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

ABSTRACT

The prosperity of IoT (Internet of Things) and IoRT (Internet of Robotic Things) and ML (Machine Learning) pushes more and more ML applications deployed on wireless devices, on which data parallel training can harness the distributed hardware resources to fast adapt to changing environments. However, in wireless networks, there typically exists network instability among wireless devices, where the network bandwidth capacity between two devices can be easily interfered by mobile obstacles and varying distances. This instability often leads to severe stall (thus, under-utilization) for devices when a model synchronization barrier (e.g., bulk synchronizes parallel) is involved. Recent staled synchronous parallel (SSP) methods attempt to reduce such stall by allowing devices to proceed training with a staled model updates from devices with unstable networks. However, all these methods are model-granulated synchronization, and a transient network instability can greatly slow down the transfer and the barrier synchronization of the model.

We present ROG, the first ROw-Granulated high-performance and robust wireless distributed training system optimized for ML training over unstable wireless networks. ROG further 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, such that 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.

ACM MobiCom '22, October 25–29, 2022, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

model adaptation, wireless distributed training, synchronization model

ACM Reference Format:

. 2022. ROG: A High Performance And Robust Distributed Training System for IoT and IoRT. In *Proceedings of The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '22).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Machine Learning (ML) models for real-world collected data are getting more prevalently deployed in a team of IoT and IoRT devices [34, 45, 45], such as objective recognition models on Mobile Phones [23] and action control models on Robots [16, 42] in outdoor tasks(e.g., rescue [], field exploration []). These models typically require real-time training to adapt their pre-trained parameters to changing environments [39, 41] (e.g., changing weather from sunny to foggy), but it is typically expensive for the devices fine-tuning these models to access a cloud data center for model training due to the the lack of internet access. For fast adaptation and to harness the distributed hardware resources of the abundant devices, such training is often distributedly deployed among the team of devices over IoT and IoRT networks [11, 31].

One open problem of the distributed training process is that its performance should be robust against the *instability* of real-world IoT and IoRT networks [**?**]: sharp bandwidth fluctuation with random duration could happen frequently and randomly on each device due to the unpredictable movement of mobile devices [30, 32], the occlusion from obstacles [15, 33], etc. For example, 40% fluctuation of bandwidth capacity happens every 1.2s on average during indoor robot surveillance (see Section 2).

A distributed training process over mobile devices typically adopts the parameter server paradigm [25]: each device keeps a model copy and computes the model's parameter update (gradients) on its own data iteratively; between iterations, a synchronization barrier (BSP [18]) is inserted, where each device pauses its computation, pushes the produced

gradients of the whole model to and pulls the averaged gradients from a parameter server (located on one mobile device) over wireless networks. The process of training iterates until the shared model converges (i.e., reaches a desired accuracy).

The instability of real-world IoT and IoRT networks turns synchronization into a major bottleneck of the distributed training process by causing the straggler effect (Fig. 1a): under the sharp and random bandwidth fluctuation due to movement, occlusion [17], etc, the transmission of gradients from some devices (i.e., stragglers) can be dramatically delayed (e.g., 8.34X longer recorded when the devices were moving in an unstable environment, see Section 2.1) by degraded network bandwidth capacity; upon the synchronization barrier, 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 the training iterations and convergence time (i.e., wall-clock time to converge).

Even when the communication traffic volume is reduced by gradient compression [27, 36], the communication bottleneck caused by straggler effect is still not removed. For example, in our evaluation, we observed that ideally the communication of the compressed gradients still took up a large portion (38.8%) of the time of the whole training process and the straggler effect in an unstable environment increased the portion of the communication to 75.2% (see Section 2.1), resulting in the convergence time to increase by 147%. We recorded that each device on average stalled (not transmitting gradients or computing due to straggler effect) for 108.1% of the time for computation (referred to as *stall cost*), implying that half of the computation power of the devices was wasted.

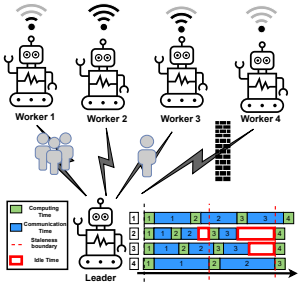
To mitigate the straggler effect, Stale Synchronous Parallel (SSP) [14, 20] allows non-stragglers to continue computing without the latest gradients from stragglers and only stall when the gradients from stragglers fall behind (stale) for a preset number of iterations (staleness threshold). In this way, non-stragglers can seamlessly proceed computing without being blocked by stragglers.

However, when coping with the instability of real-world IoT and IoRT networks, SSP is confronted with a dilemma about the choice of staleness threshold. High convergence efficiency (i.e., less number of iterations to converge) and less stall cost both contribute to minimizing convergence time, but these two requirements are contradictory in SSP because high convergence efficiency (i.e., less number of iterations to converge) of SSP requires a moderate staleness threshold [20], while less stall cost requires a large staleness threshold. In our evaluation in an unstable environment, SSP with a moderate threshold (2) required similar (1.05X) number of iterations as BSP for the training model to converge but still suffered 40.9% stall cost; increasing the threshold to 10 reduced the stall cost to 20.3%, at the cost of 1.73X number of iterations to converge compared with BSP. As a result, convergence time of SSP with moderate threshold and high threshold still increased by 98.3% and 92.1% compared with ideal environment.

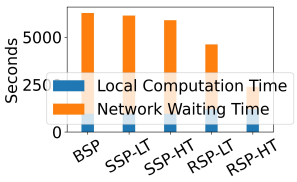
Subsequent studies [12, 21] extend SSP by dynamically assigning the staleness threshold. Specifically, they estimate the bandwidth capacity of different devices and assign higher staleness threshold to devices with low bandwidth capacity and moderate staleness threshold to devices with high bandwidth capacity, so that the straggler effect does not affect the convergence efficiency and stall cost of the non-stragglers. Unfortunately, the random and rapid nature (see Section 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. For example, assume a device is estimated to have high bandwidth capacity and thus assigned a small staleness threshold (e.g., 2), because the time to transmit the gradients of a whole model is relatively larger than the frequency of bandwidth capacity fluctuation in real-world IoT and IoRT networks (see Section 2.1), it can actually take multiplied time compared with the average to actually transmit the gradients of the whole model and result in stall. Consequently, such methods caused on average 80.3% stall cost on each device in an unstable environment and the convergence time was still increased by 88.4% in our evaluation.

The key reason for the problem of the above methods is that they all synchronize the model gradients on the granularity of a whole model, which typically is larger than the granularity (frequency) of bandwidth fluctuation in real-world IoT and IoRT 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 capacity and will be frequently invalidated, causing more stall cost and prolonged convergence time. From the view of model convergence, the scheduling treats all gradients of the training model from a device as a whole. and neglects that gradients from a device have different contribution to model convergence. Transmitting gradients that have less contribution to model convergence (e.g., small absolute value) lowers the overall convergence efficiency. Thus, it is a must to break up the gradient transmission and schedule the transmission of the gradients in a finer granularity

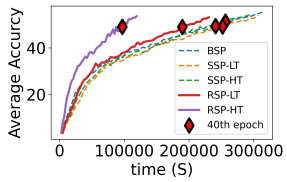
In this paper, we present ROw-Granulated (ROG), a high-performance and robust wireless distributed training system optimized for real-world IoT and IoRT networks. From three possible granularity choices: elements, rows and layers, we find that element granularity requires indexing each element of the whole model for synchronization, taking up data volume comparable to the size of the whole model (high



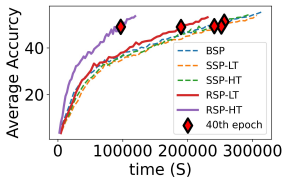
(a) Straggler effect. The communication time is interfered by distance, occlusion, etc.



(b) Stall time comparison.



(c) Accuracy comparison against iterations.



(d) Accuracy comparison against wall-clock time.

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

management cost); layer granularity is large at size and is still comparable with the granularity of bandwidth fluctuation (low transmission flexibility). Thus, RoG decouples each row in the whole model and individually and flexibly synchronizes the gradients of each row on different devices in each iteration to balance between management cost and transmission flexibility. In this way, whenever bandwidth fluctuation happens during gradients transmission, RoG in real-time adapts its scheduling of the number of rows to be transmitted, at a negligible cost of transmitting a row in degraded bandwidth capacity.

Since in RoG the gradients of different rows from each device are synchronized individually and can have different staled versions, a major concern would be how to guarantee convergence in RoG. In view that SSP guarantees convergence by limiting the divergence of training models among different devices within a staleness threshold [20], we find that the same convergence guarantee as SSP can be reached if we also break up the granularity of staleness control to each row (see Section 5.3). Thus we design Row Synchronous Parallel synchronization model (RSP) which extends the staleness control in SSP to each row of a model across different devices and to different rows within a same device. RSP provably achieves the same convergence guarantee as SSP (see Section 5.3).

A major challenge of the design of RoG is how to properly schedule the transmission of each row from different devices based on RSP to minimize stall cost and the convergence time. Following the estimate-then-schedule strategy of the existing work [12], a strawman approach would be scheduling the rows to be transmitted based estimated bandwidth capacity before transmission; such a strategy faces the similar problem that the scheduling will be invalidated due to the bandwidth capacity fluctuation during the scheduled transmission. Instead, RoG adaptively aligns the transmission time of each device with our *Adaptive Transmission Protocol* (ATP): RoG closely monitors the transmission time taken by the transmitted rows and in real-time adapts the scheduling of the pending rows to be transmitted in each iteration, such that all devices roughly spend equal time transmitting gradients in an iteration and no devices will fall behind and cause straggler effect. 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 [6] with nearly 1200 LoC and evaluated RoG under two representative real-world application paradigms on robots with different model sizes: Motion Planning Coordinated Online Learning (MPCOL, gradients sized 3.2 MByte), where a team of robots learn to plan route to specific target positions while avoiding collisions, and Environment Adaptation Coordinated Online Learning (EACOL, 64.8 MByte), where a team of monitoring robots adapt their visual objective recognition model when weather changes. We compared RoG with BSP [18], SSP [20] and a state-of-the-art estimate-then-schedule method specified for wireless networks [12] (referred to as FLOWN) in different real-world IoT and IoRT networks environments. It is notable that we minimized the communication volume with gradient compression [36] (the compressed gradient sizes of the two applications were only 0.1 MByte and 2.1 MByte), so as to conduct the strictest comparison between RSP and the baselines. Evaluation shows that:

- RoG achieved the lowest convergence time. RoG on average reduced the convergence time by xxX, xxX and xxX, compared with BSP, SSP and FLOWN.
- RoG achieved comparable convergence efficiency as BSP and SSP. RoG on average required xxX, xxX and xxX iterations to converge compared with BSP, SSP and FLOWN.
- RoG is easy to use. It took only tens of LoC effort to integrate RoG with the above-mentioned ML learning application based on Pytorch.

Our main contribution is RSP, a new stale synchronous parallel model and the scheduling strategies optimized for minimizing the convergence time of distributed training over real-world IoT and IoRT networks. Based on RSP, ROG conducts fine-grained (i.e., parameter rows) staleness control and transmission scheduling with ATP, so that the transmission time of different devices in each iteration can be balanced and the stall cost is minimized. We prove that RSP provides the same strong convergence guarantee with the notable SSP models [25], and ROG is much more robust to the instability of real-world IoT and IoRT networks. ROG benefits diverse IoT and IoRT applications in the field, such as robot rescue [], disaster response [], and robot surveillance [], making them fast adapt to changing environments under an extremely instable local network, without the access to a cloud data center. 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

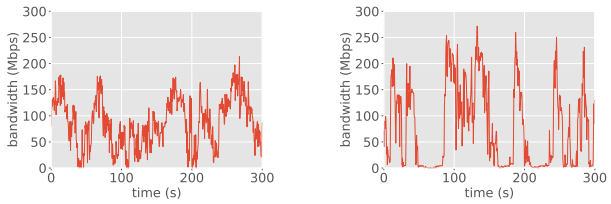
2 BACKGROUND

2.1 Characteristics of IoT and IoRT Wireless Networks

In real-world IoT and IoRT applications, devices typically need to move around. While wireless networks suffice for high mobility, the occlusion of obstacles and the change of distances among devices bring *instability* to the bandwidth capacity: non-negligible change of bandwidth capacity happens in short duration.

To demonstrate such instability, we set up a robot surveillance task: two four-wheel robots navigate around several given points in our lab (indoors) and campus garden (outdoors). The robots carry NVIDIA Jetson Xavier NX [3] as the representative IoT and IoRT hardware, since Jetson modules are well-recognized as the enabler of intelligent IoT and IoRT applications [40, 43, 44], and Jetson Xavier NX is one of the most recent models [2]. We believe our choice represents the state-of-the-art computation and communication capabilities of IoT and IoRT devices.

We saturated the wireless network connection with iperf [1] and recorded the average bandwidth capacity between these two robots every 0.1s for 5 minutes, shown in Fig. 2. The average bandwidth capacity was 95Mbit/s indoors and 74Mbit/s outdoors. We also analyzed the time for the bandwidth capacity to have at least a specific percentage of change, and found that, on average, a 20% change of bandwidth capacity happens every 0.4s, and a 40% change typically happens every 1.2s. Such times are comparable to the time of transmitting compressed gradients recorded with ideal wireless networks (1.43 s). Worse, since the average bandwidth capacity is also degraded in real-world IoT and IoRT networks, the transmission time of the compressed gradients becomes even longer, and the fluctuation of bandwidth capacity happens more frequently in real-world IoT and IoRT networks during each single transmission.



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

Figure 2: The instability of IoT and IoRT networks. It typically takes less than 0.6s for a 20% change of bandwidth capacity, comparable to the time of transmitting compressed model gradients.

2.2 Existing Distributed Training Methods

Vanilla Distributed Training. Distributed training 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. 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 each worker sees others' latest updates (BSP). Such a process iterates continuously until the model is good enough (usually determined with loss). The vanilla distributed training method (i.e., BSP) could easily get blocked by stragglers [20], thus has attracted much research effort.

Stale Synchronous Parallel and its Variants. To mitigate the straggler effect while guaranteeing model convergence, Stale Synchronous Parallel (SSP) is usually adopted [20] in practice. SSP loosens the synchronization barrier, allowing 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 [8, 9, 13, 22].

With the relatively flexible synchronization model (i.e., SSP, compared to the original BSP), subsequent studies (including federated learning) [12, 21, 35] extensively explored the scheduling of synchronization among workers according to network conditions and training process. For example, [35] proposed allocating more bandwidth for workers that have lower bandwidth capacity. [12] makes scheduling plans by jointly solving an optimization problem and essentially prioritizes workers that have high bandwidth capacity or are close to the staleness threshold.

Gradient Compression. Gradient compression greatly reduces the communication traffic volume and is indeed essential for practical distributed training over wireless networks. Although there exist lossy (information is lost during compression and can cannot be recovered) gradient compression methods [27] that achieve up to 0.1% compression rate (i.e., size after compression divided by original size), they cannot provide convergence guarantee [27]. Thus in this paper, we only consider lossless (no information is lost during compression through e.g., error compensation [36]) compression methods which have a typical compression rate of around 3% [36].

Unfortunately, even with gradient compression, the straggler effect cannot be removed on state-of-the-art IoT and IoRT devices. In our experiments, a Jetson Xavier NX [3] device out of a team of four devices computed gradients on a batch of data in 2.18 s and ideally needed to wait for 1.47 s to push and pull the compressed gradients upon the synchronization barrier, which is comparable to the computation time. During an indoors surveillance task, the straggler effect caused each device to on average stall for 2.23 s in each iteration, taking up 102.2% of the computation time (referred to as *stall cost*); almost half of the computation power is wasted.

3 OVERVIEW

Fig. 3 presents the workflow of RSP and compares it with BSP and SSP. BSP enforces strict zero-staleness and will stall (idle time in Figure 3) whenever the completion time to transmit gradients on any device is deferred due to unstable network bandwidth capacity. SSP loosens the staleness constraint by following a loosened staleness threshold. However, SSP with a moderate threshold is expecting the gradients of a whole model from stragglers to catch up in severely downgraded bandwidth capacity before causing other devices to stall, which is often not feasible in real-world IoT and IoRT networks as shown in Figure 3; increasing the threshold of SSP will in turn increase the number of iterations for the training model to converge, affecting convergence time.

To cope with the straggler effect, RoG breaks up the gradient synchronization granularity into rows and aligns the transmission time among stragglers and non-stragglers by scheduling stragglers to synchronize less rows than non-stragglers in real time in an iteration. In this way, although wireless bandwidth capacity of all these devices is fluctuating randomly and sharply, stragglers are more likely to spend less time transmitting the gradients than non-stragglers in this iteration and thus will catch up with non-stragglers. The design of RoG needs to tackle three questions: 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. We chose rows out of three possible granularity choices: elements VS rows VS layers, so as to balance 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 full index of each element, taking up data volume comparable to the model itself; on the other hand, although a model often consists of only tens to hundreds of layers and the indexing of layers costs 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 chose in RoG is a balanced choice between management overhead and flexibility of transmission. First, the total data volume of indexes 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 several to hundreds of elements. If we adapt the gradient transmission scheduling of rows from stragglers when their bandwidth capacity fluctuates, we only spend a negligible cost of transmitting gradients of a row in degraded bandwidth capacity, 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 as mentioned above, the completely asynchronous method without any staleness control of asynchronous stale parallel (ASP) is also infeasible as it has no convergence guarantee [**?**]. In view that SSP enforces a staleness threshold constraint to contain the difference of training models among different devices that have run different numbers of iterations, so as to guarantee convergence, we notice that the same convergence guarantee still holds if we use the similar way to contain the difference of rows among different devices (see Section 5.3).

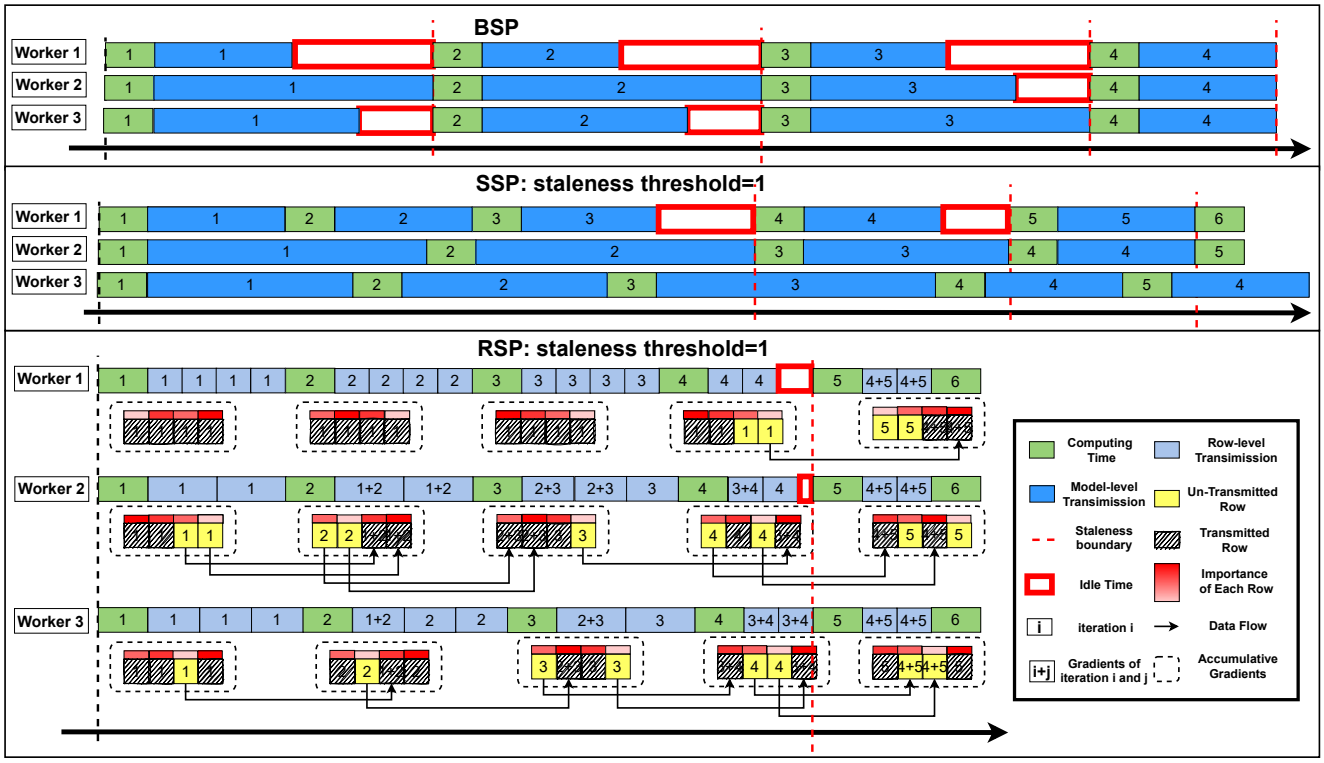


Figure 3: Workflow of RSP. We assume that bandwidth capacity on the three devices are identical in the three cases and the bandwidth capacity is interfered by distance, occlusion, etc.

Based on this finding, we design RSP which extends the staleness control of SSP by adopting two-level staleness control: for a same row on the shared model across different workers, the staled version should be within a preset staleness threshold; for different rows within a 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 Figure 3. RSP provably achieves the same level of convergence guarantee as SSP (see Section 5.3).

Adaptive Transmission Protocol (ATP). Instead of estimate-then-schedule methods that cannot adapt to real-time bandwidth capacity fluctuation, ROG leverages 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 to be exchanged every iteration to avoid triggering the staleness threshold in RSP, referred to as minimum transmission amount (MTA). For example, given a threshold 2, 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. Thus we say 50% is the MTA for threshold 2 in RSP.

Second, to minimize stall time and optimize convergence efficiency, the selection and transmission of rows should jointly consider the rows' possibility to cause stall (e.g., the staled version of this row), the quality of wireless connection (e.g., the bandwidth capacity) and the contribution of gradients of these row to model convergence (e.g., the absolute value of the gradients). Specifically, ATP on each device maintains the importance of each row based on its staled version and the average absolute value of its 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 number of rows and reports the transmission time of MTA (MTA time) on the straggler to other devices; other non-stragglers will then keep 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 about extra transmission overhead. To enable non-stragglers to keep transmitting rows for MTA

time, a straight-forward approach would be to insert bubbles between the transmission of two rows to check whether MTA time is reached; if so, quit transmitting the succeeding rows. This method is fine if each transmission takes comparatively long (e.g., 1.43 s transmitting the whole model), but would severely under-utilize the bandwidth capacity and empirically multiply the total transmission time because we are merely transmitting a row (sized only several to hundreds of elements) in each transmission. Instead, we co-design ATP with the underlying transmission protocol by enabling speculative transmission: the device continuously transmits the gradients of rows of the whole model in the priority determined by their importance without any bubble and discards the on-going transmission once the MTA time is reached (see Section 6). In this way, in the worst case the last row transmitted would be discarded if its transmission is not completed, which is also a negligible cost, and ATP manages to in real time adaptively schedule the transmission of each row according to MTA time without under-utilizing the bandwidth capacity in small granularity.

4 ARCHITECTURE OF ROG

Figure 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; RoGwill increase the staled version of rows not transmitted by one and set the staled version and gradients of those transmitted to zero, so that only gradients of rows not transmitted 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 is sent to each worker in Version Storage, so that RSP can determine whether the versions of this row satisfies 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 diverge 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.

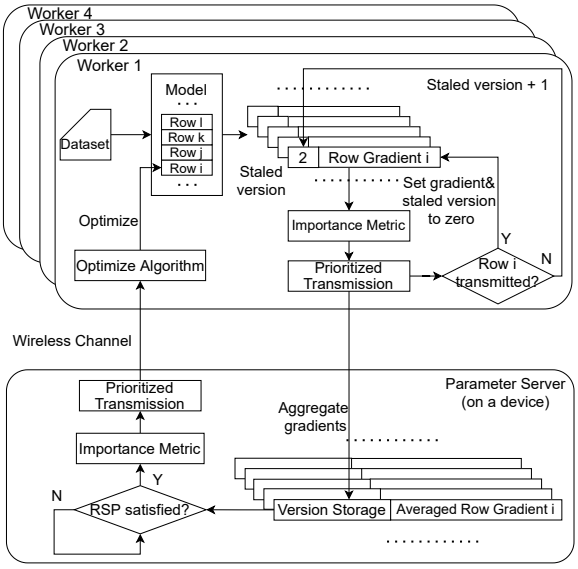


Figure 4: Architecture of RoG. On the worker side, RoG will set the gradients of the transmitted rows to zero so that the gradients will not accumulate; gradients of the not transmitted rows will accumulate as designed by Torch. Note that on the parameter server, similar to the worker side, RoG will 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.

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

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, 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 [10] and prove convergence of each row and further convergence ot the entire model. Since RoG either synchronizes or aggregates each row of parameter updates, no parameter update in a row is lost; thus the final convergence of the whole training model can provably have the same convergence guarantee as SSP.

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

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

Proof. The analysis mainly follows [46], 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 $R[X] \leq \sum_{t=1}^{T}\left\langle \tilde{g}_t, \tilde{x}_t - x^\star \right\rangle \leq M * MAX(\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \right\rangle)$.

Then, we are going to prove the convergence of each rows by finding the upper boundary of $\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \right\rangle$ for each row. Fortunately, we can learn lemma 2 from SSP [20].

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 $\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \right\rangle \leq 4F\prime L\prime\sqrt{2(S_i+1)PT}$. So, we can get $MAX(\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \right\rangle) = 4F\prime L\prime\sqrt{2(S_{max}+1)PT}$ and expand the regret $R[X] \leq M * MAX(\sum_{t=1}^{T}\left\langle \tilde{g}_t^i, \tilde{x}_t^i - x^{i,\star} \right\rangle) = M * 4F\prime L\prime\sqrt{2(S_{max}+1)PT}$. 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 [6] 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 [5]) 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 [4] 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 [38] 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 [36] (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 [20] and the framework proposed in [12] (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 [19, 29]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 [7] 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 [26] and we randomly allocates 50 shards for each robot without overlap. We choose ConvMLP [24] 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 [37], 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 [28] 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

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 ???

(a) (b) clean scenario

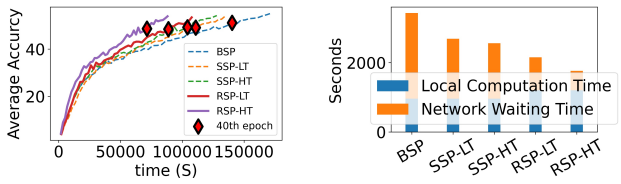
Figure 5: Accuracy - wall clock time; EACOL under different scenarios. X axis is time, y axis is training accuracy, and several lines are baselines and RoG

(a) (b) lab scenario

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

in real-wrold application paradigms on robots, and we will further study how RoG speeds up the model convergence.

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 Figure 7, we evaluated RSP and baselines in a clean WLAN without any obstacles existing based on our testbed.

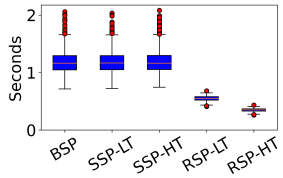


(a) Average accuracy over time. (b) Collective communication time.

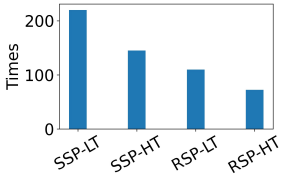
Figure 7: In a clean WLAN without obstacles.

Figure 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 Figure 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. Figure 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 Figure 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 Figure 7b without downgrading the training accuracy in Figure 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.



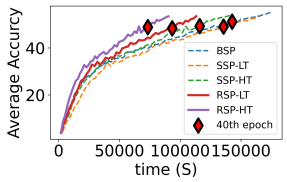
(a) Transmission time.



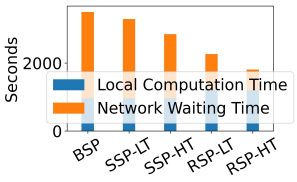
(b) The number of times triggering threshold.

Figure 8: Microanalysis of RSP and baselines.

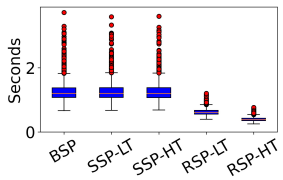
Then, we give some detailed information about microanalysis during ML training to elaborate how RSP reduced collective communication time. Network waiting time includes the transmission time for exchanging gradients or parameters and the blocked time due to the synchronization barrier (i.e., when the worker is not doing computation nor communication). Figure 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 Figure 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 Figure 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 Figure 8a.



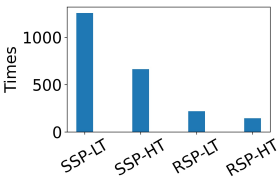
(a) Average accuracy over time.



(b) Collective communication time.

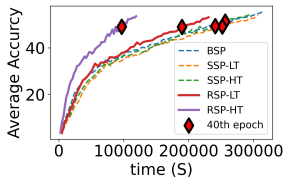


(c) Transmission time.

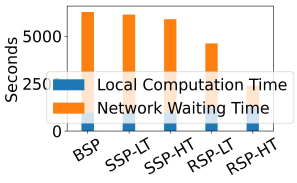


(d) The number of times triggering threshold.

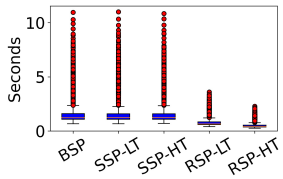
Figure 9: In a through-metal-sheet WLAN.



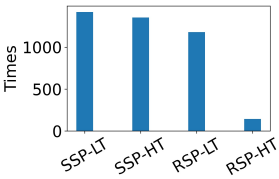
(a) Average accuracy over time.



(b) Collective communication time.



(c) Transmission time.



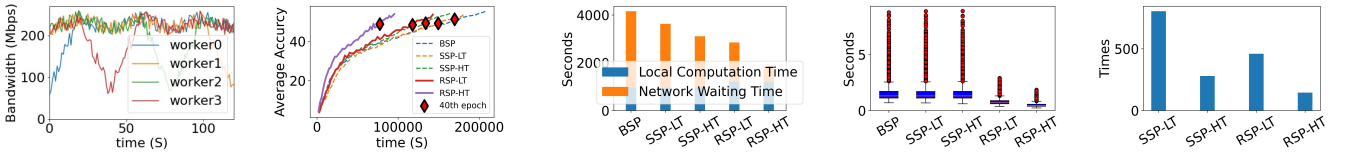
(d) The number of times triggering threshold.

Figure 10: In a through-wall WLAN.

7.4 Sensitivity Studies

ROG different threshold

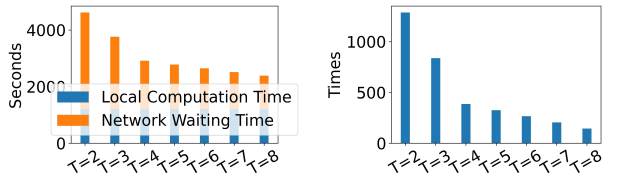
ROG different compression rate; too long



(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.

We further evaluated RSP with different staleness thresholds (i.e., from 2 to 8), whose environment configuration followed the setting as in Figure 10, because we found that in the case of Figure 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. Figure 12 shows the performance of RSP with different staleness thresholds in a through-wall WLAN.



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

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

In Figure 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 Figure 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 [20].

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.

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

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

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

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

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

[5] 2021. *Enterprise Open Source and Linux*. https://ubuntu.com/

[6] 2021. *PyTorch*. https://www.pytorch.org

[7] 2022. *tff.simulation.datasets.cifar100.load_data | TensorFlow Federated*. https://www.tensorflow.org/federated/api_docs/python/tff/simulation/datasets/cifar100/load_data

[8] 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

[9] 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.

[10] 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

[11] 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.

[12] 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.

[13] 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

[14] 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

[15] 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

[16] 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.

[17] Daniel B Faria. [n.d.]. Modeling Signal Attenuation in IEEE 802.11 Wireless LANs - Vol. ([n. d.]), 5.

[18] 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

[19] 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.

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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

[34] 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

[35] 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.

[36] 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

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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

[42] 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

[43] 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.

[44] 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.

[45] 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.

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

A RESEARCH METHODS

A.1 Part One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi malesuada, quam in pulvinar varius, metus nunc fermentum urna, id sollicitudin purus odio sit amet enim. Aliquam ullamcorper eu ipsum vel mollis. Curabitur quis dictum nisl. Phasellus vel semper risus, et lacinia dolor. Integer ultricies commodo sem nec semper.

A.2 Part Two

Etiam commodo feugiat nisl pulvinar pellentesque. Etiam auctor sodales ligula, non varius nibh pulvinar semper. Suspendisse nec lectus non ipsum convallis congue hendrerit vitae sapien. Donec at laoreet eros. Vivamus non purus placerat, scelerisque diam eu, cursus ante. Etiam aliquam tortor auctor efficitur mattis.

B ONLINE RESOURCES

Nam id fermentum dui. Suspendisse sagittis tortor a nulla mollis, in pulvinar ex pretium. Sed interdum orci quis metus euismod, et sagittis enim maximus. Vestibulum gravida massa ut felis suscipit congue. Quisque mattis elit a risus ultrices commodo venenatis eget dui. Etiam sagittis eleifend elementum.

Nam interdum magna at lectus dignissim, ac dignissim lorem rhoncus. Maecenas eu arcu ac neque placerat aliquam. Nunc pulvinar massa et mattis lacinia.