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

ABSTRACT

The prosperity of IoT (Internet of Things) and ML (Machine Learning) pushes more and more ML applications deployed on wireless mobile robots (i.e., robotic IoT), on which data parallel training can harness the distributed hardware resources to quickly adapt to changing environments. Unfortunately, the instability in wireless networks (i.e., fluctuating bandwidth due to mobile obstacles and varying distances) often leads to severe stall for devices when a model synchronization barrier is involved. Recent stale synchronous parallel (SSP) methods only partially alleviate such stall because they are all model-granulated. Their scheduling on the model granularity would be invalidated during the actual transmission due to a transient network instability; they also ignore different gradients can have different contribution to the training process.

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 extends the staleness control of SSP methods to row granularity. In this way the ML training process can update partial and the most important gradients of a stale device to avoid triggering stall, while provably achieving the same level of convergence guarantee as the SSP methods. The evaluation shows that RSP reduced up to 60% of the training time compared with the SSP baselines and achieved the same final training accuracy.

KEYWORDS

model adaptation, wireless distributed training, synchronization model

1 INTRODUCTION

Machine Learning (ML) models for real-world data are getting more prevalently deployed in a team of robotic IoT devices [39, 52, 52], such as objective recognition models [27] and action control models on robots [21, 48] in outdoor tasks (e.g., rescue [37] and disaster response [50]). These models typically require real-time training to adapt their pre-trained parameters to the changing environments [45, 47] (e.g., from sunny to foggy), but it is often expensive for the devices to access a cloud data center for model training due to the the lack of stable internet access. Therefore, such training is often distributedly deployed among the team of devices over robotic IoT networks [16, 35].

Such distributed training typically adopts the parameter server paradigm [29]: each device keeps a model copy 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).

Unfortunately, the instability of real-world robotic IoT networks turns synchronization into a major bottleneck of the distributed training process by causing the *straggler effect* (Fig. 1a): sharp bandwidth fluctuation with random duration happens frequently and randomly on each device due to movement of the devices [34, 36], occlusion from obstacles [20, 38], etc (see Sec. 2.1); under such bandwidth fluctuation, the transmission of gradients from some devices (i.e., stragglers) can be dramatically delayed (e.g., from 1.43s to 12.9s recorded when the devices were moving in an unstable environment, see Sec. 2.1); the devices that finish transmission (i.e., non-stragglers) have to stall until the delayed gradients from stragglers are transmitted in severely downgraded bandwidth, damaging training throughput (training iterations per unit time) and prolonging convergence time.

Even if the communication traffic volume is reduced by SOTA gradient compression methods [31, 41], the straggler effect remains because the computation time is still comparable to the communication time due to recent rapid advancement of computation power of SOTA robotic IoT devices [4, 8]. In our evaluation, we observed that for a Jetson NX device [6], the time it took to transmit compressed gradients in a stable environment was comparable to (taking up 63.4% of) its time to compute gradients. And the straggler effect in an unstable environment (see Sec. 2.1) caused each device on average to stall for 108.1% of the computation time in each iteration (referred to as *stall cost*). Such high stall cost reduced the training throughput by 55.34% and increased the convergence time from 132.3 minutes to 312.84 minutes.

Although mainly designed for datacenter networks, Stale Synchronous Parallel (SSP) [19, 24] has the potential to resolve such straggler effect. SSP allows non-stragglers to continue computing without the latest gradients from stragglers

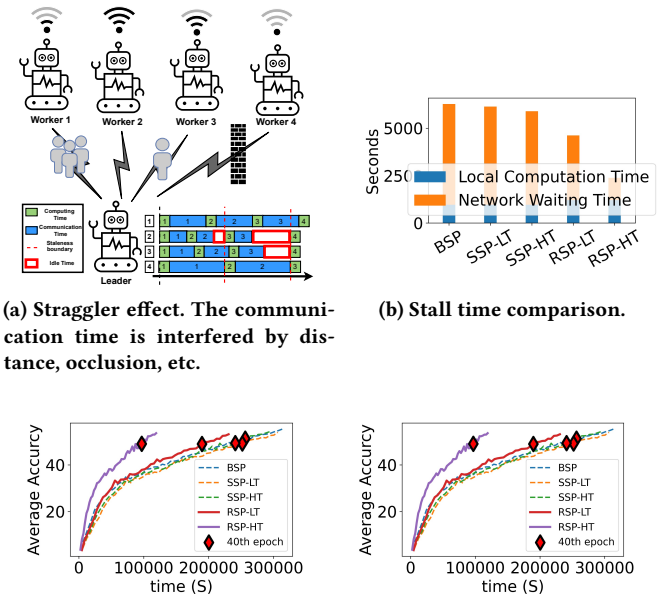
and only stall when the gradients from stragglers fall behind (stale) for a preset number of iterations (staleness threshold).

However, when coping with the instability of real-world robotic IoT networks, SSP is in a dilemma between high statistical efficiency (i.e., high accuracy gain per training iteration) and high training throughput (i.e., less stall cost). Both properties contribute to minimizing convergence time, but are contradictory in SSP because high statistical efficiency requires a moderate staleness threshold [24], while high training throughput requires a large staleness threshold. In our evaluation in an unstable environment, SSP with a moderate threshold (2) achieved similar statistical efficiency as BSP but suffered high (40.9%) stall cost; a larger threshold (10) reduced the stall cost to 20.3%, at the cost of lower statistical efficiency (reduced by 36.2%) compared with BSP. As a result, the convergence time of SSP with a moderate threshold and a high threshold were still increased to 243.5 minutes and 263.2 minutes.

Subsequent studies [17, 25] extend SSP by dynamically assigning (scheduling) the staleness threshold: higher staleness threshold for devices estimated to have low bandwidth and moderate threshold for devices estimated to have high bandwidth. However, 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, invalidating the scheduling and resulting in higher stall cost. Consequently, in our experiments, such methods caused 80.3% stall cost on average on each device in an unstable environment and the convergence time was increased to 283.4 minutes.

The key reason for the problem of the above methods is that they cannot resolve the straggler effect when they are synchronizing the model gradients on the granularity of a whole model, which is typically coarser than the granularity (or frequency) of bandwidth fluctuation in real-world robotic IoT networks. From the view of stall cost, 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 higher stall cost and prolonged convergence time. From the view of model convergence, the scheduling treats all computed gradients from a device as a whole and neglects that gradients from a device have different contribution to model convergence (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. From three possible granularity choices: elements, rows and layers, we find that element granularity requires indexing each element of the whole model for synchronization, taking up data



(a) Straggler effect. The communication time is interfered by distance, occlusion, etc.

(b) Stall time comparison.

(c) Accuracy comparison against iterations.

(d) Accuracy comparison against wall-clock time.

Figure 1: The straggler effect in real-world robotic IoT networks and the comparison between Rog and the baselines.

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). Thus, to best tradeoff between management cost and transmission flexibility, Rog breaks up the whole model into rows and individually and flexibly synchronizes the gradients of each row from different devices . In this way, whenever bandwidth fluctuation happens during gradient transmission, Rog in real-time adjusts its scheduling of the number of rows to be transmitted, at a negligible cost of transmitting only one row in degraded bandwidth.

A major concern is how to guarantee convergence in Rog when synchronizing gradients on row granularity. We propose Row Synchronous Parallel (RSP) that breaks up 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. 5.3).

A major challenge of Rog is under RSP, how to properly schedule the transmission of each row from different devices to minimize stall cost and the convergence time. As discussed

above, schedule on a large granularity would be invalidated due to the bandwidth fluctuation during the scheduled transmission. Instead, RoG adaptively aligns the transmission time of each device during the transmission of each row with *Adaptive Transmission Protocol* (ATP). 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 in transmitting gradients in an iteration. In this way no devices will fall behind and cause stall. 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.

We implemented RoG in PyTorch [11] and evaluated RoG on a team of mobile robots under two representative real-world application paradigms with different model sizes: adapting a route planning model (sized 3.2 MByte) to moving obstacles and adapting an objective recognition model (sized 64.8 MByte) to changing weather. We compared RoG with BSP [22], SSP [24] and a SOTA dynamic threshold method specified for wireless networks [17] (referred to as FLOWN) in different real-world robotic IoT networks environments. It is notable that we minimized the communication volume with gradient compression [41] (the compressed gradient of the two applications were only sized 0.1 MByte and 2.1 MByte) to conduct the strictest comparison between RoG and the baselines. Our evaluation with diverse robotic applications and real-world unstable networks shows that:

- RoG is easy to use. It took only tens of lines of code to apply RoG to each of the above ML applications.
- RoG achieved the highest training throughput. Compared with BSP, SSP and FLOWN, RoG achieved 2.12X, 1.76X and 1.89X training throughput and thus reduced their convergence time by 65.4%, 43.7% and 54.3%.
- RoG eliminated the straggler effect. RoG reduced the occurrences of stall by 99.7%, 85.3% and 90.6% compared with BSP, SSP and FLOWN.
- RoG reduced battery power consumed by stall waiting. Compared with BSP, SSP and FLOWN, RoG on average reduced the battery power consumption of the distributed training process by 13.3%, 5.4% and 7.2% due to less stall waiting.
- RoG empirically achieved the same high final accuracy as baselines. It is notable that the statistical efficiency of RoG only diverged from that of SSP by at most 2.2%.

Our main contribution is RSP, a new row-granulated synchronization model and ATP, a fine-grained scheduling strategy designed for minimizing the stall cost and the convergence time of distributed training over real-world robotic IoT networks. While guaranteeing convergence with RSP, RoG conducts fine-grained (i.e., parameter rows) staleness control and transmission scheduling with ATP, so that the transmission time of different devices under fluctuating bandwidth can be balanced and the straggler effect is eliminated. We envision that RoG will nurture diverse ML applications deployed on mobile robots in the field, such as robot rescue [37], disaster response [50], and robot surveillance [44, 53], 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/sigcomm22p692/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 RSP and RoG in Sec. 5, evaluate RoG in Sec. 7, and finally conclude in Sec. 8

2 BACKGROUND

2.1 Characteristics of Robotic IoT Networks

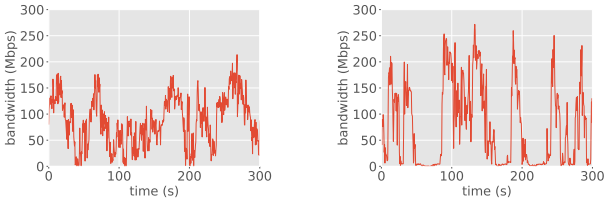
In real-world robotic IoT applications, devices typically need to move around for rescue, search, etc. Although wireless networks suffice for high mobility, the occlusion of obstacles and the change of distances among devices cause *instability* in the bandwidth capacity: sharp bandwidth fluctuation with random duration happens frequently and randomly.

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 same representative robotic IoT hardware, an NVIDIA Jetson Xavier NX [6], since Jetson modules are well-recognized as the enabler of intelligent robotic IoT applications [46, 49, 51], and Jetson Xavier NX is one of the most recent models [5]. To maximize bandwidth capacity, the robots were directly connected (without a router) using WiFi 5 (IEEE802.11ac [2]) over 80MHz channel at 5GHz frequency. We believe our setup represents the 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. 2.

The records in Fig. 2 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 the 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.

Compared with robotic IoT networks, the Internet (for federated learning) and datacenter networks (for distributed training in datacenter) often exhibit much lower bandwidth fluctuation. In datacenter networks, bandwidth fluctuation is typically caused by contention on intermediate switches, and could be mitigated by scheduling traffic on switches [1]. In the Internet, bandwidth fluctuation is often caused by the variation in the number of active users, and typically happens at the scale of hours [7].



(a) Indoors. **(b) Outdoors.**

Figure 2: 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 Existing Distributed Training Methods

Vanilla Distributed Training. Distributed training in a team of robotic IoT devices typically makes multiple workers (i.e., devices) train a shared ML model on different fractions of data and synchronizes the model's parameters across workers in an iterative process, as demonstrate in Fig. 3. A parameter server (located on one of the devices) maintains a global copy of the model being trained. In each iteration, each worker first *pull*s the model's latest parameters from a parameter server. Then, each worker computes parameter updates (i.e., gradients) over current model parameters, and *push*es the parameter update to the parameter server. The parameter server collects parameter updates from the workers and adds them to the model's parameters. A synchronization barrier is set at the end of each iteration to ensure that each worker sees others' latest updates. Such a process iterates continuously until the model is good enough (usually determined by loss or accuracy). The vanilla distributed training method (i.e., BSP) could easily get blocked by stragglers [24].

Stale Synchronous Parallel and its Variants. To mitigate the straggler effect in datacenter networks while guaranteeing model convergence, Stale Synchronous Parallel (SSP) is usually adopted [24] in practice. SSP loosens the synchronization barrier and allows fast workers to continue their iterations when the updates from slow workers are staled until the staled version reaches a *staleness threshold*. Leveraging this 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 converge efficiently and achieve high accuracy [13, 14, 18, 26].

With the relatively flexible synchronization model (i.e., SSP, compared to the original BSP), subsequent studies (including federated learning) [17, 25, 40] extensively explored the scheduling of synchronization among workers according to network conditions and training process.

Gradient Compression. Gradient compression greatly reduces the communication traffic and is indeed essential for practical distributed training over wireless networks. Although lossy gradient compression methods [31] (information is lost during compression and can cannot be recovered) could achieve up to 0.1% compression rate (i.e., size after compression divided by original size), they cannot provide convergence guarantee [31]. Thus in this paper, we only consider lossless compression methods (no information is lost during compression through e.g., error compensation [41]) which have a typical compression rate of around 3% [41].

Even with gradient compression, the communication of gradients is still a bottleneck in distributed training over a team of robotic IoT devices for two reasons. First, the devices typically share a same wireless channel, incurring communication traffic volume proportional to the number of devices on the same channel. 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 yet typically comparable to the computation time. In our experiments, a Jetson Xavier NX [6] device out of a four-device team computed gradients in 2.18s and ideally needed to wait for 1.47s upon the synchronization barrier (four devices push and pull the compressed gradients sized 2.1MByte, summing up to 134.4Mbit), which is comparable to (taking up 67.4%) the computation time. Consequently, the straggler effect, which severely prolongs the communication time due to the sharp and frequent bandwidth fluctuation in real-world robotic IoT networks, still has a major impact on the distributed training process.

3 OVERVIEW

Fig. 3 presents the workflow of RSP and compares it with BSP and SSP. The main idea of RoG is to in real-time align the transmission time among stragglers and non-stragglers by breaking up gradient synchronization granularity and scheduling stragglers to synchronize less rows than non-stragglers

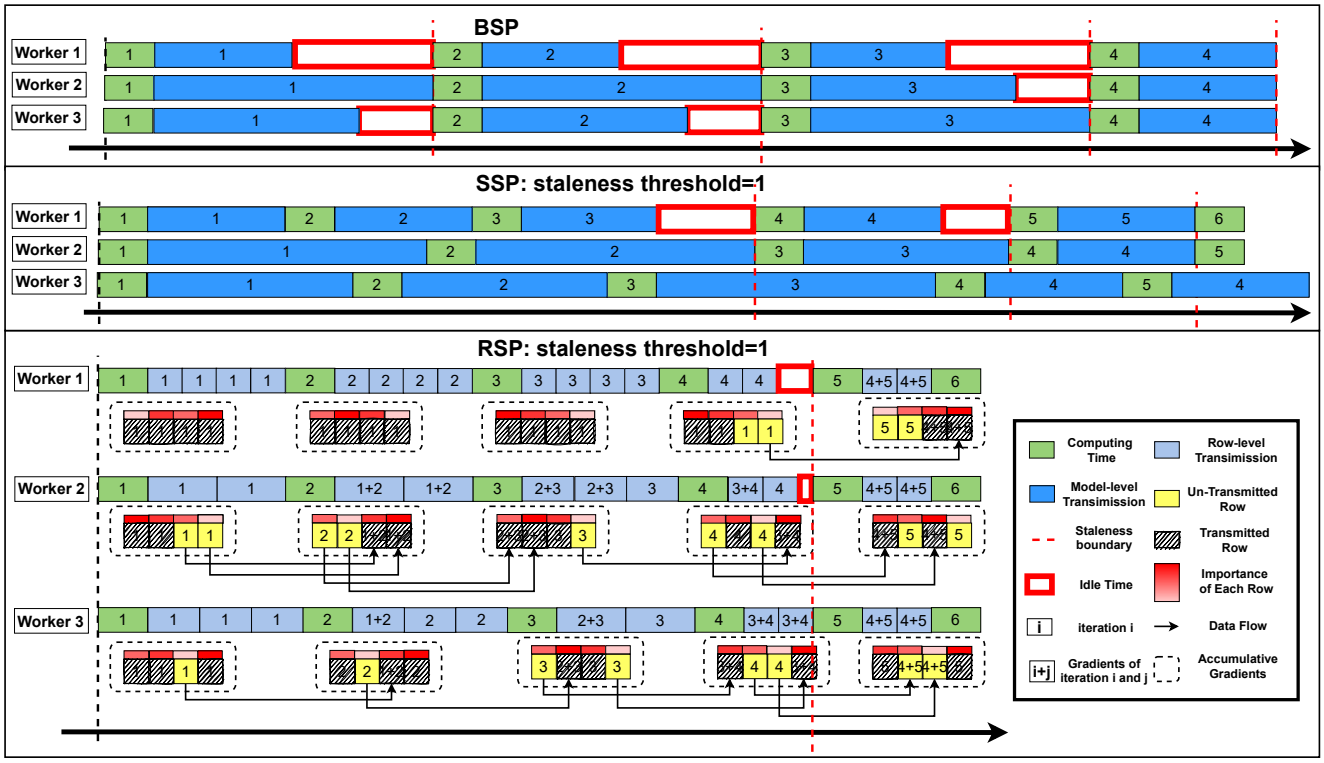


Figure 3: Workflow of RSP. We assume that bandwidth on the three devices are identical at the same time point in the three cases and the bandwidth is interfered by distance, occlusion, etc.

in an iteration. In this way, although wireless bandwidth of all these devices is fluctuating randomly and sharply, stragglers are more likely to spend less time transmitting the gradients than non-stragglers in an iteration and thus will catch up with non-stragglers. 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 one hand, synchronizing gradients of the training model in element granularity requires indices of all elements, taking up data volume comparable to the gradients of a whole model. On the other hand, although a model often consists of only tens to hundreds of layers and the indices of layers cost little, a layer can be large at size (e.g., millions of parameters) and thus the delay in transmission of a layer can cause a straggler to severely fall behind.

The model rows (matrix rows) we choose for RoG is a balanced choice between management overhead and flexibility of transmission. First, the total data volume of indices of rows is empirically small compared with the model size (e.g., 0.38% of the model size in our evaluation). Second, each row of a model is typically sized to several to hundreds of elements. If we adapt the gradient transmission scheduling of rows from stragglers when their bandwidth fluctuates, we only spend a negligible cost of transmitting gradients of a row in degraded bandwidth, making row granularity the best choice of RoG.

Row Stale Synchronous (RSP). Since RoG does not synchronize all gradients of each device in a iteration, gradients of different rows of the training model on a device can have staled for different iterations, making convergence guarantee a challenge. While BSP is infeasible because it causes severe stall cost, a completely asynchronous method without any staleness control is also infeasible as it has no convergence guarantee [54]. Instead, 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 [24] (see Sec. 5.3).

Based on this finding, we design RSP which 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. Otherwise, idle time will be inserted until the above two-level staleness control requirements are met as shown in Fig. 3. RSP provably achieves the same level of convergence guarantee as SSP (see Sec. 5.3).

Adaptive Transmission Protocol (ATP). Instead of scheduling on a large granularity that cannot adapt to real-time bandwidth fluctuation, ROG leverages a real-time adaptive scheduling method named Adaptive Transmission Protocol (ATP). First, while ROG allows stragglers to synchronize a portion of their rows with other devices, there is a lower bound to the number of rows (referred to as Minimum Transmission Amount (MTA)) to be exchanged every iteration to avoid triggering the staleness threshold in RSP. For example, the MTA for the threshold 2 is 50%, because if less than 50% of the total number of rows in a model on a device are synchronized every iteration, then after two iterations there will inevitably be rows not pushed to the parameter server for two iterations, causing other devices to stall.

Second, to minimize stall time and optimize convergence efficiency, ATP on each device maintains the importance of each row based on its possibility to cause stall (e.g., the staled version) and the contribution of gradients of each row to model convergence (e.g., the absolute value of the gradients); the rows with highest importance will be transmitted first. For a straggler in an iteration, ATP controls it to transmit MTA of the total rows and reports the transmission time of MTA (MTA time) on the straggler 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 no device would fall behind.

Third, besides the management overhead, smaller granularity also brings extra transmission overhead. To control non-stragglers to keep transmitting rows for MTA time, a straight forward approach is inserting bubbles between the transmission of every two successive rows to 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 by enabling 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. 6). In this way, the transmission overhead is reduced to possibly discarding the last row transmitted if its transmission is incomplete, which is also negligible.

4 ARCHITECTURE OF ROG

Fig. 4 shows the architecture of ROG and ATP. On each worker device 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 ROG will determine the importance of each row in Importance Metric according to the row's staled version and the average absolute value of the gradients. Prioritized Transmission ranks the rows based on their importance and transmits those with highest importance first. Prioritized Transmission keeps transmitting for the aforementioned MTA time that 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 of rows, and updates Version Storage. Specifically, ROG records the latest iteration number (version) of the received gradients (the iteration that generates the gradients) for each worker and maintains the latest versions of the gradients that are sent to each worker in Version Storage, so that RSP can determine whether the versions of this row satisfy the requirements of RSP. If so, ROG will similarly determine the importance of the rows in Importance Metric and transmit the most important rows' gradients in Prioritized Transmission for MTA time; 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 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.

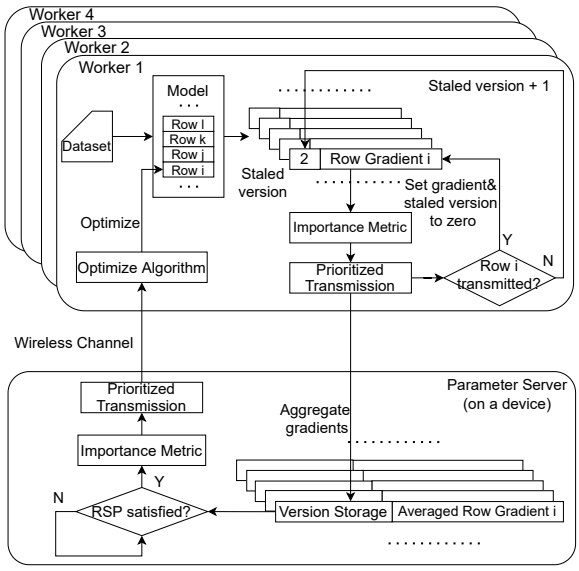


Figure 4: Architecture of RoG. On the worker side, RoG will set the gradients of the transmitted rows to zero so that the gradients will not accumulate; gradients of the not transmitted rows will accumulate as designed by Torch. Note that on the parameter server, similar to the worker side, RoG will check whether the row is transmitted by Prioritized Transmission; if so RoG will set the gradients to zero and modify the version storage accordingly. We leave out this part on the figure for simplicity.

5 DETAILED DESIGN

5.1 RoG's algorithm

We propose RoG's algorithm to provide a more fine-grained staleness control in order to meet the need to simultaneously achieve fewer worker stalls and better statistical efficiency (i.e., reduction of loss per training iteration). The parameter server part is given in Alg.1, while the local worker part is given in Alg.2.

Algorithm 1 Parameter Server of RoG

Function *ParameterServer_RoG()*: // On server
Data: w: global model; w_i: ith row of w; v_i^r: the times of w_i updates by worker r; V_i: set of v_i^r with the same i; η: learning rate
1 **upon** *receive gradients from worker r* **do**
2 **for** *each* $g\prime_i, n_i$ *received* **do**
3 $v_i^r \leftarrow v_i^r + n_i \;\; w_i \leftarrow w_i - \eta g\prime_i$
4 **for** *each row* w_i *in* w **do**
5 send w_i, V_i together to worker r with ATP control

Algorithm 2 Local Worker of RoG

Function *LocalWorker_RoG()*: // On workers
Data: w: local model
6 $g \leftarrow 0$ **for** *each iteration* **do**
7 $i \leftarrow$ iteration times PullModel(w, i, g) gradient $g \leftarrow$ Training(w) PushGradients(g)
8 **Function** *PullModel(w, iter, g)*: // On workers
Data: w: local model; $iter$: iteration times; g: last iteration's gradient; T_i: threshold of ith row;
9 **for** *each* $w\prime_i$ *in* w **do**
10 $w_i \leftarrow w_i - \eta g_i$
11 **end**
12 **for** *each* $w\prime_i, V\prime^i$ *received from server* **do**
13 $w_i \leftarrow w\prime_i \;\; V_i \leftarrow V\prime_i$
14 **end**
15 **for** *each row* w_i *in* w **do**
16 **if** *TriggerThreshold*(V_i, $iter$, T_i) **then**
17 wait for new w_i
18 **end**
19 **end**
20 **Function** *PushGradients(g)*: // On workers
Data: g: gradients; r: the rank of this worker; $g\prime$: the sum of gradients that have not yet been transmitted successfully; n: the number of gradients included in $g\prime$
21 **for** *each row* g_i *in* g **do**
22 $g\prime_i \leftarrow g\prime_i + g_i \;\; n_i \leftarrow n_i + 1$ send $g\prime_i, n_i$ together from worker r to server with ATP control
23 **end**
24 **for** *each row* $g\prime_i$ *sent successfully* **do**
25 $g\prime_i \leftarrow 0 \;\; n_i \leftarrow 0$
26 **end**

It should be noted that, in Function *PullModel*, we update the local model with local-computed gradients in *line 9* before receiving the new model in *line 12*. It is because the worker will wait until it receives the parameters from the parameter server, and we can take advantage of such waiting time in this way.

5.2 Adaptive Transmission Protocol

ATP has two main problems to solve: assigning priorities to different rows and handling the unstable bandwidth capacity among wireless connections over different devices.

First of all, we define the urgency degree of each row as how far from this row triggers its staleness threshold, which actually is the number of the corresponding worker's iterations minus this row's most minor updated of all the workers. In addition, RoG set the urgency degree of the row, which is one step away from triggering threshold, to infinity,

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

ensuring such row is chosen to been transferred in this time to avoid the trigger of the threshold.

For parameter rows at parameter server side, Rog directly assign priorities based on each row's urgency degree. For gradient rows at the worker side, considering the differences between rows' gradient, Rog multiplies the sum of the absolute values of the gradient by the urgency degree and arrange priorities based on this calculation result. Since infinity times any number is still infinity, the most urgent row has the highest priority.

Assuming that the staleness threshold is S and the gradients calculated each time are independent of each other, the priority assigned to each row is random without triggering the threshold. If the percentage of rows to be transferred each time is fixed to P, the probability that a row keeps remaining until one step away from triggering threshold is $(1-P)^{S-1}$. In order to ensure such an urgent row can be transferred at this time, there should be $(1-P)^{S-1} < P$. 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. In particular, we set the MTA to 0.5 when the threshold is 1.

Then, we leverage the most widely used feature in wireless scenarios: *throught* to measure the bandwidth capacity of each worker, which is responsible for determining the speed of the transmission. ATP monitors the throughput of all workers and calculates the ratio q of the throughput of the worker, where this ATP progress is, to the minimum throughput among workers during run time. By letting each worker transfer $q \times MTA$ percent of the rows, Rogmakes all workers complete the transfer simultaneously. Since the upper bound of $q \times MTA$ is 100%, when $q \geq \frac{1}{MTA}$, there is still a difference in the transmission time between workers. In other words, when the ratio of maximum throughput to minimum throughput is less than $\frac{1}{MTA}$, Rog can handle the network instability, and when the ratio is more than $\frac{1}{MTA}$, Rog could mitigate tails with the help of ATP.

5.3 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 [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, assuming that $||\nabla f_t(x)|| \leq L$ for all t (i.e. f_t are L-Lipschitz), and that $MAX_{x,x\prime \in \chi} D(x||x\prime) \leq F^2$ (the optimization problem has bounded diameter), we claim that*

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

Proof. The analysis mainly follows [55], except where the Lemma 1 is involved. By the properties of convex functions, $R[x] = \sum_{t=1}^{T}(f_t(\tilde{x}_t) - f(x^\star)) \leq \sum_{t=1}^{T}\left\langle \nabla f_t(\tilde{x}_t), \tilde{x}_t - x^\star \right\rangle = \sum_{t=1}^{T}\left\langle \tilde{g}_t, \tilde{x}_t - x^\star \right\rangle$, where we have defined $\tilde{g}_t = \nabla f_t(\tilde{x}_t)$. The high-level idea is to show that $R[x] \leq o(T)$, which implies $E_t[f_t(\tilde{x}_t) - f_t(x^\star)] \to 0$ and thus convergence. □

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

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

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

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

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 [24].

Lemma 2. *For P workers under SSP with staleness threshold $S\prime$, we assume that $||\nabla f_{\prime t}(x)|| \leq L\prime$ and $max_{x^i, x\prime^i \in X^i} D(x^i||x\prime^i) \leq F\prime^2$. If we set the initial step size $\sigma = \frac{F}{L\sqrt{2\kappa}}$, where $\kappa = (s+1)P$, 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 2 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 $R[X] \leq M * MAX(\sum_{t=1}^{T} \langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \rangle) = M * 4F\prime L\prime \sqrt{2(S_{max}+1)PT}$. Since $F\prime$ and $L\prime$is still unknown, we need to go further and find a more specific upper boundary.

Lemma 3. *If $max_{x,x\prime \in X} D(x||x\prime) \leq F^2$, and $max_{x^i, x\prime^i \in X^i} D(x^i||x\prime^i) \leq F\prime^2$ 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 $R[X] \leq M*4F\prime L\prime \sqrt{2(S+1)PT} = M*4\frac{F}{\sqrt{M}}\frac{L}{\sqrt{M}}\sqrt{2(S+1)PT} = 4FL\sqrt{2(S+1)PT}$.

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.

6 IMPLEMENTATION

RoG is implemented in PyTorch [11] with nearly 1200 LoC. It inspects the underlying tensors storing ML model parameters. As ML model parameters in PyTorch are stored with multiple tensors, RoG divides each tensor by row and keeps recording each row's updated versions. Since rows from different tensors usually have different sizes and the number of elements in different tensors vary greatly, RoG detects the maximum packet size (i.e., 1500 bytes for Ubuntu 20.04 [10]) and packs several rows with small sizes as a whole, making the overall size close to, but not exceeding, the maximum packet size, increasing the utilization of each packet. RoG's design and implementation are common for all distributed ML training based on parameter server and can be adopted by modifying only tens of LoC.

Our method to handle the transmission overhead since conceptually we are scheduling in a row-by-row manner.

7 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. The Wi-Fi connections among these devices were configured on channel 36 (5GHz) and achieved an average bandwidth capacity of around 180Mbps in a clean environment. Since the heterogeneity of computation power between laptop and robots is out of our scope, we adopt [43] to make all the involved devices spend the same time computing on a batch of data in each iteration.

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 facilities 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 [41] (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 and Applications. We compared ROG with SSP [24] and the framework proposed in [17] (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. Both paradigms (EACOL and MPCOL) are fundamental [23, 33]cite more to 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.

In 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. 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 [12] 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 [30] and we randomly allocates 50 shards for each robot without overlap. 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. We follow the methods of DeepTest [42], 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%).

In MPCOL, a team of robots plan route to specific locations while avoiding collisions in a working space (e.g., a damaged building). When robots enter a new working space, distributed training is required to adapt the robots' route planning models to new obstacle patterns in order to recover route planning performance as soon as possible. In MPCOL, we adopted [32] as the workload and performed distributed training on our testbed with real robots and wireless networks. More specific? Model size? performance?

A form 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.

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?

7.1 Performance under EACOL

line chart: Accuracy - wall clock time: SSP-HT, SSP-LT, FLOWN(-MT), ROG-HT LT

line chart: Accuracy - iteration

bar chart: Computation time, Communication time + time server waiting for gradients from worker

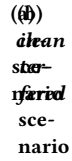


Figure 5: Accuracy - wall clock time; EACOL under different scenarios. X axis is time, y axis is training accuracy, and several lines are baselines and ROG

7.2 Performance under MPCOL

7.3 Ablation Study of ROG

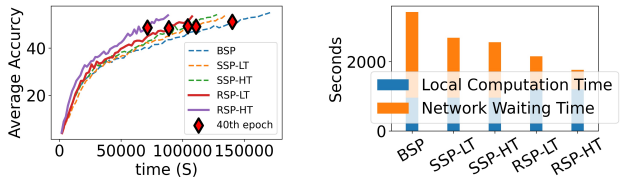
disable importance schedule disable gradient disable threshold

From the previous two sections, we can see that ROG accelerates the speed of adapting to the new environment by ??? in real-wrold application paradigms on robots, and we will further study how ROG speeds up the model convergence.

((a)) lab-scenario

Figure 6: MPCOL under different scenarios. X axis is time, y axis is training accuracy, and several lines are baselines and RoG

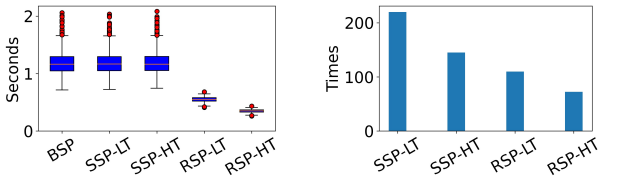
We compared RSP with the baselines in terms of model training accuracy and collective communication time per epoch to validate the effectiveness of RSP. In Fig. 7, we evaluated RSP and baselines in a clean WLAN without any obstacles existing based on our testbed.



(a) Average accuracy over time. **(b) Collective communication time.**

Figure 7: In a clean WLAN without obstacles.

Fig. 7a shows RSP and baselines' convergence rate over time, where the y-axis is the average training accuracy, and the red dot indicates the 40th epoch's finish time and average training accuracy. The red dots of RSP and SSP are basically at the same horizontal line, which means RSP did not downgrade the final training accuracy compared with SSP. With the highest red dot in Fig. 7a, BSP has the highest statistical efficiency (i.e., increase of model accuracy per training iteration) because it stalls whenever the completion time to transfer model updates on any device defer due to unstable network bandwidth capacity. Fig. 7b shows RSP and baselines' collective communication time and local computation time, and the sum of them is the average epoch time taken by RSP and baselines. We see that RSP-HT achieved the fastest convergence: reducing 75% collective communication time of BSP and 49% average epoch time of BSP eventually. No matter how big the threshold is, the RSP increased nearly 25% overhead of local computation time in Fig. 7b. It is because local computation time includes calculating model update gradients and controlling RSP's model update by ATP, executed during each iteration regardless of staleness threshold. Collective communication time of RSP was greatly reduced in Fig. 7b without downgrading the training accuracy in Fig. 7b, so that RSP achieved 1.25X speedup as compared to SSP under the same low staleness threshold, and 1.45X speedup under the same high one.



(a) Transmission time. **(b) The number of times triggering threshold.**

Figure 8: Microanalysis of RSP and baselines.

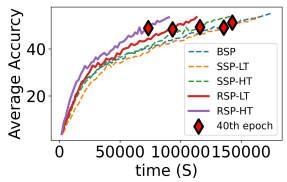
Then, we give some detailed information about microanalysis during ML training to elaborate how RSP reduced collective communication time. Network waiting time includes the transmission time for exchanging gradients or parameters and the blocked time due to the synchronization barrier (i.e., when the worker is not doing computation nor communication). Fig. 8a shows the time for each transmission of gradients and model under RSP and other baselines, where the red dots represent outliers of tail latency from wireless network jitters. Since all baselines adopt a model-granulated synchronization, they permanently transferred the entire model under the same wireless network conditions and followed the same pattern with a similar mean and variance of transmission time. With the help of ATP, RSP reduced the volume of each transmission and reduced mean and variance of transmission time in Fig. 8a. The smaller the mean, the faster each transfer completes, while the smaller the variance, the less likely it is to trigger the threshold. Such that the number of times triggering thresholds was reduced by 50% for RSP-LT and 70% for RSP-HT in Fig. 8b. Since it is in a clean WLAN, workers' bandwidth capacity was basically the same, and the outliers mainly caused the triggering of threshold in the Fig. 8a.

7.4 Sensitivity Studies

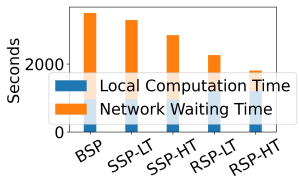
ROG different threshold

ROG different compression rate; too long

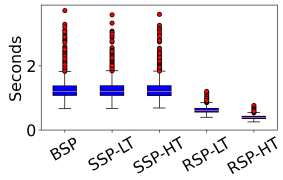
We further evaluated RSP with different staleness thresholds (i.e., from 2 to 8), whose environment configuration followed the setting as in Fig. 10, because we found that in the case of Fig. 10, RSP-LT could not handle such a dramatic instability. Therefore, we wanted to find out the upper bounds on the instability among wireless connections that RSP with different staleness thresholds can handle. Fig. 12 shows the



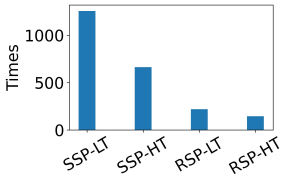
(a) Average accuracy over time.



(b) Collective communication time.

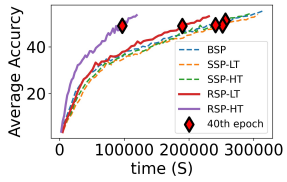


(c) Transmission time.

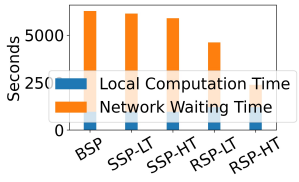


(d) The number of times triggering threshold.

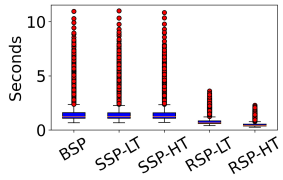
Figure 9: In a through-metal-sheet WLAN.



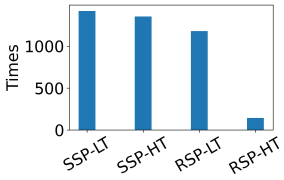
(a) Average accuracy over time.



(b) Collective communication time.



(c) Transmission time.



(d) The number of times triggering threshold.

Figure 10: In a through-wall WLAN.

performance of RSP with different staleness thresholds in a through-wall WLAN.

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 5.2, 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.

7.5 Lessons learned

Finer granularity brings more mangement overhead and transmission overhead. As discussed in Section 3, a smaller granularity of synchronization brings extra management overhead (e.g., index) and transmission overhead in the wireless distributed training process. Although we chose a balanced granularity of rows, which takes up small data volume compared with the model size and we minimize the transmission overhead by enabling speculative transmission with the co-design of ATP and the underlying transmission protocol as discussed in Section 6, such overhead cannot be completely eliminated and potentially limits the performance gain of RoG over the baselines.

The selection of staleness threshold. As shown in Sensitivity Study above, RoG's performance gain over SSP is evident (e.g., 44% reduction of the convergence time) with the staleness threshold greater than 5, but RoG suffered a similar level of stall cost as SSP with a smaller staleness threshold, e.g., 2. That is because RoG is expecting a minimum portion of the gradients of the whole model (MTA) to catch up in each iteration to avoid violating RSP and triggering stall, and the MTA is inversely proportional to the staleness threshold. It is difficult for half (MTA is 0.5 with threshold 2) of the total gradients of the model from stragglers to catch up in unstable environments with a small threshold. It is recommended to use a moderate staleness threshold (e.g., 5) in unstable environments, which is also recommended by SSP [24].

Limitations.

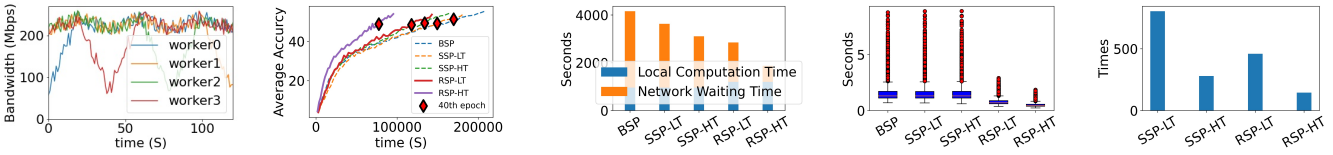
8 CONCLUSION

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

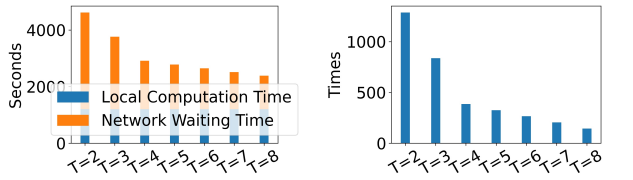
[1] [n.d.]. *Datacenter Traffic Control: Understanding Techniques and Tradeoffs | IEEE Journals & Magazine | IEEE Xplore.* https://ieeexplore.ieee.org/abstract/document/8207422

[2] [n.d.]. IEEE 802.11ac-2013. https://en.wikipedia.org/w/index.php?title=IEEE_802.11ac-2013&oldid=1076948038 Page Version ID: 1076948038.



(a) Bandwidth capacity. **(b) Average accuracy over time.** **(c) Collective communication time.** **(d) Transmission time.** **(e) The number of times triggering threshold.**

Figure 11: In a WLAN with dynamic obstacles.



(a) Transmission time. **(b) The number of times triggering threshold.**

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

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

[4] [n.d.]. *Jetson AGX Orin.* https://developer.nvidia.com/embedded/jetson-agx-orin

[5] [n.d.]. *NVIDIA Embedded Systems for Next-Gen Autonomous Machines.* https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[6] [n.d.]. NVIDIA Jetson Xavier NX GPU Specs. https://www.techpowerup.com/gpu-specs/jetson-xavier-nx-gpu.c3642

[7] [n.d.]. *On the shoulders of giants: recent changes in Internet traffic.* http://blog.cloudflare.com/on-the-shoulders-of-giants-recent-changes-in-internet-traffic/

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

[9] 2020. Jetson Modules. https://developer.nvidia.com/embedded/jetson-modules

[10] 2021. *Enterprise Open Source and Linux.* https://ubuntu.com/

[11] 2021. *PyTorch.* https://www.pytorch.org

[12] 2022. *tff.simulation.datasets.cifar100.load_data | TensorFlow Federated.* https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/cifar100/load_data

[13] Ricardo Barandela, Rosa M. Valdovinos, J. Salvador Sánchez, and Francesc J. Ferri. [n.d.]. The Imbalanced Training Sample Problem: Under or over Sampling?. In *Structural, Syntactic, and Statistical Pattern Recognition* (Berlin, Heidelberg, 2004) *(Lecture Notes in Computer Science)*, Ana Fred, Terry M. Caelli, Robert P. W. Duin, Aurélio C. Campilho, and Dick de Ridder (Eds.). Springer, 806–814. https://doi.org/10.1007/978-3-540-27868-9_88

[14] Sakyajit Bhattacharya, Vaibhav Rajan, and Harsh Shrivastava. [n.d.]. ICU Mortality Prediction: A Classification Algorithm for Imbalanced Datasets. 31, 1 ([n. d.]). https://ojs.aaai.org/index.php/AAAI/article/view/10721 Number: 1.

[15] Léon Bottou. [n.d.]. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT'2010* (Heidelberg, 2010), Yves Lechevallier and Gilbert Saporta (Eds.). Physica-Verlag HD, 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16

[16] Mingzhe Chen, Deniz Gündüz, Kaibin Huang, Walid Saad, Mehdi Bennis, Aneta Vulgarakis Feljan, and H. Vincent Poor. 2021. Distributed Learning in Wireless Networks: Recent Progress and Future Challenges. *arXiv:2104.02151 [cs, math]* (April 2021). http://arxiv.org/abs/2104.02151 arXiv: 2104.02151.

[17] Mingzhe Chen, Zhaohui Yang, Walid Saad, Changchuan Yin, H. Vincent Poor, and Shuguang Cui. 2021. A Joint Learning and Communications Framework for Federated Learning Over Wireless Networks. *IEEE Transactions on Wireless Communications* 20, 1 (Jan. 2021), 269–283. https://doi.org/10.1109/TWC.2020.3024629 Conference Name: IEEE Transactions on Wireless Communications.

[18] Xiangtao Chen, Lan Zhang, Xiaohui Wei, and Xinguo Lu. [n.d.]. An effective method using clustering-based adaptive decomposition and editing-based diversified oversamping for multi-class imbalanced datasets. 51, 4 ([n. d.]), 1918–1933. https://doi.org/10.1007/s10489-020-01883-1

[19] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing. [n.d.]. Exploiting Bounded Staleness to Speed Up Big Data Analytics. 37–48. https://www.usenix.org/conference/atc14/technical-sessions/presentation/cui

[20] Ming Ding, Peng Wang, David Lopez-Perez, Guoqiang Mao, and Zihuai Lin. [n.d.]. Performance Impact of LoS and NLoS Transmissions in Dense Cellular Networks. 15, 3 ([n. d.]), 2365–2380. https://doi.org/10.1109/TWC.2015.2503391 arXiv:1503.04251

[21] Michael Everett, Yu Fan Chen, and Jonathan P. How. 2021. Collision Avoidance in Pedestrian-Rich Environments With Deep Reinforcement Learning. *IEEE Access* 9 (2021), 10357–10377. https://doi.org/10.1109/ACCESS.2021.3050338 Conference Name: IEEE Access.

[22] Alexandras V. Gerbessiotis and Leslie G. Valiant. [n.d.]. Direct bulk-synchronous parallel algorithms. In *Algorithm Theory — SWAT '92* (Berlin, Heidelberg, 1992) *(Lecture Notes in Computer Science)*, Otto Nurmi and Esko Ukkonen (Eds.). Springer, 1–18. https://doi.org/10.1007/3-540-55706-7_1

[23] Ruihua Han, Shengduo Chen, and Qi Hao. 2020. Cooperative Multi-Robot Navigation in Dynamic Environment with Deep Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 448–454. https://doi.org/10.1109/ICRA40945.2020.9197209 ISSN: 2577-087X.

[24] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. [n.d.]. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *Advances in Neural Information Processing Systems*

(2013), Vol. 26. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/hash/b7bb35b9c6ca2aee2df08cf09d7016c2-Abstract.html

[25] Hanpeng Hu, Dan Wang, and Chuan Wu. [n.d.]. Distributed Machine Learning through Heterogeneous Edge Systems. 34, 5 ([n. d.]), 7179–7186. https://doi.org/10.1609/aaai.v34i05.6207

[26] Harsurinder Kaur, Husanbir Singh Pannu, and Avleen Kaur Malhi. [n.d.]. A Systematic Review on Imbalanced Data Challenges in Machine Learning: Applications and Solutions. 52, 4 ([n. d.]), 79:1–79:36. https://doi.org/10.1145/3343440

[27] Nicolas Kourtellis, Kleomenis Katevas, and Diego Perino. 2020. FLaaS: Federated Learning as a Service. In *Proceedings of the 1st Workshop on Distributed Machine Learning (DistributedML'20)*. Association for Computing Machinery, New York, NY, USA, 7–13. https://doi.org/10.1145/3426745.3431337

[28] Jiachen Li, Ali Hassani, Steven Walton, and Humphrey Shi. [n.d.]. ConvMLP: Hierarchical Convolutional MLPs for Vision. ([n. d.]). arXiv:2109.04454 http://arxiv.org/abs/2109.04454

[29] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. [n.d.]. Scaling Distributed Machine Learning with the Parameter Server. 583–598. https://www.usenix.org/conference/osdi14/technical-sessions/presentation/li_mu

[30] Wei Li and Andrew McCallum. [n.d.]. Pachinko allocation: DAG-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning - ICML '06* (Pittsburgh, Pennsylvania, 2006). ACM Press, 577–584. https://doi.org/10.1145/1143844.1143917

[31] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. [n.d.]. DEEP GRADIENT COMPRESSION: REDUCING THE COMMUNICATION BANDWIDTH FOR DISTRIBUTED TRAINING. ([n. d.]), 14.

[32] Tianyu Liu. 2018. Robot Collision Avoidance via Deep Reinforcement Learning. https://github.com/Acmece/rl-collision-avoidance.git.

[33] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. 2018. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6252–6259. https://doi.org/10.1109/ICRA.2018.8461113 ISSN: 2577-087X.

[34] Antoni Masiukiewicz. [n.d.]. Throughput comparison between the new HEW 802.11ax standard and 802.11n/ac standards in selected distance windows. 65, 1 ([n. d.]), 79–84. http://www.ijet.pl/index.php/ijet/article/view/10.24425-ijet.2019.126286 Number: 1.

[35] Jihong Park, Sumudu Samarakoon, Anis Elgabli, Joongheon Kim, Mehdi Bennis, Seong-Lyun Kim, and Mérouane Debbah. 2021. Communication-Efficient and Distributed Learning Over Wireless Networks: Principles and Applications. *Proc. IEEE* 109, 5 (May 2021), 796–819. https://doi.org/10.1109/JPROC.2021.3055679 Conference Name: Proceedings of the IEEE.

[36] Yuanteng Pei, Matt W. Mutka, and Ning Xi. [n.d.]. Connectivity and bandwidth-aware real-time exploration in mobile robot networks. 13, 9 ([n. d.]), 847–863. https://doi.org/10.1002/wcm.1145 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcm.1145.

[37] Jorge Pena Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. [n.d.]. Collaborative Multi-Robot Search and Rescue: Planning, Coordination, Perception, and Active Vision. 8 ([n. d.]), 191617–191643. https://doi.org/10.1109/ACCESS.2020.3030190 Publisher: IEEE.

[38] Nurul I. Sarkar and Osman Mussa. [n.d.]. The effect of people movement on Wi-Fi link throughput in indoor propagation environments. In *IEEE 2013 Tencon - Spring* (2013-04). 562–566. https://doi.org/10.1109/TENCONSpring.2013.6584508

[39] Uday Shankar Shanthamallu, Andreas Spanias, Cihan Tepedelenlioglu, and Mike Stanley. 2017. A brief survey of machine learning methods and their sensor and IoT applications. In *2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)*. 1–8. https://doi.org/10.1109/IISA.2017.8316459

[40] Wenqi Shi, Sheng Zhou, and Zhisheng Niu. 2020. Device Scheduling with Fast Convergence for Wireless Federated Learning. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC40277.2020.9149138 ISSN: 1938-1883.

[41] Jun Sun, Tianyi Chen, Georgios Giannakis, and Zaiyue Yang. 2019. Communication-Efficient Distributed Learning via Lazily Aggregated Quantized Gradients. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc. https://papers.nips.cc/paper/2019/hash/4e87337f366f72daa424dae11df0538c-Abstract.html

[42] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[43] Sahil Tyagi and Prateek Sharma. [n.d.]. Taming Resource Heterogeneity In Distributed ML Training With Dynamic Batching. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)* (2020-08). 188–194. https://doi.org/10.1109/ACSOS49614.2020.00041

[44] E. Vidal, J. D. Hernández, N. Palomeras, and M. Carreras. [n.d.]. Online Robotic Exploration for Autonomous Underwater Vehicles in Unstructured Environments. In *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)* (2018-05). 1–4. https://doi.org/10.1109/OCEANSKOBE.2018.8559224

[45] Mei Wang and Weihong Deng. 2018. Deep visual domain adaptation: A survey. *Neurocomputing* 312 (Oct. 2018), 135–153. https://doi.org/10.1016/j.neucom.2018.05.083

[46] Yan Wang, Zihang Lai, Gao Huang, Brian H. Wang, Laurens van der Maaten, Mark Campbell, and Kilian Q. Weinberger. [n.d.]. Anytime Stereo Image Depth Estimation on Mobile Devices. ([n. d.]). arXiv:1810.11408 http://arxiv.org/abs/1810.11408

[47] Garrett Wilson and Diane J. Cook. 2020. A Survey of Unsupervised Deep Domain Adaptation. *ACM Transactions on Intelligent Systems and Technology* 11, 5 (July 2020), 51:1–51:46. https://doi.org/10.1145/3400066

[48] Joohyun Woo and Nakwan Kim. 2020. Collision avoidance for an unmanned surface vehicle using deep reinforcement learning. *Ocean Engineering* 199 (March 2020), 107001. https://doi.org/10.1016/j.oceaneng.2020.107001

[49] Yecheng Xiang and Hyoseung Kim. [n.d.]. Pipelined Data-Parallel CPU/GPU Scheduling for Multi-DNN Real-Time Inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)* (2019-12). 392–405. https://doi.org/10.1109/RTSS46320.2019.00042 ISSN: 2576-3172.

[50] Kyon-Mo Yang, Jong-Boo Han, and Kap-Ho Seo. [n.d.]. A Multi-robot Control System based on ROS for Exploring Disaster Environment. In *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)* (2019-11). 168–173. https://doi.org/10.1109/ICCMA46720.2019.8988650

[51] Shuochao Yao, Yifan Hao, Yiran Zhao, Huajie Shao, Dongxin Liu, Shengzhong Liu, Tianshi Wang, Jinyang Li, and Tarek Abdelzaher. [n.d.]. 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)* (2020-08). 1–10. https://doi.org/10.1109/RTCSA50079.2020.9203676 ISSN: 2325-1301.

[52] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris. 2019. A Review of Machine Learning and IoT in Smart Transportation. *Future Internet* 11, 4 (April 2019), 94. https://doi.org/10.3390/fi11040094 Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[53] Yaowen Zhang, Dianxi Shi, Yunlong Wu, Yongjun Zhang, Liujing Wang, and Fujiang She. 2021. Networked Multi-robot Collaboration in Cooperative–Competitive Scenarios Under Communication Interference. In *Collaborative Computing: Networking, Applications and Worksharing*, Honghao Gao, Xinheng Wang, Muddesar Iqbal, Yuyu Yin, Jianwei Yin, and Ning Gu (Eds.). Vol. 349. Springer International Publishing, Cham, 601–619. https://doi.org/10.1007/978-3-030-67537-0_36

[54] Xing Zhao, Aijun An, Junfeng Liu, and Bao Xin Chen. [n.d.]. Dynamic Stale Synchronous Parallel Distributed Training for Deep Learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)* (2019-07). 1507–1517. https://doi.org/10.1109/ICDCS.2019.00150 ISSN: 2575-8411.

[55] Martin Zinkevich, Alexander J. Smola, and John Langford. [n.d.]. Slow Learners are Fast. https://openreview.net/forum?id=SkV6_uW_-B