FedHeN: Federated Learning in Heterogeneous Networks

Durmus Alp Emre Acar [1] **Venkatesh Saligrama** [1]

Abstract

We propose a novel training recipe for federated learning with heterogeneous networks where each device can have different architectures. We introduce training with a side objective to the devices of higher complexities to jointly train different architectures in a federated setting. We empirically show that our approach improves the performance of different architectures and leads to high communication savings compared to the state-of-the-art methods.

1. Introduction

McMahan et al. (2017a) propose Federated Learning (FL) as a new distributed optimization problem where a server gradually trains a global model on datapoints from many devices. Due to privacy concerns, it is not allowed to transfer datapoints. Instead, FL transmits local and global models between devices and the server. Since transmitting models is a costly operation (Halgamuge et al., 2009), FL aims to find a global model with as less number of communication as possible.

Failure of traditional FL in heterogeneous devices. In standard FL, the server trains one model for all devices. A more practical problem is to allow devices to have different architectures based on device resources. For instance, consider a FL setting with many cellphone users where we want to distributively train a model on all user data (McMahan et al., 2017b). Some users might have the latest release of a cellphone whereas the rest use old versions. The users with the latest releases would prefer a different, potentially more complex, model than the user with old cellphones. Conventional FL as in McMahan et al. (2017a) fails in this case because devices have different model architectures.

In this work, we investigate the above practical FL problem where we allow devices to have different architectures based on their capacities. We propose FedHeN that modifies device training by introducing a novel side objective to the devices with complex models. The side objective allows FedHeN to jointly train complex and simple architectures.

We test our method in real-world datasets of CIFAR10 and CIFAR100 and compare it to a naive baseline and the current state-of-the-art method. We show that FedHeN achieves significant communication savings as well as better performance compared to the competitors.

Related Work.

FL. FedAvg (McMahan et al., 2017a) is proposed as an extension of decentralized SGD (Zinkevich et al., 2010) . The convergence of FedAvg depends on device data distribution (Li et al., 2020b). Different modifications are proposed to improve convergence guarantees such as FedProx (Li et al., 2020a), SCAFFOLD (Karimireddy et al., 2019), FedDyn (Acar et al., 2021a). All of these methods average parameters of device models in the server. This is not allowed heterogeneous network setting since the architectures are not the same. Differently, FedHeN targets FL with heterogeneous architectures.

HeteroFL (Diao et al., 2021) introduces FL with heterogeneous networks problem. HeteroFL considers a setting where simple architecture is mapped to a subset of the complex architecture. The server constructs new models by averaging model weights of all devices based on the mapping. Different from HeteroFL, FedHeN introduces novel side objectives for complex device training.

Early Exit. Due to their size, big DNNs consume more energy and they are slow to operate. Hubara et al. (2016); Yang et al. (2019) propose to quantize/binarize the weights of DNNs to improve memory and computation costs. Bolukbasi et al. (2017) propose to train a big DNN so that the network adaptively early exits in 'easy' examples to decrease inference costs. Kaya et al. (2019) propose to add a side objective on DNN to prevent 'overthinking' of DNNs for easy examples. Different from these works, we are interested in FL. FedHeN introduces side objectives to jointly train models in FL with different architectures.

Our Contributions.

- We present FedHeN to jointly train a server model with

[1]Boston University, Boston, MA. Correspondence to: Durmus Alp Emre Acar <alpacar@bu.edu>, Venkatesh Saligrama <srv@bu.edu>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

Algorithm 1 FL in Heterogeneous Networks - FedHeN

Input: T, E, η, N, $\{\mathcal{D}_i\}_{i\in[N]}$, initial models $\boldsymbol{w}_s^1$, $\boldsymbol{w}_c^1$,
for $t = 1 \dots T$ **do**
  Randomly sample active devices, $\mathcal{Z} \subset [N]$,
  Divide $\mathcal{Z}$ into simple and complex devices, $\mathcal{Z}_s$, $\mathcal{Z}_c$
  # Client Optimization
  for $i \in \mathcal{Z}_s$ **do**
    Receive the server simple model, $\boldsymbol{w}_s^t$,
    $\boldsymbol{w}_{s,i}^{t+1} = \text{ClientTraining}\left(\boldsymbol{w}_s^t, \mathcal{D}_i, E, \eta\right)$
    Transmit $\boldsymbol{w}_{s,i}^{t+1}$ back to the server.
  end for
  for $j \in \mathcal{Z}_c$ **do**
    Receive the server complex model, $\boldsymbol{w}_c^t$,
    $\boldsymbol{w}_{c,j}^{t+1} = \text{ClientTrainingSideObj}\left(\boldsymbol{w}_c^t, \mathcal{D}_j, E, \eta\right)$
    Transmit $\boldsymbol{w}_{c,j}^{t+1}$ back to the server.
  end for
  # Server Optimization
  Set $\boldsymbol{w}_s^{t+1}$ using weights from all active devices,
  $\boldsymbol{w}_s^{t+1} = \frac{1}{|\mathcal{Z}|}\left(\sum_{i\in\mathcal{Z}_s} \boldsymbol{w}_{s,i}^{t+1} + \sum_{j\in\mathcal{Z}_c} \left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}}\right)$
  Set $\boldsymbol{w}_c^{t+1}$'s sub-net as the updated simple model,
  $\left[\boldsymbol{w}_c^{t+1}\right]_{\mathcal{M}} = \boldsymbol{w}_s^{t+1}$
  Set rest of the $\boldsymbol{w}_c^{t+1}$ using complex active devices,
  $\left[\boldsymbol{w}_c^{t+1}\right]_{\mathcal{M}'} = \frac{1}{|\mathcal{Z}|_c} \sum_{j\in\mathcal{Z}_c} \left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}'}$
end for

different architectures based on device capacities.

- We empirically show that FedHeN achieves significant communication savings compared to the baselines.

2. Method

FL setting consists of one server node and N device nodes. Each device i has a different dataset $\mathcal{D}_i$. Let $f_i : \mathcal{W} \to \mathcal{R}$ be the loss of using a model on device i's dataset. FL solves,

$$\min_{\boldsymbol{w}\in\mathcal{W}} \frac{1}{N} \sum_{i\in[N]} f_i\left(\boldsymbol{w}\right)$$

where $\boldsymbol{w}$ is the NN parameters.

Different from the conventional FL, we are interested in having different architectures in devices. For simplicity, consider a setting where we have a simple architecture, $\boldsymbol{w}_s \in \mathcal{W}_s$, and a complex architecture $\boldsymbol{w}_c \in \mathcal{W}_c$. Let $\mathcal{S} \subset [N]$ and $\mathcal{C} = [N] - S$ be the devices that have simple and complex architectures respectively. We reformulate our problem as,

$$\min_{\substack{\boldsymbol{w}_s\in\mathcal{W}_s \\ \boldsymbol{w}_c\in\mathcal{W}_c}} \frac{1}{|\mathcal{S}|} \sum_{i\in S} f_i\left(\boldsymbol{w}_s\right) + \frac{1}{|\mathcal{C}|} \sum_{j\in C} f_j\left(\boldsymbol{w}_c\right)$$
$$\text{such that } \mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = 0 \tag{1}$$

where $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c)$ captures the relationship between simple and complex architectures.

Algorithm 2 FedHeN Device Optimizations

function ClientTraining $(\boldsymbol{w}_s, \mathcal{D}_i, E, \eta)$:
  Start from $\boldsymbol{w}_i$=$\boldsymbol{w}_s$, train E epochs,
  for E epochs, batch $B \subset \mathcal{D}_i$ **do**
    Compute batch gradient, $\hat{\nabla} f_i(\boldsymbol{w}_i)$,
    Update, $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta\hat{\nabla} f_i(\boldsymbol{w}_i)$,
  Return trained model $\boldsymbol{w}_i$
end function
function ClientTrainingSideObj $(\boldsymbol{w}_c, \mathcal{D}_j, E, \eta)$:
  Start from $\boldsymbol{w}_j$=$\boldsymbol{w}_c$, train E epochs with side obj.,
  for E epochs, batch $B \subset \mathcal{D}_j$ **do**
    Compute batch gradient along with side obj.,
    $\hat{\nabla} f_j(\boldsymbol{w}_j)$, $\hat{\nabla} f_j\left(\left[\boldsymbol{w}_j\right]_{\mathcal{M}}\right)$,
    Update, $\boldsymbol{w}_j \leftarrow \boldsymbol{w}_j - \eta\left(\hat{\nabla} f_j(\boldsymbol{w}_j) + \hat{\nabla} f_j\left(\left[\boldsymbol{w}_j\right]_{\mathcal{M}}\right)\right)$,
  Return trained model $\boldsymbol{w}_j$
end function

We note that Eq. 1 is an ERM objective. If there is no condition (no $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = 0$ constraint), one could minimize the objective by separating complex and simple losses. However, we are not interested in training data, we would like to train models that perform well on test data. Hence, we introduce $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c)$ as a way of regularizing the models.

To relate simple and complex architectures, we assume,

Assumption 2.1. Simple architecture is a sub-network of the complex architecture. There exists a set of indices, $\mathcal{M}$, of complex architecture such that $\mathcal{W}_s = \{[\boldsymbol{w}_c]_{\mathcal{M}} | \boldsymbol{w}_c \in \mathcal{W}_c\}$ where $[\boldsymbol{w}_c]_{\mathcal{M}}$ selects the weights of $\boldsymbol{w}_c$ based on the index set $\mathcal{M}$.

We encourage weight sharing between simple architecture and the corresponding sub-network of the complex architecture as in Assumption 2.1. We let $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = \|\boldsymbol{w}_s - [\boldsymbol{w}_c]_{\mathcal{M}}\|^2$.

To further regularize models, we add a side objective to the complex device training. Complex devices minimize their losses along with the corresponding sub-network of simple architecture as,

$$\min_{\substack{\boldsymbol{w}_s\in\mathcal{W}_s \\ \boldsymbol{w}_c\in\mathcal{W}_c}} \frac{1}{|\mathcal{S}|}\sum_{i\in S} f_i\left(\boldsymbol{w}_s\right) + \frac{1}{|\mathcal{C}|}\left(\sum_{j\in C} f_j\left(\boldsymbol{w}_c\right) + f_j\left(\left[\boldsymbol{w}_c\right]_{\mathcal{M}}\right)\right)$$
$$\text{such that } \mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = 0 \tag{2}$$

We would like highlight some properties of Eq. 2,

- Simple architecture is trained on all datapoints instead of on the datapoints only from the simple devices. Hence, the generalization of simple architecture is improved.
- $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c)$ correlates simple and complex architecture. A better simple model leads to a better complex model through $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = 0$ condition.

Table 1. IID split, 50 simple and 50 complex devices, 10% participation rate. The number of communication rounds required to achieve the target test performance for different methods. The gain in using FedHeN compared to best baseline method is given.

Dataset	Accuracy	FedHeN	Decouple	NoSide	Gain
Simple Model					
CIFAR-10	84.4	289	943	805	**2.8**×
	83.4	249	731	669	**2.7**×
CIFAR-100	46.4	296	864	984	**2.9**×
	45.4	250	588	807	**2.4**×
Complex Model					
CIFAR-10	88.5	649	991	941	**1.4**×
	87.5	456	739	669	**1.5**×
CIFAR-100	46.8	468	963	614	**1.3**×
	45.8	376	752	472	**1.3**×

Table 2. Non-IID split, 50 simple and 50 complex devices, 10% participation rate. The number of communication rounds required to achieve the target test performance for different methods. The gain in using FedHeN compared to best baseline method is given.

Dataset	Accuracy	FedHeN	Decouple	NoSide	Gain
Simple Model					
CIFAR-10	79.4	295	986	810	**2.7**×
	78.4	256	816	676	**2.6**×
CIFAR-100	43.8	278	978	914	**3.3**×
	42.8	239	813	762	**3.2**×
Complex Model					
CIFAR-10	84.2	596	1000	857	**1.4**×
	83.2	519	887	751	**1.4**×
CIFAR-100	44.8	450	997	498	**1.1**×
	43.8	372	754	456	**1.2**×

FedHeN Algorithm. FedHeN solves Eq. 2 with the steps summarized in Algorithm 1.

In each round, a random subset of devices become active, $\mathcal{Z}$. We divide set $\mathcal{Z}$ into simple active and complex active device sets as $\mathcal{Z}_s$ and $\mathcal{Z}_c$ respectively.

Simple active devices receive the server simple model, $\boldsymbol{w}_s^t$. We compute a local model $\boldsymbol{w}_{s,i}^{t+1}$ by starting from $\boldsymbol{w}_s^t$ and training it for E epochs on local dataset $\mathcal{D}_i$ displayed as 'ClientTraining' method (Alg. 2). The trained model is transmitted back to the server.

Complex active devices receive the server complex model, $\boldsymbol{w}_c^t$. We train a local model starting from $\boldsymbol{w}_c^t$ and training it for E epochs using their local dataset, $\mathcal{D}_j$ with gradients of the complex and the simple model shown as 'ClientTrainingSideObj' method (Alg. 2). Namely, we update the model with summation of batch gradient of complex model, $\hat{\nabla} f_j(\boldsymbol{w}_j)$, and batch gradient of the corresponding sub-network (simple) model, $\hat{\nabla} f_j\left([\boldsymbol{w}_j]_{\mathcal{M}}\right)$. The trained model is transmitted back to the server.

The server collects models from participating devices. The server simple model is constructed by averaging weights from all active devices, .i.e the simple devices $\{\boldsymbol{w}_{s,i}^{t+1}\}_{i\in\mathcal{Z}_s}$ as well as the common sub-net of the complex devices $\left\{\left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}}\right\}_{j\in\mathcal{Z}_c}$, ln. 18 in Alg. 1. The common sub-network of the server complex model is set equal to the constructed server simple model, ln. 20 in Alg. 1. Finally, the rest of the server complex model is constructed by averaging the corresponding weights of the active complex models, .i.e $\left\{\left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}'}\right\}_{j\in\mathcal{Z}_c}$ where $\mathcal{M}'$ corresponds to the sub-network that is not common with the simple architecture, ln. 22 in Alg. 1.

This completes one round of training of FedHeN. We iterate the same process for T communication rounds.

Cost of side objective. In passing, we note that side objective adds minimal cost to the complex devices. Firstly, it is a light weight operation. Complex devices calculate gradients of the full model, $\boldsymbol{w}_c$. Calculating the gradient with respect to the simple model, $\boldsymbol{w}_s$, requires less computation. Secondly, the main energy consumption occurs during transmission of models (Halgamuge et al., 2009).

3. Experiments

In this section, we compare FedHeN method to baselines in real-world dataset settings. We refer to Appendix A for a description of the hyperparameters.

FL dataset. We test our method using CIFAR-10 and CIFAR-10 (Krizhevsky et al., 2009). We split the datasets into 100 clients and randomly activate 10 clients in each round. We consider both IID and non-IID splits in our experiments. IID split is constructed by randomly partitioning data into clients. Non-IID split is constructed using a Dirichlet prior on the labels as in Yurochkin et al. (2019).

We assume the first 50 devices have simple architecture and the last 50 devices have complex architecture in all experiments.

Models. We use PreActResNet18[1] (He et al., 2016) for the complex architecture. PreActResNet18 has 4 residual blocks and total of 11.1M parameters.

If we centralize all client datapoints, the complex architecture gets 93% and 73.5% performance for CIFAR-10 and CIFAR-100 datasets respectively. If we centralize half of the datapoints as in 50 devices, the complex architecture gets 90.5% and 62.6% performance for CIFAR-10 and CIFAR-100 datasets respectively.

As a simple architecture, we consider the first 2 residual blocks of PreActResNet18. Then, we add a mix pooling layer (Lee et al., 2016) that learns a weighted combination of avg pooling and max pooling layers as in Kaya et al. (2019).

[1]BatchNorm layers store data statistics which results in privacy leakage. We use GroupNorm layers (Wu & He, 2018) instead in all ResNet models.

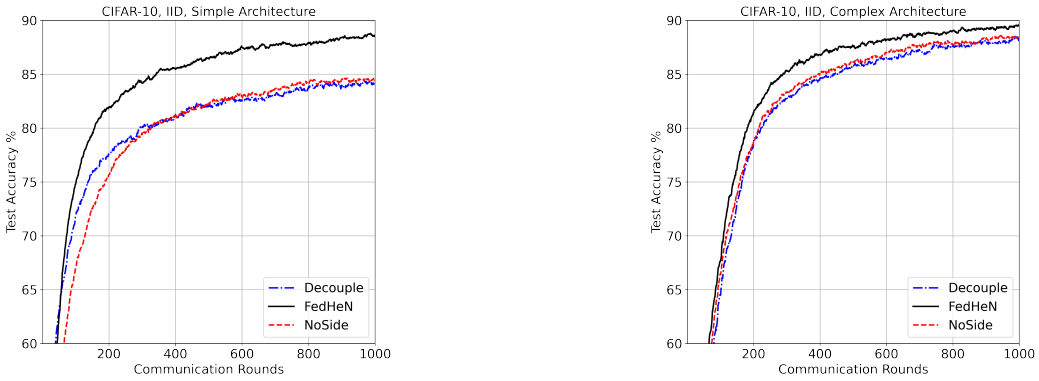


Figure 1. Test accuracy vs. communication rounds on CIFAR-10 IID split. **Left**: Simple, **Right**: Complex.

The simple architecture has overall 0.7M parameters.

If we centralize all client datapoints, the simple architecture gets 86% and 63.2% performance for CIFAR-10 and CIFAR-100 datasets respectively. If we centralize half of the datapoints as in 50 devices, simple model gets 84.5%, 55.7% performance for CIFAR-10 and CIFAR-100 respectively.

Methods. We compare FedHeN to two baselines as,

- *Naive Decouple*. Decouple minimizes Eq. 1 without any $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c) = 0$ constraint. It decouples complex and simple device training. It separately trains a complex and a simple model using FedAvg. Decouple is summarized in Algorithm 3.
- *NoSide*[2] *(Diao et al., 2021)*. NoSide is motivated from HeteroFL (Diao et al., 2021). It minimizes Eq. 1 with the same $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c)$ as FedHeN. The key difference is that it does not use side objective in the complex architecture training. NoSide is summarized in Algorithm 4.

Evaluation Metric. We fix a target test accuracy for server simple and server complex models. We compare the number of communications rounds to achieve the target test accuracy for all methods.

Results. Table 1 & 2 show the number of communication rounds to get target accuracies in all methods. We highlight the gain of using FedHeN compared to the best competitor in the last column. We present convergence curves vs. communication rounds in Figure 1, 2 & 3 (Appendix A).

FedHeN leads to significant communication savings. We observe that FedHeN trains better models uniformly in all the experiments shown in the gain columns of Table 1 & 2. For instance, the same simple performance of 43.8% is obtained using $3.3\times$ less communication with FedHeN in CIFAR-100 non-IID setting. The communication savings ranges from $1.1\times$ to $3.3\times$ in our experiments.

Decouple vs NoSide. Decouple is a naive algorithm. However, it performs close to NoSide algorithm in CIFAR-10 IID and non-IID settings as shown in Figure 1 & 2. For instance, the same test accuracy for complex model is achieved in 991 and 941 rounds for Decouple and NoSide models in CIFAR-10 IID setting as shown in Table 1.

Simple model in FedHeN achieves similar to centralized accuracy. FedHeN achieves better simple performance because simple architecture is trained on all datapoints due to the side objective in complex devices. Moreover, weight sharing between complex and simple architecture improves simple model's generalization. For instance, FedHeN's simple model in CIFAR-10 achieves 88.6% test accuracy in IID split within 1000 communication rounds as shown in Figure 1 which is higher than the centralized accuracy of simple model. Differently, NoSide and Decouple gets worse simple performance compared to FedHeN and the centralized model as shown in Figure 1, 2 & 3.

Complex model in FedHeN achieves better performance compared to competitors. Training with side objective improves complex model performance in FedHeN. For instance, in CIFAR-10 IID setting, FedHeN's complex model achieves 89.6% test performance within 1000 communication rounds which is $>1\%$ better compared to the competitors. This is also reflected in the gain values. FedHeN leads to $1.5\times$ communication savings for complex model.

4. Conclusion

We consider FL with heterogeneous networks where devices use different architectures based on their capacities. Our method, FedHeN, modifies device objectives of the complex device models to jointly train with different architectures. FedHeN leads to high communication savings compared to the baseline methods.

[2]HeteroFL is proposed in a setting where simple model is obtained by shrinking CNN channels of the complex model different from our setting. Except from the simple model definition, HeteroFL uses the same $\mathcal{R}(\boldsymbol{w}_s, \boldsymbol{w}_c)$ as FedHeN and it does not add side objective. We name HeteroFL in our setting as NoSide.

Acknowledgements

This research was supported by the Army Research Office Grant W911NF2110246, the National Science Foundation grants CCF-2007350 and CCF-1955981, and the Hariri Data Science Faculty Fellowship Grants, and a gift from the ARM corporation.

References

Acar, D. A. E., Zhao, Y., Matas, R., Mattina, M., Whatmough, P., and Saligrama, V. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=B7v4QMR6Z9w.

Acar, D. A. E., Zhao, Y., Zhu, R., Matas, R., Mattina, M., Whatmough, P., and Saligrama, V. Debiasing model updates for improving personalized federated training. In *International Conference on Machine Learning*, pp. 21–31. PMLR, 2021b.

Bolukbasi, T., Wang, J., Dekel, O., and Saligrama, V. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pp. 527–536. PMLR, 2017.

Chen, F., Luo, M., Dong, Z., Li, Z., and He, X. Federated meta-learning with fast convergence and efficient communication. *arXiv preprint arXiv:1802.07876*, 2018.

Diao, E., Ding, J., and Tarokh, V. Hetero{fl}: Computation and communication efficient federated learning for heterogeneous clients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=TNkPBBYFkXg.

Fallah, A., Mokhtari, A., and Ozdaglar, A. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/finn17a.html.

Halgamuge, M. N., Zukerman, M., Ramamohanarao, K., and Vu, H. L. An estimation of sensor energy consumption. *Progress in Electromagnetics Research*, 12:259–295, 2009.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

Jee Cho, Y., Wang, J., and Joshi, G. Towards understanding biased client selection in federated learning. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I. (eds.), *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 10351–10375. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/jee-cho22a.html.

Karimireddy, S. P., Kale, S., Mohri, M., Reddi, S. J., Stich, S. U., and Suresh, A. T. SCAFFOLD: stochastic controlled averaging for on-device federated learning. *CoRR*, abs/1910.06378, 2019. URL http://arxiv.org/abs/1910.06378.

Kaya, Y., Hong, S., and Dumitras, T. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pp. 3301–3310. PMLR, 2019.

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. *Technical report*, 2009.

Lee, C.-Y., Gallagher, P. W., and Tu, Z. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics*, pp. 464–472. PMLR, 2016.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems 2020*, pp. 429–450, 2020a.

Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HJxNAnVtDS.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282, 2017a.

McMahan, B., Ramage, D., and Scientists, R. Federated learning: Collaborative machine learning without centralized training data, Apr 2017b. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

Nishio, T. and Yonetani, R. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2019. doi: 10.1109/ICC.2019.8761315.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Thrun, S. and Pratt, L. *Learning to learn*. Springer Science & Business Media, 2012.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., and Hua, X.-s. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., and Khazaeni, Y. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, pp. 7252–7261, 2019.

Zhang, S. Q., Lin, J., and Zhang, Q. A multi-agent reinforcement learning approach for efficient client selection in federated learning, 2022. URL https://arxiv.org/abs/2201.02932.

Zinkevich, M., Weimer, M., Smola, A. J., and Li, L. Parallelized stochastic gradient descent. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pp. 2595–2603. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper/2010/hash/abea47ba24142ed16b7d8fbf2c740e0d-Abstract.html.

A. Appendix

Algorithm 3 Algorithm Decouple

Input: T, E, η, initial models $\boldsymbol{w}_s^1, \boldsymbol{w}_c^1$,
for $t = 1 \dots T$ **do**
  Randomly sample active devices, $\mathcal{Z} \subset [N]$,
  Divide $\mathcal{Z}$ into simple and complex devices, $\mathcal{Z}_s, \mathcal{Z}_c$
  # Client Optimization
  for $i \in \mathcal{Z}_s$ **do**
    Receive the server simple model, $\boldsymbol{w}_s^t$,
    $\boldsymbol{w}_{s,i}^{t+1} = \text{ClientTraining}\left(\boldsymbol{w}_s^t, \mathcal{D}_i, E, \eta\right)$
    Transmit $\boldsymbol{w}_{s,i}^{t+1}$ back to the server.
  end for
  for $j \in \mathcal{Z}_c$ **do**
    Receive the server complex model, $\boldsymbol{w}_c^t$,
    $\boldsymbol{w}_{c,j}^{t+1} = \text{ClientTraining}\left(\boldsymbol{w}_c^t, \mathcal{D}_j, E, \eta\right)$
    Transmit $\boldsymbol{w}_{c,j}^{t+1}$ back to the server.
  end for
  # Server Optimization
  Set $\boldsymbol{w}_s^{t+1}$ using weights from simple active devices,
  $\boldsymbol{w}_s^{t+1} = \frac{1}{|\mathcal{Z}_s|} \sum_{i \in \mathcal{Z}_s} \boldsymbol{w}_{s,i}^{t+1}$
  Set $\boldsymbol{w}_c^{t+1}$ using weights from complex active devices,
  $\boldsymbol{w}_c^{t+1} = \frac{1}{|\mathcal{Z}_c|} \sum_{i \in \mathcal{Z}_c} \boldsymbol{w}_{c,i}^{t+1}$
end for

Algorithm 4 Algorithm NoSide

Input: T, E, η, initial models $\boldsymbol{w}_s^1, \boldsymbol{w}_c^1$,
for $t = 1 \dots T$ **do**
  Randomly sample active devices, $\mathcal{Z} \subset [N]$,
  Divide $\mathcal{Z}$ into simple and complex devices, $\mathcal{Z}_s, \mathcal{Z}_c$
  # Client Optimization
  for $i \in \mathcal{Z}_s$ **do**
    Receive the server simple model, $\boldsymbol{w}_s^t$,
    $\boldsymbol{w}_{s,i}^{t+1} = \text{ClientTraining}\left(\boldsymbol{w}_s^t, \mathcal{D}_i, E, \eta\right)$
    Transmit $\boldsymbol{w}_{s,i}^{t+1}$ back to the server.
  end for
  for $j \in \mathcal{Z}_c$ **do**
    Receive the server complex model, $\boldsymbol{w}_c^t$,
    $\boldsymbol{w}_{c,j}^{t+1} = \text{ClientTraining}\left(\boldsymbol{w}_c^t, \mathcal{D}_j, E, \eta\right)$
    Transmit $\boldsymbol{w}_{c,j}^{t+1}$ back to the server.
  end for
  # Server Optimization
  Set $\boldsymbol{w}_s^{t+1}$ using weights from all active devices,
  $\boldsymbol{w}_s^{t+1} = \frac{1}{|\mathcal{Z}|} \left(\sum_{i \in \mathcal{Z}_s} \boldsymbol{w}_{s,i}^{t+1} + \sum_{j \in \mathcal{Z}_c} \left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}} \right)$
  Set $\boldsymbol{w}_c^{t+1}$'s sub-net as the updated simple model,
  $\left[\boldsymbol{w}_c^{t+1}\right]_{\mathcal{M}} = \boldsymbol{w}_s^{t+1}$
  Set rest of the $\boldsymbol{w}_c^{t+1}$ using complex active devices,
  $\left[\boldsymbol{w}_c^{t+1}\right]_{\mathcal{M}'} = \frac{1}{|\mathcal{Z}|_c} \sum_{j \in \mathcal{Z}_c} \left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}'}$
end for

Decouple Algorithm. We present Decouple methods in Algorithm 3. Decouple fully decouples simple and complex architecture training. We explain the method in detail below.

In each round, a random subset of devices become active, $\mathcal{Z}$. $\mathcal{Z}$ is then divided into simple active and complex active device sets as $\mathcal{Z}_s$ and $\mathcal{Z}_c$ respectively. Simple active devices receive the server simple model. A local model is trained using 'ClientTraining' method (Alg. 2). The trained model is transmitted back to the server. Complex active devices follow a similar process where they receive the server complex model. Then, a local model is trained using 'ClientTraining' method. The trained model is transmitted back to the server.

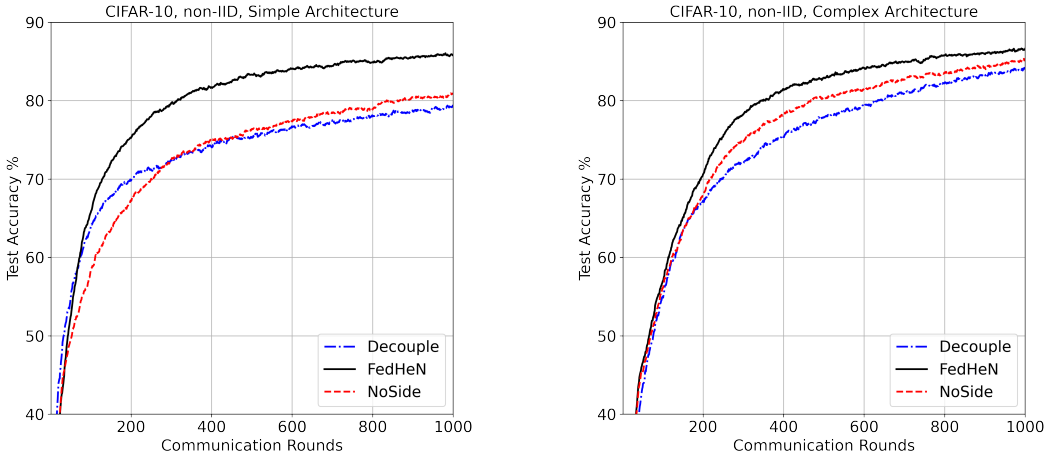
The server simple model is constructed by averaging weights from all active simple devices, $\{\boldsymbol{w}_{s,i}^{t+1}\}_{i \in \mathcal{Z}_s}$. The server complex model is constructed using the all active complex devices, $\left\{\boldsymbol{w}_{c,j}^{t+1}\right\}_{j \in \mathcal{Z}_c}$.

NoSide Algorithm. NoSide method is presented in Algorithm 4. We explain the method in detail below.

In each round, a random subset of devices become active, $\mathcal{Z}$. We divide set $\mathcal{Z}$ into simple active and complex active device sets as $\mathcal{Z}_s$ and $\mathcal{Z}_c$ respectively. Simple and complex device training is the same as in Decouple method where each active device receive the current server model based on their capacity, then train a local model using 'ClientTraining' method and transmit the trained model back to the server.

The server step is the same as in FedHeN method. The server simple model is constructed by averaging weights from all active devices, .i.e the simple devices $\{\boldsymbol{w}_{s,i}^{t+1}\}_{i \in \mathcal{Z}_s}$ as well as the common sub-net of the complex active devices $\left\{\left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}}\right\}_{j \in \mathcal{Z}_c}$. The common sub-architecture of the server complex model is set equal to the constructed server simple model. The rest of the server complex model is constructed by averaging the corresponding weights of the active complex models, .i.e $\left\{\left[\boldsymbol{w}_{c,j}^{t+1}\right]_{\mathcal{M}'}\right\}_{j \in \mathcal{Z}_c}$.

Related Works. We mention more dimensions of related work in this subsection.



(a) (b)

Figure 2. Test accuracy vs. communication rounds on CIFAR-10 non-IID split. 2(a): Simple, 2(b): Complex.

Personalized federated learning. Personalized federated learning (Chen et al., 2018) extends meta learning (Thrun & Pratt, 2012) into FL. The server trains a meta model and the devices customize the meta model using the local dataset. Fallah et al. (2020) propose to use FedAvg and MAML (Finn et al., 2017) meta adaptation. Acar et al. (2021b) use debiasing algorithms to improve convergence of FedAvg. Differently, FedHeN trains different complexity models for devices based on the capacities. This can be thought as customizing the device models based on the device resources. FedHeN can be improved with personalized federated learning methods.

Client selection. There are works that target to reduce communication costs through client selection (Zhang et al., 2022; Nishio & Yonetani, 2019; Jee Cho et al., 2022). A smart client selection further decreases the number of iterations and the communication costs compared to random activation. Different from client selection, FedHeN allows simple devices to participate federated learning which have lower communication footprint that complex device models. FedHeN can be adapted in to client selection algorithms.

Hyperparameters. We train each method for 1000 communication rounds. Each active device trains models for $E = 5$ epochs with learning rate as $\eta = 0.1$. We use SGD optimizer during training and clip gradients (at norm 10) to improve stability. If a device model fails in training, .i.e gets NaN weights, we ignore that device in server model construction only for that round. The methods are implemented using PyTorch library (Paszke et al., 2019).

Figures and Algorithms. Decouple and NoSide are summarized in Algorithm 3 and 4 respectively. The convergence curves of FedHeN, Decouple and NoSide are presented in Figure 1, 2 and 3.

Reporting Results. Methods average only active devices to set server models. This introduces noise in convergence curves and communication gain calculations. We report/present model performances when we average all devices (all complex devices for server complex model and all simple devices for server simple model). We note that this is just for the reporting purposes and the training is performed as stated in Algorithm 1, 3 & 4.

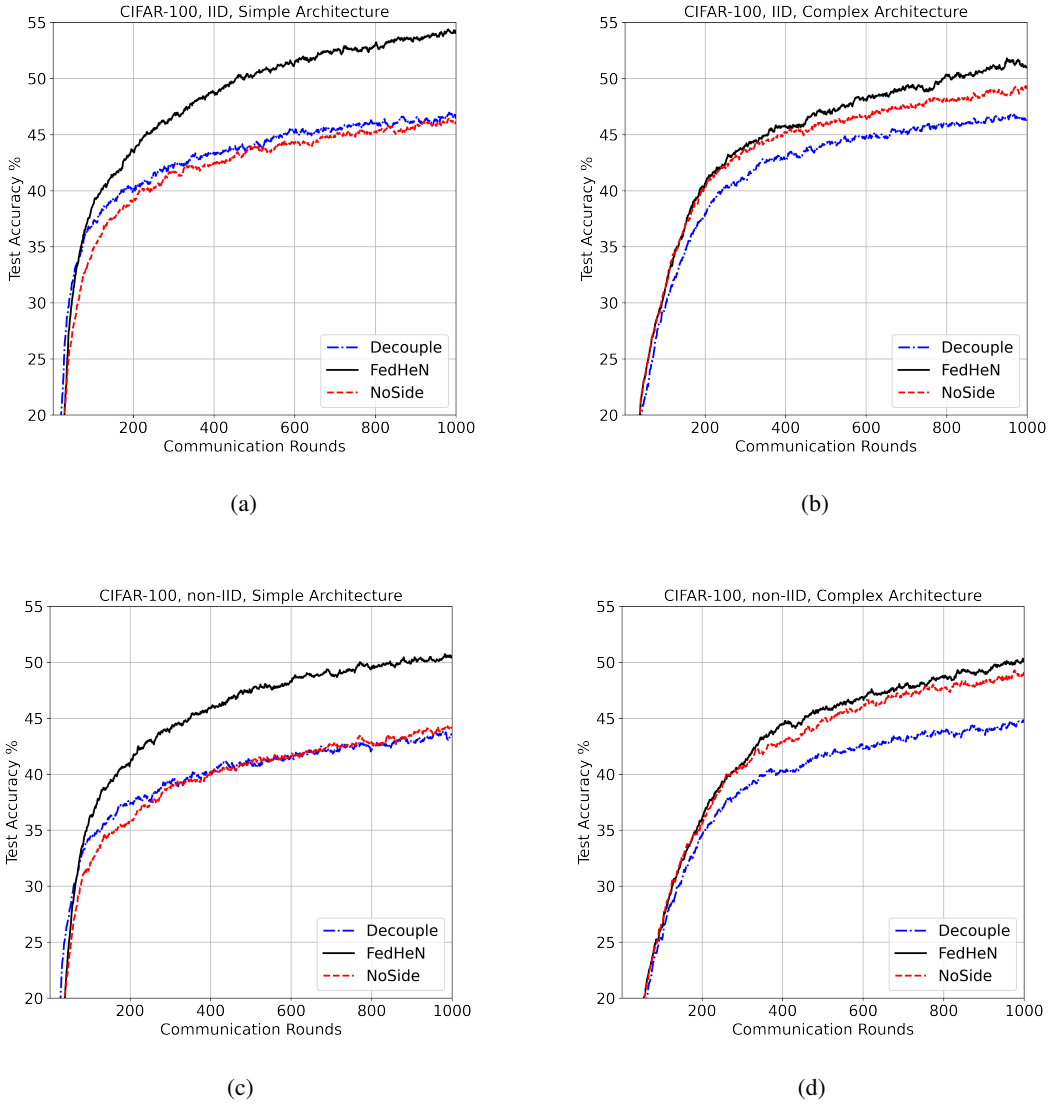


Figure 3. Test accuracy vs. communication rounds on CIFAR-10. 3(a) & 3(b): IID split simple and complex. 3(c) & 3(d): non-IID split simple and complex.