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

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 existence of DP synchronization barriers, 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 accuracy of training model within tight time budget and wastes energy stalling. Existing methods to cope with instability of datacenter networks in distributed training are incapable of handling such straggler effect. Because they are conducting model-granulated transmission scheduling, which is much more coarser-grained than the granularity of transient network instability in real-world robotic IoT networks, making a previously reached scheduling 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 parameter 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 stall, while provably guaranteeing convergence. The evaluation shows that given same training time, ROG achieved up to 4.3% training accuracy gain compared with the baselines and saved 32.5% of the energy to reach the same training accuracy.

1. INTRODUCTION

Critical robotic tasks such as rescue [34] and disaster response [49] are more prevalently leveraging machine learning models (e.g., objective recognition models [26] or action control models [20, 45]) deployed over a team of mobile robots. These models typically require real-time training to adapt pretrained parameters to changing environments [42, 44] (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 the lack of stable internet access. Therefore, such training for critical robotic tasks is often distributedly deployed among the team of robots over robotic IoT networks [15, 32].

Such distributed training typically adopts the parameter server paradigm [28]: 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 [21]) 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, 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). Resilient distributed training is important 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 reach 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 is important for the training model to reach as an high as possible accuracy given a tight training time budget, while preserving battery energy to reach a desired accuracy.

Unfortunately, although such distributed training with BSP empirically achieves high statistical efficiency (**R3**) [22], the instability of real-world robotic IoT networks hinders it from meeting **R1** and **R2** by causing the ***straggler effect*** (Fig. **??**): 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. 2.1) by sharp bandwidth degradation due to movement of the devices [31, 33], occlusion from obstacles [19, 35], 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**).

Although mainly designed for datacenter networks, Stale Synchronous Parallel (SSP) [18, 22] has the potential to resolve such straggler effect. SSP allows non-stragglers to continue computing without latest gradients from stragglers

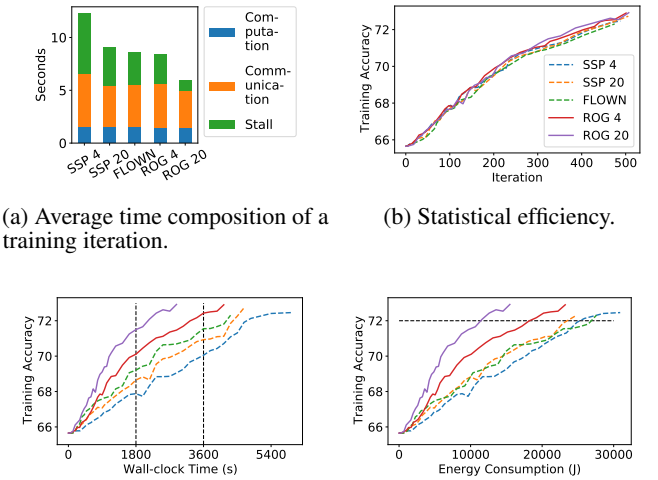
and only stall when the gradients from stragglers fall behind (stale) for a preset number of iterations (staleness threshold).

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 [22], 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 55.7% of the duration of a training iteration); a larger threshold (20) reduced the stall time to 20.3% of a training iteration, at the cost of lower statistical efficiency (reduced by on average 36.2%) compared with BSP. As a result, given 30 minutes training time, they reached xx% and xx% lower training accuracy and consumed xx% and xx% more energy to reach such accuracy compared with an ideal case (BSP without straggler effect).

Subsequent studies [16, 24] extend SSP by dynamically assigning (scheduling) the staleness threshold to simultaneously fulfill **R1** and **R2**: higher staleness threshold for devices estimated to have low bandwidth and less contribution to training accuracy; small threshold for the opposite. However, they are designed for datacenter networks and edge networks and cannot fulfill **R1**, because the random and rapid nature (see Sec. 2.1) 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 varying bandwidth during the actual transmission. In our evaluation (Fig. 1), compared with the ideal case, such methods caused stall time to on average take up 62.3% of a training iteration and 5.3% lower training accuracy after training for 30 minutes; 35% more energy was consumed to reach such accuracy .

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 contribution 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.

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 level of granularity to break up the parameters of an ML model: layers (matrixes), rows (matrixes rows) and elements (individual parameters). We find that 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 at size and is still comparable with the granularity of bandwidth fluctuation



(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 in the outdoor environments.

(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 to different rows within a same device. RSP guarantees convergence by limiting the divergence of rows within the staleness threshold and thus limiting the divergence of the whole training model on different devices. RSP provably achieves the same convergence guarantee as SSP (see Sec. 4.3).

The second challenge is under RSP, how to properly schedule the transmission of each row from different devices to fulfill **3Rs**. In view of the fluctuating bandwidth in real-world robotic IoT networks, scheduling based on transmission volume cannot achieve **R1**. Instead, ROG adaptively aligns the transmission time of each device by speculatively transmitting each row with *Adaptive Transmission Protocol* (ATP). In an training iteration, ATP closely 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., absolute value of the gradients), minimizing the chance of stall and accelerating convergence (**R3**).

We implemented ROG in PyTorch [10] and evaluated ROG on a team of mobile robots under different representative

real-world application paradigms with different model sizes. We compared ROG with BSP [21], SSP [22] and a SOTA dynamic threshold method [16] (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 [38] (the compressed gradients of the two applications were only sized 0.1 MByte and 2.1 MByte) to conduct the strictest comparison between ROG and the baselines. Evaluation shows that:

- ROG achieves high accuracy. ROG achieved 2.8% to 3.1% accuracy gain over the baselines after training for 30 minutes. When training for 60 minutes the accuracy gain increased to 2.9% to 3.3%.
- ROG is energy-efficient. When the training model reached a same high accuracy, ROG reduced battery energy consumption by 29.5% to 32.4% compared with the baselines.
- ROG is scalable. When increasing the number of robots involved or increasing the batchsize of training, ROG still at least achieved 4.2% accuracy gain and 32.5% energy consumption reduction over the baselines.
- ROG is easy to use. It took only tens of lines of code to apply ROG to each of the above 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 [34], disaster response [49], and robot surveillance [41, 51], making them fast adapt to changing environments under an extremely instable local wireless network without being affected by straggler effect. ROG's code is released on https://github.com/microP156/rog.

In the rest of this paper, we introduce the background of this paper in Sec. 2, give an overview of ROG in Sec. 3, present detailed design of ROG in Sec. 4, evaluate ROG in Sec. 6, and finally conclude in Sec. 7

2. BACKGROUND

2.1 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.

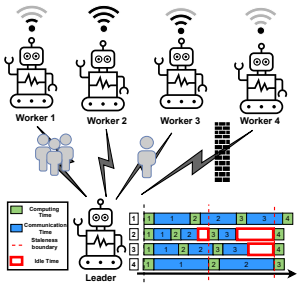


Figure 2: The instability of real-world robotic IoT networks.

This causes divergence in gradient transmission time from different robots and the straggler effect.

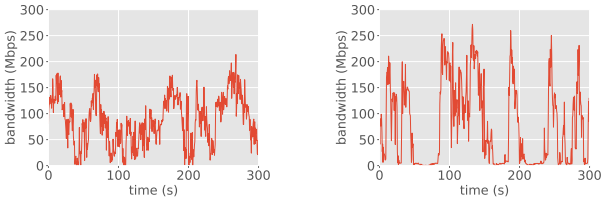
To demonstrate such instability, we set up a robot surveillance task: two four-wheel robots navigate around several given points at 5~40cm/s speed in our lab (indoors) and campus garden (outdoors). Each of the robots carries an NVIDIA Jetson Xavier NX [5], the representative robotic IoT hardware since Jetson modules are well-recognized as the enabler of intelligent robotic IoT applications [43, 48, 50], and Jetson Xavier NX is one of the most recent models [4]. To maximize bandwidth capacity, the robots were directly connected (without a router) using IEEE802.11ac [2] over 80MHz channel at 5GHz frequency. We believe our setup represents the state-of-the-art (SOTA) computation and communication capabilities of robotic IoT devices. Under the above settings, we saturated the wireless network connection with iperf [3] and recorded the average bandwidth capacity between these two robots every 0.1s for 5 minutes, shown in Fig. 3.

The records in Fig. 3 show frequent and sharp bandwidth fluctuation in both indoors and outdoors. 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, less 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) 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 [1]. In edge networks, bandwidth fluctuation is often caused by the variation of overall traffic volume, and typically happens at the scale of hours [6]. Existing methods target these two types of networks, and are not designed for handling instability in robotic IoT networks.

2.2 Related Work

BSP, SSP and their Variants. Bulk Synchronous Paral-



(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.

lel (BSP) enforces synchronization between each iteration. Therefore, it could easily get blocked by stragglers [22]. To mitigate the straggler effect in datacenter networks while guaranteeing model convergence, SSP is usually adopted [22] 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 [12,13,17,25]. 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) [16,24,36] 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 [30] (information is lost during compression and can 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 [30]. In this paper, we only consider lossless compression methods (e.g., the lost information during compression is compensated with error compensation [38]) which have a typical compression rate of around 3% [38].

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 [9], 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 [5] 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.

3. OVERVIEW

3.1 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 in real-time dynamically and adaptively scheduling the transmission of rows to align the transmission time among all devices as shown in Fig. 4. The design of ROG needs to tackle 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 chose rows to better tradeoff between the management overhead and flexibility in transmission (duration of transmission of the smallest unit). On the one hand, rows, which typically consist of several to hundreds of elements, is not too small. The indexing of rows for management typically takes up data volume much smaller than the model size (e.g., empirically 0.38% of the model size in our evaluation). On the other hand, a row is also not too big. When wireless bandwidth is degraded during a row's transmission and would prolong its transmission time, updating the transmission scheduling of the pending rows only costs a short time transmitting this row at degraded bandwidth, which is negligible. Consequently, row granularity is the best choice of ROG.

Row Stale Synchronous (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 staled versions, making convergence guarantee a challenge. We find that breaking up the staleness control of SSP to each row of the training model would contain the divergence of the same row across different devices and thus contain the divergence of the training model across different devices, which is key to convergence of the distributed training process [22]. Consequently, we design RSP that adopts a two-level row-granulated staleness control: for the same row on the training

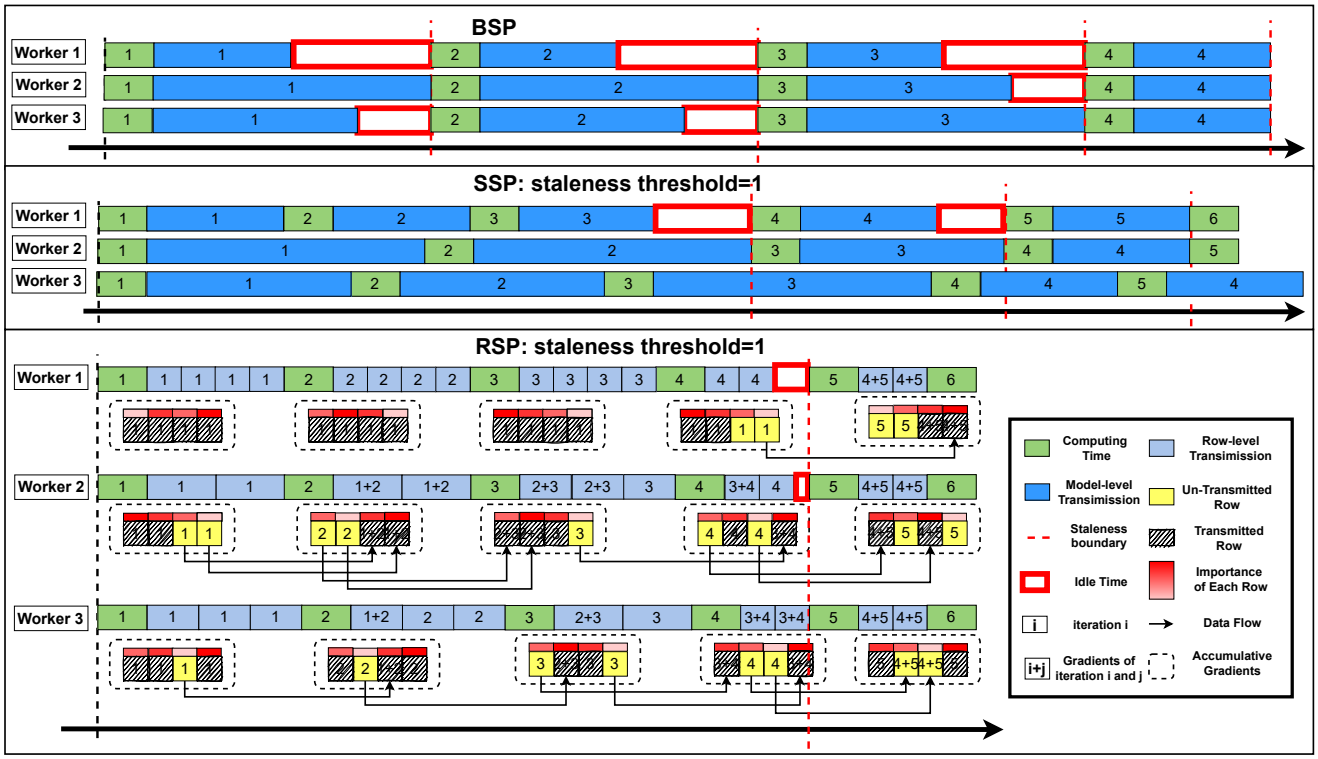


Figure 4: Workflow of ROG. The bandwidth on the three devices are identical at the same time point in the three cases and the bandwidth is interfered by distance, occlusion, etc.

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. Otherwise, idle time will be inserted until the above two-level staleness control requirements are met as shown in Fig. 4. RSP provably achieves the same level of convergence guarantee as SSP (see Sec. 4.3).

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 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 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 control non-stragglers to keep transmitting rows only for MTA time, a straight forward approach is inserting bubbles between the transmission of every two successive rows and check whether MTA time is reached. However, such approach is infeasible in ROG because empirically the transmission time of a row is comparable to the time cost of the inserted bubble, 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 any bubble and discards the on-going transmission once the MTA time is reached (see Sec. 4). In this way, the transmission overhead is reduced to possibly discarding the last row transmitted if its transmission is incomplete, which is also negligible.

3.2 Architecture of ROG

Fig. 5 shows the architecture of ROG and ATP. On each worker and the device acting as 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 dataset and Importance Metric will sort the order of transmission of each row according to the row's staled version and the average absolute value of the gradients. Speculative Transmission keeps transmitting for the aforementioned MTA time on non-straggler or transmits MTA on stragglers, which balances the transmission time of each worker. ROG will increase 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 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.

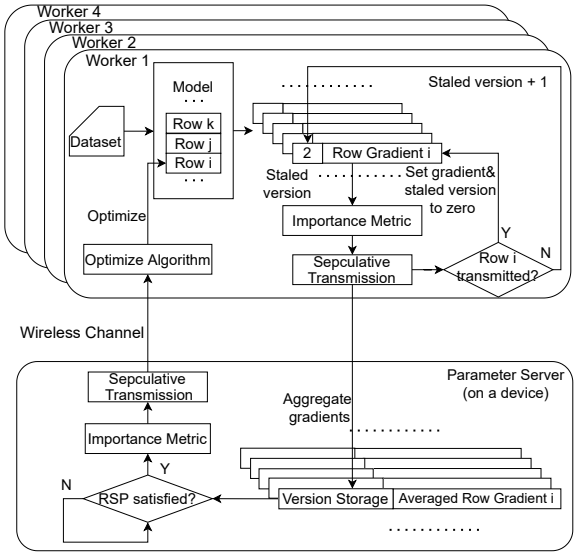


Figure 5: Architecture of ROG. Note that on the parameter server, similar to the worker side, ROG checks whether a row is transmitted and manages its accumulated gradients and the version storage accordingly. We leave out this part on the figure for simplicity.

4. DETAILED DESIGN

4.1 Algorithms of ROG

Here we present how ROG integrate 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, ATP is mainly described in functions of ImportanceMetric() and SpeculativeTransmission() and we will describe them in detail in the next section.

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**

On the worker side in Algo. 1, when gradients is computed in an 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 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 shwon 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.

4.2 Adaptive Transmission Protocol

Algorithm 2 Parameter Server

Function *ParameterServer*_ROG*()*:

Data: $\bar{g}^r$: averaged gradient for worker r; $\bar{g}_i^r$: i-th row of $\bar{g}^r$; v_i^r: the latest training iteration on worker r that updates row i; V: $\{v_i^r\}$; *num*: the number of worker; P: rows' priority; t: staleness threshold

1 **upon** *receive gradients g′ from worker r* **do**

// `Worker r push gradients`

2 $g\prime, n \leftarrow$ RecvGradients(r)

3 **for** *each row i in g′* **do**

4 $v_i^r \leftarrow n$

5 **for** *each worker s* **do**

6 $\bar{g}_i^s \leftarrow \bar{g}_i^s + \frac{g\prime_i}{num}$

7 **for** *each row i in g′* **do**

// v_i^r `triggers the finer-grained threshold`

8 **while** $v_i^r - min(V) \geq t$ **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 $\bar{g}_i^r \leftarrow 0$

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′* **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*)

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** $Transmitted$

Table 1: MTA values under different thresholds

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

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 treats workers and the parameter server differently (line 3 to 6). Since the staleness threshold is triggered and handled at the parameter server side, workers pushing gradients to the parameter server need to specially give priority (bigger j) to the staled rows (line 4), so as to reduce the possibility to trigger the staleness threshold and cause stall. On the contrary, pulling gradients from the parameter server will not affect the triggering of the staleness threshold, and thus we give priority to fresher rows (line 6) that typically have higher contribution to training accuracy. These rows are then sorted in descend order according to their assigned importance.

After sorting these rows, Speculative Transmission (Algo. 4) retrives the scheduled transmission time (t_{MTA}) and enforces the transmission time 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 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 discarded, which is a negligible cost thanks to the small size of a row.

4.3 Proof of guaranteed convergence

In what follows, we shall informally refer to x as the "system state", and the operation $x \leftarrow x+u$ as "writing an update", where u as 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 the tth update written by worker p at iteration c through the write operation $x \leftarrow x+u_t$. The updates u_t are a function of the system state x, and under the RSP model, different workers will "see" different, noisy versions of the true state x. Let $\tilde{x}_t$ be the noisy state read by worker p at clock c, implying that $u_t = G(\tilde{x}_t)$ for some deep learning optimization algorithm G. According to the RSP's row-granulated synchronization model with staleness thresholds for each row, $S_i \geq 0, i = 1,2,...,M$, we divide the 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 noisy state of the ith row of the model parameters.

In this paper, we focus on SGD [14] 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 $||\nabla f_t(x)|| \le L$ for all t (i.e. f_t are L-Lipschitz), and that $MAX_{x,x\prime\in x} D(x||x\prime) \le F^2$ (the optimization problem has bounded diameter), we claim that*

$R[x]=\sum_{t=1}^{T}(f_t(\tilde{x}_t)-f(x^\star)) \le 4FL\sqrt{2(S_{max}+1)PT}$

PROOF. The analysis mainly follows [52], except where the Lemma 1 is involved. By the properties of convex functions, $R[x]=\sum_{t=1}^{T}(f_t(\tilde{x}_t)-f(x^\star)) \le \sum_{t=1}^{T}\langle \nabla f_t(\tilde{x}_t), \tilde{x}_t - x^\star\rangle = \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle$, where we have defined $\tilde{g}_t=\nabla f_t(\tilde{x}_t)$. The high-level idea is to show that $R[x] \le o(T)$, which implies $E_t[f_t(\tilde{x}_t)-f_t(x^\star)] \to 0$ and thus convergence. □

First, we shall say something about $\sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle$.

LEMMA 1. *If $\tilde{g}_t=\left[\tilde{g}_t^1, \tilde{g}_t^2, ..., \tilde{g}_t^M\right]^T$, then for all $\tilde{g}_t$, $\sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle \le M * MAX(\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle)$.*

PROOF. Because $\tilde{g}_t=[\tilde{g}_t^1, \tilde{g}_t^2, ..., \tilde{g}_t^M]^T$ and the parameters of each row are independent of each other, there is $\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{i=1}^{M}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$. So, we can easily have $\sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{t=1}^{T}\sum_{i=1}^{M}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$. And since i and t are independent, we can switch the order of summation, $\sum_{t=1}^{T}\sum_{i=1}^{M}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle = \sum_{i=1}^{M}\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$. However, for any i, $\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$ is bounded, which we will prove later in the lemma 2, such that $\sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{i=1}^{M}\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle \le M * MAX(\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle)$. □

Returning to the proof of theorem 1, we use lemma 1 to expand the regret $R[X] \le \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle \le M * MAX(\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle)$.

Then, we are going to prove the convergence of each rows by finding the upper boundary of $\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$ for each row. Fortunately, we can learn lemma 2 from SSP [22].

LEMMA 2. *For P workers under SSP with staleness threshold $S\prime$, we assume that $||\nabla f_{\prime t}(x)|| \le L\prime$ and $max_{x^i,x\prime^i\in X^i} D(x^i||x\prime^i) \le F\prime^2$. If we set the initial step size $\sigma=\frac{F}{L\sqrt{2\kappa}}$, where $\kappa=(s+1)P$, then*

$R[X] \le F\prime L\prime\sqrt{2\kappa T}\left[3+\frac{1}{2\kappa}+\frac{\kappa}{\sqrt{T}}\right]$.

Assuming T large enough that $\frac{1}{2\kappa}+\frac{\kappa}{\sqrt{T}} \le 1$, we get

$R[X] \le 4F\prime L\prime\sqrt{2(S\prime+1)PT}$.

Because RSP keep the same staleness threshold for each row as SSP does, for any single row of parameters, each RSP's row gets the same constraints as SSP from lemma 2 that $\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle \le 4F\prime L\prime\sqrt{2(S_i+1)PT}$. So, we can get $MAX(\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle) = 4F\prime L\prime\sqrt{2(S_{max}+1)PT}$ and expand the regret $R[X] \le M * MAX(\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle) = M * 4F\prime L\prime\sqrt{2(S_{max}+1)PT}$. Since $F\prime$ and $L\prime$is still unknown, we need to go further and find a more specific upper boundary.

LEMMA 3. *If $max_{x,x\prime\in X} D(x||x\prime) \le F^2$, and $max_{x^i,x\prime^i\in X^i} D(x^i||x\prime^i) \le F\prime^2$ where X^i is the ith row of X, then $F\prime=\frac{F}{\sqrt{M}}$.*

PROOF. Because $x=[x^1, x^2, ..., x^M]^T$ and the parameters of each row are independent of each other, there is $\|x-x\prime\|^2 = \sum_{i=1}^{M}\left\|x^i - x\prime^i\right\|^2$. We can further have $D(x||x\prime)=\frac{1}{2}\|x-x\prime\|^2 = \sum_{i=1}^{M}\frac{1}{2}\left\|x^i - x\prime^i\right\|^2 = \sum_{i=1}^{M} D(x^i||x\prime^i)$. Since $D(x^i||x\prime^i)$ is independent of each other, in order for $D(x||x\prime)$ to be maximized, all $D(x^i||x\prime^i)$ needs to be maximized, which therefore means $MAX(D(x||x\prime)) = \sum_{i=1}^{M} MAX(D(x^i||x\prime^i)) = M * MAX(D(x^i||x\prime^i))$. In other words, there is $F^2 = M * F\prime^2$ and $F\prime=\frac{F}{\sqrt{M}}$can be obtained after deformation. □

Similar to lemma 3, we can have $L\prime=\frac{L}{\sqrt{M}}$. Returning to the proof of theorem 1, we substitute $F\prime$,$L\prime$ into the regret $R[X] \le M*4F\prime L\prime\sqrt{2(S+1)PT} = M*4\frac{F}{\sqrt{M}}\frac{L}{\sqrt{M}}\sqrt{2(S+1)PT} = 4FL\sqrt{2(S+1)PT}$

Until now, we have completed the proof of theorem 1, which means that the noisy worker views $\tilde{x}_t$ converge in expectation to the true view $x^\star$. Furthermore, because the parameters of different rows are updated completely independently of each other, RSP's convergence rate has a potentially tighter constant factor with the help of varying staleness thresholds of each row.

Note that, RSP neither let the fast worker update more times nor lose any updates of slow workers and aggregates all gradients computed on all devices within the staleness threshold. Thus, RSP guarantees uniform-sampling.

5. IMPLEMENTATION

ROG is implemented as an optimizer in PyTorch [10] with nearly 1200 lines of code. ROG exposes similar APIs as existing PyTorch optimizers (`torch.optim` [7]), so that it can be integrated by simply replacing 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 ML model parameters and keeps tracking 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.

6. 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 [8] carrying a 6-core NVIDIA Carmel ARM v8 64-bit CPU, a 384-core NVIDIA Volta GPU, and a Realtek RTL88X2CE Wi-Fi adapter, running Ubuntu 18.04 from NVIDIA's official SDK. Each laptop was equipped with an Intel Core i7-8565U CPU@1.80GHz CPU, an RTX2070 GPU and a MediaTek MT7612U USB Wi-Fi adapter, running Ubuntu 20.04. One laptop was chosen as the parameter server and

setup a IEEE802.11ac [2] hotspot over 80MHz channel at 5GHz frequency to maximize the bandwidth capacity. Since the computation heterogeneity between laptop and robots is out of our scope, we adopt [40] to make all the involved devices spend the same time computing on a batch of data in each iteration.

Experiment Setup. We setup two real-world scenarios 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. In both scenarios, robots are configured to move repeatedly among several points to generate variant wireless network conditions close to real-world deployments.

In the distributed training process, the communicated gradients are compressed before transmission and decompressed after reception using [38] (referred to as DEFSGDM). DEFSGDM typically reduces the transmission volume to 3.2% of the uncompressed counterpart and our implemented DEFSGDM typically takes 0.1 0.3s to compress and decompress gradients, much less than the time taken by computation and communication of gradients. DEFSGDM eliminates information loss by maintaining compression error (difference between the compressed the gradients and the uncompressed gradients) in each iteration and adding the compression error back to the gradients produced in the next iteration before compression; DEFSGDM was originally designed for synchronized training (BSP). We adapted DEFSGDM to support un-synchronized training by simply maintaining compression error for each worker individually on the parameter server.

Baselines. We compared ROG with SSP [22] and the framework proposed in [16] (referred to as FLOWN) under two representative real-world robotic application paradigms: Environment Adaptation Coordinated Online Learning (EACOL) and Motion Planning Coordinated Online Learning (MPCOL). FLOWN is one of the most SOTA federated-learning-based methods specified for distributed training over unstable wireless networks.

Evaluation Workload. We evaluate with a typical paradigm which we name as environmental adaptation coordinated robotic learning (EACOL): a team of robots are recognizing the objectives captured by their cameras with a shared objective recognition model, and the recognition accuracy is accidentally degraded by environmental noises such as fog. Such a paradigm is fundamental and representative [23, 46, 47] 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 through wireless distributed training.

We use the well-studied Fed-CIFAR100 [11] as the image dataset of objectives, which contains 100 types of objectives and 500 images each. 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 [29] and we randomly allocates 50 shards for each robot without overlap. We choose ConvMLP [27] 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. We follow the methods of DeepTest [39], a DNN testing framework, to add image blur effect to the Fed-CIFAR100 dataset to simulate the environmental noise of fog, and the noise leaded to a lowered recognition accuracy (65.67%). Details of the workload are listed in Table **??**.

Table 2: A table of number of parameters, the batchsize, computation time, compress time, decompress time, compressed size, average communication time under BSP in lab or corridor of both applications.

Evaluation Metrics. In real-world deployments, the major concern of robotic systems are usually time budget (for fast response to their missions) [] and energy consumption (for guaranteed battery life time) []. Therefore, we mainly compare with baselines for two aspects: model accuracy given the same time budget, and model accuracy given the same energy budget.

The evaluation questions are as follows:

- RQ1: How much does ROG benefit real-world robotic applications compared to baseline systems in terms of *model convergence*?
- RQ2: How does ROG handle the unstable wireless networks?
- RQ3: How sensitive is ROG to different staleness thresholds and different transmission volumes / compression rates?
- RQ4: What are the limitations and potential of ROG?

6.1 End-to-end Performance

(a) out-doors (b) in-doors

Figure 6: EACOL, acc-time, may merge figures with MPCOL to reduce num of figures

(a) out-doors (b) in-doors

Figure 7: MPCOL, acc-time

Convergence time. We first compare the trained model accuracy over time, as is shown in Figure 6. Given the same time budget, RSP typically achieves ??% higher accuracy than baselines in outdoors setting, and ??% higher accuracy in indoors setting. Such accuracy gains have been reported as critical in real-world robotic applications [37, 47].

Figure 8: Mitigation of stall. bar chart of comp time, comm time + ?? to show micro event when bandwidth degrade

Figure 9: Statistical efficiency.

The key reason of ROG's high performance is its mitigation of stall time without sacrificing statistical efficiency. Figure 8 shows that ROG effectively mitigates stall time. [Further explain the figure.] 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.

Figure 9 shows that ROG does not sacrifice statistical efficiency. In order to avoid being blocked by degraded bandwidth during synchronization, it is inevitable to reduce the transmit volume and delay some rows to synchronize later. However, such delay would cause subsequent gradient computation base on outdated parameters, reducing statistical efficiency. ROG's ATP identifies rows with large gradients and prioritizes them. Therefore, even if stragglers transmit less gradients than non-stragglers, important changes to the model are always synced, resulting in a comparable statistical efficiency as SSP, the tightest synchronization model.

ROG's accuracy-time remain similar between indoors and outdoors settings, showing its robustness. We leave the analysis for RSP's robustness later in Section **??**.

(a) outdoors (b) indoors

Figure 10: EACOL, acc-energy, may merge figures with MPCOL to reduce num of figures

Energy consumption. We then compare RSP's energy efficiency as shown in Figure 10. We measured the energy consumption of the whole development board including CPU, GPU, memory, wireless card, with jtop [], a well-recognized monitoring tool for NVIDIA Jetson boards. We measure the whole board instead of standalone components since distributed training involves all these major components. RSP typically achieved ??% higher accuracy given the same energy consumption, implying a higher energy efficiency.

To further understand the energy consumption statistics, we identify major states of a system during training, and measure the power consumption of each state. There are mainly three states involved in distributed training, namely *computation*, *communication*, and *stall*. During computation, a device mainly leverages its GPU for DNN forward and backward. During communication, a device transmits gradients with the parameter server. During stall, a device waits for the parameter server for messages to continue work. Note that the device cannot be put into low power sleep mode even in stall state, as it has to wait for messages from the parameter server and promptly continue work when stragglers catch up.

The power consumption of different states and systems are shown in Table 3. Different training systems show little

(a) *lab* scenario (b) *corridor* scenario

Figure 11: MPCOL, acc-energy

	computation	communication	stall
SSP			
FLOWN			
RSP			

Table 3: Power (W) in different states.

variance, as these systems do not change how computation work and stall. While during communicaton, the overhead to scheduling is minor. Due to the static power consumption rooted in transistors' leakage current [**?**], chips like CPU, GPU, and memory consumes non-negligible power even when not computing (i.e., in stall state). As is shown, the stall state power (around 4W) was nearly 28% of the computation state power (around 14W). Since RSP reduces stall times up to ??%, the corresponding energy spent during stall time is reduced accordingly, amounting to ??% reduction of energy consumption. We also note that communication and stall have similar power consumption (within ??% difference), this may be due to the relatively low wireless transmission rate involves little energy consuming operations from CPU etc.

6.2 Micro-Event Analysis

To further understand the performance gain of ROG, we recorded the real-time bandwidth and amount of synchronized model parameters each time, shown in Figure **??**. Since proactive methods (e.g., measuring with `iperf` []) contends with the application traffic, we passively measure the real-time bandwidth with the expected throughput reported by `iw` []. Note that `iw`'s output is an estimation of the physical layer bitrate which deviates from the actual bandwidth the application could exploit, we normalize the output with its average.

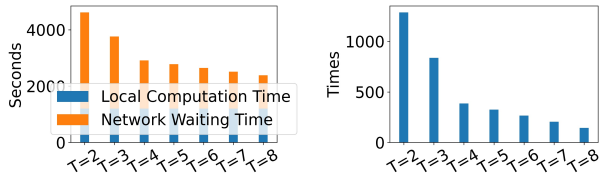
When bandwidth degradation happens, ROG adjusts .... , and a straggler's transmission volume is reduced, and any non-transmitted rows' gradients are accumulated locally to be transmitted latter. In this way, ROG prevents stragglers from being stuck in transmitting.

Figure **??** shows that even ROG cannot work in extreme conditions when bandwidth degradation last for a long time. It is impossible to perform even minimal necessary synchronization under such conditions, and no system could keep in sync. Such conditions are out of this work's scope, but occurred in our evaluation. For a real evaluation, we did not rule out such conditions.

6.3 Sensitivity Studies

ROG different threshold
ROG different batchsize

We further evaluated RSP with different staleness thresholds (i.e., from 2 to 8), whose environment configuration followed the setting as in Fig. **??**, because we found that in the case of Fig. **??**, RSP-LT could not handle such a dramatic instability. Therefore, we wanted to find out the upper bounds on the instability among wireless connections that RSP with different staleness thresholds can handle. Fig. 12 shows the performance of RSP with different staleness thresholds in a through-wall WLAN.



(a) Transmission time.

(b) The number of times triggering threshold.

Figure 12: Sensitivity to RSP's staleness thresholds. 'T' represents 'staleness threshold'.

In Fig. 12b, we see that when the threshold went from 2 to 5, the number of times triggering the threshold dropped dramatically; when the threshold went from 5 to 8, the number of times triggering the threshold decayed slowly. It is because the worker's bandwidth capacity dropped to 30% (i.e., nearly 80 Mbps in our evaluation) due to the wall's signal attenuation to wireless signals, resulting in a nearly 3.3X times difference between the fastest and slowest workers' transmission speed. According to 4.2, the MTA must be at least 0.3, where the threshold must be at least 5, to handle such dramatic network instability, which fits Fig. 12.

6.4 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 [], 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 a area), while there is not yet a well-recognized method for distributed training over non-IID datasets.

Finer granularity brings more mangement 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 completely eliminated and potentially limits the performance gain of ROG over the baselines.

The selection of staleness threshold. On the selection of staleness threshold, ROG achieves a better trading-off between statistical efficiency and training throughput than SSP.

Limitations. ROG can be ineffective when communication is not a bottleneck in the wireless distributed training process.

7. CONCLUSION

In this paper, we present ROG, a Row-granulated Stale Synchronous Parallel model optimized for IoT ML training. ROG benefits diverse IoT application scenarios such as disaster response, search & rescue, and environmental monitoring by enabling fast adaptation of ML models in changing and unknown environments to improve end-to-end task performance.

REFERENCES

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

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

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

[4] NVIDIA embedded systems for next-gen autonomous machines. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

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

[6] 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/

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

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

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

[10] (2021) PyTorch. [Online]. Available: https://www.pytorch.org

[11] (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

[12] 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.

[13] 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

[14] 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.

[15] 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

[16] 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.

[17] 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

[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. 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

[20] 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.

[21] 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.

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

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

[28] 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

[29] 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

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

[31] 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

[32] 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.

[33] 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

[34] 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.

[35] 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.

[36] 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.

[37] 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

[38] 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

[39] 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

[40] 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.

[41] 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.

[42] 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

[43] Y. Wang, Z. Lai, G. Huang, B. H. Wang, L. van der Maaten, M. Campbell, and K. Q. Weinberger, "Anytime stereo image depth estimation on mobile devices." [Online]. Available: http://arxiv.org/abs/1810.11408

[44] 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

[45] 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

[46] M. Wulfmeier, A. Bewley, and I. Posner, "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.

[47] 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.

[48] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 392–405, ISSN: 2576-3172.

[49] 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.

[50] S. Yao, Y. Hao, Y. Zhao, H. Shao, D. Liu, S. Liu, T. Wang, J. Li, and T. Abdelzaher, "Scheduling real-time deep learning services as imprecise computations," in *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 1–10, ISSN: 2325-1301.

[51] 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.

[52] M. Zinkevich, A. J. Smola, and J. Langford, "Slow learners are fast." [Online]. Available: https://openreview.net/forum?id=SkV6_uW_-B