Energy Optimization for Federated Learning on Consumer Mobile Devices With Asynchronous SGD and Application Co-Execution

Cong Wang, *Member, IEEE*, and Hongyi Wu, *Fellow, IEEE*

***Abstract*—Federated learning relies on distributed training on mobile device. The previous research mainly focuses on addressing the heterogeneity from computation and data distributions. As battery life remains to be the performance bottleneck on mobile devices, energy consumption from the persistent training tasks poses great challenges. In this paper, we propose an online scheduler to optimize energy usage by leveraging application co-execution and asynchronous gradient updates. Motivated by a series of empirical studies, we find that placing the training process in the background while co-running a foreground application gives the system a large energy discount. Based on these findings, we first study an offline baseline assuming all the application occurrences are known in advance, and propose a dynamic programming solution. Then we propose an online scheduler using the Lyapunov framework to exploit the energy-staleness/slowdown trade-offs and prove the convergence at the rate of $1/\sqrt{K}$. We conduct extensive experiments on a mobile testbed with devices from different vendors. The results indicate 10-30% energy saving and much faster convergence compared to FedAvg and FedProx with 3-4% higher testing accuracy under the non-IID data setting. The design is also validated in terms of resource utilization, memory bandwidth and Frame-Per-Second rates.**

***Index Terms*—Asynchronous federated learning, energy-efficiency, on-device training, power-aware online optimization.**

I. Introduction

THESE years, we are witnessing our planet warming up at an unprecedented rate, causing irreversible change to the climate [1]. Unfortunately, the recent advance in AI plays an increasing role in this tragedy [2]. For example, training a large transformer model would generate the same amount of carbon emissions as five fuel vehicles in their entire lifetime [3]. The study in [4] shows that 40% of the energy comes from centralized cooling while training is performed in data centers. On the other hand, *Federated Learning* (FL) rises as a new paradigm to preserve user privacy by conducting training on distributed mobile/edge devices [6], [7], [9], [10]. It is shown that federated learning potentially offers better energy efficiency because centralized cooling is no longer required [5].

Although FL is promising to reduce the overall carbon footprint, by migrating high-intensity training tasks to the mobile CPUs on consumer devices, the complex energy implication and useability impact are far from clear [12], [13]. In this paper, we seek energy minimization of FL tasks while making the device still usable. With the prevalent dominance of ARM's architecture in the mobile market [59], we focus on the big.LITTLE CPU cores [29] that are designed to achieve energy-efficient multi-tasking. Big.LITTLE uses the big cores of high throughput for foreground applications and the low-power little cores for system and background processes. However, the UI framework on mobile devices is still event-driven, user-centric and the hardware/software architecture is optimized to respond to the foreground activities with less parallelism, whereas training acceleration requires thread-level parallelism. To avoid interfering with user's foreground activities, we designate the training thread as a background service [34], which can be activated once a set of conditions such as networking, battery energy are met. Such user-driven approach raises new questions about performance and energy when both resource-intensive foreground and background processes co-exist in the system.

Through some empirical studies, we found that *when and how to dispatch the training tasks to what cores have contrastive energy footprint on the mobile devices*. Surprisingly, once the highly paralleled training threads are running in the background (pinned to the little cores), simultaneous execution of a foreground application gives the entire system a deep energy discount (about 30-50%) compared to running the foreground application and training separately, with little performance impact measured by Frame per Second (FPS) and memory access. Thus, it is preferred to co-execute application and training for energy saving purposes.

Such energy-saving incentives pose new challenges to the existing FL pipelines since the mainstream frameworks are built on Synchronous Stochastic Gradient Descent (Sync-SGD) [6], [7], [10]. All participants proceed in lock-step and their parameters are averaged at the parameter server which is subject to the computational and statistical heterogeneity from a dynamic mobile environment [7], [10]. Worst-case stragglers (slowest

Manuscript received 17 April 2023; revised 20 November 2023; accepted 6 February 2024. Date of publication 20 March 2024; date of current version 3 October 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFE0112600, in part by the National Natural Science Foundation of China under Grant 62394341, Grant 62003304, and Grant 62293511, in part by the Key Research and Development Program of Zhejiang Province under Grant 2021C03037, and in part by Enflame Tech Company Ltd. Recommended for acceptance by D. Koutsonikolas. *(Corresponding author: Cong Wang.)*

Cong Wang is with the College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: cwang85@zju.edu.cn).

Hongyi Wu is with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: mhwu@arizona.edu).

Digital Object Identifier 10.1109/TMC.2024.3379236

workers) could be orders of magnitude slower than the average execution per epoch, especially under severe thermal throttling [12], where the computing power is underutilized. *Asynchronous SGD* (ASync-SGD) is a natural solution to tackle computational heterogeneity [14], [20]. It allows fast participants to proceed in lock-free steps while the global parameters are exchanged and kept with the most updated ones. Without such barrier from the stragglers, the system enjoys higher throughput over fixed wall-clock period. However, its potential is yet to be fully explored for federated learning on mobile devices.

In contrast to a large body of works, in this paper, we combine system-level opportunities with machine learning algorithms to achieve fine-grained energy optimization for federated mobile systems. Such integration faces several cross-level challenges: 1) the success of asynchronous learning relies on well-managed *staleness* in the system, especially for non-IID data, that the stale updates from the stragglers should not diverge too much from the current directions [15], [20], i.e., the staleness is bounded with low variance and properly penalized to improve model accuracy and convergence [48], [49]. Hence, the first challenge comes from the statistical instability while waiting for energy-saving opportunities. 2) Since the patterns and future occurrences of applications are non-deterministic, the system needs to make real-time decisions based on the known priori. 3) The multi-faceted interplay among control decision, training throughput, energy and performance slowdown requires a thorough optimization formulation to account for all these factors. In addition to power saving, we investigate how the system-level control decision would propagate upwards to affect model convergence, wall-clock training time and the foreground applications.

To tackle these challenges, we start with an offline scheduling problem assuming the access to all future occurrences of the applications, which serves as the optimal upper bound. We adopt a recently proposed metric called *gradient gap* to measure the difference between model parameters in their norm magnitude [48], [49], and formulate offline optimization into a *Two Dimensional Knapsack Problem* [45] with a pseudo polynomial-time dynamic programming solution. To enable real-time decision making, we further propose an online optimization algorithm based on the Lyapunov framework [46], which balances the trade-off between energy, staleness and performance slowdown. For control knob V, it is proved to achieve the $[\mathcal{O}(1/V), \mathcal{O}(V)]$ energy-staleness/slowdown trade-off, which only requires the current information of system dynamics and queue backlogs. The convergence of the online scheme is also proved. The contribution of this work is summarized below.

1) Motivated by a series of key findings in the experiments, we leverage ASync-SGD for energy optimization of federated training tasks. To the best of our knowledge, this is one of the few works that integrate high-level machine learning algorithms with low-level system dynamics on consumer mobile devices.
2) We formulate both offline, online optimization problems and design an efficient online scheduler while ensuring staleness and performance slowdown in the long term. We also prove the online scheme converges at the $1/\sqrt{K}$ rate under bounded latency.
3) We conduct extensive experiments on a mobile testbed with 6 different devices using the MNIST and CIFAR10 datasets [51], [52]. The results demonstrate up to 30% energy saving and faster convergence compared with FedAvg [6], FedProx [7] in both IID and non-IID data distributions and only 10% away from the ideal offline solution. We also evaluate the adaptation under daily usage and collect system traces from the hardware counters to validate our design.

II. Background

A. Energy Optimization

Battery optimization on consumer mobile device develops solution across both hardware and software stacks, e.g., ARM's asymmetric CPU architecture [29], dynamic voltage and frequency scaling of the processing cores [30], resolving "energy bugs" from unexpected energy consumption [31] and App-level energy monitoring in Android [32]. Most on-device learning research focuses on prominent challenges of minimizing inference latency and memory footprint [33].

As FL requires on-device training [13], the pressure of these persistent workloads on battery energy and usability is not quite clear. A viable way to avoid interrupting normal usage is to dispatch the long-running training workload as a background service [34], so the big cores can respond to the foreground Apps promptly. From the performance perspective, the separation of workloads across the big and little clusters reduces the coherence traffic across the heterogeneous cores [35]; from the energy perspective, since a running foreground App has already activated shared resources on the big cores, co-execution of training on the little cores could take advantages of such energy disproportionality. In [38], it is shown that optimal energy saving of task bundling is realized when the overall power state is not elevated. In fact, the idea of bundling different tasks dates back to piggyback sensing with Apps such as web browsing and phone calls on symmetric core systems [39] and coalescing network packets to reduce tail energy on the wireless interface [36], [37]. However, these early works cannot be readily applied to federated learning. The closest works to ours are [36], [37] that develop online schedules based on the Lyapunov framework for packets coalescing. This paper takes a step forward to consider multi-faceted trade-offs between energy and staleness/performance slowdown, and fills the gap between federated learning and mobile systems to achieve higher energy efficiency.

B. Federated Learning

The principal FL schemes are built on *synchronous SGD* [6], in which the clients proceed with a barrier until everyone finishes or the slower updates are discarded. As pointed out in [7], [9], [10], such coordinate-wise synchronization is subject to heterogeneity in a mobile environment due to diverse hardware configuration, network bandwidth, user behavior and data distributions. A plethora of works focus on addressing the objective inconsistency between the local and global models. FedProx introduces a proximal term to augment local gradient directions and reduces solution bias [7]. An objective re-formulation is proposed in [8] that confines the local models around the mean values with a quadratic penalty term. Similarly, variance reduction method is used to correct diverging gradient updates with fewer communication rounds [9]. Another line of works optimize the local training epoches. FedNova develops a new aggregation rule to allow local variations such as different number of epoches and optimizers [10]. A control algorithm is proposed in [11] to find the optimal aggregation frequency under the resource budget. These studies improve the computational and statistical efficiency under the Sync-SGD framework.

On the other hand, *asynchronous SGD* is a natural solution to tackle computational heterogeneity. The original implementation can be traced back to *HOGWILD!* [14] in multicore systems, where multiple threads are allowed access to shared memory and updating the model with a fast convergence rate at $1/k$. Although asynchronous updates improve overall throughput by eliminating synchronization overheads, the slower clients may work on a staled copy of the model and their updates would have negative impact on the overall convergence. Considerable efforts have been devoted to understanding and mitigating staleness [15], [16], [17], [18], [20], [48], [49]. Taylor Expansion and Hessian approximation are utilized to compensate the delay from stale gradients [15]. A regularized term is introduced to reduce the variance due to staleness [16]. Different forms of penalties have been applied to staled gradients to facilitate model convergence [18], [48], [49]. From the momentum perspective, stale gradients can be considered as an implicit momentum that dampens oscillation by adding a portion of the previous (stale) updates to the current update [17]. As shown in [20], optimizers have different degrees of robustness to staleness. Some recent works attribute the convergence speedup to such implicit momentum and a large pool of clients [24], but the tension between potential divergence and training acceleration is not fully understood at this stage. Unfortunately, most of the research in Sync-SGD [6], [7], [8], [9], [10], [11] and ASync-SGD [14], [15], [16], [20], [24] lie in the confined areas of machine learning and optimization theories, but the interaction to the underlying system is not fully explored, particularly for achieving energy-efficient computation on consumer mobile device. Different from these works on the algorithm level, we combine ASync-SGD with system-level opportunities to reduce energy footprint in federated mobile systems.

Different from the earlier version [25], we formulate execution time into the optimization to account for the trade-offs between energy consumption and gradient staleness/performance slowdown while making online decisions. While [25] only contains empirical evaluations, we prove convergence under a bounded latency by connecting with the Lyapunov-based online schedules. On the system side, this work provides new findings to unveil the relation between memory access and performance slowdown of co-execution contentions. By targeting resource-constrained mobile architectures, this work also adds to the existing efforts of scheduling conflict in multi-tenant CPU clusters [26].

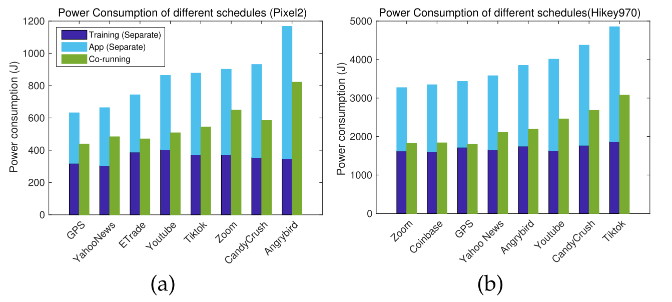


Fig. 1. Power consumption of different schedules (a) Pixel2 (b) Hikey970 Dev. Board. The green bars (co-running) represent about 35-50% power saving compared to the sum of blue bars (separate).

III. Motivation

A. Preliminary Experiment

We motivate the design by conducting preliminary power measurements on the HiKey 970 Development Board [40] and Pixel2 smartphone (see Section VII for implementation details). We compare the power consumption of two approaches: 1) Schedule training as a service in the background, separately from the application (denoted by *separate*). 2) Schedule training to co-run with the foreground application (*co-execution*). Since applications have diverse resource demands and user interaction patterns, we choose some popular applications from Google Play, and summarize important observations in the following.

Observation 1. (Power Saving): Compared to separate scheduling, co-execution can potentially offer 35-50% power saving (Fig. 1).

Explanation: The power saving originates from temporal sharing of the hardware resources. Though the big/little cores have their own L1/L2 cache, the memory bandwidth is shared in the SoC. Once the highly-paralleled training process is started on the little cores, the memory resource is activated and kept at certain power state. Having a foreground application executed on the big cores does not elevate the total power state too much, thus resulting in a substantial energy saving compared to separate execution. This finding is cross-validated with the homogeneous cores of Nexus6 as resource contention on the same CPU cluster degrades the power saving benefits (see Section VIII). Though shared resources offer better energy efficiency, they also lead to potential slowdown as illustrated below.

Observation 2.1 (Performance Slowdown - Background Training) Performance of background training is proportional to the accessing rates to the main memory (Fig. 2(a)), which serves as a hardware indicator of the interference between background/foreground applications.

Explanation: 2.1 is similar to the observations in [27] on desktop/servers, where the performance of memory-bound applications are found to be proportional to how fast the memory requests are served. This is due to memory access priority assigned to the applications by the memory controller, and the foreground applications are typically given higher priority. E.g., Tiktok tends to have higher resource utilization while the user is swiping for the next video and we observe less memory access

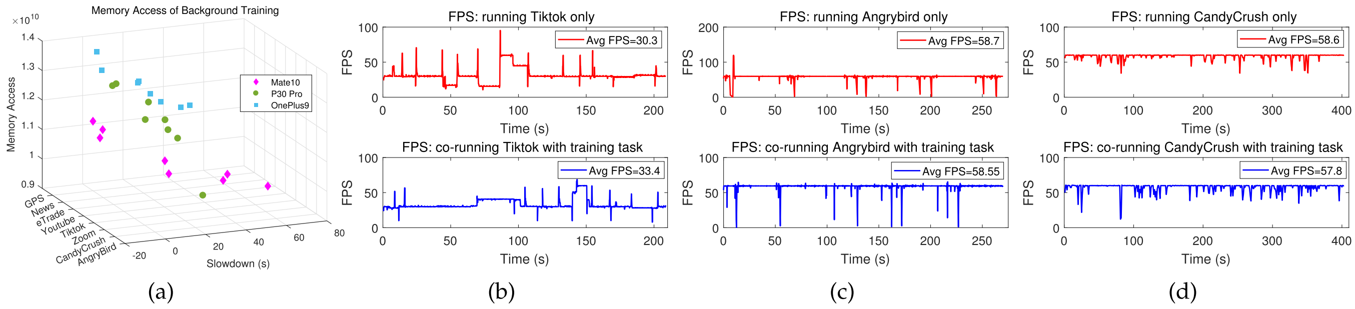


Fig. 2. Performance impact from co-executing the background training and foreground applications: (a) Performance slowdown versus memory access from the background training co-running with different applications; (b)-(d) FPS of Tiktok, Angrybird and Candycrush co-executed with background training.

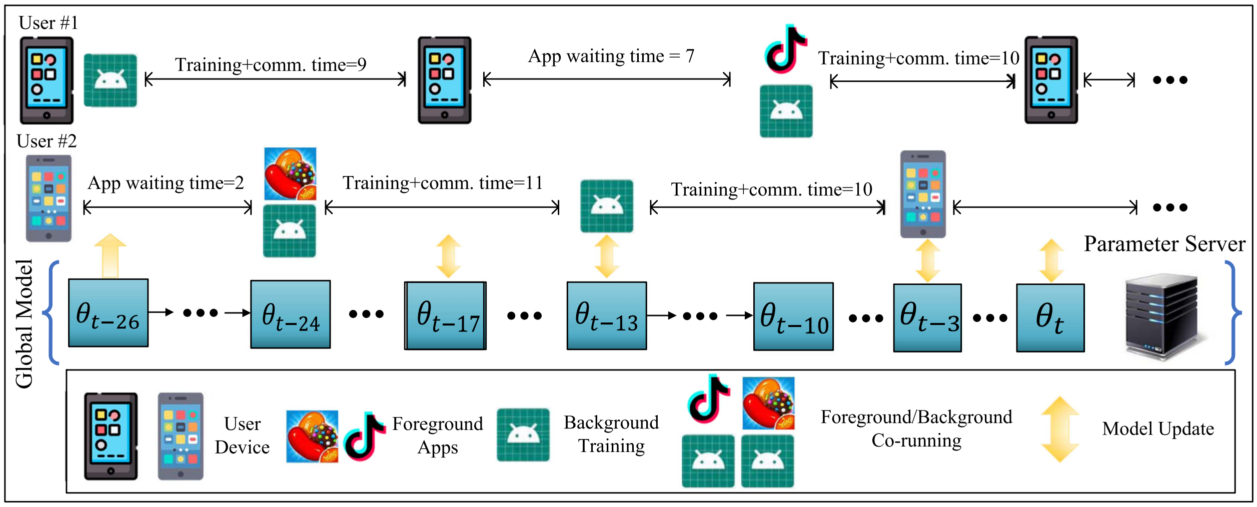


Fig. 3. Example of asynchronous federated learning. The scheduler decides whether to launch training immediately or co-execute with the foreground applications.

from the background training process. Co-running with these applications causes temporary contention on the memory bus and reduces the memory service rate of the background training process.

Observation 2.2 (Performance - Foreground Apps): Co-execution has negligible slowdown measured by FPS rendering to the foreground applications (Fig. 2(b–d)).

Explanation: With the co-execution of training and gaming, we notice a bit more frequent stuck while gaming, especially at loading different background, that the FPS temporarily drops near zero. However, as shown the figure, most of the FPS values stay around 60 frames/s to deliver real-time user experience. More experimental results are available in Fig. 10.

IV. System Model

This section describes the system model and definitions. Training consists of the following procedures: 1) A device pulls the latest model from the server when it is ready; 2) then it is scheduled for training either immediately or waiting until an application co-running opportunity; 3) the model is pushed to the server after the local training finishes; 4) the global model is updated and ready to be downloaded by other users. Fig. 3 shows an example with two users in the time interval $[t-26, t]$. At $t-26$, User 1 performs training immediately for 9 time units, and User 2 executes training with CandyCrush after waiting for 2 time units. The global model is updated at $t-17$ and $t-13$ when Users 1 and 2 finish training respectively. Next, User 1 waits Tiktok for 7 time units and User 2 performs training immediately. Likewise, the global model is updated at $t-3$ and t.

For simplicity, we use the average power consumption: co-execution takes P_i^c power at the i-th device; separate execution of application and training consume P_i^a and P_i^b respectively; otherwise, P_i^d power is consumed while idling. According to device profiling, we typically have $P_i^d < P_i^b < P_i^a < P_i^c$ and $P_i^c < P_i^b + P_i^a$. We assume the application would at least last for the duration of the training task in the background. While co-running, background training would have a potential slowdown of d_i and the foreground slowdown is measured by FPS (evaluated in Sec.VIII-C2). To quantify gradient staleness, we formally define *delay* and *gradient gap* below.

Definition 1. (Lag): Lag L_τ is defined as the number of updates that have been made to the global model within delay τ, e.g., a device downloads the model at $t-\tau$ and it takes τ (waiting, training and communication) until the next update.

TABLE I
LIST OF IMPORTANT NOTATIONS

Notation	Definition
P_i^a, P_i^b	Average power consumption of separate executions of foreground *application* and background *training*
P_i^c, P_i^d	Average power consumption of training/application *co-execution*, and device *idling*.
τ_t, g_t, θ_t	*Delay*, *gradient gap* and *model parameters* at time t.
$\alpha(t), s(t)$	Control decision {"schedule", "idle"}, application status {"app", "no app"}.

In the example of Fig. 3, there is one update from User 1 at $t-17$ between $[t-26, t-13]$, so the lag of is $L_{\tau_1} = 1$. Lag only gives a coarse estimation of staleness, since the gradient difference between two updates is not captured. We further leverage the concept of gradient gap introduced by [48], [49].

Definition 2. (Gradient Gap): According to the L-Lipschitz continuity, the gradient gap g_k is bounded by the norm difference between the model parameters at $\theta_{t-\tau}$ and θ_t [48], [49],

$$g_\tau = \left\|\nabla f(\theta_{t-\tau}) - \nabla f(\theta_t)\right\|^2 \leq L \left\|\theta_{t-\tau} - \theta_t\right\|^2. \tag{1}$$

Table I summarizes the important notations used in this paper.

V. OFFLINE SCHEDULING BASELINE

We first study an offline problem by assuming all the application occurrences are known. This serves as a baseline for the online algorithm proposed next. For the offline problem, our goal is to maximize power saving of all n users in a time slot with bounded staleness and performance slowdown. The power saving is denoted by $s_i = P_i^a + P_i^b - P_i^c$, if the decision is co-execution (decision variable $x_i = 1$); otherwise, $s_i = 0$ ($x_i = 0$). The total performance slowdown is d_i, when $x_i = 1$; otherwise, it is 0.

Offline Problem Formulation:

$$\mathbf{P1}: \quad \max \sum_{i=1}^{n} s_i x_i \tag{2}$$

s.t.

$$\frac{1}{n}\sum_{i=1}^{n} g_i(\tau_i; x_i) \leq G_m, \tag{3}$$

$$\frac{1}{n}\sum_{i=1}^{n} d_i x_i \leq D_m. \tag{4}$$

Constraints (3) and (4) impose that the average gradient gap and performance slowdown are bounded by the maximum gradient difference G_m and slowdown D_m respectively. The problem is a natural extension to the *Two-Dimensional Knapsack Problem* [45], which maximizes the total value of items under an area capacity (length and width) of packing rectangle blocks in a container. A unique challenge to our problem is the dependence between the gradient gap and scheduling decisions noted by $g_i(\tau_i; x_i)$ in Constraint (3), i.e., the gradient gap of a user depends on the scheduling decisions from other users, and this unfortunately, forms a difficult looping situation. To resolve this problem, we derive an upper bound of the lag given the information of application arrivals and training/communication

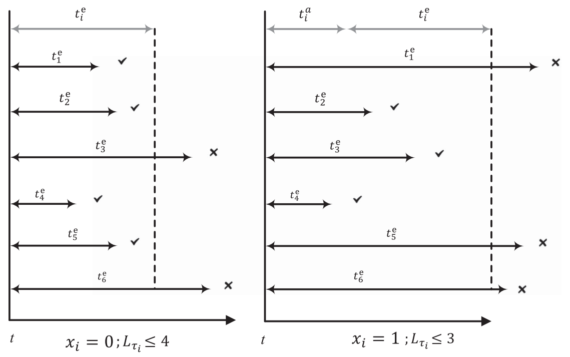


Fig. 4. Illustration of Property 1 for different scheduling decisions of user i.

time. Once this upper bound is satisfied, we have a feasible, near-optimal solution. The derivation is illustrated in the following property.

Property 1: At time t, given the waiting time for application t_i^a, execution time t_i^e (training and communication) of all the users $\{1, 2, \ldots, n\}$, L_{τ_i} is bounded by,

$$L_{\tau_i} \leq \begin{cases} \sum\limits_{j\in\{1,\ldots,n-1\}} \mathbb{1}_{\{t+t_j^e \leq t+t_i^e\}}, & x_i = 0, \\ \sum\limits_{j\in\{1,\ldots,n-1\}} \mathbb{1}_{\{t+t_j^e \leq t+t_i^a+t_i^e\}}, & x_i = 1. \end{cases} \tag{5}$$

$\mathbb{1}_{\{\cdot\}}$ is 1 if the condition is satisfied. Depending on the scheduling decision x_i, the lag is bounded by the maximum number from the rest users $j \in \{1, \ldots, n-1\}$ who finish before i, i.e., the worst case is when they are all scheduled immediately at t; otherwise, they might wait for the applications first and update to the global model after i finishes, which does not count towards the lag value. Fig. 4 shows an example for user i with execution time t_i^e and $t_i^a + t_i^e$: when $x_i = 0$, there are 4 users {1,2,4,5} who finishes before i so $L_{\tau_i} \leq 4$; when $x_i = 1$, there are 3 users {2,3,4} who finishes before i so $L_{\tau_i} \leq 3$.

The gradient gap can be calculated by (1), where the number of discrete model updates between $[t-\tau, t]$ is bounded by (5). Hence, we can approximate $g_i(\tau_i; x_i)$ in Constraint (3) and solve the 2D Knapsack problem by utilizing its recursive sub-structure:

$$S_i(x,y) = \begin{cases} S_{i-1}(x,y), \ \ 0 < x \leq g(\tau_i; x_i), 0 < y \leq d_i, \\ \max\left\{S_{i-1}(x,y), S_{i-1}(x - g(\tau_i; x_i), y - d_i) + s_i\right\}, \\ g(\tau_i; x_i) \leq x \leq nG_m, d_i \leq y \leq nD_m. \end{cases} \tag{6}$$

$S_i(x, y)$ represents the maximal power saving given the two states (x, y), where x and y represent gradient staleness and performance slowdown respectively. The maximal value $S_i(x, y)$ is calculated by comparing $S_{i-1}(x, y)$ with the previous values of $S_{i-1}(x - g(\tau_i; x_i), y - d_i)$ plus the power saving s_i. The solution can be tabulated by a 2D matrix with the rows from 0 to nG_m and the columns from 0 to nD_m, thereby having a total $\mathcal{O}(n^3 G_m D_m)$ computation complexity for n users.

VI. Online Scheduling

Offline scheduling assumes the future application arrival as a priori. In this section, we propose an online scheduling based on the Lyapunov framework that only relies on the current observation. The Lyapunov framework defines a task queue for the entire system.

Definition 3. (Queue Dynamics): The task queue represents the number of users waiting to be scheduled. Their arrival is considered as a random process. The queue backlog will increase by $A(t)$ if a number of $A(t)$ users are ready to start training at t. If $b(t)$ users finish their training, the backlog is reduced by $b(t)$.

The system makes a control decision $\alpha(t) = \{\text{‘}schedule\text{’}, \text{‘}idle\text{’}\}$ at time t. Recall that the power consumption $P_i(t)$ of the i-th device depends on how training is scheduled and the current application status $s(t) = \{\text{‘}app\text{’}, \text{‘}no\ app\text{’}\}$, i.e., $P_i(t) = P_i(\alpha(t), s(t))$:

$$P_i(t) = \begin{cases} P_i^a, & \alpha(t) = \text{‘}idle\text{’}, s(t) = \text{‘}app\text{’} \\ P_i^b, & \alpha(t) = \text{‘}schedule\text{’}, s(t) = \text{‘}no\ app\text{’} \\ P_i^c, & \alpha(t) = \text{‘}schedule\text{’}, s(t) = \text{‘}app\text{’} \\ P_i^d, & \alpha(t) = \text{‘}idle\text{’}, s(t) = \text{‘}no\ app\text{’}. \end{cases} \tag{7}$$

The corresponding service rate is,

$$b_i(t) = \begin{cases} 1, & \alpha(t) = \text{‘}schedule\text{’} \\ 0. & \alpha(t) = \text{‘}idle\text{’} \end{cases} \tag{8}$$

The total service rate is $b(t) = \sum_{i=1}^n b_i(t)$.

Online Problem Formulation: Our goal is to minimize the time-averaged energy consumption of training tasks in the system,

$$\mathbf{P2}: \quad \limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} \mathbb{E}\big[P_i(t)\big] \tag{9}$$

s.t.

$$\limsup_{T\to\infty} \frac{1}{nT} \sum_{t=1}^{T} \sum_{i=1}^{n} g_{t-\tau_i,t} \le G_m \tag{10}$$

$$\limsup_{T\to\infty} \frac{1}{nT} \sum_{t=1}^{T} \sum_{i=1}^{n} d_i(t) \le D_m \tag{11}$$

Constraints (10), (11) bound the gradient gaps and performance slowdown from all the users in a time-averaged sense. **P2** can be transformed into the queue stability problem under the Lyapunov optimization framework. Given the arrival rate $A(t)$ and service rate $b(t)$, the queueing dynamics is,

$$Q(t+1) = \max\left(Q(t) - b(t), 0\right) + A(t) \tag{12}$$

with the initial $Q(0) = 0$. We define *virtual queues* $H(t), J(t)$ for the two constraints,

$$H(t+1) = \max\left(H(t) + G(t) - nG_m, 0\right) \tag{13}$$

$$J(t+1) = \max\left(J(t) + D(t) - nD_m, 0\right) \tag{14}$$

where $G(t) = \sum_{i=1}^n g_{t-\tau_i,t}$, $D(t) = \sum_{i=1}^n d_i(t)$, and the initial $H(0), J(0) = 0$. We concatenate the actual and virtual queues into $\Theta(t) = [Q(t), H(t), J(t)]$. The Lyapunov function $L(\Theta(t))$ for the backlogged training tasks is,

$$L(\Theta(t)) \triangleq \frac{1}{2}\big(Q(t)^2 + H(t)^2 + J(t)^2\big), \tag{15}$$

and the Lyapunov drift function $\Delta(\Theta(t))$ is:

$$\Delta(\Theta(t)) \triangleq \mathbb{E}\big[L(\Theta(t+1)) - L(\Theta(t)) | \Theta(t)\big] \tag{16}$$

It represents the change in the Lyapunov function in t as the scalar volume of queue congestion and transforms **P2** into a new problem of minimizing the drift-plus-penalty:

$$\mathbf{P3}: \quad \min \Delta(\Theta(t)) + V\mathbb{E}\big[P(t)|\Theta(t)\big] \tag{17}$$

V is the control parameter to balance energy and staleness/performance slowdown. According to the Lyapunov framework, the key is to obtain the upper bound of the drift as described in the following Lemma.

Lemma 1: Given the queue backlogs $\Theta(t)$, arrival rates $A(t)$, service rate $b(t)$, gradient gaps and performance slowdown, the upper bound for the drift-plus penalty term is defined as,

$$\begin{aligned} &\Delta(\Theta(t)) + V\mathbb{E}\big[P(t)|\Theta(t)\big] \le B + V\mathbb{E}\big[P(t)|\Theta(t)\big] \\ &+ Q(t)\mathbb{E}\big[A(t) - b(t)|\Theta(t)\big] + H(t)\mathbb{E}\big[G(t) - nG_m|\Theta(t)\big] \\ &+ J(t)\mathbb{E}\big[D(t) - nD_m|\Theta(t)\big] \end{aligned} \tag{18}$$

where the constant $B = \frac{1}{2}(n^2 + G_{\max}^2 + D_{\max}^2 + n^2 G_m^2 + n^2 D_m^2)$ is a positive constant.

Proof: Applying (15) to (16) we have,

$$\begin{aligned} \Theta(t) &= \mathbb{E}\big[L(\Theta(t+1)) - L(\Theta(t))|\Theta(t)\big] \\ &= \frac{1}{2}\mathbb{E}\big[\Theta(t+1)^2 - \Theta(t)^2\big] \\ &= \frac{1}{2}\mathbb{E}\big[Q(t+1)^2 + H(t+1)^2 + J(t+1)^2 - Q(t)^2 - H(t)^2 \\ &\quad - J(t)^2\big] \end{aligned} \tag{19}$$

Since $\max^2\{x, 0\} \le x^2$, from (12) and (13) we have,

$$\begin{aligned} &Q^2(t+1) + H^2(t+1) + J(t+1)^2 \le Q^2(t) + \big(A(t) - b(t)\big)^2 \\ &+ 2Q(t)\big(A(t) - b(t)\big) + H^2(t) + G(t)^2 + 2H(t)G(t) \\ &+ n^2 G_m^2 + J^2(t) + D^2(t) + 2J(t)D(t) + n^2 D_m^2 \end{aligned} \tag{20}$$

Combine (19) and (20),

$$\begin{aligned} \Theta(t) \le\ & B + \mathbb{E}\big[Q(t)(A(t) - b(t))|Q(t)\big] + \mathbb{E}\big[H(t)G(t)|H(t)\big] \\ &+ \mathbb{E}\big[J(t)D(t)|J(t)\big] \end{aligned} \tag{21}$$

where the constant $B = \frac{1}{2}(A_{\max}^2 + G_{\max}^2 + D_{\max}^2 + n^2 G_m^2 + n^2 D_m^2)$. $A_{\max}$ is the maximum of $A(t) - b(t)$, which equals to the total number of users n. $G_{\max}$ and $D_{\max}$ are the maximum gradient gap and performance slowdown in the system. Thus, (21) completes the proof of Lemma 1. □

Our algorithm observes the queue backlogs of $Q(t), H(t), J(t)$ and the current application usage $s(t)$ to make a decision of $\alpha(t) \triangleq \{\text{'}schedule\text{'}, \text{'}idle\text{'}\}$ that minimizes the right-hand side (R.H.S.) of the drift bound (18), which is equivalent to the following objective.

$$\min\left(V\sum_{i=1}^{n}P_i(t) - Q(t)\sum_{i=1}^{n}b_i(t) + H(t)\sum_{i=1}^{n}g_{t-\tau_i,t_i}\right.$$
$$\left.+J(t)\sum_{i=1}^{n}d_i(t)\right) \tag{22}$$

This formulation makes online decisions based on the current observations and does not need a-priori knowledge of the arrival rates. With the information of application usage, scheduling decisions can be made at the parameter server in a centralized manner within $\mathcal{O}(n)$. However, centralization carries certain privacy risks since it requires private information of application usage, which can be used to re-identify specific users [47].

A. Distributed Implementation

To preserve privacy, we transform the scheme into a distributed implementation as described below. The server maintains a checkpoint for each user i's most recent update $\theta_{t-\tau_i}$ and the queue backlogs. In each time slot, it computes the gradient gap based on all the previous updates $\theta_{t-\tau_i}$ and the current model θ_t, then transmits the values of gradient gaps and queue backlogs to the users. Each user calculates to minimize (22) based on application usage and the information received from the server. Hence, application status is retained on-device and the information shared are gradient gaps and queue backlogs, which reveal little private information. It also reduces the uplink communication overhead from the users. If the control decision is $\alpha(t) = \{\text{'}schedule\text{'}\}$, the user performs training and the global model is updated as,

$$\theta_t = \theta_{t_p} - \frac{\eta_t}{\gamma_t}\nabla F_{\xi_{t-\tau}}(\theta_{t-\tau}). \tag{23}$$

where θ_{t_p} is the model time-stamped from the previous model update at t_p. $\theta_{t-\tau}$ is the staled model. γ_t is a variable to penalize the staled updates, which can have various forms in the implementation. For example, set the penalty to the lag from delayed updates [18], cosine similarity of gradient directions [20] or gradient gaps [49]. The entire procedure on the server and user side are described in Algorithms 1 and 2.

Algorithm 1: Online Algorithm (Parameter Server).

```
1 Input: Control decision $\alpha_i(t)$, model checkpoints
    $\theta_{t-\tau_i}, \forall i$, learning rate $\eta_t$ and staleness penalty $\gamma_t$.
2 Output: Updated model $\theta_t$.
3 for $i = 1$ to $n$ do
4     Compute queue backlogs based on $\alpha_i(t)$.
5     Compute gradient gap $g_{t-\tau_i,t} = \|\theta_{t-\tau_i} - \theta_t\|^2$.
6     Transmit queue backlogs and gradient gap to the
        corresponding users.
7     Receive gradient updates $F_{\xi_{t-\tau_i}}(\theta_{t-\tau_i})$ from the
        user and update the global model:
8     $\theta_t \leftarrow \theta_{t_p} - \frac{\eta_t}{\gamma_t}\nabla F_{\xi_{t-\tau_i}}(\theta_{t-\tau_i})$.
```

Algorithm 2: Online Algorithm (User i).

```
1 Input: Receive queue backlogs $Q(t), H(t), J(t)$ and
    gradient gap $g(t-\tau_i,t)$ from server, control
    parameter $V$, and action space $\Omega$.
2 Output: Scheduling decisions.
3 $\alpha_i(t) \leftarrow \arg\min_{P_i,b_i,g_{t-\tau_i,t},d_i} VP_i(t) - Q(t)b_i(t) + H(t)g(t-\tau_i,t) + J(t)d_i(t)$.
4 Inform control decision $\alpha_i(t)$ to server.
5 if $\alpha_i(t) =$ '$schedule$' then
6     Update the local model for all $N$ batches:
        $\theta_t \leftarrow \theta_{t-\tau_i} - \eta_t\sum_{b=1}^{N}\nabla F_{\xi_{t-\tau_i,b}}(\theta_{t-\tau_i})$
7     Upload the gradients to the server.
```

B. Illustration of Control Decisions

To minimize (22), $H(t)\sum_{i=1}^{n}g_{t-\tau_i,t}$ and $J(t)\sum_{i=1}^{n}d_i(t)$ can be viewed as penalty terms when there are backlogs in the virtual queues. When there is no backlog at all $(Q(t), H(t), J(t) = 0)$, we only have the first term in (22) and the control decision is to always set the device to idle because idling has the minimum power consumption. This matches with the intuition to wait for better co-running opportunities if there is enough space for backlogged workloads.

Without Virtual Queue: There could be cases that there are queue backlogs in $Q(t)$, but for the virtual queue $H(t)$, the cumulative gradient gap and performance slowdown are within the bounds of nG_m, nD_m. Using (13) and (14), both $H(t), J(t) = 0$ so the only backlogs are from $Q(t)$,

$$\alpha_i(t) = \underset{\alpha_i(t)}{\text{argmin}}\begin{cases}(VP_i^c - Q(t), VP_i^a), & s(t) = \text{'}app\text{'}\\(VP_i^b - Q(t), VP_i^d), & s(t) = \text{'}no\ app\text{'}\end{cases} \tag{24}$$

The decision can be made by simply observing $Q(t)$: for $s(t) =$ 'app', the decision is to co-execute if $Q(t) \geq V(P_i^c - P_i^a)$; otherwise, the decision is idling. This can be interpreted as the control decision is to always back off until the queue exceeds the power difference between co-running and separate execution scaled by the control knob V. Similarly, for $s(t) =$ '$no\ app$', the decision is to execute background training when $Q(t) \geq V(P_i^b - P_i^d)$ or remain idle otherwise. Hence, without virtual queue backlogs, the decision is to prioritize energy conservation.

With Virtual Queue: When $H(t), J(t) > 0$, the penalty terms are activated,

$$\alpha_i(t) = \underset{\alpha_i(t)}{\text{argmin}}\begin{cases}\left(VP_i^c - Q(t) + J(t)d_i(t), VP_i^a + H(t)g_{t-\tau_i,t}\right),\\ s(t) = \text{'}app\text{'}\\ \left(VP_i^b - Q(t), VP_i^d + H(t)g_{t-\tau_i,t}\right),\\ s(t) = \text{'}no\ app\text{'}\end{cases} \tag{25}$$

If there is an application, the decision is to co-execute if

$$Q(t) \geq V(P_i^c - P_i^a) + J(t)d_i(t) - H(t)g_{t-\tau_i,t}; \tag{26}$$

otherwise, the decision is to run the application only. Similarly, if there is no application, the decision is to run training if,

$$Q(t) \geq V(P_i^b - P_i^d) - H(t) g_{t-\tau_i,t}; \tag{27}$$

otherwise, the decision is idling and waiting for better opportunities. The control decision is to observe the power difference between different actions scaled by the control knob V plus/minus the relaxation terms from the virtual queue of staleness and performance slowdown. In the above two conditions, if $Q(t) \to n$ and is full, but still less than R.H.S in order to schedule training, the negative term from the staleness queue $H(t)g_{t-\tau_i,t}$ grows to reduce the R.H.S. such that training would be scheduled if staleness rises.

For each user i, $\tau_{\max}$ is the maximum delay until the next schedule. From the conditions above, we have the following property.

Property 2: The maximum delay $\tau_{\max}$ is bounded if the control knob V is selected such that the R.H.S. of both (26) and (27) are less than the maximum queue length of $Q(t) = n$. Hence, training would be always scheduled and $\tau_{\max}$ is bounded.

C. Optimality of Control Decisions

The optimality of online scheduling is derived in Theorem 1.

Theorem 1: Let $L(\Theta(t))$ defined by (15) and $L(\Theta(0)) = 0$. P^* is the optimal power consumption. For constants $B, V \geq 0$, the queues of $\Theta(t)$ are mean rate stable and the time-averaged power consumption and queue backlogs are bounded by:

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\big[P(t)\big] \leq \frac{B}{V} + P^* \tag{28}$$

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\big[\Theta(t)\big] \leq \frac{B}{\epsilon} + \frac{V(P^* - P_{\min})}{\epsilon} \tag{29}$$

Proof: Given that the optimal decision $\alpha^*(t)$ can stabilize the queue,

$$\mathbb{E}\big[P(\alpha^*(t))\big] = P^*. \tag{30}$$

and the power consumption is lower-bounded by a finite value $P_{\min}$ (idling power of P_i^d)

$$\mathbb{E}[P(t)] \geq P_{\min} \tag{31}$$

Assume there exists $\epsilon > 0$ such that the difference of the service and arrival rates of all queues are larger than ϵ.

$$\begin{aligned}&\mathbb{E}\big[b(t) - A(t)|Q(t)\big], \mathbb{E}\big[nG_m - G(t)|H(t)\big],\\ &\mathbb{E}\big[nD_m - D(t)|J(t)\big] > \epsilon\end{aligned} \tag{32}$$

Plug the above equations into (18), we have

$$\begin{aligned}
&\Delta(\Theta(t)) + V\big[P(t)|\Theta(t)\big] \leq B + VP^* - \epsilon\big(\mathbb{E}\big[Q(t)\big]\\
&+ \mathbb{E}\big[H(t)\big] + \mathbb{E}\big[J(t)\big]\big)\\
&\sum_{t=0}^{T-1} \Delta(\Theta(t)) + \sum_{t=0}^{T-1} V\mathbb{E}\big[P(t)|\Theta(t)\big] \leq T(B + VP^*)\\
&- \epsilon \sum_{t=0}^{T-1} \big(\mathbb{E}\big[Q(t)\big] + \mathbb{E}\big[H(t)\big] + \mathbb{E}\big[J(t)\big]\big)\\
&\frac{\mathbb{E}\big[L(\Theta(T-1)) - L(\Theta(0))\big]}{TV} + \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\big[P(t)\big] \leq \frac{B}{V}\\
&+ P^* - \frac{\epsilon}{TV}\big(\mathbb{E}\big[Q(t)\big] + \mathbb{E}\big[H(t)\big] + \mathbb{E}\big[J(t)\big]\big)\\
&\limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\big[P(t)\big] \leq \frac{B}{V} + P^*
\end{aligned} \tag{33}$$

The second inequality takes the summation over $t \in \{0, \ldots, T-1\}$. The third inequality plugs in (16), and divides both sides by TV. The last inequality takes $T \to \infty$ and because $L(\Theta(0)) = 0$. Thus, (28) of Theorem 1 is proved. The time-averaged queue length can be derived by dividing both sides of the second inequality by ϵT and re-arranging the terms,

$$\begin{aligned}
&\frac{1}{T}\sum_{t=0}^{T-1}\big(\mathbb{E}\big[Q(t)\big] + \mathbb{E}\big[H(t)\big] + \mathbb{E}\big[J(t)\big]\big)\\
&\leq \frac{B + V(P^* - \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[P(t)\big])}{\epsilon}\\
&\quad - \frac{\mathbb{E}\big[L(\Theta(T-1))\big]}{\epsilon T} + \frac{\mathbb{E}\big[L(\Theta(0))\big]}{\epsilon T}\\
&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[\Theta(t)\big] \leq \frac{B}{\epsilon} + \frac{V(P^* - P_{\min})}{\epsilon} + \frac{\mathbb{E}\big[L(\Theta(0))\big]}{\epsilon T}\\
&\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[\Theta(t)\big] \leq \frac{B}{\epsilon} + \frac{V(P^* - P_{\min})}{\epsilon}
\end{aligned} \tag{34}$$

The left-hand side of the second step summarizes all the queues using $\Theta(t) = [Q(t), H(t), J(t)]$. The R.H.S. uses (31) and removes the negative term for the upper bound. The last step takes the limits of $T \to \infty$. Thus, (29) of Theorem 1 is proved. ☐

The performance bounds (28), (29) demonstrate $[\mathcal{O}(1/V), \mathcal{O}(V)]$ energy-staleness and slowdown trade-offs: by arbitrarily increasing V, we can make $\frac{B}{V} \to 0$ and the time-averaged power consumption close to the optimal value, whereas the staleness and slowdown grow linearly with V.

D. Convergence Analysis

Theorem 2: If Assumption 1 (see below), Property 2 and the following condition holds,

$$\frac{T_m L^2 \eta_k^2}{\gamma_k^2} + \frac{L\eta_k}{\gamma_k} \leq 1. \tag{35}$$

the iteration satisfies,

$$\begin{aligned}
&\min_{k\in\{1,\ldots,K+1\}} \mathbb{E}\Big[\|\nabla f(\theta_k)\|^2\Big]\\
&\leq \frac{f(\theta_1) - f(\theta_*) + \sum_{k=1}^{K}\Big(\frac{L\eta_k^2}{2\gamma_k^2} + \frac{T_m L^2 \eta_k^3}{2\gamma_k^3}\Big)\sigma^2}{\sum_{k=1}^{K}\frac{\eta_k}{2\gamma_k}}.
\end{aligned} \tag{36}$$

Proof: To facilitate the convergence proofs, we migrate the previous notation in continuous time t into discrete update steps k towards the global model. With a little abuse of notation, we redefine τ_k as the delay at the k-th step, where delay is the sum of application wait time, training time and communication time. ξ_k is the mini-batch from the data samples $\{1, \ldots, B\}$ and the gradient $\nabla F(\theta_{k-\tau_k})$ is computed on the parameters $\theta_{k-\tau_k}$. γ_k is the penalty at the k-th step. We re-write the update rule in (23) using step k,

$$\theta_{k+1} = \theta_k - \frac{\eta_k}{\gamma_k}\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k}). \tag{37}$$

Assumption 1: We make the following common assumptions, *L-Smoothness.*

$$f(y) - f(x) \le \langle \nabla f(x), y - x\rangle + \frac{L}{2}\|y - x\|^2 \tag{38}$$

Bounded Delay: According to Property 2, the control knob V is selected such that the maximum delay is bounded (denote the bound by T_m).

$$\max_k\{\tau_1, \tau_2, \ldots, \tau_k\} \le T_m \tag{39}$$

Bounded variance: The variance is bounded,

$$\mathbb{E}[\|\nabla F(\theta) - \nabla f(\theta)\|^2] \le \sigma^2 \tag{40}$$

Unbiased gradient: The stochastic gradient is an unbiased estimator of the full gradient,

$$\mathbb{E}[\nabla F(\theta)] = \nabla f(\theta) \tag{41}$$

From the L-smoothness property in (38), we have

$$\begin{aligned} f(\theta_{k+1}) - f(\theta_k) &\le \langle \nabla f(\theta_k), \theta_{k+1} - \theta_k\rangle + \frac{L}{2}\|\theta_{k+1} - \theta_k\|^2 \\ &= -\frac{\eta_k}{\gamma_k}\langle \nabla f(\theta_k), \nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k})\rangle \\ &\quad + \frac{L\eta_k^2}{2\gamma_k^2}\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k})\right\|^2 \end{aligned} \tag{42}$$

Taking expectations on both sides of (42), we have

$$\begin{aligned} \mathbb{E}[f(\theta_{k+1})] - \mathbb{E}[f(\theta_k)] &\le -\frac{\eta_k}{\gamma_k}\mathbb{E}[\langle \nabla f(\theta_k), \nabla f(\theta_{k-\tau_k})\rangle] \\ &+ \frac{L\eta_k^2}{2\gamma_k^2}\mathbb{E}\left[\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k})\right\|^2\right] \end{aligned} \tag{43}$$

Using $\langle a, b\rangle = \frac{1}{2}(\|a\|^2 + \|b\|^2 - \|a - b\|^2)$, we have

$$\begin{aligned} \mathbb{E}[f(\theta_{k+1})] - \mathbb{E}[f(\theta_k)] &\le -\frac{\eta_k}{2\gamma_k}\mathbb{E}\Big[\|\nabla f(\theta_k)\|^2 + \|\nabla f(\theta_{k-\tau_k})\|^2 \\ &- \underbrace{\|\nabla f(\theta_k) - \nabla f(\theta_{k-\tau_k})\|^2}_{A_1}\Big] + \frac{L\eta_k^2}{2\gamma_k^2}\mathbb{E}\Big[\underbrace{\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k})\right\|^2}_{A_2}\Big] \end{aligned} \tag{44}$$

Let us compute A_2 first so the partial results can be applied to A_1 as well.

$$\begin{aligned} A_2 &= \mathbb{E}\left[\left\|\left(\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k}) - \nabla f(\theta_{k-\tau_k})\right) + \nabla f(\theta_{k-\tau_k})\right\|^2\right] \\ &= \mathbb{E}\Big[\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k}) - \nabla f(\theta_{k-\tau_k})\right\|^2 + \|\nabla f(\theta_{k-\tau_k})\|^2 \\ &\quad + 2\langle \nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k}) - \nabla f(\theta_{k-\tau_k}), \nabla f(\theta_{k-\tau_k})\rangle \\ &\le \mathbb{E}\left[\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k}) - \nabla f(\theta_{k-\tau_k})\right\|^2 + \|\nabla f(\theta_{k-\tau_k})\|^2\right] \\ &\le \sigma^2 + \mathbb{E}[\|\nabla f(\theta_{k-\tau_k})\|^2] \end{aligned} \tag{45}$$

where the last inequality takes the bounded variance property. A_1 bounds the norm difference of the parameters during delay τ_k. Then using the L-Lipchitz property on A_1, we have

$$A_1 = \|\nabla f(\theta_k) - \nabla f(\theta_{k-\tau_k})\|^2 \le L^2\|\theta_k - \theta_{k-\tau_k}\|^2 \tag{46}$$

By telescoping i from $k - \tau_k + 1$ to k on the update rule in (37), we have

$$\theta_k = \theta_{k-\tau_k} - \sum_{i=k-\tau_k+1}^{k}\frac{\eta_i}{\gamma_i}\nabla F_{\xi_{i-\tau_i}}(\theta_{i-\tau_i}) \tag{47}$$

Plugging (47) into (46), we have

$$\begin{aligned} A_1 &\le L^2\left\|\sum_{i=k-\tau_k+1}^{k}\frac{\eta_i}{\gamma_i}\nabla F_{\xi_{i-\tau_i}}(\theta_{i-\tau_i})\right\|^2 \\ &\le \frac{T_mL^2\eta_k^2}{\gamma_k^2}\left\|\nabla F_{\xi_{k-\tau_k}}(\theta_{k-\tau_k})\right\|^2 \\ &\le \frac{T_mL^2\eta_k^2}{\gamma_k^2}(\sigma^2 + \|\nabla f(\theta_{k-\tau_k})\|^2) \end{aligned} \tag{48}$$

where the second inequality comes from the bounded maximum delay T_m. The last inequality applies the results from (45). Then plug in A_1 and A_2 into (44), and rearrange the terms, we have

$$\begin{aligned} \mathbb{E}[f(\theta_{k+1})] - \mathbb{E}[f(\theta_k)] &\le -\frac{\eta_k}{2\gamma_k}\mathbb{E}\left[\|\nabla f(\theta_k)\|^2\right] + \Bigg(\frac{T_mL^2\eta_k^3}{2\gamma_k^3} \\ &+ \frac{L\eta_k^2}{2\gamma_k^2} - \frac{\eta_k}{2\gamma_k}\Bigg)\mathbb{E}\left[\|\nabla f(\theta_{k-\tau_k})\|^2\right] + \left(\frac{L\eta_k^2}{2\gamma_k^2} + \frac{T_mL^2\eta_k^3}{2\gamma_k^3}\right)\sigma^2 \\ &\le -\frac{\eta_k}{2\gamma_k}\mathbb{E}\left[\|\nabla f(\theta_k)\|^2\right] + \left(\frac{L\eta_k^2}{2\gamma_k^2} + \frac{T_mL^2\eta_k^3}{2\gamma_k^3}\right)\sigma^2 \end{aligned} \tag{49}$$

in which the second inequality follows under the condition that,

$$\frac{T_mL^2\eta_k^2}{\gamma_k^2} + \frac{L\eta_k}{\gamma_k} \le 1. \tag{50}$$

By telescoping k from 1 to K,

$$\begin{aligned} f(\theta_{K+1}) - f(\theta_1) \le &-\sum_{k=1}^{K}\frac{\eta_k}{2\gamma_k}\mathbb{E}\left[\|\nabla f(\theta_k)\|^2\right] \\ &+ \sum_{k=1}^{K}\left(\frac{L\eta_k^2}{2\gamma_k^2} + \frac{T_mL^2\eta_k^3}{2\gamma_k^3}\right)\sigma^2 \end{aligned} \tag{51}$$

and denote θ_{K+1} as θ_*, we can conclude that,

$$\min_{k\in\{1,\ldots,K\}}\mathbb{E}\left[\|\nabla f(\theta_k)\|^2\right] \le$$

$$\frac{\Big(f(\theta_1) - f(\theta_*)\Big) + \sum_{k=1}^{K}\Big(\frac{L\eta_k^2}{2\gamma_k^2} + \frac{T_m L^2 \eta_k^3}{2\gamma_k^3}\Big)\sigma^2}{\sum_{k=1}^{K}\frac{\eta_k}{2\gamma_k}} \tag{52}$$

This finishes the proof of Theorem 2 and we have the following corollary. □

Corollary 1: When we set the ratio between step size and staleness penalty as $\frac{\eta_k}{\gamma_k} = \frac{1}{T_m L\sqrt{k}}$, the online scheme converges at the rate of $\frac{1}{\sqrt{K}}$.

Proof: We set $\frac{\eta_k}{\gamma_k} = \frac{1}{T_m L\sqrt{k}}$. (52) becomes,

$$\begin{aligned}
&\min_{k\in\{1,\ldots,K\}} \mathbb{E}\Big[\|\nabla f(\theta_k)\|^2\Big] \le \\
&\le \frac{2\Big(f(\theta_1) - f(\theta_*)\Big) + \sum_{k=1}^{K}\frac{\sigma^2}{T_m^2 Lk}(1+\frac{1}{\sqrt{k}})}{\sum_{k=1}^{K}\frac{1}{T_m L\sqrt{k}}} \\
&\le \frac{2\Big(f(\theta_1) - f(\theta_*)\Big) + \sum_{k=1}^{K}\frac{2\sigma^2}{T_m^2 Lk}}{\sum_{k=1}^{K}\frac{1}{T_m L\sqrt{k}}} \\
&\le \frac{2T_m L\Big(f(\theta_1) - f(\theta_*)\Big) + \frac{2\sigma^2}{T_m}\log K}{\sqrt{K}}
\end{aligned} \tag{53}$$

where the second inequality takes the fact that $(1+\frac{1}{\sqrt{k}}) \le 2$ for $k \in \{1,\ldots,K+1\}$ and last step uses $\sum_{k=1}^{K}\frac{1}{\sqrt{K}} \ge K\cdot\frac{1}{\sqrt{K}} = \sqrt{K}$. □

The $1/\sqrt{K}$ convergence rate is consistent with the previous studies of asynchronous paralleled SGD [19], [20]. Note that the analysis is under the assumption of bounded delay of scheduling decisions in Property 2. There are also some recent works that provide tighter convergence analysis without the bounded-delay assumption [21], [22], [23], where the convergence rate is a function of the maximum or average delay, e.g., the improved convergence $\mathcal{O}(\frac{\sigma^2}{\epsilon^2} + \frac{\tau_{\max}}{\epsilon})$ in [22], where ϵ is the bound of the squared gradient norm. We aim to investigate how these theoretical analysis can shed light on system-level designs in the future.

VII. System Implementation

To conduct training on Android, we adopt DL4J [41], a Java-based Deep Learning framework for Android. The backend neural computations are conducted by OpenBLAS cross-compiled for the ARM platforms. We pre-load CIFAR10 datasets [52] into the flash storage. Background training is implemented with `JobScheduler`, originally designed for long-running operations in the background. Execution time window such as networking connectivity (Wifi/4 G), device status (idling or charging) can be specified to offer fine-grained control. Once the job scheduler is started using `onStart`, a new thread is created to initialize the neural network. We enable the `largeHeap` to give the App 512 MB memory to avoid memory errors. Since the DL4J libraries and dependencies are not optimized, the training app occupies more than 300 MB, leaving the neural network little less than 200 MB space during runtime. To this end, the current implementation can only support lightweight training such as LeNet5 with small batch sizes at this point.

The number of CPU cores designated for background services is specified by the vendor, which can be found under `/dev/cpuset/background/cpus`. E.g., Pixel2 and Mate10 utilize the two little cores; Nexus6P, Hikey970 only run on the one little core and the rest of the three little cores are reserved for system processes. The default kernel (e.g., CPU affinity, priority, frequency scaling) is used and no root access is required for our framework. We set the number of training threads to 2 or 1 based on the vendor specifics, because a large value would conversely lead to potential contentions and coherence overhead.

The Android kernel might kill the background training process to save memory and optimize battery lifetime. We do not find the service being killed while training, but introducing more convolutional layers with large filter size would invoke the automatic background limitation because those layers are the major resource consumers. In practice, there also exists a few "diehard" tricks such as escalating the app priority, service binding [53], whereas a fundamental solution on the OS level is out of the scope of this paper.

The communication module is developed based on the Retrofit Framework [54], which easily packages asynchronous HTTP requests to a Python-based HTTP server. For Async-SGD, once a device completes a local epoch, it creates a Retrofit `FileuploadService` to upload the local model with meta information (device ID, round #) to the server. The server replaces the current copy of the global model upon receiving it. When the device becomes available, it downloads the current model using the `FileDownloadService` as a starting point for the next local epoch.

VIII. Evaluation

Testbed/Parameter Settings: The evaluation is conducted on devices from different vendors: Nexus 6/6P, HiKey970 Dev. Board, Pixel2, Huawei P30 Pro and OnePlus9. We adopt the LeNet5 architecture for training and each device trains on a partition of 2 K images of the CIFAR10 dataset with a mini-batch size of 20.

A. Energy Measurement

First, we measure the energy consumptions of different control decisions as summarized in Table II: background *training only* (1st row), *application only* (1st col) and *co-running* (2nd col). To avoid breaking the devices while removing the battery and screen connectors, we use a combination of software profilers and power monitor: Trepn [42], Perfetto [43] and Monsoon Power Monitor [44]. Trepn is used for Nexus6/6P and Perfetto is built into Android 10 in Pixel2, P30 Pro and Oneplus9. For the Hikey970 Dev. board, we directly power it with 12 V DC input from the Monsoon Power Monitor.

We measure the system-wide energy consumption from the device which includes all the system background threads. To reduce the variances, we disable all irrelevant applications that might have processes lingering in the background. We choose

TABLE II
AVERAGED ENERGY MEASUREMENTS - BATTERY POWER (W), ENERGY SAVING RATIO ↑ (NEGATIVE MEANS MORE ENERGY CONSUMPTION), EXECUTION TIME (S) AND SLOWDOWN ↓ (NEGATIVE MEANS PERFORMANCE SPEEDUP) RUNNING LENET-5 OF CIFAR10 DATASET

	Nexus6					Nexus6P					Hikey970				
Apps	app	co-run	saving	time	slowdown	app	co-run	saving	time	slowdown	app	co-run	saving	time	slowdown
Training	1.8	–	–	204s	–	0.9	–	–	211s	–	7.87	–	–	213s	–
Map	3.4	3.5	10%	274s	34%	0.5	1.3	1%	225s	6.6%	8.82	9.42	**51%**	186s	**-12%**
News	1.7	2.2	**26%**	239s	17%	0.44	1.2	-54%	362s	71%	9.17	9.76	44%	210s	-1.4%
Trading	1.4	2.4	13%	236s	**16%**	0.48	0.96	**25%**	228s	8%	8.50	9.15	49%	195s	-8.4%
Youtube	0.5	1.9	-15%	284s	39%	0.53	1.2	13%	220s	**4.2%**	9.15	11.45	34%	210s	-1.4%
Tiktok	1.6	2.3	2%	296s	45%	1.0	1.1	**-85%**	675s	**220%**	11.0	11.2	**24%**	271s	**27%**
Zoom	1.2	2.1	-27%	370s	81%	1.4	1.6	-12%	340s	61%	7.89	8.53	47%	209s	-1.8%
CandyCrush	1.3	2.3	**-263%**	997s	**388%**	0.7	1.3	-8%	280s	32%	11.1	11.26	35%	233s	9.4%
Angrybird	2.5	2.8	-28%	400s	96%	1.1	1.2	-76%	620s	193%	10.1	10.7	44%	200s	-6.1%

	Pixel2					P30 Pro					OnePlus9				
Apps	app	co-run	saving	time	slowdown	app	co-run	saving	time	slowdown	app	co-run	saving	time	slowdown
Training	1.35	–	–	223s	–	0.776	–	–	123s	–	0.684	–	–	129s	–
Map	1.6	2.2	34%	196s	**-12%**	2.48	2.76	**-3%**	154s	**25.8%**	1.04	1.12	**35%**	130s	**0.77%**
News	1.82	2.4	33%	197s	-11%	2.61	3.02	1%	141s	14%	1.29	1.39	24%	138s	6.98%
Trading	1.72	2.23	33%	206s	-8%	1.70	2.22	13%	125s	**1.6%**	0.91	1.09	26%	140s	8.52%
Youtube	2.04	2.21	34%	226s	1.3%	1.88	2.39	11%	127s	3.2%	0.94	1.14	25%	138s	6.98%
Tiktok	2.37	2.52	36%	212s	-4.9%	2.94	3.37	4%	135s	9.7%	1.22	1.58	**6%**	146s	**13.18%**
Zoom	2.57	3.11	27%	206s	-7.6%	4.36	4.41	2%	144s	17%	1.19	1.28	27%	138s	6.98%
CandyCrush	2.89	2.92	**39%**	199s	-10.8%	3.12	3.22	11%	137s	11%	1.26	1.50	17%	139s	7.75%
Angrybird	2.86	2.88	**13%**	285s	**27.8%**	2.96	2.98	**15%**	135s	9.75%	1.18	1.31	26%	136s	5.42%

Each device conducts training on 4000 images with batchsize 20. The best value is marked in blue and the worst in red.

a number of 8 popular applications that users usually spend considerable time. The percentage of energy saving is calculated as, $1-\frac{P_i^c t_c}{P_i^b t_b+P_i^a t_a}$, which involves the execution time t_a, t_b, t_c for application, background training and co-execution. We may have less power consumption but higher execution time (performance slowdown). In these cases, the energy saving percentage is diminished - e.g., P30 Pro were supposed to provide around 20-30% power saving as we measured in the experiment, but the extra slowdown of the training execution actually reduces the total energy saving to around 10% in Table II. If the performance slowdown is too large, it would totally offset the energy saving efforts such as some of the cases in Nexus 6/6P and the optimal decision should be to execute the training alone without co-execution.

We notice that the newer generations of devices offer higher energy saving about an average of 10–30% and even slight speedup of background training. However, for older chipsets such as Nexus 6 with homogeneous cores, co-running only offers marginal energy improvements depending on the application. Some applications even have energy surge due to contention on cache resources, which further leads to CPU throttling and elongated training time. In these cases, the online controller is expected to avoid co-execution.

B. Simulation Evaluation

Evaluation Settings: We set the probability of application arrival to 0.001 in each time slot, i.e., an average of 1 app arrival for every 1000 s. The user randomly picks a device from the list and chooses an app from the 8 representative apps. We set the number of users to 25 (equal data partition) and slowdown bound $D_m = 500$. For non-IID settings, each user has a random collection of 3 and 6 classes for MNIST and CIFAR10. The total training time is set to 3 hours and each time slot is 1 s. We adopt the *offline scheduling* and fixed policy of *immediate scheduling* as the baselines. Immediate scheduling runs the background training when a device is available regardless of the application arrivals. A look-ahead time window is set for the offline solution, which invokes the algorithm every 500 s. We also compare the online scheme with FedAvg [6] and FedProx [7].

Control Knob V: The control knob V can be considered as a hyperparameter that balances the energy consumption and the (virtual) queue length in the system. In principle, a small V (near zero) tend to schedule training more aggressively without waiting for the foreground application, thus resulting in high energy consumption with less backlog in the queues. As V grows larger, users would act conservatively to wait for co-running opportunities, which corresponds to the decline of energy consumption and the increase of the queue backlogs. This is observed in Fig. 5(a-c) as the energy consumption drops quickly, and $Q(t)$, $H(t)$ increase with a larger V. Meanwhile, Fig. 5(d) shows that $J(t)$ of performance slowdown also exhibits an uprising trend because of additional latency imposed by co-execution. The staleness bound G_m implies different levels of tolerance to staleness. With a larger G_m, more devices are put into idling to wait for applications, so the energy consumption is lower. Since a larger V would have marginally reduced energy saving compared to the increase of queue length, practitioners could focus on the values below 5000 to achieve a reasonable energy-performance balance.

Energy and Gradient Staleness: Fig. 6(a) compares the energy consumption of different scheduling policies. Immediate scheduling has the maximum energy consumption as it quickly turns on training regardless of application arrival. Offline scheduling offers the minimum energy consumption by looking ahead of time. Since FedProx is still a synchronous scheme, the energy consumption is the same with FedAvg. Once we set $V = 5000$, by adjusting G_m, online scheduling could save more

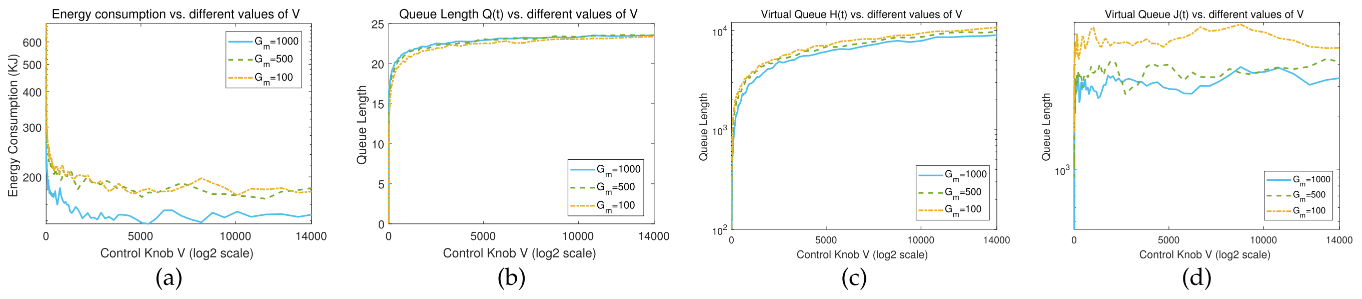


Fig. 5. Impact of adjusting control knob V on (a) Energy consumption; (b) Queue length $Q(t)$; (c) Virtual queue length $H(t)$; (d) Virtual queue length $J(t)$.

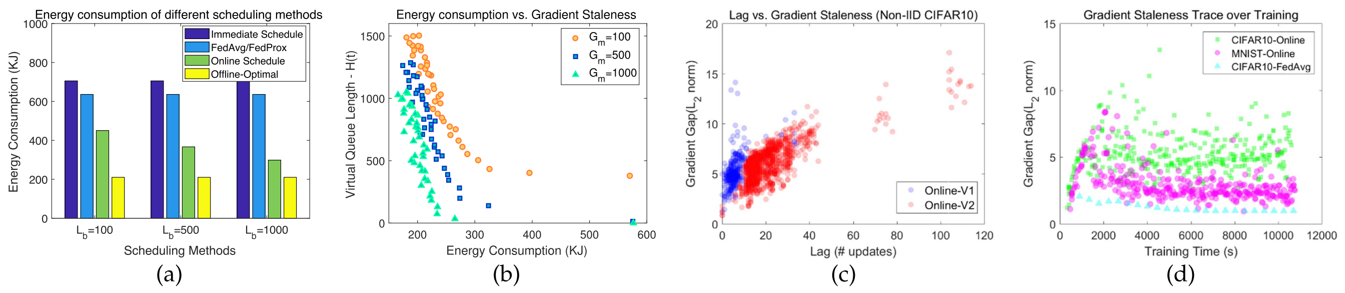


Fig. 6. Energy consumption and gradient staleness (a) comparison of total energy consumption (green bar is the proposed online scheme); (b) trade-offs between energy consumption and queue delay; (c) lag versus gradient gap of different online settings; (d) trace of gradient gaps for different datasets and methods.

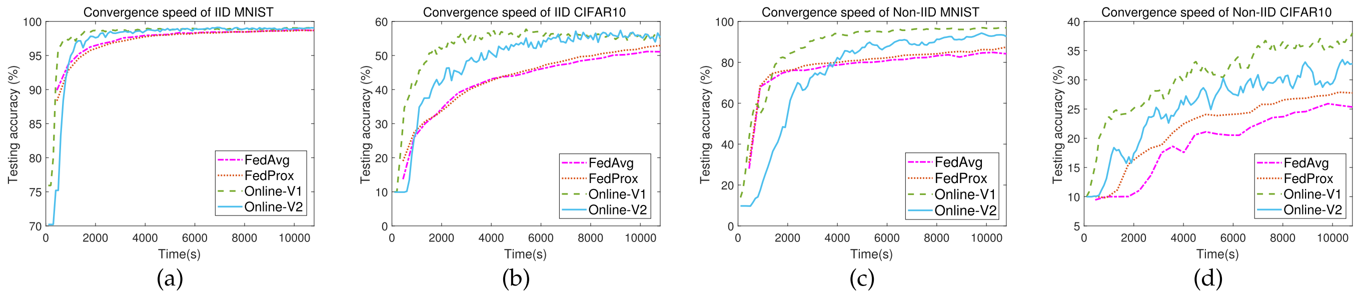
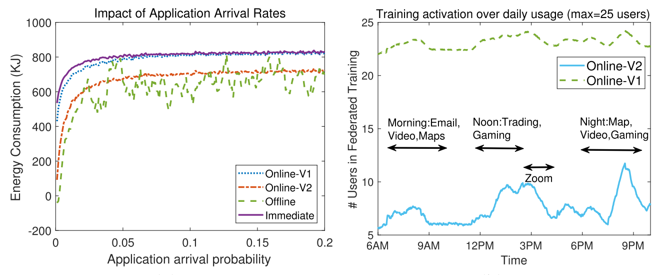


Fig. 7. Comparison of training convergence between the online method with FedAvg [6] and FedProx [7]. (a) MNIST with IID data; (b) CIFAR10 with IID data; (c) MNIST with non-IID data; (d) CIFAR10 with non-IID data.

than 50% energy compared to FedProx/FedAvg and is only 10% away from the optimal offline solution. Fig. 6(b) further validates the $[\mathcal{O}(1/V), \mathcal{O}(V)]$ energy-staleness/slowdown trade-offs as the attempt of energy reduction would ultimately lead to congestion of the virtual queues.

Then we use two sets of parameters for the online scheme: $G_m = 100$ and $V = 1000$ denoted as *Online-V1* and $G_m = 1000$ and $V = 2 \times 10^5$ denoted as *Online-V2*, where Online-V1 schedules training more aggressively with higher energy consumption. Fig. 6(c) first shows that gradient staleness is generally proportional to the lags. Since Online-V2 is more conservative, the gradient staleness is much larger compared to Online-V1. Note that the same lag value could correspond to different staleness at different stages of training. Thus, gradient gap provides a more accurate measure of staleness. Fig. 6(d) compares the gradient staleness between different datasets and FedAvg. The staleness values of FedAvg are sampled at model averaging with a declining trend. In contrast, the online scheme forms an upward trend at the beginning but sideway movements in the later iterations due to the difference between local parameters, and their values grow with the complexity of the datasets (CIFAR10 has higher staleness compared to MNIST).

Model Convergence: Since the control decisions also affect the ML algorithms, the overall performance should be also measured in terms of model convergence. For the online scheme, the speedups are determined by: 1) the number of meaningful updates contributed by individual users in fixed time intervals (i.e., throughput); 2) the accumulation and variance of gradient staleness. Fig. 7 compares the test accuracy of different schemes in both IID and non-IID settings. For IID data, FedProx is on par with FedAvg; for non-IID data, it offers about 3-4% improvements of accuracy. The online scheme surpasses both



(a) (b)

Fig. 8. Evaluation of application arrival rates (a) total energy consumption; (b) number of users in training with daily application usage [28].

FedProx and FedAvg by a large margin (5-10% on CIFAR10) with additional 50% energy saving. For non-IID data, staled updates from users would have large variance in their gradient directions so asynchronous updates are inherently more noisy. Although synchronous approaches such as FedAvg and FedProx have faster and stable theoretical convergence, they face computational heterogeneity especially in a mobile environment, i.e., waiting for stragglers with slow execution. Asynchronous method allows faster users to proceed without waiting for the stragglers, thus having higher utilization and system throughput. This is consistent with [20] that the trade-offs between system throughput and gradient staleness are the keys in asynchronous algorithms. Our empirical finding also supports that the asynchronous primitive is more suitable for mobile devices with heterogeneous computing power compared to FedAvg and FedProx.

Impact of Application Arrival: Our strategy depends on the intensity of application usage for energy saving. We further evaluate the impact of different application arrival rates from $0 - 0.2$ per time slot. Fig. 8(a) shows the application arrival rate versus energy consumptions. With more running applications, the general energy consumption follows an increasing trend for all three schemes. Immediate scheduling is independent of application arrivals and the energy saving comes from the coincident co-execution. In contrast, the online scheme is able to utilize the application arrival more wisely as we can see the initial gap from immediate scheduling is large. As the application rate rises, co-execution quickly saturates and the online scheme has degraded to the immediate scheme.

Because the offline scheme foresees the co-running opportunities, it is able to achieve the lowest energy consumption when applications are scarce but will aggressively schedule with the applications when the arrival rate increases. Due to random arrivals, the actual number of application arrivals is different under various probabilities, hence the fluctuations in the offline baseline. For the online setting, the scheduling decision is determined several analytical equations. The controller schedules training according to a weighted balance, thus having less fluctuations compared to the pure algorithmic dynamic programming solution. Meanwhile, it is interesting to see that sometimes the online approach even achieves lower energy than the offline baseline. This is because the offline approach is also sub-optimal because of Property 1 and other approximations.

TABLE III
ENERGY OVERHEAD OF ONLINE OPTIMIZATION (W)

	Nexus 6	Nexus 6P	Pixel 2
Power(idle)	0.238	0.486	0.689
Power(comp.)	0.245	0.525	0.736
Overhead (%)	3.0%	7.4%	6.3%
	HiKey970	P30	OnePlus9
Power(idle)	5.67	0.490	0.395
Power(comp.)	5.98	0.510	0.422
Overhead (%)	5.4%	4%	6.8%

As application usage depends on a variety of contextual cues such as time and location, we conduct more experiments based on the Sequential Mobile App usage dataset [28]. Fig. 8(b) shows the number of devices in the training mode from 6AM-10PM with certain events at specific time, e.g., watching video during the breakfast/dinner and GPS Maps during commute. We can see that the online schemes put more users into training while there are more activities (during the lunch break and evening time); otherwise, when application usage is scarce, it will adjust the control decisions based on the virtual queue backlogs.

Scheduling Overhead: The online scheme involves lightweight computation to evaluate (22). The energy overhead is shown in Table III. Here, "idle" means the power with no scheduling or training (as well as no apps running in the background); "comp" means while the device is taking extra computations to reach an online decision. The actual implementation requires to sense the foreground apps with `getRunningTasks` in `Activity` and a few if-else branches with light computation. The results indicate that the overhead is below 10% in each time slot. To reduce the overhead, we can optimize the scheduling granularity with a larger decision interval.

C. System Evaluation

1) CPU Utilization: We gather more system traces from Mate10 (same SoC with HiKey970) and P30 Pro to validate our design. Fig. 9 shows the CPU utilizations of all 8 cores when perform training and co-running with CandyCrush on Mate10. It is observed that the training threads are utilizing the small cores 2,3 while the cores 0,1 are reserved for other system processes. Similarly, during co-execution, CandyCrush is running on the big cores 0,1 and the average utilization is under 50%. This validates our multitasking design based on the big.LITTLE microarchitecture. By taking a closer look at the utilization of small cores, it periodically drops below 100% due to access to the external memory in neural computations - a common challenge in optimizing deep learning [56], [57]. Here, the memory bottleneck becomes more prominent on mobile devices.

2) Impact on Foreground Apps: To provide an overall assessment of slowdown of the foreground apps, we gather more statistics shown in Fig. 10. For different apps, there are two FPS targets: 30 and 60 FPS. If the FPS rates fall below 24, there will be a noticeable lag on the user's end. In our experiment, most of the frame drops are caused by frequent user interactions such as

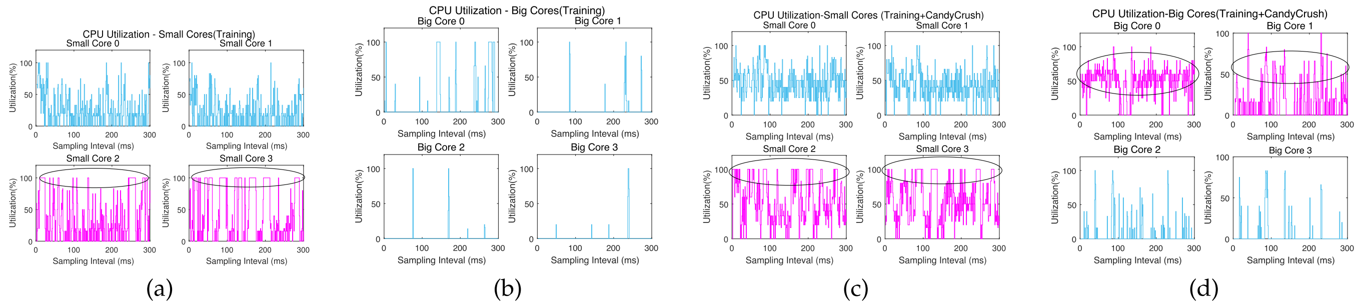


Fig. 9. (a) and (b) Visualization of per-core CPU utilizations on Mate10 performing training on the background and (c) and (d) co-running training with CandyCrush. The cores with the designated workloads are circled.

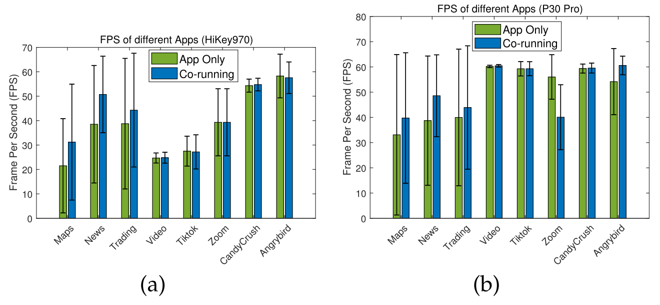


Fig. 10. FPS when co-executed with different applications (a) Mate10 (b) P30 Pro.

clicking, fast switching between web pages while the threads are fetching the content or slowly responding due to system stall. By judging the difference between the bars in Fig. 10, we see that co-running has little impact on the foreground FPS as it achieves comparable (above 24) or even higher FPS rates (e.g. News, Trading and Maps). We conjecture the higher FPS is due to more resources are activated such as higher memory frequency by Dynamic Voltage and Frequency Scaling (DVFS), once the foreground app is activated. On the other hand, gaming mainly uses the Mali GPU with less memory requests, thus we do not see much contention or FPS lags on the foreground games.

3) Memory Bandwidth: Finally, we measure the memory bandwidth as shown in Fig. 11, where (a)(e) provide a baseline of the background training only. For the rest plots, the top subfigure is running the foreground app only and the bottom one is co-running with training. Mate10 has 4 bandwidth states at $\{4, 8, 12, 18\}$ Gbps and P30 Pro has 7 states at $\{4, 6.5, 8, 12, 16, 18, 20.5\}$ Gbps. We show the histogram of bandwidth distributions in different states during execution. In practice, the achievable bandwidth is determined by the row buffer hit ratio of memory requests, which is a characteristic of the spatial memory locality of the running apps [58]. Each app has its own memory access patterns: in (a)(e), the foreground and background training exhibit identical bandwidth distributions; for co-running, CandyCrush requires less memory access; Tiktok/Trading involve more user interactions, thus resulting in higher overall bandwidth. While co-running, the bandwidth pattern is dominated by the foreground app because it is given higher priority to access the shared resources. Background training mainly resides at the low-bandwidth states. E.g., P30 Pro and Mate10 have 86% and 100% of bandwidth states less than 12 Gbps. To this end, compared with running the app alone, co-execution only has slightly higher bandwidth (with some exceptions in (c)(e)), so an overall energy saving is accomplished.

IX. Discussion

The generality of our design on ARM's big.LITTLE microarchitecture: As evidenced by [59], "over 99% of the premium smartphones are powered by ARM", because of its extraordinary energy efficiency and thermal management on battery-powered devices. The prevalent dominance of ARM SoCs also extends to AIoT boards and other embedded devices. Hence, we expect the applicability of the proposed scheme on most of the mobile devices powered by ARM as experimented on a wide range of devices in this paper. On the other hand, the asymmetric/hybrid CPU technology is also transformative to the world of desktops and servers. E.g., Intel's Alder Lake x86 CPUs featured the performance cores (big) and efficient cores (small). AMD also releases the first series of hybrid CPU processors Ryzen 7040 U mobile processors for notebook. We believe this would be an ongoing trend with major chip makers to commit resources in energy-efficient computing for consumer electronics and data centers.

The support of our design on different neural network models: We have also experimented Mobilenet and VGG-4 in our design. Although Mobilenet is known to have high inference performance on mobile CPUs, training is much slower compared to LeNet-5 (on the similar scale of FLOPs). This is consistent with some reports that training Mobilenet is much slower on the GPUs due to the depthwise convolution designs [13]. Training these architectures on high-end mobile CPUs exhibits slowdown as well. E.g., training with the foreground threads spends around 10 minutes per epoch compared to 2 minutes of LeNet-5 with a slowdown of 5×. The background training on the little cores almost tripled the execution time compared to the big cores. Due to the extended execution time, the process (background service) is more prone to be killed by the `low memory killer`, or preempted by other background processes, thus having much higher variance and fluctuations for different mini-batches. With higher FLOP count, VGG-4 even executes faster on the big cores, but the overall memory footprint is approaching the 500 MB uplimit.

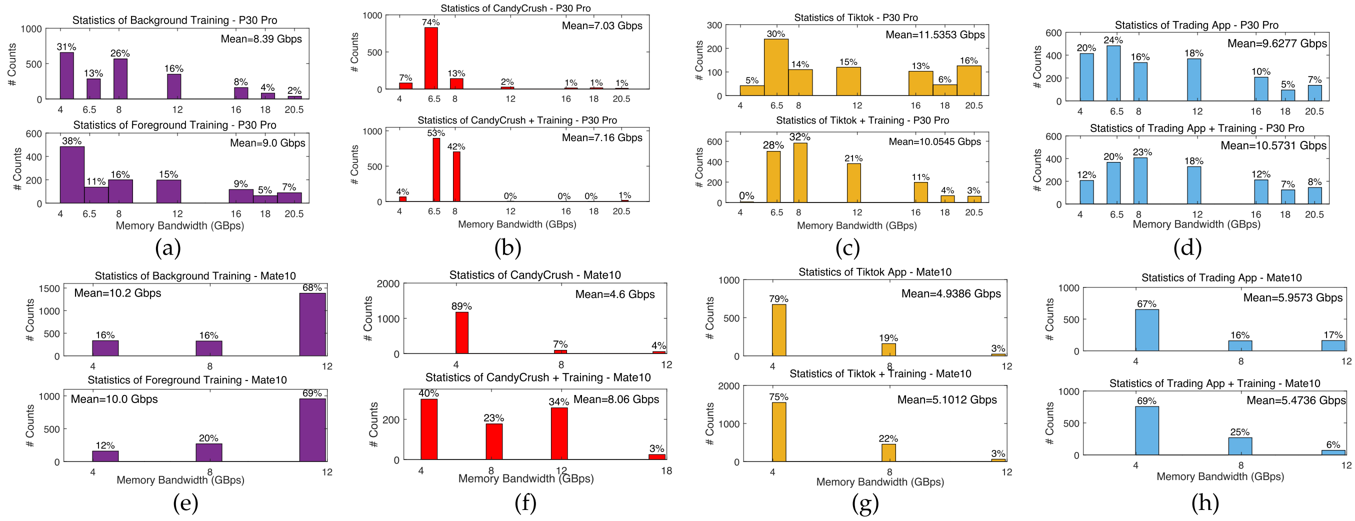


Fig. 11. Memory bandwidth states while performing (a) and (e) background/foreground training alone; (b), (c), (d), (f), (g), and (h) Candycrush, Tiktok, Trading (top subfigure) and co-execute with training (bottom subfigure). P30 Pro (first row) and Mate10 (second row).

X. Conclusion

In this paper, we combine asynchronous federated learning with application co-execution to minimize energy consumptions on consumer mobile devices. We motivate this work by real measurements and explore the system dynamics for energy saving. Based on the offline problem solution, we develop an online scheme to address the energy-staleness/slowdown trade-offs with low computational overhead. We provide theoretical proofs of such trade-off and training convergence. Our extensive evaluation demonstrates that the online scheme achieves 50% energy saving compared to FedAvg and FedProx, and only 10% away from the optimal solution. The proposed scheme can adapt to different application usage patterns based on daily activities, while keeping the devices in low power state during the rest of the time. System traces collected from the hardware counters also validate our design.

Acknowledgment

The authors are grateful to the anonymous reviewers for their insights and detailed comments.

References

[1] AR6 Synthesis Report of Climate Change, 2023. Accessed: Mar. 28, 2024. [Online]. Available: https://www.ipcc.ch/report/ar6/syr/
[2] E. Garcia-Martin, C. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *J. Parallel Distrib. Comput.*, vol. 134, pp. 75–88, 2019.
[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proc. Conf. Empir. Methods Natural Lang. Process.*, 2019, pp. 3645–3650.
[4] A. Capozzoli and G. Primiceri, "Cooling systems in data centers: State of art and emerging technologies," *Energy Procedia*, vol. 83, pp. 484–493, 2015.
[5] X. Qiu, T. Parcollet, D. Beutel, T. Topal, A. Mathur, and N. Lane, "A first look into the carbon footprint of federated learning," 2021, *arXiv: 2010.06537*.
[6] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[7] T. Li, A. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, 2020, pp. 429–450.
[8] F. Hanzely and P. Richtarik, "Federated learning of a mixture of global and local models," 2021, *arXiv: 2002.06440*.
[9] S. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. Suresh, "SCAFFOLD: Stochastic controlled averaging for federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5132–5143.
[10] J. Wang, Q. Liu, H. Liang, G. Joshi, and V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 7611–7623.
[11] S. Wang et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.
[12] C. Wang, Y. Yang, and P. Zhou, "Towards efficient scheduling of federated mobile devices under computational and statistical heterogeneity," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 394–410, Feb. 2021.
[13] C. Wang, Y. Xiao, X. Gao, L. Li, and J. Wang, "Close the gap between deep learning and mobile intelligence by incorporating training in the loop," in *Proc. ACM Int. Conf. Multimedia*, 2019, pp. 1419–1427.
[14] F. Niu, B. Recht, C. Re, and S. Wright, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
[15] S. Zheng et al., "Asynchronous stochastic gradient descent with delay compensation," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 4120–4129.
[16] C. Xie, O. Koyejo, and I. Gupta, "Asychronous federated optimization," in *Proc. 12th Annu. Workshop Optim. Mach. Learn.*, 2020, pp. 1–11.
[17] I. Mitliagkas, C. Zhang, and S. Hadjis S, "Asynchrony begets momentum, with an application to deep learning," in *Proc. IEEE Annu. Allerton Conf. Commun., Control, Comput.*, 2016, pp. 997–1004.
[18] W. Zhang, S. Gupta, X. Lian, and J. Liu, "Staleness-aware Async-SGD for distributed deep learning," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 2350–2356.
[19] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2737–2745.
[20] W. Dai, Y. Zhou, N. Dong, H. Zhang, and E. Xing, "Toward understanding the impact of staleness in distributed machine learning," in *Proc. Int. Conf. Learn. Representations*, 2018.
[21] L. Nguyen, P. Nguyen, M. Dijk, P. Richtarik, K. Scheinberg, and M. Takac, "SGD and hogwild! convergence without the bounded gradients assumption," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3747–3755.
[22] S. Stich and S. Karimireddy, "The error-feedback framework: Better rates for SGD with delayed gradients and compressed updates," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 9613–9648, 2020.
[23] A. Koloskova, S. Stich, and M. Jaggi, "Sharper convergence guarantees for asynchronous SGD for distributed and federated learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2022, pp. 17202–17215.

[24] H. Yang, Z. Liu, Y. Fu, T.Q. Quek, and H. Poor, "Federated stochastic gradient descent begets self-induced momentum," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, 2022, pp. 9027–9031.
[25] C. Wang, B. Hu, and H. Wu, "Energy minimization for federated asynchronous learning on battery-powered mobile devices via application co-running," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2022, pp. 939–949.
[26] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "VELTAIR: Towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling," in *Proc. Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2022, pp. 388–401.
[27] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "MISE: Providing performance predictability and improving fairness in shared main memory systems," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Architecture*, 2013, pp. 639–650.
[28] M. Aliannejadi, H. Zamani, F. Crestani, and W. Bruce Croft, "Context-aware target apps selection and recommendation for enhancing personal mobile assistants," *ACM Trans. Inf. Syst.*, vol. 39, no. 3, pp. 1–30, 2021.
[29] ARM's big.LITTLE architecture. Accessed: Mar. 28, 2024. [Online]. Available: https://bit.ly/3ZaVoIM
[30] ARM's DVFS scheme. Accessed: Mar. 28, 2024. [Online]. Available: https://bit.ly/3LPSGFX
[31] A. Pathak, C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: A first look at energy bugs in mobile devices," in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, pp. 1–6.
[32] Android optimize for doze and app standby. Accessed: Mar. 28, 2024. [Online]. Available: https://bit.ly/3TSavGp
[33] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μLayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1–15.
[34] JobScheduler. Accessed: Mar. 28, 2024. [Online]. Available: https://bit.ly/2V7M8Ku
[35] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput CNN inference on embedded ARM Big.LITTLE multicore processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2254–2267, Oct. 2020.
[36] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, set, go: Coalesced offloading from mobile devices to the cloud," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 2373–2381.
[37] T. Zhang, X. Zhang, F. Liu, H. Leng, Q. Yu, and G. Liang, "eTrain: Making wasted energy useful by utilizing heartbeats for mobile data transmissions," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 113–122.
[38] M. Zhu and K. Shen, "Energy discounted computing on multicore smartphones," in *Proc. USENIX Annu. Tech. Conf.*, 2016, pp. 129–141.
[39] N. Lane et al., "Piggyback crowdsensing (PCS): Energy efficient crowdsourcing of mobile sensor data by exploiting smartphone app opportunities," in *Proc. 11th ACM Conf. Embedded Netw. Sensor Syst.*, 2013, pp. 7:1–7:14.
[40] Hikey970 board. Accessed: Mar. 28, 2024. [Online]. Available: https://www.96boards.org/product/hikey970/
[41] Deep learning for java. Accessed: Mar. 28, 2024. [Online]. Available: https://deeplearning4j.org
[42] Trepn profiler. Accessed: Mar. 28, 2024. [Online]. Available: https://bit.ly/2UvELwl
[43] Google perfetto. Accessed: Mar. 28, 2024. [Online]. Available: https://github.com/google/perfetto
[44] Moonsoon power monitor. Accessed: Mar. 28, 2024. [Online]. Available: https://shorturl.at/gjEP0
[45] K. Dudzinski and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *Eur. J. Oper. Res.*, vol. 28, no. 1, pp. 3–21, 1987.
[46] M. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, 2010.
[47] Z. Tu et al., "Your apps give you away: Distinguishing mobile users by their app usage fingerprints," in *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 2, no. 3, pp. 1–23, 2018.
[48] I. Hakimi, S. Barkai, M. Gabel, and A. Schuster, "Taming momentum in a distributed asynchronous environment," 2019, *arXiv: 1907.11612*.
[49] S. Barkai, I. Hakimi, and A. Schuster, "Gap-aware mitigation of gradient staleness," in *Proc. Int. Conf. Learn. Representations*, 2020.
[50] A. Kosson, V. Chiley, A. Venigalla, J. Hestness, and U. Koster, "Pipelined backpropagation at scale: Training large models without batches," in *Proc. Mach. Learn. Syst.*, 2021, pp. 479–501.
[51] MNIST dataset. Accessed: Mar. 28, 2024. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[52] CIFAR10 Dataset. Accessed: Mar. 28, 2024. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[53] H. Zhou et al., "Demystifying diehard Android apps," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 187–198.
[54] Retrofit for Android. Accessed: Mar. 28, 2024. [Online]. Available: https://github.com/square/retrofit
[55] Q. Meng, W. Chen, J. Yu, T. Wang, Z. Ma, and T. Liu, "Asynchronous accelerated stochastic gradient descent," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1853–1859.
[56] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *Proc. IEEE/ACM 49th Annu. Int. Symp. Microarchitecture*, 2016, pp. 18:1–18:13.
[57] A. Boroumand et al., "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proc. Int. Conf. Architect. Support Program. Lang. Operating Syst.*, 2018, pp. 316–331.
[58] S. Srikanthan, S. Dwarkadas, and K. Shen, "Data sharing or resource contention: Toward performance transparency on multicore systems," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 529–540.
[59] ARM smartphone market. Accessed: Mar. 28, 2024. [Online]. Available: https://www.arm.com/markets/consumer-technologies/smartphones

Cong Wang (Member, IEEE) received the BEng degree in information engineering from the Chinese University of Hong Kong, in 2008, the MS degree in electrical engineering from Columbia University, in 2009, and the PhD in computer and electrical engineering from Stony Brook University, NY, USA, in 2016. He worked as a tenure-track assistant professor in computer science with Old Dominion University and George Mason University, VA, USA from 2017–2023. He is now a professor with Zhejiang University, Hangzhou, China under the Hundred Talents Program. His research focuses on mobile and edge computing. He is the recipient of NSF CAREER Award, in 2021.

Hongyi (Michael) Wu (Fellow, IEEE) received the BS degree in scientific instruments from Zhejiang University, Hangzhou, China, in 1996, the MS degree in electrical engineering and the PhD degree in computer science from the State University of New York at Buffalo, Buffalo, NY, USA, in 2000 and 2002, respectively. He is a professor and the department head of electrical and computer engineering with the University of Arizona, Tucson, AZ, USA. He has been a PI or the Co-PI for more than 50 projects which have received more than \$24 M in funding from various sources, including the NSF, NSA, DOD, DOE, NATO, State governments, and industry partners. He has published more than 160 technical articles in journals and conference proceedings and acquired a U.S. patent. His current research focuses on security and privacy in intelligent computing and communication systems. He has received several awards and honors throughout his career, including an NSF CAREER Award, in 2004, the UL Lafayette Distinguished Professor Award, in 2011, the IEEE Percom Mark Weiser Best Paper Award, in 2018, and the UB CSE Distinguished Alumni Award, in 2022. He has chaired several conferences, including the prestigious IEEE Infocom 2020. He has also served on the editorial board of several journals, including as a lead topic editor of *IEEE Transactions on Computers* since 2023, an area editor for *IEEE Internet of Things Journal* since 2022, and an associate editor for *IEEE Transactions on Computers* since 2022, *IEEE Transactions on Mobile Computing* since 2018, *IEEE Internet of Things Journal* from 2014 to 2018, and *IEEE Transactions on Parallel and Distributed Systems* from 2013 to 2016.