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 need of 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 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 stalls, while provably guaranteeing convergence. The evaluation shows that, given the same training time, ROG achieved about 1.2%~5.2% training accuracy gain compared with the baselines and saved 17.3%~58.1% of the energy to achieve the same training accuracy.

1. INTRODUCTION

Critical robotic tasks such as rescue [36] and disaster response [49] are more prevalently leveraging machine learning models (e.g., objective recognition models [27] or action control models [21, 46]) deployed over a team of mobile robots. These models typically require real-time training to adapt pre-trained parameters to changing environments [44, 45] (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 a team of robots over the robotic IoT networks [16, 34].

Such distributed training typically adopts the parameter server paradigm [29]: 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 [22]) 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). 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 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 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 empirically achieves high statistical efficiency (**R3**) [23], 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. 2.1) by sharp bandwidth degradation due to movement of the devices [32, 35], occlusion from obstacles [20, 37], 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) [19, 23] has the potential to mitigate such straggler effect. SSP allows non-stragglers to

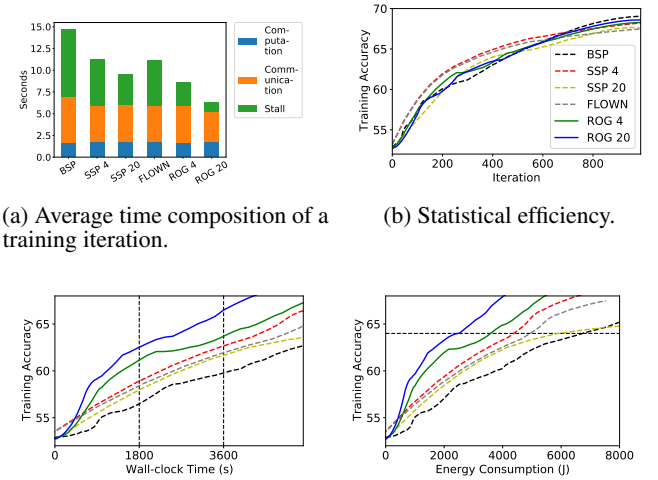
continue computing without latest gradients from stragglers and only stall when the gradients from stragglers fall behind (stale) for a preset number of iterations (staleness threshold).

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 [23], 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 [17, 25] 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. 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 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 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 (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 at 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.



(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.

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 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. 4.3).

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 an 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., 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 a representative real-world application paradigm. We compared ROG with BSP [22], SSP [23] and a SOTA dynamic threshold method [17] (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 [40] (the compressed gradients were only sized at 2.1 MByte) to conduct the tightest comparison between ROG

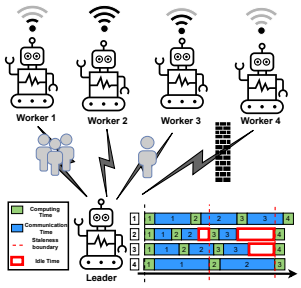


Figure 2: The instability of real-world robotic IoT networks.

and the baselines. Evaluation shows that:

- ROG achieves high accuracy. ROG achieved a 1.2%~5.2% 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 17.3%~58.1% 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 batchsize of training, ROG still achieved 1.3%~4.2% accuracy gain and a 32.2%~64.2% 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 [36], disaster response [49], and robot surveillance [43, 50], making them fast 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/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

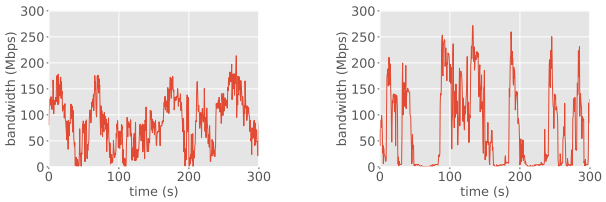
2.1 Characteristics of Robotic IoT Networks

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. 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. 6. 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 [4] 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, 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) 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 [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.



(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.

2.2 Related Work

BSP, SSP and their Variants. Bulk Synchronous Parallel (BSP) enforces synchronization between each iteration. Therefore, it could easily get blocked by stragglers [23]. To mitigate the straggler effect in datacenter networks while guaranteeing model convergence, SSP is usually adopted [23] 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 [12,13,18,26]. 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) [17,25,38] 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 [31] (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 [31]. In this paper, we only consider lossless compression methods (e.g., the lost information during compression is compensated with error compensation [40]) which have a typical compression rate of around 3% [40].

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 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 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, are 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. Therefore, 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 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 convergence of the distributed training process [23]. 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. 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

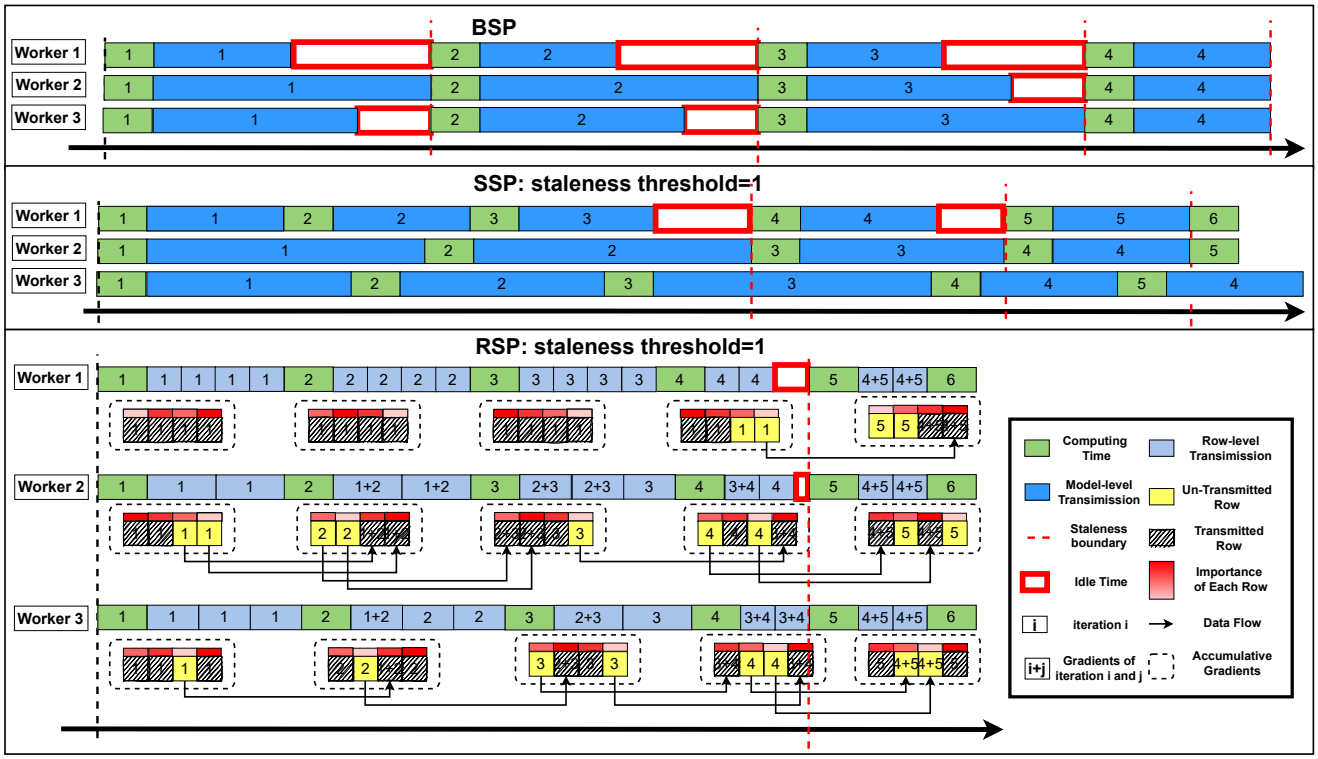


Figure 4: Workflow of ROG. The bandwidth on the three devices are identical at the same time point in the three cases and the bandwidth is interfered by distance, occlusion, etc.

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 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 on-going transmitting row 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 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 sorts 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.

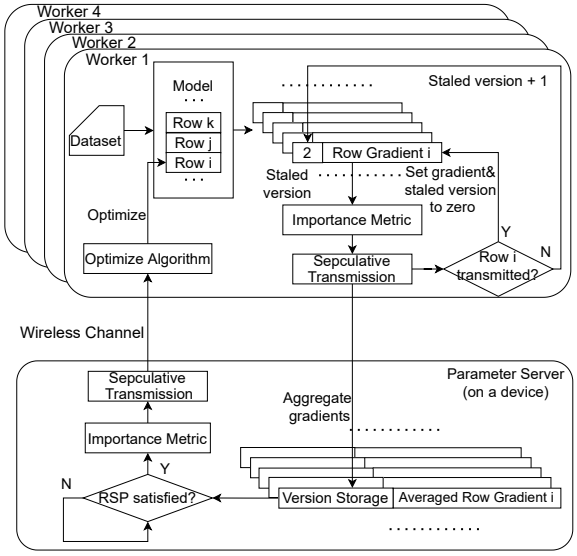


Figure 5: Architecture of ROG. Note that on the parameter server, similar to the worker side, ROG checks whether a row is transmitted and manages its accumulated gradients and the version storage accordingly. We leave out this part on the figure for simplicity.

4. DETAILED DESIGN

4.1 Algorithms of ROG

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

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

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 worker; 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 $\bar{g}^r_i \leftarrow 0$

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 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($g\prime$, $importance$)

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** $Transmitted$

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.

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

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 at least P percent of row 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 1. 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

Following the convention in [23], we refer to x as the "system state", and the operation $x \leftarrow x+u$ as "writing an update", where u as a "model update". Let u_t be the the tth update written by workers, and according to the RSP's row-granulated synchronization model for each row, 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.

In this paper, we focus on SGD [15] 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, we define* $D(x||x\prime) = \frac{1}{2}\|x - x\prime\|^2$ *and assume that P workers write model updates to parameter server independentl,* $||\nabla f_t(x)|| \leq L$ *for all t (i.e.* f_t *are L-Lipschitz), and that* $MAX_{x,x\prime \in \chi} D(x||x\prime) \leq F^2$ *(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. By the properties of convex functions, $R[x]$ $= \sum_{t=1}^{T}(f_t(\tilde{x}_t) - f(x^\star)) \leq \sum_{t=1}^{T}\langle \nabla f_t(\tilde{x}_t), \tilde{x}_t - x^\star\rangle$ $= \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle$, where we have defined $\tilde{g}_t = \nabla f_t(\tilde{x}_t)$. Because $\nabla f_t(\tilde{x}_t) = [\nabla f_t(\tilde{x}_t)^1, \nabla f_t(\tilde{x}_t)^2, ..., \nabla f_t(\tilde{x}_t)M]^T$, $\tilde{g}_t = [\tilde{g}_t^1, \tilde{g}_t^2, ..., \tilde{g}_t^M]^T$ and $\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{i=1}^{M}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$. 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, $\sum_{t=1}^{T}\sum_{i=1}^{M}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle = \sum_{i=1}^{M}\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$, such that $R[X] \leq \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{x}_t - x^\star\rangle = \sum_{i=1}^{M}\sum_{t=1}^{T}\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star}\rangle$. □

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. Thanks to SSP's pioneering work [23], we learn the following lemma 1.

LEMMA 1. *For P workers with staleness threshold S\prime, we assume that* $||\nabla f_t\prime(x)|| \leq L\prime$ *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* $\kappa = (s+1)P$, *then*

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

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

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

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

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 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 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. 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 the regret $R[X] \leq M * 4F\prime L\prime\sqrt{2(S_{max}+1)PT} = M * 4\frac{F}{\sqrt{M}} * \frac{L}{\sqrt{M}}\sqrt{2(S_{max}+1)PT} = 4FL\sqrt{2(S_{max}+1)PT}$. Until now, we have completed the proof of theorem 1, which means $\tilde{x}_t$ converge in expectation to the minimizer $x^\star$.

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 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.

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 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.

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] 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 [3] 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 [42] to make all the involved devices spend equal time computing gradients in each iteration (see Table 2).

Experiment Scenarios. Our evaluation was under a typical paradigm where 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 such as fog. Such a paradigm is fundamental and representative [24,47,48] 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 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 and this scenario imposes higher level of instability as discussed in Sec. 2.1.

Baselines. We compared ROG with SSP [23] and the framework proposed in [17] (referred to as FLOWN). FLOWN is one of the most SOTA scheduling-based methods specified for distributed training over unstable wireless networks.

Dataset. We use the well-studied Fed-CIFAR100 [11] 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 [30] and we equally divided the dataset equally to four parts without overlap, each for one of the worker. We follow the methods of DeepTest [41], a DNN testing framework, to various noises to the Fed-CIFAR100 dataset to simulate fog and brightness changes.

Gradient Compression. In the distributed training process, the communicated gradients are compressed before transmission and decompressed after reception using [40] which typically reduces the transmission volume to 3.2% of the uncompressed counterpart.

Model. We choose ConvMLP [28] 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.

Default setup and statistics are listed in Table 2.

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

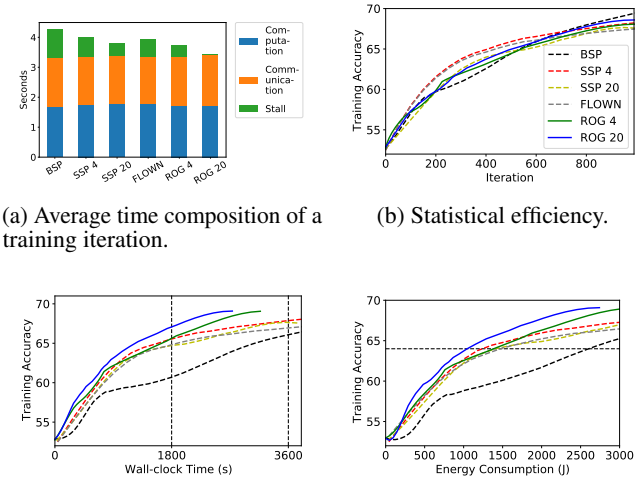
Table 2: Default Setup

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 number of devices and batchsizes?
- RQ4: What are the limitations and potential of ROG?

6.1 End-to-end Performance

Training accuracy and energy consumption. We first compare the accuracy of the training model and energy consumption of the training process over time, as is shown in Fig. 1 (outdoor) and Fig. 6 (indoor). The training accuracy



(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 in the indoor environment.

was obtained by checkpointing and validating the training model on each worker every 50 training iterations and then averaging the validated accuracy. We measured and averaged the energy consumption of the whole development board including CPU, GPU, memory, wireless card on all robots, with jtop [14], a well-recognized monitoring tool for NVIDIA Jetson boards.

Our evaluation shows that ROG achieved both high training accuracy and high energy efficiency. When training for 30 minutes, ROG achieved 3.2%~5.0% higher accuracy than the baselines in the outdoor setting in Fig. 1c, and up to 3.0% accuracy gain in the indoor setting in Fig. 6c due to reduced instability. When training for 60 minutes, ROG achieved 1.2%~5.2% higher accuracy than the baselines in outdoors setting. Such accuracy gains have been reported as critical in real-world robotic applications [39, 48]. In terms of energy consumption, when the training model reached an accuracy of 64.0%, ROG saved 17.3%~58.1% of the battery energy in outdoors setting (Fig. 1d). The reduction of energy consumption was up to 25.7% in indoor setting still due to reduced instability.

The key reason of 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 level of instability (robustness, **R1**). Fig. 1a and Fig. 6a shows 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 the outdoor setting and 0.4s stall in the indoor setting. ROG reduced the stall time by 52.1% to 89.8% in outdoor setting. Still, in indoor setting, ROG was less effective due to the reduced instability in indoor environments. By breaking down the whole model into

	computation	communication	stall
Power (W)	13.35	4.25	4.04

Table 3: Power (Watt) in different states.

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 delay some rows to synchronize later, 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 less 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. 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 are shown in Table 3 and they apply to all evaluated systems because 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, because due to the static power consumption rooted in transistors' leakage current [33], chips like CPU, GPU, and memory consumes non-negligible power even when not computing (i.e., in stall state). 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.

6.2 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, shown in Fig. 7. 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` [2]. Note that `iw`'s output is an estimation of the physical layer

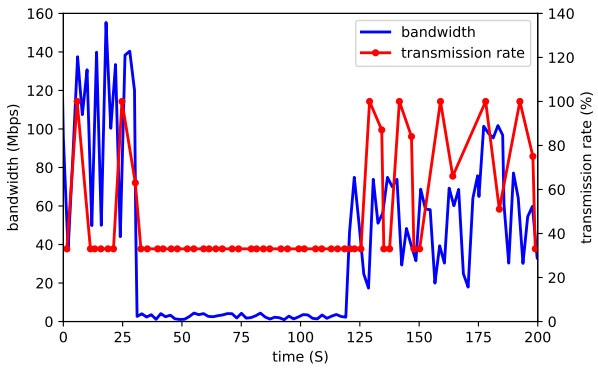


Figure 7: Real-time bandwidth and the percentage of rows transmitted by ROG

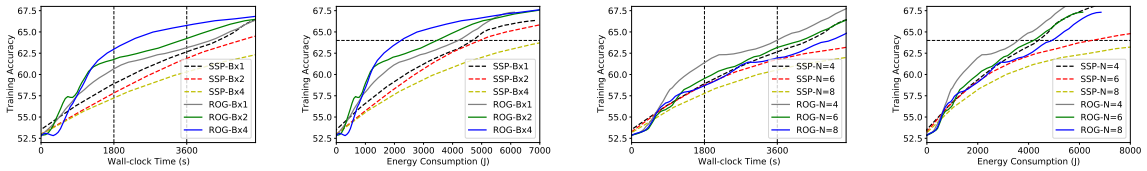
bitrate which deviates from the actual bandwidth the application could exploit, we normalize the output with its average.

When bandwidth was fluctuating in the former part of Fig. 7, ROG responded immediately and adjusted the transmission rate on a robot accordingly. This aligned the transmission time between this robot and the fastest worker, avoiding possible straggler effect and co staleness is in a low level (0 to 1). In this way, ROG prevented a robot from being a straggler and stalling the training process and prevented staleness from affecting statistical efficiency. When bandwidth degraded to a extremely low level and lasted for a long time in the middle part of Fig. 7, it is 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. 7, this robot caught up quickly (staleness decreased) because it was allowed to transmit partial of its rows.

6.3 Sensitivity Studies

Batchsize. We varied the batchsize (x2, x4) of training to examine how ROG performs with different ratio of computation and communication in the outdoor environment in Fig. 8. When the batchsize 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 can be less severe (stall time slightly decreased in the Fig. 9a) and ROG's potential gain over the baselines is limited. As shown in Fig. 8a and Fig. 8b, when training for 30 minutes, ROG achieved 4.2% accuracy gain over the SSP-4 and 32.2% energy consumption reduction when training accuracy reached 64% in the doubled batchsize case; When the batchsize was increased to four times, ROG achieved 6% accuracy gain over the baselines and energy consumption reduction when training accuracy reached 64%.

Number of workers. Increasing the number of workers (4, 6, 8 workers) caused more severe straggler effect. 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



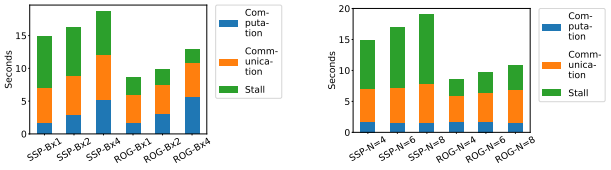
(a) Accuracy in various batchsize.

(b) Energy consumption with various batchsize.

(c) Accuracy with various number of workers.

(d) Energy consumption with various number of workers.

Figure 8: Sensitivity Studies



(a) Batchsize.

(b) Number of workers.

Figure 9: Average time composition in sensitivity studies

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 (Fig. 8c and Fig. 8d), when training for an hour, ROG achieved 1.3% accuracy gain over the SSP-4 and 25% 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 1.5% accuracy gain over the baselines and 46% energy consumption reduction when training accuracy reached 64%.

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 extra 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 effect of staleness threshold. From evaluation, we can learn that the statistical efficiency of ROG with a large staleness threshold (20) only slightly degraded than ROG with a small staleness threshold (4). The degradation complies with the report of SSP [23] but the level of degradation is smaller than SSP. The reason could be that ATP of ROG will try to align the transmission time among all workers by adjusting their communication traffic volume, which will make stragglers to catch up and non-stragglers to pace down. As a result, in most of the time the staleness of all workers is confined in a low level regardless of either a small staleness threshold and the impact of staleness on statistical efficiency is minimized.

7. CONCLUSION

In this paper, we present ROG, a Row-granulated distributed training system optimized for robotic IoT networks. Through 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 instable local wireless network without being affected by straggler effect.

REFERENCES

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

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

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

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

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

[15] 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.

[16] 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

[17] 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.

[18] 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

[19] 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

[20] 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

[21] 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.

[22] 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.

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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

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

[29] 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

[30] 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

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

[32] 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

[33] 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/

[34] 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.

[35] 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

[36] 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.

[37] 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.

[38] 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.

[39] 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

[40] 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

[41] 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

[42] 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.

[43] 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.

[44] 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

[45] 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

[46] 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

[47] 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.

[48] 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.

[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] 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.