OPTIMIST: Opportunistic Federated Transfer Learning for Mobile Devices Using Sequential Independent Subnetwork Training

Victor Romero II *§, Tomokazu Matsui *†, Yuki Matsuda ‡*†, Hirohiko Suwa *† and Keiichi Yasumoto *†
* Nara Institute of Science and Technology, Ikoma, Nara, Japan
† RIKEN Center for Advanced Intelligence Project AIP, Chuo, Tokyo 103-0027, Japan
‡Okayama University, Okayama, Okayama, Japan
§University of the Philippines Tacloban College, Tacloban City, Philippines
Emails: {romero.victor_militante.rs1, m.tomokazu, h-suwa, yasumoto}@is.naist.jp, yukimat@okayama-u.ac.jp

***Abstract*—This paper introduces OPTIMIST, a federated transfer learning framework designed for opportunistic settings, where stable networks are unavailable and centralized servers are impractical. OPTIMIST leverages pre-trained backbones and enables the collaborative training of classifier heads by resource-constrained, sensor-rich edge devices. In the absence of centralized orchestration, model parameters are exchanged between mobile nodes over transient connections. To avoid catastrophic interference resulting from varying node mobility and encounter patterns, OPTIMIST adapts a conflict-free strategy that splits the classifier head into smaller non-overlapping subnetworks. These subnetworks are trained locally, exchanged during encounters for incremental refinement, and seamlessly reassembled into a complete classifier head. We implemented a prototype for Android devices to evaluate its practicality in cross-device scenarios. Our experiments demonstrate its effectiveness in varying degrees of data heterogeneity. Additionally, under assumptions of relatively sparse data sets on edge devices, batch training time on the order of seconds supports the feasibility of this technique in real-world environments.**

***Index Terms*—opportunistic networks, federated learning, mobile computing**

I. Introduction

The growing ubiquity of sensor-rich edge devices has spurred interest in deploying machine learning (ML) models for a wide array of pervasive applications. However, many of these applications must operate in environments where traditional centralized data aggregation and processing requirements are impractical. These include rural smart agriculture [1], wildlife conservation [2], and response systems in disaster-struck areas [3]. The lack of requisite infrastructure in these environments limits the effective use of data from sensor-rich devices, necessitating alternative approaches.

Federated learning (FL) [4] enables collaborative model training and distributes computational workload among participants, fostering a democratic training process. Unlike centralized approaches that require powerful servers to collect and process data, cross-device FL allows resource-limited edge devices to jointly build high quality models. However, traditional FL relies on frequent communication with a central orchestrator for model aggregation, which is infeasible in environments characterized by opportunistic connectivity and dynamic node mobility. In such cases, opportunistic federated learning (OppFL) [5]–[8] adapts FL protocols to accommodate sporadic communications and lack of end-to-end connectivity between nodes.

While OppFL addresses the challenge of distributed training, producing high-quality models remains difficult when individual devices face computational constraints for training complex models. Transfer learning (TL) [9] offers a complementary solution to this challenge. Feature-based transfer learning, in particular, leverages backbone models that have been pre-trained on extensive datasets to extract features that can be used for diverse downstream tasks. This approach reduces the computational burden of local training, thereby improving the feasibility of deploying FL-based solutions in resource-constrained environments.

Motivated by the inherent complementarity between feature-based transfer learning and opportunistic federated learning, this study explores the practical feasibility of combining them to realize a robust framework for deploying machine learning solutions in challenging, infrastructure-limited scenarios. This study builds on recent work by [8], which trains neural networks in opportunistic settings by dividing the classifier head into multiple non-overlapping subnetworks. Each subnetwork is uniquely assigned to a node, locally trained, and exchanged during transient encounters. This approach facilitates efficient adaptation of the backbone model to downstream tasks by reducing model size and synchronization needs, while also preventing aggregation conflicts from parameter overlap.

Contributions: Our main contributions are as follows:

1) We develop a prototype infrastructure for creating and training transfer learning models in a conflict-free manner on real-world Android-based devices;
2) We conduct various simulations using mobile-ready models, exploring different levels of data heterogeneity within controlled and realistic scenarios;
3) We test the proposed system on a diverse set of Android devices with varying system specification.

Study was supported by JSPS KAKENHI JP21H03431 & JP19H05665.

II. Related Work

In this section, we discuss opportunistic federated learning, its key challenges, and the state of the art. We also overview transfer learning and its categories to refine and narrow the scope of this work. Finally, we contextualize the significance of our study within this landscape.

A. Opportunistic Federated Learning

The prospect of a more democratic training process has inspired numerous studies on federated learning in scenarios where participants are present but lack traditional networking infrastructure [5]–[8], [10]. In such contexts, participants use available communication interfaces to obtain model updates from nearby nodes during transient encounters. Unlike conventional decentralized federated learning [11], which addresses the absence of an orchestrator by enabling successive rounds of communication to agree on a common model state [12], [13], opportunistic settings face challenges from parameter overlap and dynamic node mobility, complicating these protocols. As a result, opportunistic federated learning often adopts egocentric or personalized approaches. Lee et al. [5] tackle training personalized models for a participant's target task, defined by a subset of class labels. Local updates occur during encounters, with nodes negotiating collaboration by exchanging label distributions and predicting encounter duration. To address data distribution skew, they employ opportunistic momentum, stabilizing training at the cost of additional storage for previous gradients. Tomita et al. [7] focus on reducing model exchanges during pairwise collaborations in the training of a tourism object detection system. Utilizing a regressor, they predict the resulting accuracy of pairwise aggregation and proceed only if the predicted accuracy meets a predetermined threshold. Ochiai et al. [6] propose the WAFL framework, which adapts federated learning for mobile ad-hoc networks. Unlike prior approaches, WAFL aims to train a generalized model without central orchestration. The authors assume that nodes can transmit and receive model updates to all neighboring nodes, a critical factor in mitigating model divergence; however, achieving this under opportunistic conditions presents challenges. Suzuki et al. [10] introduce a framework designed to enable artificial intelligence applications in extraterrestrial networks, assuming a network of spacecraft that participate in training using data from sensors and telemetry. Given the sparse encounters between spacecrafts, the authors utilize store-carry-and-forward protocols to transmit model updates to Earth observation centers for aggregation. Lastly, Romero et al. [8] present an alternative approach to training neural networks in opportunistic settings. Rather than training the model in full, their method employs independent subnetwork training [14] and sequential federated learning [15]. The neural network is divided into non-overlapping subnetworks, which are assigned, trained, and exchanged among neighboring nodes. This strategy facilitates parallel, conflict-free progress and enhances participation due to the smaller sizes of the subnetworks.

B. Feature-based Transfer Learning

Transfer learning is categorized into four approaches based on the solution type [9]: instance-based, reweighting or modifying source instances; feature-based, utilizing extracted features from pre-trained models; parameter-based, fine-tuning model parameters; and relation-based, transferring knowledge of data point relationships. This study focuses on feature-based approaches, leveraging pre-trained backbone models trained on large, diverse datasets. Such models enable solutions in scenarios where heavy data or computational demands hinder training. For instance, [16] develops a multi-task model for disaster-related tasks using different pre-trained models, while [17] employs a ResNet50 pre-trained on CT image datasets to guide COVID-19 classifier training. In opportunistic federated learning, leveraging pre-trained backbones for computing intermediate features sidesteps the computational expense of training feature extractors [18].

C. Position of this Study

Building on the efficacy of nonoverlapping subnetworks and sequential federated learning for training neural networks in opportunistic environments, this study integrates these methods with pre-trained backbone networks as feature extractors to enable opportunistic federated transfer learning. While prior work [19] highlighted the potential of this approach in transfer learning applications, this study advances the infrastructure for real-world deployment on Android-based devices.

III. OPTIMIST: OPportunistic federated Transfer learnIng for Mobile devices using sequential Independent Subnetwork Training

A. Preliminaries

Consider a pre-trained model with parameters θ, which can be decomposed into two parts: the feature extractor, denoted as θ_{feat}, and the classifier head, denoted as θ_{cls}. In the context of federated transfer learning, θ_{feat} is fixed, while the objective is to optimize θ_{cls} collaboratively across multiple nodes.

Let $C = \{c_0, c_1, \ldots, c_{N-1}\}$ represent the set of N participating nodes. Each node c_i possesses its own dataset $\mathcal{D}_i$, and for any pair of nodes $c_i, c_j \in C$, where $i \neq j$, $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$. The virtually merged dataset is denoted as $\mathcal{D} = \bigcup_{i=0}^{N-1} \mathcal{D}_i$.

The federated transfer learning process aims to find the optimal parameterization of the classifier head, θ^*_{cls}, that minimizes the loss function $\mathcal{L}(\theta_{\text{cls}})$ over the virtually merged dataset $\mathcal{D}$:

$$\theta^*_{\text{cls}} = \arg\min_{\theta_{\text{cls}}} \left(\mathcal{L}(\theta_{\text{cls}}) = \frac{1}{N} \sum_{i=0}^{N-1} \ell(\theta_{\text{cls}}, \mathcal{D}_i) \right) \quad (1)$$

Here, $\ell(\theta_{\text{cls}}, \mathcal{D}_i)$ represents the loss incurred by the classifier head with parameters θ_{cls} on the local dataset $\mathcal{D}_i$, defined as:

$$\ell(\theta_{\text{cls}}, \mathcal{D}_i) = \frac{1}{|\mathcal{D}_i|} \sum_{(x,y)\in\mathcal{D}_i} \ell\left(y, f_{\text{cls}}(f_{\text{feat}}(x; \theta_{\text{feat}}); \theta_{\text{cls}})\right) \quad (2)$$

where $f_{\text{feat}}(x; \theta_{\text{feat}})$ denotes the output of the fixed feature extractor for input x, and $f_{\text{cls}}(\cdot; \theta_{\text{cls}})$ represents the classifier head's output, parametrized by θ_{cls}.

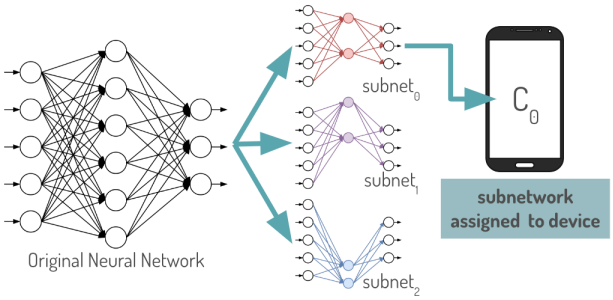

Fig. 1: Extraction of multiple subnetworks for $K = 3$: input and output neurons are shared across all subnetworks.

B. Generating the Classifier Head Subnetworks

In this work, the classifier head is modeled as a fully connected neural network with a single hidden layer. The number of neurons in the hidden layer is denoted by n_h, while n_{in} and n_{out} represent the dimensions of the input and output layers, respectively. The input layer size is determined by the feature dimensions extracted from the pre-trained backbone, while the output layer size is adapted to the requirements of the target task. To construct non-overlapping subnetworks, n_h neurons in the hidden layer are randomly partitioned into K distinct groups (here, $K \ll n_h$) as described in [14]. The k-th subnetwork is then formed by connecting the input and output neurons of the original classifier head to the corresponding partitioned group of hidden neurons. In effect, each subnetwork functions as a weaker version of the full classifier head. Fig. 1 presents an example illustrating the process of splitting a neural network into three non-overlapping subnetworks.

C. Sequential Training of Independent Subnetworks

For clarity, we divide the training phase into T discrete time steps. At each time step $t \in \{0, 1, ..., T-1\}$, each node performs local updates on its assigned subnetwork for E epochs. Let $\hat{\theta}^t_{i,e}$ denote the weights of the subnetwork assigned to node i after $e \in \{0, 1, ..., E\}$ local steps in the t-th time step. Using Stochastic Gradient Descent (SGD) as the local optimizer, subnetwork weights are updated as follows:

$$\hat{\theta}^t_{i,e+1} = \hat{\theta}^t_{i,e} - \eta \nabla \ell(\hat{\theta}^t_{i,e}, \mathcal{D}_i) \tag{3}$$

where $\nabla \ell(\hat{\theta}^t_{i,e}, \mathcal{D}_i)$ refers to the gradient of the loss of the current subnetwork parameters on the local dataset $\mathcal{D}_i$ and η is the learning rate. The final updated weights for the current time step $\hat{\theta}^t_{i,E}$ are then integrated into the node's copy of the full model following a coordinate-wise copy operation.

Nodes leverage opportunistic encounters to exchange models with the goal of incrementally exposing each subnetwork to more local datasets and integrating additional subnetworks into their full model. Let $\mathcal{N}^t_i$ denote the neighborhood of node c_i at time t. Here, we assume neighbor relationships are symmetric, such that $\forall c_i, c_j \in C : c_j \in \mathcal{N}^t_i \iff c_i \in \mathcal{N}^t_j$. Consider the scenario shown in Fig. 2 where at time $t = 1$, c_1 only has c_0 as its neighbor. After model exchange, at time $t = 2$, c_1 takes custody of c_0's subnetwork, and vice versa, such that $\hat{\theta}^2_{1,0} \leftarrow \hat{\theta}^1_{0,E}$ and $\hat{\theta}^2_{0,0} \leftarrow \hat{\theta}^1_{1,E}$. Local training is then conducted using the newly received subnetworks. When $|\mathcal{N}^t_i| = 0$, c_i does not update its subnetwork to prevent overfitting.

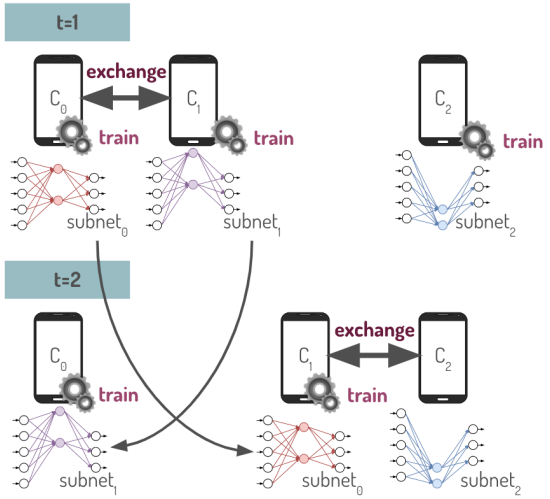

Fig. 2: Subnetwork exchange and sequential training: Nodes train their assigned subnetworks and exchange them upon encountering each other.

IV. Prototyping Conflict-free Federated Transfer Learning in Android

We developed a prototype framework for conflict-free federated transfer learning on Android by extending the Flower federated [20] code base. This framework supports custom TensorFlow Lite transfer learning models by specifying a frozen base model for feature extraction and a trainable classifier head. The design is adaptable, allowing various combinations of base models and classifier heads. Like Flower, the framework embeds four core functions in the tflite model: *infer*, *train*, *parameters*, and *restore*. However, we modified *parameters* and *restore* to interact only with the classifier head subnetwork, as the base model's parameters are not transmitted during training. The model creation infrastructure uses TensorFlow version 2.14.

The current Flower federated Android application requires all client training and testing data to be loaded into memory on initialization, which is impractical for transfer learning scenarios where data preprocessing increases memory usage. For instance, adapting MobileNetV2 to CIFAR-10 involves resizing input images to 224×224 and converting pixel values from 8-bit integers to 32-bit floats, significantly raising memory requirements. To address this, we implemented a custom data loading routine that loads input data on demand, circumventing the heap size limits of Android applications.

A. Provisions for Simulation

To emulate pairwise exchanges between nodes, we modify the server-side *configure_fit* routine to send differentiated subnetwork weights based on simulated node interactions. Using a

connection log from the One Simulator [21], the routine tracks the network state as a time-varying graph and extracts random matches from the current network state. Node pairs from each match are assumed to exchange subnetwork parameters, and the *fit_configuration* for the next round is updated accordingly.

We also introduce an experimental mode to emulate an arbitrary number of participants using limited devices. This required modifications to both server and client applications, including adding an *emulated_id* field to the *fit_configuration* dictionary. Upon receiving the *fit_configuration* dictionary containing the *emulated_id* and new model parameters, the client application loads the training data corresponding to the emulated device and updates the TensorFlow model parameters accordingly.

V. Experiments

We conduct simulations using models built with the infrastructure from Section IV. Our evaluations assess the system's ability to support transfer learning with pre-trained models optimized for mobile devices under varying levels of data heterogeneity. We track the performance of a hypothetical *oracle classifier head*, constructed by virtually incorporating the latest subnetwork updates. Additionally, we record accuracy trends for individual subnetworks and the local full classifier heads at each node. These metrics reveal how exposure to different datasets affects subnetwork performance and how successive subnetwork integration impacts the full classifier heads.

A. Dataset and Distributions

We use the CIFAR-10 dataset [22] to represent the target task for our transfer learning scenario, which consists of $60,000$ images that are distributed across 10 classes. The training set, consisting of $50,000$ images are split across $N = 100$ participating nodes by dirichlet sampling using concentration parameter α values of 100, 0.5, and 0.05, respectively. Smaller values of α result in more heterogeneous local datasets.

B. Model Architectures

In our experiments, we use three pre-trained backbones as feature extractors, all trained on the ImageNet dataset: MobileNetV2, EfficientNetB0, and ResDense, which combines ResNet101 and DenseNet121. ResDense serves as a performance benchmark for more powerful models not optimized for mobile devices. Following [18], input images pass through the feature extraction layers of ResNet101 and DenseNet121 in parallel, producing feature vectors of dimensions 2048 and 1024, respectively, which are concatenated into a 3072-dimensional vector. MobileNetV2 and EfficientNetB0 output 1280-dimensional vectors directly. The classifier head is a neural network with one hidden layer of 5000 neurons, followed by batch normalization and ReLU activations.

C. Node Connections

Bidirectional connections between nodes are established using two methods: random pairing and the random waypoint model (RWP). In random pairing, nodes are split into two sets, and connections are formed between randomly sampled nodes from each set, ensuring full participation in subnetwork exchanges. While not reflective of real-world mobility, this approach explores maximum node participation. The RWP method uses mobility traces from the One Simulator [21], with road network layers configured for areas (in meters) of 500×500, 1000×1000, and 2000×2000, as shown in Fig. 3. Nodes start at random positions, move to randomly chosen destinations at speeds of $[0.5, 1.5]$ m/s, pause briefly, and repeat the process. Connection links form when nodes come within a 50-meter range.

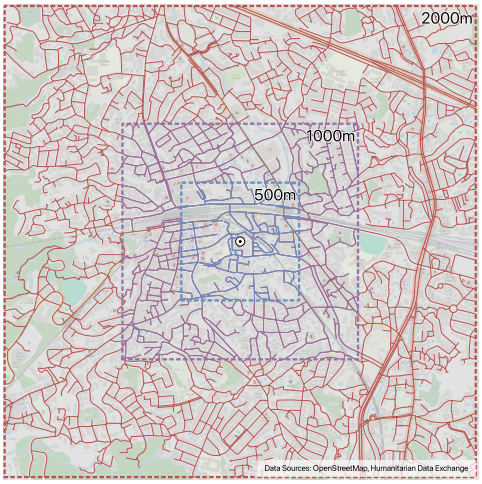

Fig. 3: Three simulation areas of increasing size centered on Ikoma City Hall, Nara Perfecture, Japan: S (500x500 m), M (1000x1000 m), and L (2000x2000 m)

D. Experimental Setups

Our experimental design combines simulations and real-world device tests. Simulations, outlined in Table I, include two parts: a controlled environment to establish baseline metrics and a realistic scenario with nodes moving under the random waypoint model. During local training, the learning rate η is 0.01 and batch size is set to 32. Each simulation is run five times with different random seeds on RTX 3080 and 3090 GPUs. Additionally, we train on diverse Android smartphones to evaluate real-world feasibility. A summary of the Android devices used for testing is provided in Table II.

VI. Results and Discussions

This section outlines the experimental results, beginning with a summary of key metrics across various conditions, followed by an in-depth analysis to extract insights.

A. Results of Experimental Setup I

Table III summarizes the results of the initial set of experiments. In a controlled environment with ensured pairwise connections, fine-tuning pre-trained models by splitting the classifier head and exchanging subnetworks proves viable. Among the three backbones evaluated, the most computationally expensive model, ResDense, consistently achieves the best

TABLE I: Experimental setups

Setup	Backbone	Topology Source	Steps	$Dir(\alpha)$	Subnet Hidden Neurons	No. of Subnets
I	ResDense, MobileNetv2, EfficientNetB0	RP	100	100, 0.5, 0.05	50	100
II	EfficientNetB0	RWP	100	100, 0.5, 0.05	50	100, 1

TABLE II: Specifications of the Android devices

Device ID	Device Name	CPU	GPU	OS	Memory (GB)
A	Redmi Note 10 JE	Qualcomm Snapdragon 480 (8-cores)	Adreno 619	Android 11	4
B	Poco X3 NFC	Qualcomm SM7150-AC Snapdragon 732G (8-cores)	Adreno 618	Android 12	6
C	Aquous Sense 7	Qualcomm SM6375 Snapdragon 695 5G (8-cores)	Adreno 619	Android 14	6
D	Poco X6 Pro	Mediatek Dimensity 8300 Ultra (8-cores)	Mali G615-MC6	Android 14	12

performance. However, the other two backbones, which are better suited for deployment on resource-constrained mobile devices, still deliver reasonable results. Interestingly, in IID scenarios, the average subnetworks accuracy may exceed that of the local full models. We attribute this phenomenon to two key factors. First, when the local datasets are homogeneous (i.e., $\alpha = 100$), the training progress of the subnetworks is stable, leading to consistent performance improvements. Second, at each time step, a node integrates only one subnetwork update into its full model. As a result, by the end of the simulation, the full model has not yet incorporated the most recent updates from all subnetworks. Nonetheless, the results show that even without redistributing the final subnetwork updates, the full model maintains relatively robust accuracy, especially under non-IID data distributions.

TABLE III: Summary of Results for Setup I

Architecture	ACC	Data Heterogeneity		
		$\alpha = 100$	$\alpha = 0.5$	$\alpha = 0.05$
ResDense	Oracle	0.8919	0.8882	0.8724
	Full	0.8823	0.8779	0.8490
	Subnet	0.8889	0.8827	0.8039
EfficientNetB0	Oracle	0.8373	0.8320	0.8012
	Full	0.8232	0.8149	0.7638
	Subnet	0.8345	0.8248	0.7432
MobileNetV2	Oracle	0.7906	0.7826	0.7402
	Full	0.7743	0.7639	0.6970
	Subnet	0.7875	0.7760	0.6854

B. Results of Experimental Setup II

In this section, the study transitions from a controlled environment to a more realistic scenario where node movement is governed by the random waypoint model within real-world road networks of increasing size. In these experiments, node connections are not guaranteed, meaning that a node may forgo training its subnetwork during certain time steps. The backbone used is EfficientNetB0 followed by a classifier head with 50 hidden neurons. To assess the effectiveness of the strategy, results are compared with an opportunistic implementation of federated averaging, referred to as OppFedAVG. In addition to the larger model capacity (49.3MB vs. 563.9KB), OppFedAVG is given an advantage by assuming that nodes receive updates from all their neighbors at each time step. The results are summarized in Table IV, demonstrating that the previous findings translate well into more realistic settings. Additionally, while performance remains comparable between the two approaches under IID assumptions and smaller simulated areas, the subnetwork-based training exhibits greater robustness, showing less sensitivity to the size of the simulation area and the degree of data heterogeneity.

TABLE IV: Summary of Results for Setup II

Area	OppFedAVG			OPTIMIST		
	0.05	0.5	100	0.05	0.5	100
L	0.3902	0.6950	0.7661	0.5655	0.7550	0.7875
M	0.5965	0.7920	0.8144	0.7165	0.8054	0.8175
S	0.7311	0.8256	0.8333	0.7868	0.8252	0.8315

Fig. 4 compares the total number of model exchanges between the two strategies across different map sizes. The evaluated strategy resulted in $8,042$ exchanges in the small area, compared to $40,400$ for the OppFedAVG approach. Similarly, in medium and large areas, the strategy led to $5,450$ and $3,634$ exchanges, while the OppFedAVG approach resulted in $18,600$ and $4,800$ exchanges, respectively. This further highlights the efficiency of the evaluated strategy in maintaining communication with fewer exchanges.

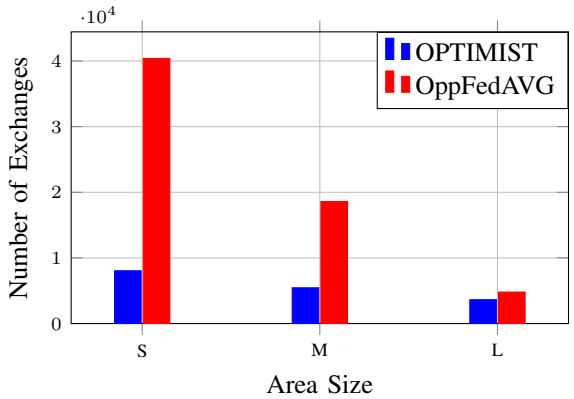

Fig. 4: Comparison in the number of exchanges across different sizes of the simulation area.

C. Results on Android Devices

Finally, we extend the evaluation to Android devices using the prototype framework. We employ MobileNetV2 with a subnetwork classifier head consisting of 50 hidden neurons. Each device is provided with 500 training samples, and the batch size is set to 32. While foreground applications are closed, no additional measures are taken to optimize performance, closely mimicking real-life usage scenarios.

We focus on three key metrics: Epoch, the total time for one epoch of training (including data loading and preprocessing); Batch, the time to process one batch of preprocessed input; and Load, the time to restore TensorFlow Lite model parameters using newly received subnetwork updates. A summary of these metrics across devices is shown in Table V.

TABLE V: On-device training statistics (ms)

Device	Load (Avg)	Batch (Avg)	Epoch (Avg)
A	1.7	2,053.1	33,593.1
B	1.5	1,912.6	31,243.7
C	2.9	1,705.4	27,989.8
D	1.4	1,559.9	25,567.9

While batch training times vary among devices, they remain consistent within a few seconds. The fastest device processes a batch in 1559.9 ms, while the slowest takes 2053.1 ms, a difference of approximately 31.7%. On-demand data loading and preprocessing contribute to a slight increase (2%) to the overall epoch duration. Finally, parameter transmission averages 650 ms. These results highlight the practical feasibility of OPTIMIST. To further enhance efficiency, precomputing features and saving them to disk could significantly reduce training costs, eliminate repeated preprocessing, and streamline training by allowing a single forward pass per input.

VII. Conclusion

In this paper, we introduced OPTIMIST, a federated transfer learning framework designed for opportunistic settings where traditional federated learning infrastructure is impractical. By leveraging pre-trained backbones and partitioning the classifier head into multiple non-overlapping subnetworks, OPTIMIST ensures conflict-free training and efficient parameter exchange during transient encounters. Simulations and on-device training confirm OPTIMIST's robust performance, surpassing federated averaging in efficiency and adaptability under data skew and limited connectivity, while maintaining practical batch processing times. Overall, OPTIMIST is a promising solution for deploying machine learning models in infrastructure-limited scenarios.

References

[1] T. A. Shaikh, T. Rasool, and F. R. Lone, "Towards leveraging the role of machine learning and artificial intelligence in precision agriculture and smart farming," *Computers and Electronics in Agriculture*, vol. 198, p. 107119, 2022.

[2] D. Tuia, B. Kellenberger, S. Beery, B. R. Costelloe, S. Zuffi, B. Risse, A. Mathis, M. W. Mathis, F. Van Langevelde, T. Burghardt *et al.*, "Perspectives in machine learning for wildlife conservation," *Nature communications*, vol. 13, no. 1, pp. 1–15, 2022.

[3] V. Chamola, V. Hassija, S. Gupta, A. Goyal, M. Guizani, and B. Sikdar, "Disaster and pandemic management using machine learning: a survey," *IEEE Internet of Things Journal*, vol. 8, no. 21, pp. 16 047–16 071, 2020.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[5] S. Lee, X. Zheng, J. Hua, H. Vikalo, and C. Julien, "Opportunistic federated learning: An exploration of egocentric collaboration for pervasive computing applications," in *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 2021, pp. 1–8.

[6] H. Ochiai, Y. Sun, Q. Jin, N. Wongwiwatchai, and H. Esaki, "Wireless ad hoc federated learning: A fully distributed cooperative machine learning," *arXiv preprint arXiv:2205.11779*, 2022.

[7] S. Tomita, J. P. Talusan, Y. Nakamura, H. Suwa, and K. Yasumoto, "Fedtour: Participatory federated learning of tourism object recognition models with minimal parameter exchanges between user devices," in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 2022, pp. 667–673.

[8] V. Romero, T. Matsui, Y. Matsuda, H. Suwa, and K. Yasumoto, "Towards opportunistic federated learning using independent subnetwork training," in *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2024, pp. 174–181.

[9] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[10] L. C. Suzuki, V. G. Cerf, J. L. Torgerson, and T. S. Suzuki, "A novel federated computation approach for artificial intelligence applications in delay and disruption tolerant networks," in *2023 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*. IEEE, 2023, pp. 1–8.

[11] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, 2023.

[12] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, 2020.

[13] D. Menegatti, A. Giuseppi, S. Manfredi, and A. Pietrabissa, "A discrete-time multi-hop consensus protocol for decentralized federated learning," *IEEE Access*, 2023.

[14] B. Yuan, C. R. Wolfe, C. Dun, Y. Tang, A. Kyrillidis, and C. Jermaine, "Distributed learning of fully connected neural networks using independent subnet training," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, pp. 1581–1590, 2022.

[15] Y. Li and X. Lyu, "Convergence analysis of sequential federated learning on heterogeneous data," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] F. Alam, T. Alam, M. A. Hasan, A. Hasnat, M. Imran, and F. Ofli, "Medic: a multi-task learning dataset for disaster image classification," *Neural Computing and Applications*, vol. 35, no. 3, pp. 2609–2632, 2023.

[17] Y. Pathak, P. K. Shukla, A. Tiwari, S. Stalin, and S. Singh, "Deep transfer learning based classification model for covid-19 disease," *Irbm*, vol. 43, no. 2, pp. 87–92, 2022.

[18] E. Hu, Y. Tang, A. Kyrillidis, and C. Jermaine, "Federated learning over images: vertical decompositions and pre-trained backbones are difficult to beat," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 19 385–19 396.

[19] V. Romero, "Phd forum: Learning at the time of disasters," in *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2024, pp. 256–257.

[20] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[21] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proceedings of the 2nd international conference on simulation tools and techniques*, 2009, pp. 1–10.

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