However, while these various wireless networks differ in throughput and communication range, decay of wireless signals due to varying distance or occlusion is still a common issue among them. The resulting throughput fluctuation will still cause straggler effect in these wireless networks, where ROG will be beneficial. Overall, ROG is optimized for distributed training over any wireless LAN with frequent bandwidth fluctuation and we leave the evaluation of ROG over a wider variety of wireless networks as future work.

Finer granularity brings extra management overhead and transmission overhead. While Finer granularity in ROG enables more flexibility in scheduling to adapt to the instability of real-world robotic IoT networks, it also brings extra management overhead (e.g., index) and transmission overhead in the wireless distributed training process. Although we minimized these overheads by choosing a balanced granularity of rows and enabling speculative transmission, such overhead cannot be eliminated and potentially limits the performance gain of ROG over the baselines.

Future work. We would like to apply and evaluate ROG in a wider variety of online learning robotic tasks and environments in the future. Also, it is of interest to explore further improvements of ROG such as pipelining communication and computation on a robot in the training process as described in [65] or even decoupling communication and computation. We believe such investigation will enable even faster and more robust wireless distributed training in real-world Robotic IoT Networks.

VII. Conclusion

In this paper, we present ROG, a Row-granulated distributed training system optimized for robotic IoT networks. By breaking up the granularity of model synchronization into rows and applying adaptive scheduling the transmission of each row, ROG is able to balance the transmission time among different workers under unstable wireless bandwidth and prevent the straggler effect from causing stall in workers. In this way, ROG optimizes training throughput while providing high statistical efficiency and achieves high accuracy and high energy efficiency in the distributed training. We envision that ROG will nurture diverse ML applications deployed on mobile robots in the field, making them fast adapt to changing environments under an extremely unstable local wireless network without being affected by straggler effect.

Acknowledgments

We thank all anonymous reviewers for their helpful comments. This work is supported in part by a Huawei Flagship Research Grant in 2021, the HKU-SCF FinTech Academy R&D Funding Scheme in 2021 and 2022, HK RGC GRF (17202318, 17207117), HK ITF (GHP/169/20SZ), NSFC (No. 62132009), the Pujiang Lab (Heming Cui is a courtesy researcher in this lab), and the HKU and IS-CAS Joint Lab for Intelligent System Software.

Appendix

A. Abstract

This artifact presents the availability, functionality and key reproducible results of this paper (ROG: A High Performance and Robust Distributed Training System for Robotic IoT). We provide bash and python scripts which would reproduce

our results on at least 5 robots, PCs, or laptops, each with at least 8GB CPU memory or 8GB GPU memory.

B. Artifact check-list (meta-information)

- **Program: Docker [66]; Traffic Control (TC) [67]**
- **Model: ConvMLP-M [41], [68].**
- **Data set: Fed-CiFar100 [53], [54]; scripts provided to generate the noised dataset from DeepTest [56].**
- **Run-time environment: Arm64 or X86/64; Ubuntu18.04; Python3.6**
- **Hardware: NVIDIA Jetson Xavier NX [47], PCs, or laptops, each with at least 8GB CPU memory or 8GB GPU memory; Wireless network interface controller (WNIC).**
- **Execution: Bash and python scripts.**
- **Output: Raw data and figures reproducing results of CRUDA in Sec. VI-A and Sec. VI-C.**
- **How much disk space required (approximately)?: 30GB disk space on each device involved.**
- **How much time is needed to prepare workflow (approximately)?: 1~2 hours.**
- **How much time is needed to complete experiments (approximately)?: 2~3 days.**
- **Publicly available?: Yes. https://github.com/hku-systems/ROG.**
- **Workflow framework used?: PyTorch [21].**
- **Archived (provide DOI)?: Yes. https://doi.org/10.5281/zenodo.6941140.**

C. Description

1) How to access: We provide access to a well-prepared environment (a cluster with 2 PC, 1 laptop and 2 NVIDIA Jetson Xavier NX) with code we used a via ZeroTier network [69] during artifact evaluation, to ease the burden of configuring. After joining our ZeroTier network, you can access the devices we used by ssh commands. All code and data are also publicly available at https://github.com/hku-systems/ROG and https://doi.org/10.5281/zenodo.6941140.

2) Hardware dependencies: At least 5 NVIDIA Jetson Xavier NX [47], PCs, or laptops, each with a WNIC and at least 8GB CPU memory or 8GB GPU memory are required. We recommend all the devices involved are homogeneous but we also support heterogeneous settings. Mobility is not necessary for the devices, since we provide scripts using TC [67] to reproduce the real-time bandwidth capability recorded in the identical settings in Sec. VI.

3) Software dependencies: Ubuntu18.04 with TC enabled [67]. Docker [66] is required to build the runtime environments with the dockerfile that we provide. The docker images that we rely on are nvidia/cuda:10.2-runtime-ubuntu18.04 for X86/64 environments and dustynv/ros:foxy-pytorch-l4t-r34.1.1 for Arm64 environments.

4) Data sets: The used Fed-CiFar100 dataset [53], [54] is pulled from Tensorflow [70] and noised with scripts modified from DeepTest [56].

5) Models: The used ConvMLP-M model and its related code are pulled from [68], the official GitHub repository of [41].

D. Installation

To ease the burden of configurations, we provide access to a well-prepared environment. Installation from a clean environment is also possible: download the artifact or clone the GitHub repository; download the data set and the model we used using utils/download_data.sh if they are not in place; change directory to the downloaded repository, build the docker image and then run the built image:

```
$ git clone https://github.com/microP156/rog
$ cd ROG
$ bash utils/download_data.sh
  # For X86/64 devices
$ docker build -f Dockerfile_x86-64 -t ROG
  # For Arm64 devices please replace
  # Dockerfile_x86-64 with Dockerfile_arm64
$ docker run -td --name rog -v /tmp/.X11-unix --gpus
    all -e DISPLAY=:0 --privileged -v /dev:/dev --
    network=host --cap-add=NET_ADMIN --ipc=host
    -v "$PWD":/home/work ROG bash
```

Some extra configurations are required:

- Enable WiFi hotspot on one of the devices involved and connect all other devices to the hotspot. The device enabling hotspot will act as the parameter server and all others act as workers.
- In the file scripts/run.py on the parameter server, replace the IP addresses and WNIC names with the wireless IP addresses and WNIC names of all the devices as instructed in the file. Also replace the code repository location of workers in that file with the actual location you cloned the repository on the workers.
- Set up SSH key based authentication between the parameter server and the workers on the hotspot network, so that you do not need to enter your username and password multiple times.

E. Experiment workflow

If you have installed and configured the evaluation environments properly (or in the environment we provide), the experiments can be started up by simply running the script run_all.sh on the parameter server (the device with hotspot enabled) with

```
$ bash run_all.sh
```

The default evaluation items will be run consecutively as defined in the script without any extra operations.

The default evaluation items include end-to-end evaluation of ROG on CRUDA in the outdoor environment (Sec. VI-A), ROG's performance with varied thresholds and varied batch sizes (Sec. VI-C). Note that we omit the cases with different numbers of workers since we are incapable of keeping that many devices online in our laboratory.

F. Evaluation and expected results

Raw evaluation results will be generated in the ./result repository; at the end of each evaluation item, figures concluding the results will be drawn automatically in the ./figure repository. The lines in the figures are expected to be similar to those in our evaluation.

G. Notes

We recommend disabling the mobility of all the devices involved during artifact evaluation (the devices we provide are stationary and being charged); to introduce instability of wireless networks in artifact evaluation, we instead provide scripts based on TC [67] to replay on each device the real-time bandwidth that we recorded in the identical settings in our evaluation. That is because we need to ensure reproducibility of the results as well as reliability of the devices during artifact evaluation, since the moving devices can easily run out of energy or crash into obstacles if not being supervised.

References

[1] J. P. Queralta, J. Taipalmaa, B. Can Pullinen, V. K. Sarker, T. Nguyen Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," vol. 8, pp. 191 617–191 643, publisher: IEEE.

[2] K.-M. Yang, J.-B. Han, and K.-H. Seo, "A multi-robot control system based on ROS for exploring disaster environment," in *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*, pp. 168–173.

[3] N. Kourtellis, K. Katevas, and D. Perino, "FLaaS: Federated Learning as a Service," in *Proceedings of the 1st Workshop on Distributed Machine Learning*, ser. DistributedML'20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 7–13. [Online]. Available: https://doi.org/10.1145/3426745.3431337

[4] M. Everett, Y. F. Chen, and J. P. How, "Collision Avoidance in Pedestrian-Rich Environments With Deep Reinforcement Learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021, conference Name: IEEE Access.

[5] J. Woo and N. Kim, "Collision avoidance for an unmanned surface vehicle using deep reinforcement learning," *Ocean Engineering*, vol. 199, p. 107001, Mar. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801820300792

[6] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, Oct. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218306684

[7] G. Wilson and D. J. Cook, "A Survey of Unsupervised Deep Domain Adaptation," *ACM Transactions on Intelligent Systems and Technology*, vol. 11, no. 5, pp. 51:1–51:46, Jul. 2020. [Online]. Available: https://doi.org/10.1145/3400066

[8] J. Park, S. Samarakoon, A. Elgabli, J. Kim, M. Bennis, S.-L. Kim, and M. Debbah, "Communication-Efficient and Distributed Learning Over Wireless Networks: Principles and Applications," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 796–819, May 2021, conference Name: Proceedings of the IEEE.

[9] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed Learning in Wireless Networks: Recent Progress and Future Challenges," *arXiv:2104.02151 [cs, math]*, Apr. 2021, arXiv: 2104.02151. [Online]. Available: http://arxiv.org/abs/2104.02151

[10] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," pp. 583–598. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[11] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/1ff1de774005f8da13f42943881c655f-Abstract.html

[12] A. V. Gerbessiotis and L. G. Valiant, "Direct bulk-synchronous parallel algorithms," in *Algorithm Theory — SWAT '92*, ser. Lecture Notes in Computer Science, O. Nurmi and E. Ukkonen, Eds. Springer, pp. 1–18.

[13] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ML via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems*, vol. 26. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper/2013/hash/b7bb35b9c6ca2aee2df08cf09d7016c2-Abstract.html

[14] A. Masiukiewicz, "Throughput comparison between the new HEW 802.11ax standard and 802.11n/ac standards in selected distance windows," vol. 65, no. 1, pp. 79–84, number: 1. [Online]. Available: http://www.ijet.pl/index.php/ijet/article/view/10.24425-ijet.2019.126286

[15] Y. Pei, M. W. Mutka, and N. Xi, "Connectivity and bandwidth-aware real-time exploration in mobile robot networks," vol. 13, no. 9, pp. 847–863, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcm.1145. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.1145

[16] N. I. Sarkar and O. Mussa, "The effect of people movement on wi-fi link throughput in indoor propagation environments," in *IEEE 2013 Tencon - Spring*, pp. 562–566.

[17] M. Ding, P. Wang, D. Lopez-Perez, G. Mao, and Z. Lin, "Performance impact of LoS and NLoS transmissions in dense cellular networks," vol. 15, no. 3, pp. 2365–2380. [Online]. Available: http://arxiv.org/abs/1503.04251

[18] H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Exploiting bounded staleness to speed up big data analytics," pp. 37–48. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/cui

[19] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A Joint Learning and Communications Framework for Federated Learning Over Wireless Networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, Jan. 2021, conference Name: IEEE Transactions on Wireless Communications.

[20] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," vol. 34, no. 5, pp. 7179–7186. [Online]. Available: https://aaai.org/ojs/index.php/AAAI/article/view/6207

[21] (2021) PyTorch. [Online]. Available: https://www.pytorch.org

[22] J. Sun, T. Chen, G. Giannakis, and Z. Yang, "Communication-Efficient Distributed Learning via Lazily Aggregated Quantized Gradients," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://papers.nips.cc/paper/2019/hash/4e87337f366f72daa424dae11df0538c-Abstract.html

[23] E. Vidal, J. D. Hernández, N. Palomeras, and M. Carreras, "Online robotic exploration for autonomous underwater vehicles in unstructured environments," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, pp. 1–4.

[24] Y. Zhang, D. Shi, Y. Wu, Y. Zhang, L. Wang, and F. She, "Networked Multi-robot Collaboration in Cooperative–Competitive Scenarios Under Communication Interference," in *Collaborative Computing: Networking, Applications and Worksharing*, H. Gao, X. Wang, M. Iqbal, Y. Yin, J. Yin, and N. Gu, Eds. Cham: Springer International Publishing, 2021, vol. 349, pp. 601–619.

[25] X. Liu, Z. Guo, S. Li, F. Xing, J. You, C.-C. J. Kuo, G. El Fakhri, and J. Woo, "Adversarial unsupervised domain adaptation with conditional and label shift: Infer, align and iterate," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10 347–10 356, ISSN: 2380-7504.

[26] M. Awais, F. Zhou, H. Xu, L. Hong, P. Luo, S.-H. Bae, and Z. Li, "Adversarial robustness for unsupervised domain adaptation," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 8548–8557, ISSN: 2380-7504.

[27] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison, "iMAP: Implicit mapping and positioning in real-time," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, pp. 6209–6218. [Online]. Available: https://ieeexplore.ieee.org/document/9710431/

[28] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "NICE-SLAM: Neural implicit scalable encoding for SLAM," number: arXiv:2112.12130. [Online]. Available: http://arxiv.org/abs/2112.12130

[29] "iPerf - Download iPerf3 and original iPerf pre-compiled binaries." [Online]. Available: https://iperf.fr/iperf-download.php

[30] Datacenter traffic control: Understanding techniques and tradeoffs | IEEE journals & magazine | IEEE xplore. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8207422

[31] On the shoulders of giants: recent changes in internet traffic. [Online]. Available: http://blog.cloudflare.com/on-the-shoulders-of-giants-recent-changes-in-internet-traffic/

[32] Nam Sung Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, Jie S. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," vol. 36, no. 12, pp. 68–75. [Online]. Available: http://ieeexplore.ieee.org/document/1250885/

[33] X. Chen, L. Zhang, X. Wei, and X. Lu, "An effective method using clustering-based adaptive decomposition and editing-based diversified oversamping for multi-class imbalanced datasets," vol. 51, no. 4, pp. 1918–1933. [Online]. Available: https://doi.org/10.1007/s10489-020-01883-1

[34] S. Bhattacharya, V. Rajan, and H. Shrivastava, "ICU mortality prediction: A classification algorithm for imbalanced datasets," vol. 31, no. 1, number: 1. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10721

[35] H. Kaur, H. S. Pannu, and A. K. Malhi, "A systematic review on imbalanced data challenges in machine learning: Applications and solutions," vol. 52, no. 4, pp. 79:1–79:36. [Online]. Available: https://doi.org/10.1145/3343440

[36] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" in *Structural, Syntactic, and Statistical Pattern Recognition*, ser. Lecture Notes in Computer Science, A. Fred, T. M. Caelli, R. P. W. Duin, A. C. Campilho, and D. de Ridder, Eds. Springer, pp. 806–814.

[37] W. Shi, S. Zhou, and Z. Niu, "Device Scheduling with Fast Convergence for Wireless Federated Learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, pp. 1–6, iSSN: 1938-1883.

[38] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "DEEP GRADIENT COMPRESSION: REDUCING THE COMMUNICATION BANDWIDTH FOR DISTRIBUTED TRAINING," p. 14.

[39] "Jetson Modules," Oct. 2020. [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules

[40] "NVIDIA Jetson Xavier NX GPU Specs." [Online]. Available: https://www.techpowerup.com/gpu-specs/jetson-xavier-nx-gpu.c3642

[41] J. Li, A. Hassani, S. Walton, and H. Shi, "ConvMLP: Hierarchical convolutional MLPs for vision." [Online]. Available: http://arxiv.org/abs/2109.04454

[42] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds. Physica-Verlag HD, pp. 177–186.

[43] torch.optim — PyTorch 1.11.0 documentation. [Online]. Available: https://pytorch.org/docs/stable/optim.html

[44] CuPy: NumPy & SciPy for GPU. [Online]. Available: https://cupy.dev/

[45] NumPy. [Online]. Available: https://numpy.org/

[46] W. Liu, L. Chen, Y. Chen, and W. Zhang, "Accelerating federated learning via momentum gradient descent," vol. 31, no. 8, pp. 1754–1766, conference Name: IEEE Transactions on Parallel and Distributed Systems.

[47] "The World's Smallest AI Supercomputer." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/

[48] "IEEE 802.11ac-2013," page Version ID: 1076948038. [Online]. Available: https://en.wikipedia.org/w/index.php?title=IEEE_802.11ac-2013&oldid=1076948038

[49] S. Tyagi and P. Sharma, "Taming resource heterogeneity in distributed ML training with dynamic batching," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 188–194.

[50] M. Wulfmeier, A. Bewley, and I. Posner, "Incremental Adversarial Domain Adaptation for Continually Changing Environments," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 4489–4495, iSSN: 2577-087X.

[51] ——, "Addressing appearance change in outdoor robotics with adversarial domain adaptation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1551–1558, ISSN: 2153-0866.

[52] H.-K. Hsu, C.-H. Yao, Y.-H. Tsai, W.-C. Hung, H.-Y. Tseng, M. Singh, and M.-H. Yang, "Progressive Domain Adaptation for Object Detection," 2020, pp. 749–757. [Online]. Available: https://openaccess.thecvf.com/content_WACV_2020/html/Hsu_Progressive_Domain_Adaptation_for_Object_Detection_WACV_2020_paper.html

[53] (2022) tff.simulation.datasets.cifar100.load_data | TensorFlow federated. [Online]. Available: https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/cifar100/load_data

[54] CIFAR-10 and CIFAR-100 datasets. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[55] W. Li and A. McCallum, "Pachinko allocation: DAG-structured mixture models of topic correlations," in *Proceedings of the 23rd international conference on Machine learning - ICML '06*. ACM Press, pp. 577–584. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143917

[56] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 303–314. [Online]. Available: https://doi.org/10.1145/3180155.3180220

[57] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[58] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys, "Nice-slam: Neural implicit scalable encoding for slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[59] R. Bonghi, "Jetson stats," original-date: 2018-11-24T19:42:07Z. [Online]. Available: https://github.com/rbonghi/jetson_stats

[60] S. Siva and H. Zhang, "Robot perceptual adaptation to environment changes for long-term human teammate following," *The International Journal of Robotics Research*, p. 0278364919896625, Jan. 2020, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364919896625

[61] en:users:documentation:iw [linux wireless]. [Online]. Available: https://wireless.wiki.kernel.org/en/users/documentation/iw

[62] sadmin. InfiniBand - a low-latency, high-bandwidth interconnect. [Online]. Available: https://www.infinibandta.org/about-infiniband/

[63] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," vol. 35, no. 6, pp. 1201–1221, conference Name: IEEE Journal on Selected Areas in Communications.

[64] S. A. Ahson and M. Ilyas, *WiMAX: Standards and Security*, ser. The WiMAX handbook. CRC Press.

[65] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipe-SGD: A decentralized pipelined SGD framework for distributed deep net training," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/2c6a0bae0f071cbbf0bb3d5b11d90a82-Abstract.html

[66] Home - docker. [Online]. Available: https://www.docker.com/

[67] tc(8) - linux manual page. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[68] SHI-labs/convolutional-MLPs: [preprint] ConvMLP: Hierarchical convolutional MLPs for vision, 2021. [Online]. Available: https://github.com/SHI-Labs/Convolutional-MLPs

[69] ZeroTier – global area networking. [Online]. Available: https://www.zerotier.com/

[70] Introduction to TensorFlow. [Online]. Available: https://tensorflow.google.cn/learn