Prototyping Opportunistic Learning in Resource Constrained Mobile Devices

Haoxiang Yu*, Hsiao-Yuan Chen*, Sangsu Lee*, Xi Zheng†, Christine Julien*
*Department of Electrical and Computer Engineering, University of Texas at Austin
{hxyu, littlecircle0730, sethlee, c.julien}@utexas.edu
†Department of Computing, Macquarie University, james.zheng@mq.edu.au

2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) | 978-1-6654-1647-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/PerComWorkshops53856.2022.9767493

***Abstract*—With the increasing capabilities of pervasive computing devices, training machine learning models *on-device* has become feasible. At the same time, demands for increased user privacy and reduced communication overhead have brought decentralized machine learning to the forefront. In these paradigms, individual devices collaborate *opportunistically* to train models using locally available data. In this paper, we examine the practical feasibility of such *opportunistic learning*. In a basic opportunistic learning approach, when a device (the *learner*) encounters another device (the *neighbor*), it can request the neighbor to perform training on the learner's behalf using the neighbor's own local data. To realize an opportunistic learning in the real world, one must solve two challenges: (1) leveraging device-to-device communication to discover neighbors and exchange models and (2) training models on the device subject to resource and latency constraints. In this paper, we examine the feasibility of implementing opportunistic learning to learn a convolutional neural network (CNN) model for an image classification task using a small network testbed of diverse iOS devices. We demonstrate success in implementing a completely decentralized approach and characterize the challenges and opportunities that lie ahead.**

***Index Terms*—mobile computing, distributed machine learning, pervasive computing, on-device training**

I. Introduction

Mobile devices capture and store huge amounts of private data, including personal health data, text messages, photos, etc. This data can be leveraged to train models that support subsequent user interactions. Consider a model on a user's smartphone that can identify objects in photos to allow keyword search over the library of photos. Training such a model clearly requires a very large number of labeled photos. However, in creating such models, privacy is always a central concern, as users oppose sharing private data, like the photos they take with their personal devices. Federated learning [1], [2] performs model training on edge devices (such as users' smartphones) without requiring private data to be shared with a third party. Federated learning is coordinated by a central server that performs model averaging for a large set of client edge devices. Decentralized approaches have also emerged that leverage opportunistic device-to-device interactions to train models without relying on a central coordinator. For example, in opportunistic learning [3], an edge device solicits neighboring devices to assist in training using their local data.

In this paper, we examine the feasibility of implementing opportunistic learning on real mobile devices. For any opportunistic learning approach, there are two basic challenges: (1) discovering neighbors and exchanging data and models and (2) training lightweight models entirely on-device without a central server. While existing approaches to opportunistic and decentralized learning have been evaluated extensively theoretically and through mathematical simulation, implementation on real devices must additionally consider resource constraints, network dynamics, and other unpredictabilities that are difficult or impossible to model in simulation. However, these real-world conditions have a significant impact on the performance of the entire process.

Our novel contributions focus not on evaluating the accuracy of opportunistic learning but rather on evaluating the practical feasibility of opportunistic learning *in general* and on real devices. Therefore, we fix a learning model and focus on the two key challenges identified above. In particular, we use a basic opportunistic learning approach [3] in which a device (the *learner*) encounters another device (the *neighbor*) and asks the neighbor to perform a round of training using the neighbor's own computational resources and local data. The neighbor maintains the privacy of its data, but the learner can leverage a larger and more diverse set of data. Recognizing that different devices in a pervasive computing environment may have different, personalized learning goals, when a learner discovers a new neighbor, the learner uses information about the neighbor's local data distribution to determine whether collaborating would be beneficial. If the learner decides to request collaboration, it sends the neighbor its model weights. The neighbor performs a round of training on the learner's model using the neighbor's local data before sending updated weights back to the learner. Finally, after the learner receives the updated weights, it updates the local model.

In this paper we evaluate the feasibility of the two key components of practical opportunistic learning: (1) opportunistic neighbor discovery and model exchange and (2) on-device training. The findings in this paper are independent of the particular learning modality and apply to any approach that relies on decentralized (device-to-device) exchange of models and/or on-device training. We report on a series of experiments using 7 different iOS devices, which train a modified version of VGG-16 model [4] to perform a simple image classification task using the CIFAR-10 dataset [5]. We capture system-level metrics like latency, memory, and thermal impacts that are impossible to capture in the large scale mathematical simulations of decentralized learning approaches. In addition, we examine

the feasibility of communicating model weights using off-the-shelf device-to-device communication technologies. The approach in this paper is an exemplar feasibility study; while we use iOS devices for our testbed, the approach and its findings are extensible to other mobile OS platforms, including Android.

II. Related Work

In this section, we discuss the background and related work along the two framing challenges: (1) using device-to-device communication to exchange model summaries and (2) supporting on-device training. We also provide an overview of the opportunistic learning we use for our study.

A. Device-to-Device Model Sharing

In opportunistic learning, devices need to discover one another and then exchange information to support on-device training (e.g., summaries of models, data distributions, etc.). A variety of discovery mechanisms exist that rely on different wireless technologies. Bluetooth Low Energy (BLE) has emerged as a commonly utilized technology [6] because of its low energy consumption and ubiquitous availability. BLE-based discovery is even the approach underlying several contact tracing protocols that have emerged in recent years [7]. *Continuous neighbor discovery* approaches have also been proposed for IoT-like environments [8]–[12], largely based on precursors from wireless sensor networks [13], [14].

Once devices are mutually discovered, opportunistic learning requires them to exchange information relevant to the learning task, which may include larger volumes of data, for instance a CNN's model weights. Direct device-to-device data sharing has also been well studied in a variety of application domains [15], [16], including applications to contact tracing [17] and machine learning using edge devices [18]. These approaches use a mixture of technologies, including BLE but also extending to direct WiFi communication channels.

Platform-specific communication approaches also exist. For instance, the Multipeer Connectivity Framework[1], built on Bonjour[2], supports device-to-device communication among Apple devices, using a combination of the local area network, WiFi, and BLE. Similarly, Google Nearby Connections API[3] allows similar connections of Android devices.

B. On-Device Training

As federated learning has exploded, significant attention has been paid to on-device training of machine learning models [19], [20]. Others have recognized potential hurdles to practical implementations and developed techniques to address these. For instance, TinyTL [21] reduces the memory pressure associated with on-device training. NestDNN [22] dynamically considers resource demands and available runtime resources to tailor a model to a mobile device, while Lin et al. [23] explore a battery-aware framework for inference on mobile devices.

With this growing interest, platform developers have also been building tools to support on-device training. In particular, the following options have high visibility:

1) **TensorFlow Federated (TFF)**[4] is designed to evaluate decentralized learning with a focus on federated learning. However, it does not yet support on-device training.
2) **Tensorflow Lite**[5] optimizes for resource-constrained devices and supports on-device *inference* for Android and iOS, embedded Linux, and microcontrollers. As of the writing of this paper, Tensorflow Lite does not yet support on-device training, but this functionality is listed in the project's roadmap.
3) **PyTorch Mobile**[6] also enables on-device inference in iOS, Android, and Linux-based systems. As of the writing of this paper, it remains in beta release and does not support on-device training.
4) **OpenMined**[7] is an open-source ecosystem for federated learning. It focuses on the federated learning setting and requires a central server for coordination.
5) **Core ML**[8] is a framework for both on-device training and inference for most of devices under the Apple umbrella, including iPhones, iPads, Macs, Apple TV, and Apple Watches.

While these approaches offer potential to support an opportunistic learning platform for mobile devices, we choose Core ML for our feasibility study since, at the time of writing, it is the most fully featured and stable platform supporting on-device training.

C. Opportunistic Collaborative Learning

Our goal of this paper is to identify challenges in implementing opportunistic learning on real-world devices. For the purposes of our experiments we use a straightforward implementation of opportunistic learning based on [3], applied to a classification task. Each device has a personalized learning *goal*, captured as a set of data labels the device's model should be able to correctly classify. In addition, each device continuously collects (labeled) data based on the device's user's behaviors. Each device continuously shares an advertisement that includes a distribution of labels in the device's locally available data. When one device (the *learner*) discovers another device (the *neighbor*), it determines whether the neighbor's advertised label distribution sufficiently overlaps the learner's goal set. If it does, the learner could benefit from training on the neighbor's data. However, because the neighbor does not want to share the potentially private raw data, the learner instead shares its model (by sharing the model's weights) with the neighbor. The neighbor loads the model into memory and provides a series of mini-batch training rounds using the learner's model and the neighbor's local data. When the training rounds are complete, the neighbor returns a set of updated model weights

[1]https://developer.apple.com/documentation/multipeerconnectivity
[2]https://developer.apple.com/bonjour/
[3]https://developers.google.com/nearby/connections/overview
[4]https://www.tensorflow.org/federated
[5]https://www.tensorflow.org/lite
[6]https://pytorch.org/mobile/home
[7]https://www.openmined.org
[8]https://developer.apple.com/documentation/coreml

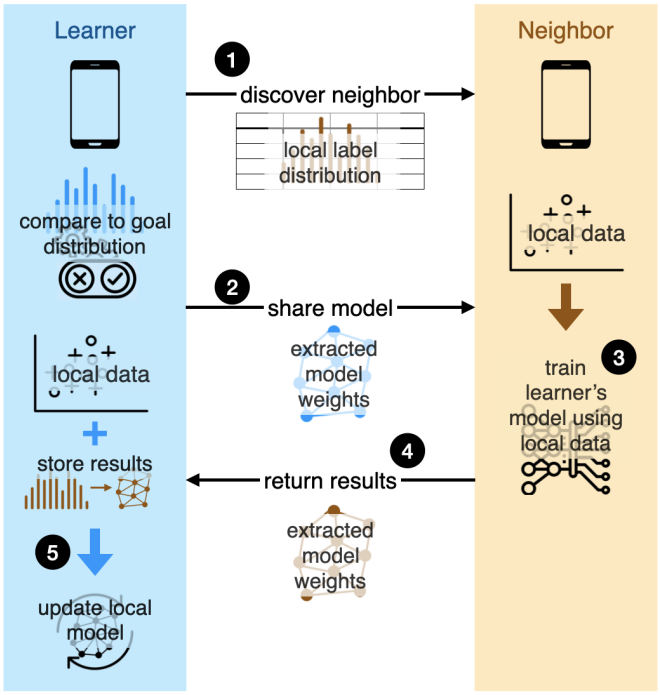


Fig. 1: The opportunistic learning process

to the learner, which the learner incorporates into its local model via a local averaging algorithm.

In this opportunistic learning approach, all of the communication (e.g., advertisement of data distributions and sharing of model weights and updated model weights) is performed opportunistically, without the assistance of any third party using only device-to-device communication. All training is performed using only resources local to the mobile devices.

III. An Opportunistic Learning Prototype

Opportunistic learning process is conceptually a modular system with two core components: (1) device-to-device model sharing and (2) on-device model training. The first of these components in turn has two functions: discovering neighbors and exchanging models. Fig. 1 shows a sequence diagram of the opportunistic learning process.

A. Device-to-device model sharing

As described previously, we use Apple's Core ML to support on-device training on Apple devices because it is the most stable and flexible option. Because of this decision, we are also motivated to use Apple's Multipeer Connectivity Framework to support device-to-device communication because it integrates well with the Apple ecosystem. The Multipeer Connectivity Framework is flexible in that it can use, in combination and depending on the underlying devices, infrastructure Wi-Fi networks, peer-to-peer Wi-Fi, and Bluetooth personal area networks for the underlying communication. In this work, we focus on localized device-to-device communication; therefore, we disable the infrastructure Wi-Fi network and force the devices to communicate through peer-to-peer Wi-Fi and Bluetooth personal area networks. In our prototype networks described in the next section, we use 7 iOS devices.

1) Discovering neighbors: When any device is participating in a opportunistic learning network, it simultaneously assumes two roles; in the terminology of the Multipeer Connectivity Framework, these roles are *advertiser* and *browser*. As an advertiser, the device continuously broadcasts its data distribution alongside its `peerid`; this functionality is implemented by the `MCNearbyServiceAdvertiser` in the Multipeer Connectivity Framework. As a browser, the `MCNearbyServiceBrowser` captures any broadcasted data distributions from neighboring devices. For instance, if the browser on one device (the *learner* in Fig. 1) receives a signal from another device (the *neighbor*) containing the neighbor's data distribution, the learner compares its goal distribution with the neighbor's data distribution. If the neighbor has data that matches the learner's goal, the learner will invite the neighbor to proceed to model exchange. Once a neighbor is discovered, a learner device proceeds to the next step: exchanging the models needed for opportunistic learning.

2) Exchanging Model Information: Using the previous example, the neighbor device will receive an invitation from the learner to establish a model exchange session. Because of practical limitations on the number of open communication sessions, if the neighbor is already connected to more than 8 devices, it rejects the invitation. Once connected, the learner will package its model and prepare for wireless model exchange. Specifically, the learner shares a Core ML-specific representation of the model weights and uses the established Multipeer Connectivity session to send the weights to the neighbor (step 2 in Fig. 1). The learner will send the model as three different files (shown in Fig. 2 and described in more detail below), using the Multipeer Connectivity Framework's `sendResourceAtURL` function. Once the neighbor receives all of the files, it checks the files to confirm that no data loss occurred during the exchange. Then the neighbor will insert the 3 files into the local Core ML Model and perform the on-device training process as described below.

Once the on-device training finishes on the neighbor device, we reverse the process to return the updated weights to the learner device. After receives the weights, it will rebuild its model and disconnect the model exchange session.

During a model exchange session, the participating devices may move out of communication range, and as a result the session will be terminated. If the session is killed at any time during the model exchange and training, the neighbor device will stop the training and both devices will release any resources allocated to this encounter.

B. On-Device Training

Core ML is the machine learning framework we choose for our system design. It offers on-device training and inference that leverages the capabilities of the low-level hardware and can improve performance by using GPUs and Apple's AI accelerator mechanisms, when available. Within our prototype's use of Core ML, we must solve two practical challenges: how

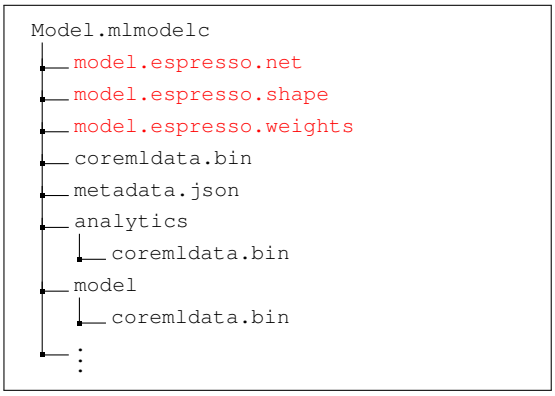


Fig. 2: File Structure of .mlmodelc

to extract a meaningful model representation that can be shared between the learner and the neighbor and how to perform training on the neighbor device.

1) Extracting Models: Core ML relies on an internal compiled model representation that is stored in a *.mlmodelc*, whose high-level structure is shown in Fig. 2. The proprietary format of the file makes it difficult to extract the model weights to share in order to enable model training on a neighbor device. However, to minimize resource consumption during model exchange we prefer to send only the portion of the *.mlmodelc* file that is relevant to the learner and the neighbor. To determine the portion that needs to be shared, we computed and compared md5 hash value for each of the internal files before, during, and after the training process and found that only `model.espresso.net`, `model.espresso.shape`, and `model.espresso.weights` change when the model is trained. Therefore, we assume these files contain the model information, including the weights, that needs to be shared between the learner and the neighbor; these are the data extracted and sent in steps 2 and 4 in Fig. 1.

2) Training Models: When the neighbor device receives the model from the learner, it loads the model into Core ML and performs the training on its own data. In the result, for each encounter, the neighbor trains 10 epochs for model with all of the data on its own. When the training process completes, the neighbor device extracts the updated model information from the *.mlmodelc* file and sends the updates back to the learner. After the learner device receives the updated weights from the neighbor, the learner rebuilds the model and can use it for inference (step 5 in Fig. 1).

IV. Feasibility of Opportunistic Learning

To measure the performance of opportunistic learning on real devices, we deployed our prototype to 7 different Apple devices. We used a customized version of VGG-16 [4] tailored to the input CIFAR-10 dataset [5] (32*32 for each image associated with a set of 10 labels) while satisfying the memory limitations of our test devices. Each device used the VGG-16 model initialized with randomized weights. Each device randomly selected 2 of the 10 labels and then received 300 different images from these 2 labels. Each device also randomly selected 2 labels as its goal distribution.

TABLE I: Devices Used (Ordered by Computational Power)

Device Type	System Version	SoC	Core ML Accelerator
iPad Pro 12.9	iOS 15.1	M1	Yes
iPhone 12 Mini	iOS 15.1	A14 Bionic	Yes
iPhone SE 2020	iOS 15.1	A13 Bionic	Yes
iPhone 8	iOS 15.1	A11 Bionic	None
iPhone 8	iOS 14.4.2	A11 Bionic	None
iPad Pro 10.5	iOS 14.6	A10X Fusion	None
iPhone 7	iOS 15.1	A10 Fusion	None

Table I shows the devices we used. The devices were arranged so that all were within communication range of each other for the entire experiment. With the exception of the iPhone SE, the devices have other applications installed (they are the everyday devices of real individuals). We closed all applications other than the facilities supporting opportunistic learning, but some background tasks were unavoidably present on the devices. On the one hand, this leads to some unpredictabilities in our results; on the other hand it simulates the "wild" environment in which we expect opportunistic learning to be deployed.

In the remainder of this section, we report on benchmark measures for the two stages in opportunistic learning: device-to-device communication and on-device model training. Across both challenges, we found that a device's *thermal state*[9] had a significant impact on the performance. Apple defines the thermal state of a device using 4 levels:

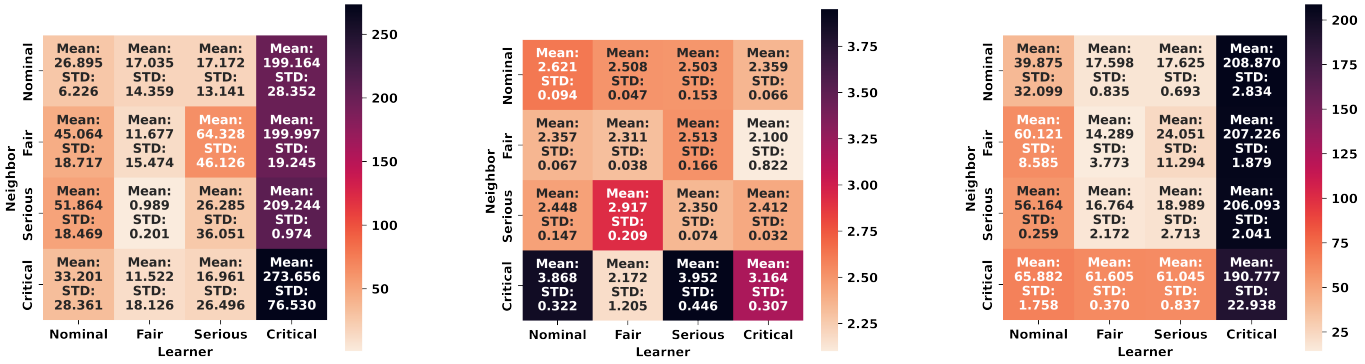
1) Nominal: The device's temperature is low. The device can finish as much work as needed.
2) Fair: The device's temperature is slightly elevated. Apple suggests developers reduce resource usage.
3) Serious: The device's temperature is high. At this level, Apple suggests developers reduce CPU/GPU usage, I/O Operation, and network and Bluetooth usage.
4) Critical: This is the highest level of device temperature. The device needs to cool down immediately. Most work will be severely impacted or even terminated.

In the below, we present the benchmarks for both communication and training in the context of a device's thermal state.

A. Communication Testing

Fig. 3 reports the average time required to perform the needed communication steps, by phase of the device-to-device communication and by the thermal state of each device. The time to discover a new neighbor is calculated as the difference between the neighbor device being online and its discovery by the learner device. The time to establish a new connection is measured as the time elapsed between when the learner sends an invitation to the neighbor and the connection is fully established. Finally, the time to exchange models is measured

[9]https://developer.apple.com/documentation/foundation/processinfo/thermalstate



(a) Discovering Neighbors (b) Establishing Connections (c) Exchanging Models

Fig. 3: Average Time Spent on Device-to-Device Model Sharing, by Phase (seconds)

as the difference between when one device begins to send a model and the second device has completely received it.

Fig. 3(a) shows that when the learner is in the more critical thermal states, the speed for discovery is serious affected. However, the thermal state of the neighbor device has limited impact on the discovery time. Fig. 3(b) shows that the devices' thermal states have little impact on the time establishing a connection. Finally, Fig. 3(c) shows that when either of the devices' thermal state becomes more critical, the time for model exchange is impacted. This is to be expected since both devices have a computational role to play in model exchange.

B. Model Training

Fig. 4 shows the training time and thermal state for each device over the experiment. Each measurement represents the time for an epoch (i.e., a single pass over the entire training dataset). The background color in each figure represents the thermal state for that device during that training round. Fig. 5 shows these same result, but in aggregate for each device.

Because newer devices' SoCs have higher efficiency than those of older devices, we observe that the older devices are much more prone to jump quickly to the "Fair" or even higher thermal level. The figures also depict a clear correlation between the model training time and a device's thermal state.

C. Insights

Our prototype and experiments have shown that it is indeed feasible to implement opportunistic learning on real world devices and device-to-device networks. However, there remain open challenges and opportunities for further innovation.

As we discussed in the related work section, there are limited frameworks that support practical implementations of on-device training. In addition, even though it is the most stable and available of the on-device training frameworks, Core ML still only supports limited layers for on-device training. In particular, for now, we can only train the traditional CNN layer and Dense layer on-device. We checked the ability to support MobileNet and MobileNetV2 as they are designed for the mobile environment, but neither of these models can be used under the Core ML framework.

Further, it is well known that mobile devices are resource constrained and their operating systems are designed to monitor the system performance and adapt on the fly as the performance deteriorates. These system level adaptations can interfere with memory and computation intensive training schemes. For the purposes of this study, we had to dramatically reduce the size of the model that we trained. In the future, the operating systems could be made more training-aware and collaborative in an on-device training process.

Finally, Fig. 4 and Fig. 5 show large differences in training times on different devices. Given these dramatic differences, and the expected continuation of device fragmentation in mobile device markets, the ability for opportunistic learning frameworks to be able to handle heterogeneous device capabilities is of paramount importance. Future work should consider adapting the model size and training intensity based on individual devices potentially dynamic training capabilities.

V. Conclusion

We demonstrated the feasibility of implementing the two key facets of opportunistic decentralized machine learning: supporting (1) localized device-to-device sharing of data that underpins local learning and (2) on-device training of real models. In the process, we identified several limitations of existing platforms and several opportunities for future contributions with significant practical impact. Our efforts demonstrate that it is essential for on-device learning schemes to directly and explicitly consider more than just hypothetical memory, computation, and bandwidth specifications of devices—one must also consider the interplay of these constraints, as well as the way the model training and communication interact with other functions on the device.

We also identified several avenues for future systems work in supporting opportunistic learning. First, clearly, platform specific solutions are limiting, and we must seek frameworks that support cross-platform collaborative learning. Second, fragmentation in the mobile device space is a very real issue, and the performance of on-device training on devices of differing capabilities varies dramatically. Opportunistic learning implementations must consider the heterogeneity of devices

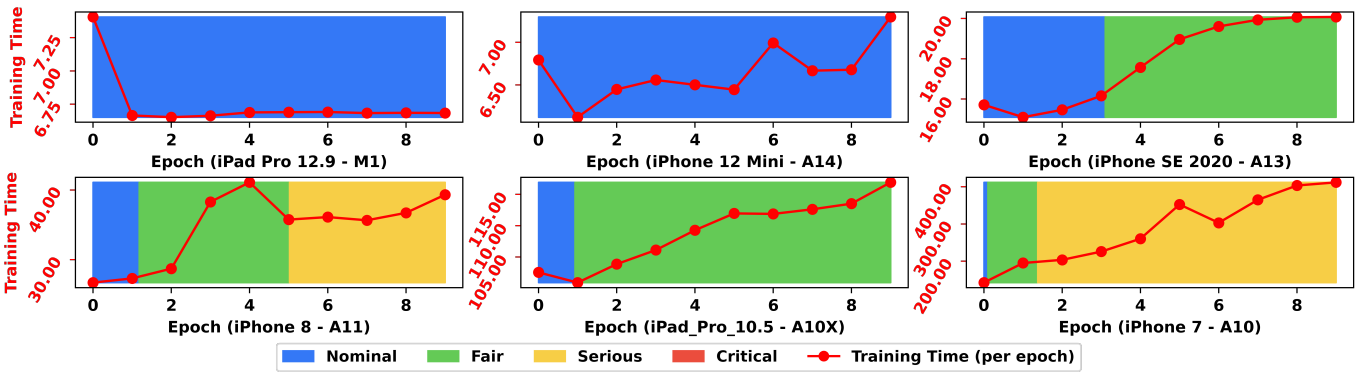


Fig. 4: Model training result

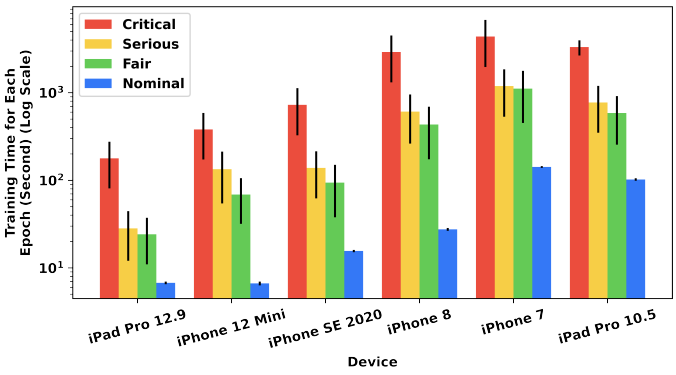


Fig. 5: Average Training Time in Different Devices Under Different Thermal State

and the networks that connect them. Ultimately, however, this work shows that real world devices are ready and able to support opportunistic decentralized learning.

Acknowledgement

This work was funded in part by the National Science Foundation grant CNS-1909221, Data61 grant CRP C020996, and Australian Research Council Project grant (ARC LP190100676). Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the NSF, Data61, and Australian Research Council.

References

[1] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[3] S. Lee, X. Zheng, J. Hua, H. Vikalo, and C. Julien, "Opportunistic federated learning: An exploration of egocentric collaboration for pervasive computing applications," in *Proc. of PerCom*. IEEE, 2021.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[6] J. Liu, C. Chen, and Y. Ma, "Modeling neighbor discovery in bluetooth low energy networks," *IEEE Communications Letters*, vol. 16, no. 9, pp. 1439–1441, 2012.

[7] N. Ahmed, R. A. Michelin, W. Xue, S. Ruj, R. Malaney, S. S. Kanhere, A. Seneviratne, W. Hu, H. Janicke, and S. K. Jha, "A survey of covid-19 contact tracing apps," *IEEE Access*, vol. 8, pp. 134 577–134 601, 2020.

[8] B.-R. Chen, S.-M. Cheng, and J.-J. Lin, "Energy-efficient BLE device discovery for Internet of Things," in *Proc. of CANDAR*, 2017, pp. 75–79.

[9] C. Drula, C. Amza, F. Rousseau, and A. Duda, "Adaptive energy conserving algorithms for neighbor discovery in opportunistic bluetooth networks," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 96–107, 2007.

[10] S. Han, Y. Park, and H. Kim, "Extending bluetooth le protocol for mutual discovery in massive and dynamic encounters," *IEEE Transactions on Mobile Computing*, vol. 18, no. 10, pp. 2344–2357, 2019.

[11] C. Julien, C. Liu, A. L. Murphy, and G. P. Picco, "BLEnd: Practical continuous neighbor discovery for bluetooth low energy," in *Proc. of IPSN*, 2017.

[12] P. H. Kindt and S. Chakraborty, "On optimal neighbor discovery," in *Proc. of SIGCOMM*, 2019, pp. 441–457.

[13] Y. Qiu, S. Li, X. Xu, and Z. Li, "Talk more listen less: Energy-efficient neighbor discovery in wireless sensor networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[14] M. Bakht, M. Trower, and R. H. Kravets, "Searchlight: Won't you be my neighbor?" in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012, pp. 185–196.

[15] M. Devos, A. Ometov, N. Mäkitalo, T. Aaltonen, S. Andreev, and Y. Koucheryavy, "D2D communications for mobile devices: Technology overview and prototype implementation," in *Proc. of ICUMT*, 2016.

[16] Q. Lin, F. Wang, and J. Xu, "Optimal task offloading scheduling for energy efficient d2d cooperative computing," *IEEE Communications Letters*, vol. 23, no. 10, pp. 1816–1820, 2019.

[17] H. Mcheick, H. H. Hassan, and H. Kteich, "D2D communication: COVID-19 contact tracing application using Wi-Fi direct," in *Proc. of SmartNets*, 2021.

[18] X. Cai, X. Mo, J. Chen, and J. Xu, "D2d-enabled data sharing for distributed machine learning at wireless network edge," *IEEE Wireless Communications Letters*, vol. 9, no. 9, pp. 1457–1461, 2020.

[19] R. Gu, C. Niu, F. Wu, G. Chen, C. Hu, C. Lyu, and Z. Wu, "From server-based to client-based machine learning: A comprehensive survey," *ACM Computing Surveys*, vol. 54, no. 1, January 2021.

[20] S. Dhar, J. Guo, J. J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Trans. on the IoT*, vol. 2, no. 3, July 2021.

[21] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce memory, not parameters for efficient on-device learning," in *Proc. of NEURIPS*, 2020.

[22] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proc. of MobiCom*, 2018.

[23] Z. Lin, Y. Gu, and S. Chakraborty, "Tuning machine-learning algorithms for battery-operated portable devices," in *Proc. of AIRS*, 2010.