FedEx: Scalable and Secure System for User-Adaptive Deep Learning Training in Mobile Devices

Ahnjae Shin*
yuyupopo@snu.ac.kr
Seoul National University

Heeseung Yun*
terry9772@snu.ac.kr
Seoul National University

Kyunggeun Lee*
initium0917@snu.ac.kr
Seoul National University

Taebum Kim*
k.taebum@snu.ac.kr
Seoul National University

ABSTRACT

All machine learning (ML) services need to retrain their models periodically to accommodate new users and use cases. Especially with a great amount of personal data generated on the application's behalf, using personal data to continuously enhance a ML model could greatly benefit app retention rate. However, previous frameworks lack scalability or security in training deep learning models with personal user data. To address this problem, we propose FedEx, a mobile-cloud hybrid system to utilize both mobile and cloud resources in a scalable and secure manner. FedEx applies transfer learning to train models with additional data. Our system is scalable by removing mobile devices out of training loop, and secure as it mitigates the security risk of sending raw data. Experimental results suggest that FedEx can be adapted for models with a variety of capacity, as well as provide more energy-efficient and faster training process compared to representative baselines.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computing methodologies** → *Distributed artificial intelligence.*

ACM Reference Format:
Ahnjae Shin, Heeseung Yun, Kyunggeun Lee, and Taebum Kim. 2020. FedEx: Scalable and Secure System for User-Adaptive Deep Learning Training in Mobile Devices. In *Proceedings of 21st International Workshop on Mobile Computing Systems and Applications (HotMobile'20)*. ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.475/123_4

*Equal contribution

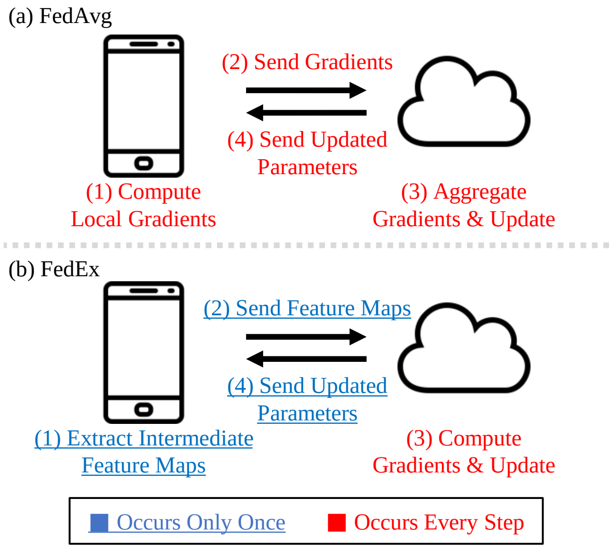
(a) FedAvg
(2) Send Gradients
(4) Send Updated Parameters
(1) Compute Local Gradients
(3) Aggregate Gradients & Update
(b) FedEx
(2) Send Feature Maps
(4) Send Updated Parameters
(1) Extract Intermediate Feature Maps
(3) Compute Gradients & Update
🟦 Occurs Only Once
🟥 Occurs Every Step

Figure 1: Comparison between FedAvg [17] and FedEx. In FedAvg, both mobile and cloud resources are used to train a model. Communication of model gradients occurs every step. Meanwhile in FedEx, only cloud is utilized, thus saving mobile resource.

1 INTRODUCTION

As smartphones have become prevalent in the past decade and are producing a greater amount of personal data than ever, there is a great chance of improving user experiences using personal data in mobile devices. In the field of machine learning, applications featuring AI services have a high potential of improving service quality using personal data from mobile devices. Mobile applications have the opportunity to collect additional training data in various ways such as user feedback or correction. These data can be used to further train the ML model. However, utilizing personal data causes privacy concerns, especially in the field of deep learning where personal data should often be sent to the cloud servers for post-training.

To guarantee privacy, some previous studies [17, 21] proposed a group training algorithms, termed *Federated Learning* in [17], that utilize personal data in mobile devices for training without collecting the personal data to the central server. In federated learning algorithms, mobile devices compute local gradients, sends the local gradients to the central server to update the global parameters, and pulls back the updated model parameters from the server. Federated learning preserves privacy by sending no personal data to the central server in the training process.

However, federated learning gravely degrades training throughput because, by its very nature, federated learning is analogous to a special form of distributed deep learning training where each mobile device serves a worker process and the central server serves as a parameter server [14]. That is, federated training not only incurs communication overhead for parameter synchronization every training step as is the case with distributed deep learning training, it also suffers from low computation power of mobile devices compared to the high-end server-side GPUs.

Therefore, in this paper, we introduce FedEx, a new training algorithm that minimizes training throughput loss while providing a similar level of privacy of federated learning. FedEx is designed for transfer training where the early layers are frozen and only the last layers are trained using the personal data. As shown in figure 1, in FedEx, instead of pushing local gradients to and pulling updated model parameters from the server, local devices send intermediate feature maps to the server, which are the outputs of the frozen layers. The rest of the training process is done in the central server, and the updated model parameters are pushed back to the local devices after the server finishes post-training. The whole training process incurs only constant communication overhead and only a constant amount of computation is done in mobile devices with respect to the number of training steps.

The main contributions of this paper are as follows:

(1) We proposed a scalable, energy-efficient end-to-end framework for mobile user-adaptive training of deep learning models while providing a certain level of privacy.
(2) We verified the effectiveness of FedEx using a sample mobile face recognition application by improving the inference accuracy with nearly optimal training throughput.

The rest of this paper is organized as follows: In Section 2, we explain the design of FedEx in more detail. In Section 4 and Section 5, we perform experiments on the training throughput, accuracy, and energy consumption of FedEx and discuss the results. In Section 6, we introduce related works, and in Section 7, we discuss the future works integrating some orthogonal approaches and more challenging scenarios.

2 APPROACH

To ensure scalability, mobile devices should be not or least involved in training a ML model. Especially recent ML models have an overwhelming number of parameters, which commercial devices can't handle. However, to ensure privacy user's raw data should not be sent to the cloud. To ensure both properties, FedEx exploits transfer learning[18] where early layers are frozen and only the remaining layers are trained. Transfer learning is a common scenario in training a model with a relatively small amount of data. The model is originally trained with a large set of data, and fine-tuned against the dataset of interest with early layers frozen. We observe that to train a ML model in the transfer learning paradigm, only the output of the frozen layer is sufficient to train the model. FedEx stores intermediate features at ML inference, and pushes the intermediate features and labels to the cloud for training.

Compared to *FedAvg*, a federated learning algorithm proposed in previous work [17], the source of performance gain of FedEx is described in Figure 1. In every training step of FedAvg, (2a-1) the mobile devices compute local gradients and (2a-2) sends the local gradients to the cloud server. Then, (2a-3) the cloud server aggregates the local gradients from many devices, updates the model parameters, and (2a-4) sends back the updated model parameters to the local devices. Since all four steps must be performed every training step, the computation in mobile devices and communication between the mobile devices and the server are repeated proportionally to the number of training steps. In other words, the amount of computation performed by mobile devices and the amount of gradient/parameter communication between the local devices and the server is $O(N)$, where N is the number of training steps.

In contrast, FedEx performs bottleneck operations only constant times. In FedEx, (2b-1) mobile devices compute and cache intermediate feature maps in the service time, not training time. Then, (2b-2) the feature maps along with the labels provided by the user feedback are sent to the server. (2b-3) All the computation and communication during training are performed inside the server, and (2b-4) the updated parameters are sent back to the mobile devices after the training process. Since the bottleneck operations (2b-1, 3, and 4) are out of the actual training process, the amount of computation performed by mobile devices and the amount

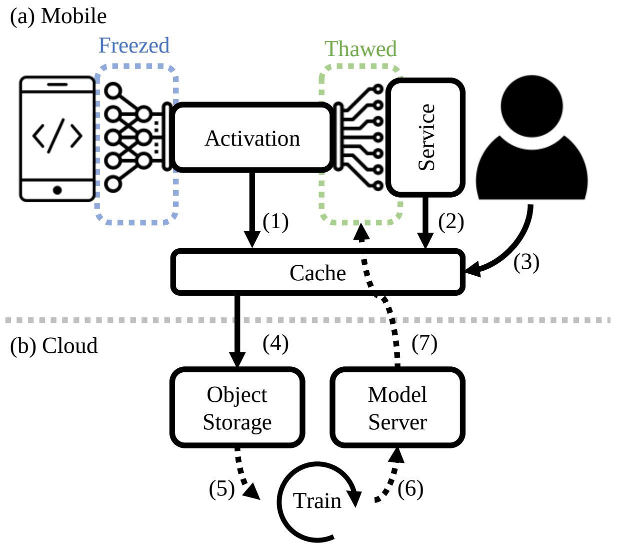

Figure 2: An overview of the FedEx architecture.

of gradient/parameter communication between the local devices and the server is $O(1)$ with respect to the number of training steps.

3 FEDEX ARCHITECTURE

In this section, we introduce our system FedEx. The training process of FedEx is described in Figure 2.

First, while serving the user's inference requests, the application stores to the internal database (1) the intermediate feature maps, (2) the inference results, and (3) the feedback from the user. Here, the feedback from the user may include whether the inference result was correct, and a new label corrected by the user if the result was wrong.

Then, when the mobile device is available (e.g. few active processes, no interaction with the user, and/or external power supply), (4) the application pushes the intermediate feature maps cached in the internal DB to the remote server. The server (5) retrains the thawed layers using the intermediate feature maps collected from many local devices, and (6) saves the updated model parameters to be pulled back to the local devices when the training is finished. Finally, when the mobile device is available, (7) the application requests the updated model parameters from the server and serves the further inference requests from the user with the improved model.

The object storage is responsible for aggregating received intermediate features, and trigger the main scheduling loop. The model server keeps track of most recently trained models in tf-lite checkpoint format. Also, it keeps track of the model that had the best validation accuracy so far. When a mobile device requests a *pull*, the model server responses with the best performing model. The scheduling loop runs several training loops in parallel. Each training loop is given a single GPU device. The scheduler only spawns new training loops if the object storage is updated.

4 EXPERIMENTS

Client Application. We implement an Android Face Recognition application as Figure 3. Using the application, the user can infer who the person is, provide feedback for the inference result, push accumulated feedback with respective intermediate features to the cloud and pull updated model parameters from the cloud. For simplicity, we only focus on face recognition problem instead of face detection problem. We use the same model as FaceNet [20] for inference, which used InceptionResNet-V1 [22] as backbone. Our pretrained model reached 92.77% validation accuracy for 8631 classes of VGGFace2 [4] dataset. Then, we deployed the model using TensorFlow Lite [1] to the android device.
Cloud Server. Our cloud server is equipped with two 18-core Intel Xeon E5-2695 @2.10 GHz, and 6 NVIDIA TITAN Xp GPU cards. It communicates with the application via HTTP request. Our source code for both client application and cloud server are made public in GitHub[1].
Baselines. We chose two different baselines: on-device model personalization [2] and FedAvg [17]. At the comparison, we did not concern the model accuracy since our technique does not modify any training configurations. On-device baseline model uses MobileNetV2 [19] as a backbone to perform a simple 4-label classification task, which contains significantly less parameter to tune than our backbone model. After sufficient amount of data is collected, only the model head is trained on-device while the rest of the model is frozen. For a fair comparison, we focus on analyzing energy consumption characteristics of the on-device baseline and FedEx. On the other hand, FedAvg utilizes preprocessed data in the cloud to train a multi-layer perceptron (MLP) for MNIST task. To the best of our knowledge, there is no implementation of FedAvg available for deep neural networks, thus we inevitably simulate its behavior except the training part to compare the latency.

To explore feasibility of adapting larger models for FedEx, we additionally compare the backbone model with varying degree of thawed layers by dividing InceptionResNet-V1 into 6 chunks, namely Freeze 1–6. Freeze 1 model is the closest model to the original model which has almost all trainable parameters. Freeze 6 model is the most lightweight, whose trainable parameters are only the last classifier layer of InceptionResNet-V1. Table 1 shows training throughput

[1]https://github.com/Mobile-and-Ubiquitous-Computing-2020-1/team1
[2]https://github.com/tensorflow/examples/blob/master/lite/examples/model_personalization

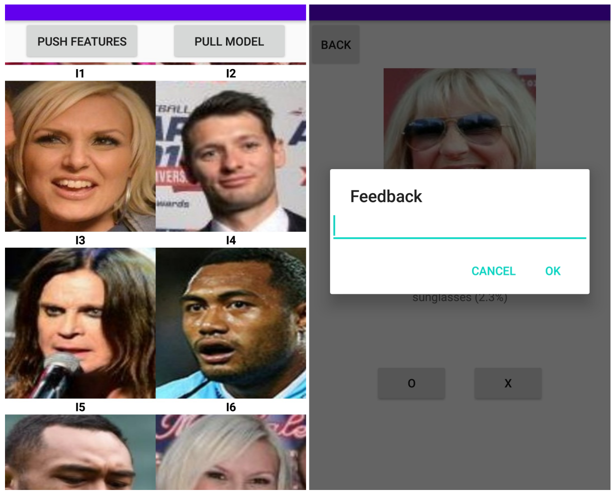

Figure 3: User interface of image gallery (left) and face recognition (right).

in our server using single Titan Xp GPU with batch size as 90.

Evaluation Metrics. We evaluate FedEx in terms of three criteria: accuracy, latency and energy consumption. We follow the accuracy metric of [20] by evaluating sparse categorical accuracy. To evaluate latency of the framework, we measure throughput as well as estimated time for sending the features, receiving updated parameters and updating the model in the cloud if applicable. Finally, we evaluate total energy consumption using Android Studio Profiler and smartphone battery monitoring. Note that the energy consumption is reported using software tools instead of precise hardware equipment.

5 RESULTS

Model	Throughput [images / sec]	# Params	Feature
Full Model	487.56	27.91M	$(160 \times 160 \times 3)$
Freeze 1	573.36	27.88M	$(38 \times 38 \times 64)$
Freeze 2	699.13	27.29M	$(17 \times 17 \times 256)$
Freeze 3	963.26	26.9M	$(17 \times 17 \times 256)$
Freeze 4	2963.88	18.3M	$(8 \times 8 \times 896)$
Freeze 5	28078.46	5.35M	(1792)
Freeze 6	41869.68	4.43M	(512)

Table 1: Training throughput of server side with varying frozen layers. Feature denotes the shape of intermediate feature tensor. Note that this throughput does not contain data-load time.

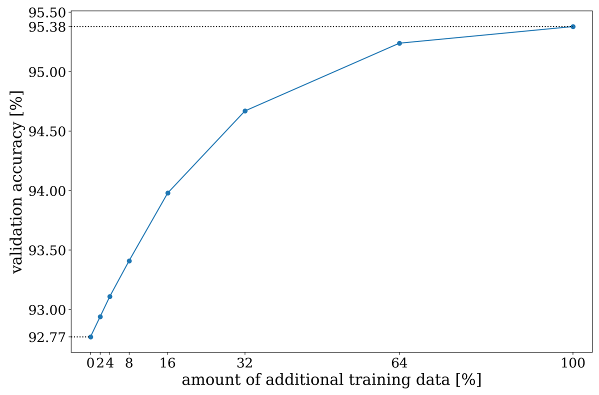

Figure 4: Validation accuracy after additional training. 100% data means using the entire prepared intermediate features (437,855) and 0% is pretrained accuracy (no additional training).

	Model	latency (ms)
Push	Base (Freeze 6)	**1525.408**
	Freeze 5	5487.633
	Freeze 4	281920.3
	Freeze 3	375950.0
	Freeze 2	375950.0
	Freeze 1	427250.4
	FedAvg	**36111.52**
Pull		**25986.07**

Table 2: Latency of push & pull operations,

Accuracy. Since one of our main contribution is that we can make user-adaptive model by additional training at the cloud, we evaluate how much accuracy can further increase. We prepared randomly sampled 437,855 data from original dataset for additional training because it is impossible to collect bunch of real-world additional data in short period. Figure 4 is our experiments result of additional training for Freeze 6 model. It shows that if we use more data, the validation accuracy increases and even 2% of additional data (8757 feature samples) could increase accuracy. The result seems quite natural because the model can see more data which the model haven't seen before. Our additional training runs for 10 epochs, but the number of epochs does not matter since 10 epochs training with 100% additional data could be done within 4 minutes in our server.

Latency. We evaluate the latency of pushing intermediate features to the cloud and pulling the updated model back to the client. As a typical usage scenario, we assume that there are 10 face recognition feedback data in the client, which is an acceptable amount of average daily selfies. That is, for each variants of FedEx, the client is sending 10 feature vectors

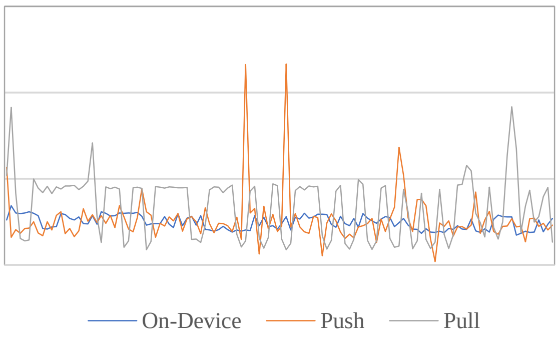

Figure 5: Energy profiling result using Android Studio Profiler, where each data point indicates 0.2 second in time.

with varying shapes (1) to the server. To imitate the behavior or FedAvg, gradients extracted from the model are likewise pushed to the cloud. Then, the client pulls updated model parameters back, where the size of data is the same for all experiments due to identical backbone. The result in Figure 2 suggests that FedEx with Freeze 6 backbone outperforms FedAvg in terms of latency, pushing the features within 1.5 seconds on average. Latency for pushing the features does increase with respect to the size of data. But considering FedEx updates the model parameters while the client is idle (*e.g.* when the user is sleeping), the latency for data tranfer can be deemed less significant. To add on, if we further compress the data before transaction, latency is going to decrease for all experiments.

Energy Consumption. Figure 5 shows energy profiling result of pull & push operation of base model (*i.e.* Freeze 6) and on-device training baseline. Although the Android Profiler only reports relative value, battery consumption evaluation on actual smartphone device (Galaxy S7) can be taken into account for rough estimation, which returned approximately 7.9mAh per minutes. In pushing operation, there are a few spikes in battery consumption, mostly attributed to network utilization. On the other hand, most of the energy consumption during pull operation is originated from cpu utilization, presumably due to loading the updated model parameters. Although the on-device baseline displays less energy consumption than the other two operations, on-device training should persist for a longer period of time, likely way more than 4 minutes reported for FedEx training of base model, considering the inefficiency of naïve on-device training approach [6].

6 RELATED WORK

On-Device Learning. On-device learning of deep learning model is highly constrained under system resource specifications of the devices [10]. [6] claims that on-device can cost up to two orders of magnitude greater response time and energy compared to cloud-based inference. For efficient on-device learning, existing researches concentrate on memory-level optimization [11, 16] or model distillation [5, 15].

[11] suggests neural weight virtualization with context-switching to fit DNNs in memory for faster execution and energy efficiency. DeepEye [16] interleaves computation-heavy convolution layers with the loading of memory-heavy fully-connected layers to alleviate memory issues. AdaDeep [15] automatically selects a combination of compression techniques for a given DNN to fit in the mobile platform under resource constraint. NestDNN [5] adaptively selects pruned deep learning model by taking resource-accuracy trade-offs under the dynamics of runtime resources into account. Compared to FedEx which alleviates local resource constraints with the assistance of cloud, on-device learning is limited in model scalability. Nontrivial pressure in terms of energy, memory, and computation restricts on-device learning in smaller models.

Federated Learning. Federated learning is a learning paradigm that trains an algorithm over a number of edge devices without explicitly exchanging data samples, whose representative application scenario is mobile keyboard prediction in mobile devices [7]. We refer the readers to [25] as a primer for concepts and applications of a federated learning framework. FedAvg [17] aggregates local gradients sent from individual devices and updates gradient for each communication round. To ensure the security of FedAvg, [3] suggests the secure aggregation strategy in which only aggregated answers of the users are visible to the server. SCAFFOLD [9] proposes a client-drift correcting strategy for FedAvg to effectively reduce the number of communication rounds without being effected by data heterogeneity or client sampling. Applying federated learning to larger deep learning models requires nontrivial computation and network resources, so most of the previous researches limited their scopes to smaller scenarios like training multi-layer perceptrons (MLPs) for MNIST dataset [17]. Also, as claimed by [2], federated learning is more susceptible to model poisoning attack and model replacement backdoor than conventional training scheme with training data.

Cloud Learning. Cloud learning utilizes cloud instead of local resources to train the model. Besides the trade-off between computation and network usage, one of the major concerns of cloud learning is to prevent security breaches when the user data is uploaded to the cloud. [8] suggests training neural networks using encrypted data and returning encrypted predictions to devices. Occlumency [12] ensures the confidentiality of the data with minimal latency by leveraging the secure Software Guard eXtension (SGX) enclave. To mitigate the performance degradation issue of privacy-preserving cloud learning, Arden [23] conjoins noisy training and strong data perturbation for fast yet accurate learning. GRACE [24] tackles network bandwidth bottleneck of IoT devices by applying deep neural network-aware compression algorithm without disturbing the performance. Contrary to existing approaches, which require sophisticated scheme to ensure the security of user data, FedEx leverages intermediate feature tensor to ensure a certain level of security without additional overhead.

7 DISCUSSION

Privacy Concerns. Leveraging external sources of computation incurs privacy concerns, and there are varying degrees of the attack surface. Naïve cloud learning involving raw data transmission is highly susceptible to major security breaches from hijack attack to model poisoning. FedEx, which transfers intermediate features to the cloud, is relatively secure from data hijacking. Though chances are slim, there still is a chance of reverse-engineering the features to reconstruct original data. Federated learning frameworks utilize a more secure form of data (*i.e.* gradients), but they are still prone to another type of attack like model poisoning [2]. As discussed in Section 6, methodologies to ensure a certain level of security against privacy concerns are actively discussed. For example, to reinforce privacy protection of FedEx data transaction, it is possible to reconcile intermediate feature anonymizer like TIPRDC [13] for improved security against the aforementioned breaches.

Multi-Client Scenarios. Under current use cases, the models are updated while the client is in the idle state, which results in a weaker timing constraint. Though we have only conducted experiments tackling single client scenarios, FedEx can be readily extended for multi-client usage cases by utilizing existing solutions like scaling up the cloud or simple load scheduling. Our pilot test involving a few devices returned no significant issues, but a thorough experiment with respect to the scalability of FedEx should be conducted. Additionally, multi-client learning scenarios can enable synergistic effects among users by applying strategies like adaptive model sharing, but we leave this to future work.

8 CONCLUSION

FedEx is a mobile-cloud hybrid system to fine-tune deployed deep learning models with personal user data in a scalable and secure way. FedEx stores the intermediate feature maps while serving user requests, and send them to the server for post-training. Our experiments show that FedEx achieves higher training throughput and lower energy consumption while improving accuracy by post-training, compared to the federated learning or on-device training algorithms.

9 ACKNOWLEDGEMENT

This research is conducted in the Mobile and Ubiquitous Computing course, 2020 spring. We thank the course staff for guidance on the project.

REFERENCES

[1] [n.d.]. TensorFlow Lite: ML for Mobile and Edge Devices. https://www.tensorflow.org/lite

[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2018. How to backdoor federated learning. *arXiv preprint arXiv:1807.00459* (2018).

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards Federated Learning at Scale: System Design. *arXiv preprint arXiv:1902.01046* (2019).

[4] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. 2018. VGGFace2: A Dataset for Recognising Faces across Pose and Age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, 67–74.

[5] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 115–127.

[6] Tian Guo. 2018. Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 184–190.

[7] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated Learning for Mobile Keyboard Prediction. *arXiv preprint arXiv:1811.03604* (2018).

[8] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Catherine Jones. 2017. Privacy-preserving machine learning in cloud. In *Proceedings of the 2017 on Cloud Computing Security Workshop*. 39–43.

[9] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. 2019. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. *arXiv preprint arXiv:1910.06378* (2019).

[10] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.

[11] Seulki Lee and Shahriar Nirjon. 2020. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *Proceedings of the 18th Annual International Conference on Mobile Systems, Applications, and Services*. 389–400.

[12] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. 2019. Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–17.

[13] Ang Li, Yixiao Duan, Huanrui Yang, Yiran Chen, and Jianlei Yang. 2020. TIPRDC: Task-Independent Privacy-Respecting Data Crowdsourcing Framework with Anonymized Intermediate Representations. *arXiv preprint arXiv:2005.11480* (2020).

[14] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Broomfield, CO) *(OSDI'14)*. USENIX Association, USA, 583–598.

[15] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 389–400.

[16] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. 68–81.

[17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Artificial Intelligence and Statistics*. 1273–1282.

[18] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Trans. on Knowl. and Data Eng.* 22, 10 (Oct. 2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[20] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 815–823. https://doi.org/10.1109/CVPR.2015.7298682

[21] R. Shokri and V. Shmatikov. 2015. Privacy-preserving deep learning. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 909–910.

[22] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, Satinder P. Singh and Shaul Markovitch (Eds.). AAAI Press, 4278–4284. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806

[23] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. 2018. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2407–2416.

[24] Xiufeng Xie and Kyu-Han Kim. 2019. Source Compression with Bounded DNN Perception Loss for IoT Edge Computer Vision. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[25] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.