Late to the party? On-demand unlabeled personalized federated learning

Ohad Amosy
Bar Ilan University, Israel
amosy3@gmail.com

Gal Eyal
Bar Ilan University, Israel

Gal Chechik
Bar-Ilan University, Israel
NVIDIA Research, Israel

Abstract

In Federated Learning (FL), multiple clients collaborate to learn a shared model through a central server while keeping data decentralized. Personalized Federated Learning (PFL) further extends FL by learning a personalized model per client. In both FL and PFL, all clients participate in the training process and their labeled data are used for training. However, in reality, novel clients may wish to join a prediction service ***after it has been deployed****, obtaining predictions for their own* ***unlabeled*** *data.*

Here, we introduce a new learning setup, On-Demand Unlabeled PFL (OD-PFL), where a system trained on a set of clients, needs to be later applied to novel unlabeled clients at inference time. We propose a novel approach to this problem, ODPFL-HN, which learns to produce a new model for the late-to-the-party client. Specifically, we train an encoder network that learns a representation for a client given its unlabeled data. That client representation is fed to a hypernetwork that generates a personalized model for that client. Evaluated on five benchmark datasets, we find that ODPFL-HN generalizes better than the current FL and PFL methods, especially when the novel client has a large shift from training clients. We also analyzed the generalization error for novel clients, and showed analytically and experimentally how novel clients can apply differential privacy

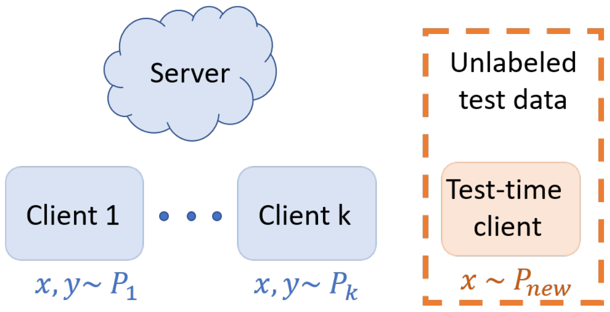


Figure 1. **The On-demand PFL problem.** A set of k clients $c_1, \ldots, c_k$ are available for training a PFL model, each with their own distribution $P_i(x, y)$. After training is completed, a new client appears, with its own distribution P_{new} over unlabeled data. The goal is to create a model f_{new} that minimizes the loss over the new client data $l(y, f_{new}(x))$.

1. Introduction

Federated Learning (FL) is the task of learning a model over multiple disjoint local datasets, while keeping data decentralized [25]. Personalized Federated Learning (PFL) [38] extends FL to the case where the data distribution varies across clients. PFL has numerous applications from a smartphone application that wishes to improve text prediction without uploading user-sensitive data, to a consortium of hospitals that wish to train a joint model while preserving the privacy of their patients. Current PFL methods assume that all clients participate in training and that their data is labeled , so once a model is trained, a novel client cannot be added.

In many cases, however, a federated model has been trained and deployed, but then novel clients wish to join. Often, such novel clients do not have labeled data, and their data distribution may shift from that of training clients. This is the case, for example, when a speech recognition federated model has been deployed and needs to be applied to new users or when a virus diagnostic has been developed for some regions or countries, and then needs to be applied to new populations while the virus spreads. This learning setup poses a hard challenge to existing approaches. Non-personalized FL techniques may not generalize well to novel clients due to domain shift. PFL techniques learn personalized models but are not designed to handle clients that were not available during training.

When a novel client joins with its *labeled* data, there are various strategies to adapt the pre-trained model to the novel client. For instance, a FL model can be fine-tuned using those labels. For PFL, it is less clear which personalized model should be fine-tuned. [29] used a hypernetwork (HN) that generates a personalized model for each client, given a descriptor of client labels. To generalize to a novel client with labeled data, they fine-tuning the descriptor using the labeled data through the hypernetwork. While all these approaches are useful, they cannot handle novel clients that have no labeled data.

Here, we define a novel problem: performing federated learning on novel clients with *unlabeled* data that are only available at inference time. We call this setup **OD-PFL** for *On-Demand Unlabeled Personalized Federated Learning*. We propose a novel approach to this problem, called ODPFL-HN. During training, our architecture learns a space of personalized models, one for each client, together with an encoder that maps each client to a point in that client space. All personalized models are learned jointly through an HN, allowing us to combine personalized data effectively. At inference time, a novel client can locally compute its own descriptor using the client encoder. Then, it sends the descriptor to the server as input to the HN and obtains its personalized model.

A key question remains for this approach to succeed: How to compute a descriptor of a novel unlabeled client? A key idea is to define a *client encoder* that maps a dataset into a descriptor, and train it jointly with the HN. We explore the properties that this encoder should have. First, its architecture should implement a function that is invariant to permutations over its inputs [35]. Furthermore, if objects in the data adhere to their own symmetries, such as images, graphs, or point clouds, the encoder architecture should also be invariant to these symmetries [24]. To the best of our knowledge, this is the first paper that discusses learning such invariant descriptors of datasets or clients.

FL is motivated by privacy, but was shown to be vulnerable [21]. In the OD-PFL setup, novel clients do not expose data or gradients, so they can better control their privacy. We show theoretically how differential privacy (DP) can be applied effectively to a new client and then experimentally measured how DP affects the accuracy of the personalized model.

This paper makes the following contributions: (1) A new learning setup, OD-PFL, learning a personalized model to **novel unlabeled clients at inference time**. (2) A new approach, learn a space of models using an encoder that maps an unlabeled client to that space, and an architecture ODPFL-HN based on hypernetworks. (3) A generalization bound based on multitask learning and domain adaptation, and analysis of differential privacy for a novel client. (4) Evaluation on five benchmark datasets, showing that ODPFL-HN performs better than or equal to the baselines.

2. Related work

Federated learning (FL). In FL, clients collaboratively solve a learning task, while preserving data privacy and maintaining communication efficiency. By collaborating through FL, clients leverage the shared pool of knowledge from other clients in the federation, and can better handle data scarcity, low data quality, and unseen classes. The literature on FL is vast and cannot be covered here. We refer the reader to recent surveys [1, 36]. We note that [20] addresses clients with unlabeled data but does not address novel (test-time) clients, which is the main focus of this paper.

Personalized Federated Learning (PFL). FL methods learn a single global model across clients, and this limits their ability to deal with heterogeneous clients. In contrast, PFL methods are designed to handle heterogeneity of data between clients by learning multiple models. [30] separates PFL methods into data-based and model-based approaches. Data-based PFL approaches aim to smooth the statistical heterogeneity of data among different clients, by normalizing the data [7] or by selecting a subset of clients with minimal class imbalance [32]. Model-based PFL approaches adapt to the diversity of data distributions across clients. As an example, a server may learn a global model and share it with all the clients. Then, each client learns its own local model on top of the global model. We refer the reader to recent surveys [16, 30].

Novel labeled clients. Several recent studies proposed to create personalized models for a novel **labeled** client. [29] used an HN to produce personalized models for training clients. Given a novel client, they tuned the embedding layer of the HN using client labels. In [22], each client interpolates a global model and a local kNN model is produced using the labels of the novel clients. [23] modeled each client as a mixture of distributions using an EM-like algorithm. A new labeled client uses its labels to calculate its own personalized mixture. All these methods depend on having access to labels of the novel client hence do not apply to the OD-PFL setup. In Sec. 6.2, we test 3 different ways to apply existing PFL models to our setup and show that ODPFL-HN outperforms all these variants.

Adapting a model to a new distribution during inference can be viewed as a variant of domain adaptation (DA) [14,19,31]. Our on-demand setup can be viewed as an adaptation of DA to FL. We emphasize that DA is fundamentally different from FL, since in FL the data is distributed across different clients. As a result, DA approaches cannot be applied directly to the FL setup. To the best of our knowledge, this is the first paper to address test-time adaptation in a FL setup.

Differential privacy (DP). The goal of DP is to share information about a dataset but to avoid sharing information about individual samples in the dataset. Although privacy is a key motivation of PFL, private information is exposed in the process: an adversarial client can infer the presence of exact data points in the data of other clients, and under certain conditions even generate the data of other clients. See a survey for more details [26]. A natural solution is to use DP to protect client privacy [10]. However, DP adds noise to the training process and may harm the model per-

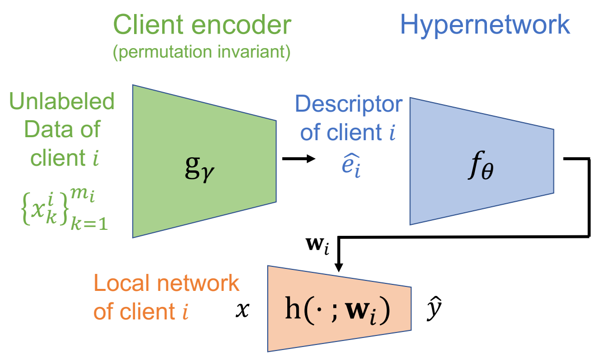


Figure 2. **The components of ODPFL-HN.** Given a client c_i with m_i unlabeled samples, the client encoder g_γ produces an embedding $\hat{e}_i$. Then, the HN f_θ predicts weights w_i for the local model h_i of client i.

formance. Here, we focus on the privacy of a novel client at test time, so the trained models remain untouched, and each novel client can choose its own privacy-accuracy trade-off at inference time.

3. The Learning Setup

We now formally define the learning setup of OD-PFL. We follow the notation in [2]. Let X be an input space and Y an output space. P is a probability distribution over the data $X \times Y$. Let l be a loss function $l : Y \times Y \to R$. $\mathcal{H}$ is a set of hypotheses with $h : X \to Y$. The error of a hypothesis h over a distribution P is defined by $err_P(h) = \int_{X\times Y} l(h(x), y)dP(x, y)$.

In OD-PFL, we train a federation of N clients $c_1, \ldots, c_N$, and other, novel, clients are added at inference time. For simplicity of notation, we consider a single novel client c_{new}. Let $\{P_i\}_{i=1}^N$ be the data distributions of training clients, and P_{new} the data distribution of the novel client c_{new}. Each training client has access to m_i IID samples from its distribution P_i, denoted as $S_i = \{(x_j^i, y_j^i)\}_{j=1}^{m_i}$.

The goal of OD-PFL is to use data from training clients $\{S_i\}_{i=1}^N$ to learn a mechanism that can assign a hypothesis $h_{new} \in \mathcal{H}$ when given unlabeled data from a novel client $S_{new} = \{x_j^{new}\}_{j=1}^m$. This hypothesis should minimize the expected error of the novel client. For any distribution of its data P_{new}, that error is defined by $err_{P_{new}}(h_{new}) = \int_{X\times Y} l(h_{new}(x), y)dP_{new}(x, y)$.

4. Our approach

We introduce OD-PFL to address the challenge of producing an "on-demand" personalized model for new unlabeled clients after a federated model has been deployed. This task is difficult because the data distribution of a novel client may differ from that of training clients and is unknown at training time. Also, the training clients are no longer available at inference time. To the best of our knowledge, these constraints were not considered in the FL setup before, and existing methods are not designed to handle this new setup. Generalizing to a novel client is hard for FL methods, because they do not specialize. It is hard for PFL methods, because no single model learned during training would not necessary fit a novel client. Even if it did, there is no clear way to select that model, because the novel client has no labels to use as a selection criteria. Here, we propose a meta-learning mechanism to produce a model that fits the new distribution of unlabeled data.

A natural candidate for such a meta-mechanism would be hypernetworks (HNs). HNs are neural networks that output the weights of another network and can therefore be used to produce "on-demand" models. Since the weights of the generated model are a (differentiable) function of the HN parameters, training the HN is achieved simply by propagating gradients from the generated (client) model. To generate a model for a client, the HN should be fed with a descriptor that summarizes the client dataset. Here we propose to learn a *client encoder* that takes as input the unlabeled data of a client and produces a dense descriptor. Figure 2 illustrates our approach. Each client feeds its input samples to a client encoder that produces an embedding vector. Then an HN takes the embedding and produces a personalized model. During training, the client uses its labels to tune the personalized model and back-propagates the gradients to the HN and the client encoder. We now describe our approach in detail.

Client encoding. The goal of client encoding is to map an entire data set into a dense descriptor. Formally, it maps unlabeled samples $S_i = \{x_j^i\}_{j=1}^{m_i}$ of client i to a descriptor e_i embedded in a representation space $\mathcal{E}$. In essence, the encoder is expected to map similar datasets to nearby descriptors in a way that balances personalization – of unique clients, with generalization – across similar clients.

For this mapping to be effective, it should obey several properties. First, the embedding should be inductive and generalizing, in the sense that the embedding function would later be applied to a novel client at test time and should generalize to that client. Second, since data samples form an unordered set, we wish that the encoder obeys the set symmetries of the data and is invariant to permutations over samples. Third, the encoder should capture the full data set. We now discuss our design decisions when building the encoder.

First, with respect to generalization. One may be tempted to explicitly regularize the representation such that similar datasets are mapped to close vectors in the embedding space. However, note that the descriptors are consumed by the downstream HN. Therefore, training the two networks jointly, while regularizing their parameters, should yield a representation that generalizes across clients. This is because the encoder tunes the representation to fit the downstream HN.

Second, with respect to permutation invariance, we tested three architectures. (1) *DeepSet* (DS) [35]. In DS, each data point is fed to the same (”siamese”) model and produces a feature vector. Then, an invariant pooling operator (usually mean) is applied to all outputs, and then processed by a second model, yielding the final descriptor. (2) DS is invariant to permutations over input samples, but it does not take into account symmetries of each element itself, such as translation invariance in images. We use *Deep Sets for symmetric elements* (DSS) [24] to handle symmetries at both the set level and the element level. (3) DS and DSS uniformly aggregate information from all individual elements in the set. Sometimes, it is beneficial to consider several elements together when computing the descriptor. To capture sample-to-sample interactions, we used a *set transformer* (ST) [18]. ST uses attention to aggregate representations of all elements into a single descriptor. The weight of each element is determined by the context of other elements in the dataset. We treated the architecture as a hyperparameter and selected it using the validation set.

Finally, with respect to describing the full dataset. The simplest approach is to use large batches that contain the entire dataset as input to the client encoder. We also tested an alternative approach that can be applied to large datasets that do not fit in a single batch in memory. In these cases, we split the data into smaller batches, encoded each batch, and used the average over batch descriptors as the final descriptor. Note that this resembles a DeepSet architecture and that when batches are sampled uniformly at random, would obey in expectation invariance to input permutation.

Hypernetworks. Our goal is to create a personalized model for a new client at inference time. Assuming that the client encoder summarized all relevant information to create such a personalized model. A natural solution is to learn a mapping from such descriptors to personalized models, and apply it to the descriptor of a novel client. This is exactly what HNs are designed for.

An HN f_θ parameterized by θ embodies a mapping from a client-embedding space to a hypotheses space $f_\theta : \mathcal{E} \rightarrow H$. The HN maps a client with an embedding vector e_i to a personalized model $h_i = h(\cdot; w_i)$, with $w_i = f_\theta(e_i)$.

Training. The client encoder and the HN are trained jointly. They produce a client descriptor and a personalized model for every client by optimizing the following loss

$$L(\theta, \gamma) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} l\left(f_\theta\left(g_\gamma(\{x_j^i\}_{j=1}^{m_i})\right)\left(x_j^i\right), y_j^i\right) \quad (1)$$

using training clients (labeled) $c_1, \ldots, c_N$, where l is a cross-entropy loss.

Algorithm 1 Training On-demand PFL-HN

```
initialize $\theta, \gamma$
**for** each round $t = 1, 2, ...$ **do**
    $i \leftarrow$ select a random client out of $N$ clients
    $e_i \leftarrow$ CLIENTENCODING($g_\gamma$)
    $w_i \leftarrow f_\theta(e_i)$     ▷ server computes a personal model
    $\Delta w_i \leftarrow$ CLIENTUPDATE($w_i$)
    apply chain rule to obtain $\Delta\theta$ and $\Delta e_i$ from $\Delta w_i$
    $\theta \leftarrow \theta - \eta\Delta\theta$
    $\Delta\gamma \leftarrow$ CLIENTBACKPROP($\Delta e_i$)
    $\gamma \leftarrow \gamma - \eta\Delta\gamma$
**function** CLIENTENCODING($g_\gamma$) //Run on client $i$
    $e_i \leftarrow g_\gamma(\{x_k\}_{k=1}^{m_i})$     ▷ client $i$ computes its embedding
    **return** $e_i$ to server

**function** CLIENTUPDATE($w_i$) //Run on client $i$
    $\mathcal{B} \leftarrow$ split $\{x_k\}_{k=1}^{m_i}$ into batches
    $w_i^{new} \leftarrow w_i$
    **for** each local epoch $e$ from 1 to $E$ **do**
        **for** each batch $b \in \mathcal{B}$ **do**
            update $w_i^{new}$ using $l(h(\cdot; w_i^{new}); b)$ ▷ training
    $\Delta w_i \leftarrow w_i^{new} - w_i$
    **return** $\Delta w_i$ to server

**function** CLIENTBACKPROP($\Delta e_i$) //Run on client $i$
    apply chain rule to obtain $\Delta\gamma$ from $\Delta e_i$
    **return** $\Delta\gamma$ to server
```

Workflow. Algorithm 1 shows the workflow of ODPFL-HN. During training, in each communication step, we repeat these 4 steps: (1) The server selects a random client and sends to it the current encoder g_γ. (2) The client locally predicts its embedding e_i and sends it back to the server. (3) Using its embedding, the server uses an HN f_θ to generate a customized network $h_i = h(\cdot; w_i)$ and communicates it to the client. (4) The client then locally trains that network on its data and communicates back to the server the delta between the weights before and after training. Using the chain rule, the server can train the hypernetwork and the encoder.

At inference time, (1) the server sends the encoder to the novel client. (2) The client uses the encoder to calculate an embedding e_{new} and sends it to the server. (3) The server uses the HN to predict the personalized model of the client $h_{new} = h(\cdot; w_{new})$ from the embedding and sends the result to the client. The client then applies its personalized model locally without revealing its data.

5. Generalization bound

The data of the new client and the clients of the federation may be sampled from different distributions. In the general case, there is no guarantee that learning a model for labeled clients would lead to a good model for a novel client. We now show that under reasonable assumptions, previous bounds developed for multitask learning (MTL) and for domain adaptation (DA), can be applied to the OD-PFL setup, to bound the generalization error of the novel client.

Intuitively, the bound has two terms; one captures the domain shift, and the other captures the generalization error.

Theorem 1. *Let $\mathcal{H}$ be a hypothesis space, P_{new} be a data distribution of a novel client, and Q be a distribution over the distributions of the clients, that is, P_i is drawn from Q. $\hat{err}_z(H)$ is the empirical loss over the training-client data and is defined in detail in the Appendix A.*

The generalization error of a novel client is bounded by $err_{P_{new}}(H) \leq \hat{err}_z(H) + \epsilon + \frac{1}{2}\int_P \inf_{h\in H} \hat{d}_{H\Delta H}(P, P_{new})dQ(P)$. Here, ϵ is an approximation error of a client in the federation from Theorem 2 in [2] and $\hat{d}_{H\Delta H}$ is a distance measure between the probability distributions defined in [3].

Proof. See Appendix A for a detailed proof. □

6. Experiments

6.1. Experiment setup and evaluation protocol

We evaluated ODPFL-HN using five benchmarks. We designed an experimental protocol the OD-PFL setup where novel clients are presented to the server during inference.

Client split: To quantify the performance of novel clients, we first randomly partition the clients into N_{train} train clients and N_{novel} novel clients (we used $N_{novel} = N/10$). Unless stated otherwise, we report average accuracy over novel clients. To conduct a fair comparison, training is limited to 500 steps for all evaluated methods. In each step, the server communicates with a 0.1 fraction of training clients following the protocol of each method.

Sample split and HP tuning: We split the samples of each *training client* into training samples and validation samples, which were used for hyperparameter tuning (details in See Appendix C).

6.2. Baselines

We evaluate using FL methods, which train a single global model, and PFL methods, which train one model per client. We compare the following FL methods: **(1) FedAVG** [25], where the parameters of local models are averaged with weights proportional to the sizes of the client datasets. **(2) FedProx** [27] adds a proximal term to the client cost functions, thereby limiting the impact of local updates by keeping them close to the global model. **(3) FedMA** [33] constructs the shared global model in a layer-wise manner by matching and averaging hidden elements with similar feature extraction signatures. For inference with FL methods, all novel clients are evaluated using the single global model.

Applying PFL methods to OD-PFL is not straightforward because **PFL methods are not designed to generalize to a novel unlabeled client.** They produce a model per training client, but it is not clear how to use these models for inference over a novel client. We tested three different ways to use PFL for inference with a novel client: **(4) PFL-sampled:** Draw a trained client model uniformly at random. We evaluate this baseline by computing the mean accuracy of all personalized models on each novel client. **(5) PFL-nearest:** We used the training client model closest to the novel client. We measure the distance using A-distance [4], which can be calculated in a FL setup. **(6) PFL-Ensemble:** (4) and (5) use a model from one of the training clients as the new client model. Here, we try a stronger baseline that uses all personalized models for a single novel client, by averaging the logits of all models for each prediction. In practice, this method is expensive in communication and computation costs. All experiments used pFedHN [29] to produce models for training clients.

6.3. Results for CIFAR

We evaluate ODPFL-HN using CIFAR10 and CIFAR100 [15] following two existing protocols.

(1) Pathological split: As proposed by [25], we sort the training samples by their labels and partition them into $N \cdot K$ shards. Then each client is randomly assigned K of the shards. This results in N clients with the same number of training samples and a different distribution over labels. In our experiments, we use $N = 100$ clients, $K = 2$ for CIFAR10 and $K = 5$ for CIFAR100.

(2) Dirichlet allocation: We follow the procedure by [12] to control the magnitude of the distribution shift between clients. For each client i, samples are drawn independently with class labels following a categorical distribution over classes with a parameter $q_i \sim Dir(\alpha)$. Here, Dir is the symmetric Dirichlet distribution.We conduct three experiments for each of the two datasets with $\alpha \in 0.1, 1, 10$. Smaller values of alpha imply larger distribution shifts between clients.

Implementation details: There are three different models in ODPFL-HN: A target model, an HN, and a client encoder. **Target model:** We use a LeNet [17] with two convolutions and two fully connected layers. To assure a fair comparison, we use the same target model across all evaluated methods and baselines. **Client encoder:** For the DS encoder, we use the same architecture as the target model, with an additional fully connected layer followed by pooling operations over batch dimension. These layers are added after

Table 1. **Accuracy on novel unlabeled clients, CIFAR10 & CIFAR100:** Values are averages and standard error across clients.

	CIFAR-10				CIFAR-100			
split	pathological	$\alpha = 0.1$	$\alpha = 1$	$\alpha = 10$	pathological	$\alpha = 0.1$	$\alpha = 1$	$\alpha = 10$
FedAvg	50.3 ± 2.9	58.7 ± 3.6	48.4 ± 2.9	66.2 ± 0.5	16.2 ± 1.3	17.9 ± 1.0	13.5 ± 1.7	30.2 ± 0.4
FedProx	54.2 ± 2.0	53.9 ± 2.2	54.2 ± 1.0	52.8 ± 0.6	5.5 ± 1.2	15.9 ± 0.7	20.6 ± 0.6	12.4 ± 0.6
FedMA	42.9 ± 1.8	49.3 ± 3.4	54.5 ± 0.9	53.8 ± 0.6	11.2 ± 0.7	12.6 ± 1.0	6.5 ± 0.4	7.3 ± 0.3
PFL-sampled	24.8 ± 1.0	61.0 ± 3.7	49.4 ± 3.0	$\mathbf{68.5 \pm 0.7}$	3.9 ± 0.4	13.5 ± 0.5	3.4 ± 1.4	32.4 ± 0.1
PFL-nearest	24.4 ± 6.2	63.1 ± 3.5	49.4 ± 0.9	$\mathbf{68.5 \pm 0.7}$	6.5 ± 2.7	14.3 ± 0.6	3.4 ± 0.4	32.1 ± 0.2
PFL-ensemble	47.6 ± 3.2	62.2 ± 3.7	49.4 ± 3.0	$\mathbf{68.5 \pm 0.7}$	7.8 ± 1.8	20.4 ± 1.2	3.4 ± 1.4	32.7 ± 0.2
ODPFL-HN (ours)	$\mathbf{59.5 \pm 3.5}$	$\mathbf{66.0 \pm 3.0}$	$\mathbf{62.9 \pm 1.0}$	68.1 ± 0.5	$\mathbf{19.5 \pm 2.1}$	$\mathbf{26.4 \pm 0.1}$	$\mathbf{32.9 \pm 0.9}$	$\mathbf{33.6 \pm 0.1}$

each convolution layer and before the fully connected layers. For the DSS and the ST encoder we use the public implementation provided by the authors. **Hypernetwork:** The HN is a fully connected network, with 3 hidden layers and linear head for each target weight tensor.

Results: Table 1 compares ODPFL-HN to baselines on CIFAR-10 and CIFAR-100. ODPFL-HN performs better than all baselines in all the evaluated scenarios, except for CIFAR-10 with the $\alpha = 10$ split.

6.4. Results for iNaturalist

iNaturalist is a dataset for Natural Species Classification based on the iNaturalist 2017 Challenge [11]. The dataset has 1,203 classes. Following [13], we evaluate ODPFL-HN using two geographical splits of iNaturalist: iNaturalist-Geo-1k with 368 clients, and iNaturalist-Geo-300 with 1,208 clients.

Implementation details: We use a MobileNetV2 [28] pre-trained on ImageNet to extract features for each image. The extracted feature vectors, of length 1280, are the input for both the target model and the client encoder. **Target model:** The target model is a fully connected network with two Dense layers and a Dropout layer. **Client encoder:** The client encoder has three fully connected layers with pooling operations after the first layer. HN implementation is the same as in Sec. 6.3.

Results: Table 2 shows ODPFL-HN outperforms current FL methods and adapted PFL methods.

6.5. Results for Landmarks

The Landmarks dataset is based on the 2019 Landmark-Recognition Challenge [34]. Following [13], we divide the dataset into clients by authorship. The resulting Landmarks-User-160k split contains 1,262 clients with 2,028 classes. Implementation is as in Sec. 6.4.

Results: Table 3 shows that ODPFL-HN outperforms current FL methods in all evaluations. ODPFL-HN outperforms the adapted PFL methods with the exception of PFL-ensemble where it ties. However, PFL-ensemble suffers from relatively large communication and computation costs compared to the proposed ODPFL-HN.

Table 2. **Accuracy on novel unlabeled clients for iNaturalist** Values are averages and SEMs across novel clients.

	iNaturalist	
split	Geo-300	Geo-1k
FedAvg	36.1 ± 1.6	36.9 ± 1.1
FedProx	17.4 ± 0.8	26.5 ± 1.7
FedMA	13.4 ± 0.7	17.5 ± 0.8
PFL sampled	25.6 ± 1.4	27.2 ± 0.9
PFL nearest	24.2 ± 3.7	26.9 ± 4.6
PFL ensemble	31.5 ± 1.6	36.6 ± 1.2
ODPFL-HN	$\mathbf{37.5 \pm 1.7}$	$\mathbf{41.6 \pm 1.2}$

6.6. Results for Yahoo Answers

The *Yahoo Answers* dataset is a question-classification dataset [37], with 1.4 million training samples from 10 classes. We divide the data into 1000 clients using the Dirichlet allocation procedure with $\alpha = 10$.

Implementation details: We used BERT [6] to extract a 768-dimension feature vector for each sample. For more details, see Section 6.4.

Results: Table 3 shows that ODPFL-HN outperforms current FL methods and adapted PFL methods.

6.7. How distribution shift affects generalization

We expect ODPFL-HN generalization to depend on the similarity between the novel client and the training clients. Novel clients that differ from training clients may perform poorly compared to clients that are similar to training clients.

To quantify this effect, we generated clients at varying similarity levels by creating new splits of CIFAR-100 using Dirichlet allocation while varying $\alpha \in$

Table 3. **Accuracy on novel unlabeled clients for Landmarks and Yahoo Answers.** Values are averages and SEMs across novel clients.

split	Landmarks User-160k	Yahoo User-1K
FedAvg	34.8 ± 1.3	27.6 ± 0.1
FedProx	13.8 ± 1.0	13.9 ± 0.1
FedMA	3.80 ± 0.4	25.0 ± 0.1
PFL sampled	37.4 ± 1.3	33.2 ± 0.1
PFL nearest	33.1 ± 3.6	13.2 ± 0.1
PFL ensemble	$\mathbf{39.1 \pm 1.4}$	33.2 ± 0.1
ODPFL-HN	$\mathbf{41.1 \pm 1.4}$	$\mathbf{35.8 \pm 0.2}$

$\{0.1, 0.25, 0.5, 1, 10\}$. To measure the similarity between a novel client and all training clients, we computed the empirical label distributions of each client and computed their KL-divergence from the novel client. Figure 3a presents the accuracy of a novel client as a function of its D_{KL} to the nearest train client. As expected, the accuracy decreases as D_{KL} grows, for all evaluated methods. ODPFL-HN demonstrates the most moderate decrease, achieving the highest accuracy in large distribution shifts.

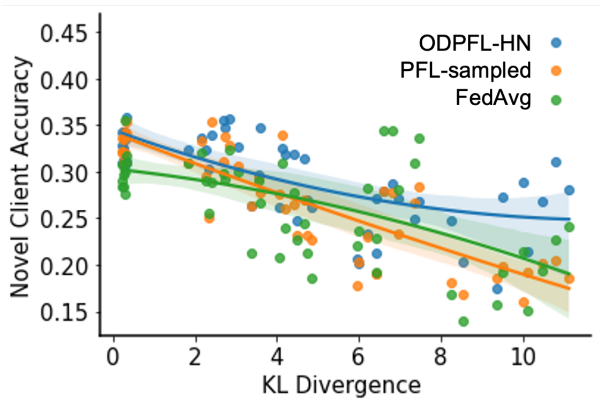
6.8. Robustness to covariate shift

Common benchmarks for PFL assume that different clients have a different distribution of labels. Here, we conduct an additional experiment to measure the effect of covariate shifts between clients. We evaluated the robustness of ODPFL-HN to a wide range of blur and rotation corruptions, using the CIFAR10 dataset. We applied the corruptions to the data of the novel client, while the data of training clients was kept uncorrupted. Figure 3b compares the accuracy for novel clients with varying levels of blur and rotation corruption. ODPFL-HN consistently out-performs all baselines.

We test ODPFL-HN robustness to covariate shift, by training an FL model on CIFAR10 (pathological split) and used STL10 as the data of the novel client. ODPFL-HN achieves $43.1\% \pm 4.1\%$, much better than all other baselines: FedProx $35.3\% \pm 2.7\%$, FedMA $32.9\% \pm 2.5\%$, FedAvg $34.7\% \pm 4.1\%$, PFL-sampled $17.6\% \pm 0.6\%$, PFL-ensemble $28.1\% \pm 1.9\%$ and PFL-nearest $31.9\% \pm 3.8\%$.

7. Differential Privacy

A key aspect of FL is data privacy, since it does not require clients to share their data directly with the hub. Unfortunately, some private information may be exposed (see a recent survey by [26]). In this section, we analyze the privacy of a novel client and characterize how it can protect its privacy by applying differential privacy (DP) [9]. We further study the trade-off between the privacy and accuracy of personalized models.



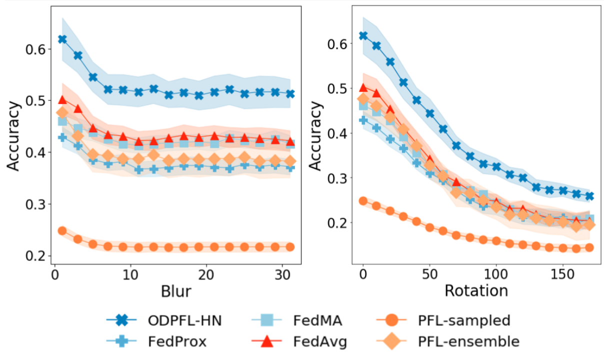


Figure 3. **Distribution shift. (Top)** Accuracy of novel clients vs. distribution shift between the novel client and training clients. Shown results for CIFAR-100 across multiple splits to $N_{train} = 90$ train clients and $N_{novel} = 10$ novel clients using symmetric Dirichlet distributions with varying parameter α (see Sec. 6.7). Accuracies of novel clients are reported against the KL-divergence (over label distribution) from the nearest train client for each method. **(Bottom)** Test mean accuracy ($\pm$ S.E.M. over 10 clients) for CIFAR-10 novel clients corrupted with blur using various size of Gaussian filters and rotation at various angles.

We first define key concepts and our notation. We use (ϵ, δ)-DP as defined by [8]. Two datasets D, D' are *adjacent* if they differ in a single instance.

Definition 7.1. A randomization mechanism $M : D \to R$ satisfies a (ϵ, δ)-differential privacy if for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that $Pr[M(d) \in S] \leq e^{\epsilon} Pr[M(d') \in S] + \delta$.

Here, ϵ quantifies privacy loss, where smaller values mean better privacy protection, and δ bounds the probability of privacy breach. The *sensitivity* of a function f is defined by $\Delta f = \max_{D,D'} ||f(D) - f(D')||$, for two datasets D and D' that differ by only one element.

[8] showed that given a model f, data privacy can be preserved by perturbing the output of the model and calibrating the standard deviation of the perturbation according to the

sensitivity of the function f and the desired level of privacy ϵ. Intuitively, if the protected model is not very sensitive to changes in a single training element, one can achieve DP with smaller perturbations.

Our focus here is to apply DP to a novel client that joins a pre-trained ODPFL-HN model. Fortunately, since the novel client does not participate in training, the only information that a novel client shares with the server is the client descriptor. This descriptor is computed locally by the client, so applying DP to the encoder can protect its data privacy.

Several mechanisms were proposed to achieve DP by adding noise. [8] describes a Gaussian mechanism that adds noise drawn from a Gaussian distribution with $\sigma^2 = \frac{2\Delta f^2 \log(1.25/\delta)}{\epsilon^2}$. Our analysis focuses on the Gaussian mechanism, but the same method can be used with other noise mechanisms.

Let a novel client apply (ϵ, δ)-DP to the encoder using the Gaussian mechanism. The server sends the encoder g to the client. The client then sends to the server $g(\{x_j^i\}_{j=1}^{m_i})+\xi$ as its embedding, where ξ is an IID vector from Gaussian distribution with $\sigma^2 = \frac{2(\Delta g)^2 \log(1.25/\delta)}{\epsilon^2}$. To do that, the client must know the sensitivity of the encoder. The following lemma shows that for a DS encoder, we can bound the sensitivity of the encoder, hence bound the noise magnitude necessary to achieve privacy. See proof in Appendix D.

Lemma 2. *Let g be a deep-set encoder, written as: $g(D) = \psi(\frac{1}{|D|}\sum_{x\in D}\phi(x))$. If ψ is a linear function with Lipschitz constant L_ψ, and ϕ is bounded by B_ψ, then the sensitivity of the encoder is bounded by $\Delta g \leq \frac{2}{|D|}L_\psi B_\phi$.*

The lemma shows that the encoder sensitivity decreases linearly with the size of the novel client dataset $|D|$. For a given (ϵ, δ), a lower sensitivity allows us to use less noise to achieve the desired privacy. This in turn means that the client can achieve better performance.

We now empirically evaluate the effect of adding Gaussian additive noise to the embedding of a novel client. To meet the conditions in lemma 2, we normalized the output of ϕ to be on a unit sphere, so $B_\phi = 1$. In addition, we average the output of ϕ, so $L_\psi = 1$. We used $\delta = 0.01$ and compared different values of ϵ and dataset sizes.

Figure 4 shows that with sufficient data, a novel client can protect its privacy without compromising its performance. For example, given a desired privacy of $\epsilon = 0.3$, if the client feeds the DP-encoder with 3000 samples, the HN creates a personalized model from a perturbed embedding that is as accurate as a non-DP model.

8. Conclusion

This paper describes a new FL setup, where a model trained in a FL workflow is transferred to novel clients whose data are not labeled and were not available during training. We describe ODPFL-HN, a novel approach to ODPFL, based on an encoder that learns a space of clients and an HN that maps clients to corresponding models in an "on-demand" way. We evaluated ODPFL-HN on five benchmark datasets, showing that it generalizes better than current FL and modified PFL methods. We also analyze and bound the generalization error for a novel client and analyze applying DP for the novel client. We hope that this paper will encourage the research community to consider generalization to novel clients when designing FL methods.

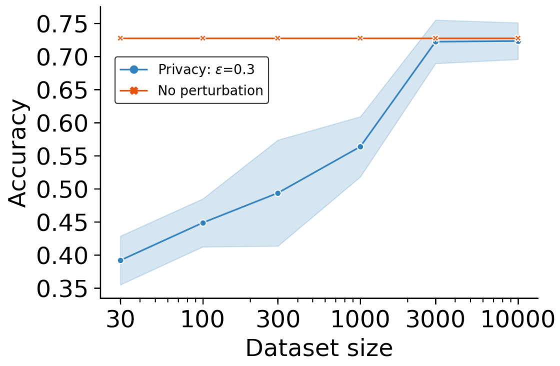


Figure 4. How much data is needed to achieve differential privacy? Test accuracy (±SEM) for CIFAR-10 novel client as a function of datasets size. Blue curve depicts models with privacy level of $\epsilon = 0.3$. With more data, smaller perturbation is needed to achieve that level of privacy, hence accuracy rises.

Ethical implications

Federated learning is designed for helping in training models without sharing user data that clients aim to keep safe. It is important to realize however that in the standard FL setup clients do send information about gradients of their local models to the server, and it is not guaranteed that no information about the data is leaked. The current paper addresses a scenario where all information about the new client is only convey through a client encoder, and this fact may provide better control to the new client over what data it shares. This work has no specific policing or military applications, and does not involve collecting or inferring personal data.

Reproducibility

We support reproducible research and will release all code, datasets and hyper parameters necessary to reproduce our results.

References

[1] Sawsan AbdulRahman, Hanine Tout, Hakima Ould-Slimane, Azzam Mourad, Chamseddine Talhi, and Mohsen Guizani. A survey on federated learning: The journey from centralized to distributed on-site learning and beyond. *IEEE Internet of Things Journal*, 8(7):5476–5497, 2020. 2

[2] Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000. 3, 5, 11

[3] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010. 5, 11

[4] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. Analysis of representations for domain adaptation. *Advances in neural information processing systems*, 19:137, 2007. 5

[5] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989. 11

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 6

[7] Moming Duan, Duo Liu, Xianzhang Chen, Renping Liu, Yujuan Tan, and Liang Liang. Self-balancing federated learning with global imbalanced data in mobile systems. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):59–71, 2021. 2

[8] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006. 7, 8

[9] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006. 7

[10] Ahmed El Ouadrhiri and Ahmed Abdelhadi. Differential privacy for deep and federated learning: A survey. *IEEE Access*, 10:22359–22380, 2022. 2

[11] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The inaturalist species classification and detection dataset. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8769–8778, 2018. 6

[12] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019. 5

[13] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Federated visual classification with real-world data distribution. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part X 16*, pages 76–92. Springer, 2020. 6

[14] Youngeun Kim, Donghyeon Cho, Kyeongtak Han, Priyadarshini Panda, and Sungeun Hong. Domain adaptation without source data. *IEEE Transactions on Artificial Intelligence*, 2021. 2

[15] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, Toronto., 2009. 5

[16] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 794–797. IEEE, 2020. 2

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 5

[18] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019. 4

[19] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning*, pages 6028–6039. PMLR, 2020. 2

[20] Nan Lu, Zhao Wang, Xiaoxiao Li, Gang Niu, Qi Dou, and Masashi Sugiyama. Federated learning from only unlabeled data with class-conditional-sharing clients. *arXiv preprint arXiv:2204.03304*, 2022. 2

[21] Priyanka Mary Mammen. Federated learning: opportunities and challenges. *arXiv preprint arXiv:2101.05428*, 2021. 2

[22] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *Advances in Neural Information Processing Systems*, 34:15434–15447, 2021. 2

[23] Othmane Marfoq, Giovanni Neglia, Richard Vidal, and Laetitia Kameni. Personalized federated learning through local memorization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15070–15092. PMLR, 17–23 Jul 2022. 2

[24] Haggai Maron, Or Litany, Gal Chechik, and Ethan Fetaya. On learning sets of symmetric elements. In *International Conference on Machine Learning*, pages 6734–6744. PMLR, 2020. 2, 4

[25] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017. 1, 5

[26] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey

on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021. 2, 7

[27] Anit Kumar Sahu, Tian Li, Maziar Sanjabi, Manzil Zaheer, Ameet Talwalkar, and Virginia Smith. On the convergence of federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 3:3, 2018. 5

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 6

[29] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502. PMLR, 2021. 1, 2, 5

[30] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. Towards personalized federated learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 2

[31] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020. 2

[32] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1698–1707. IEEE, 2020. 2

[33] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations (ICLR)*, 2020. 5, 13

[34] Tobias Weyand, André Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2 - a large-scale benchmark for instance-level recognition and retrieval. In *conference on computer vision and pattern recognition*, 2020. 6

[35] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. 2, 4

[36] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021. 2

[37] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015. 6

[38] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018. 1

Supplemental information

A. A Generalization Bound

Proof. The main idea of the proof is to first bound the error of all (labeled) training clients, using results from multitask learning. Then, treat the novel client as a target domain in a domain adaptation problem. This allow us to use results from domain adaptation to bound the novel client error.

The error of a hypothesis h over a distribution P is defined by $err_P(h) = \int_{X \times Y} l(h(x), y) dP(x, y)$. The error of the novel client for a given hypothesis **space** H is defined by

$$err_{P_{new}}(H) := \inf_{h \in H} err_{P_{new}}(h) \tag{2}$$

Since P_{new} is independent of Q, we can integrate over all $P \sim Q$ and obtain

$$err_{P_{new}}(H) = \int_P \inf_{h \in H} err_{P_{new}}(h) dQ(P). \tag{3}$$

Using Theorem 2 from [3] with P_{new} treated as the target domain and P as the source domain, gives that $\forall h, \forall P$: $err_{P_{new}}(h) \leq err_P(h) + \frac{1}{2}\hat{d}_{H\Delta H}(P, P_{new})$. Plugging into Eq. (3) gives

$$err_{P_{new}}(H) \leq \tag{4}$$

$$\int_P \inf_{h \in H} \left[err_P(h) + \tfrac{1}{2}\hat{d}_{H\Delta H}(P, P_{new}) \right] dQ(P) \tag{5}$$

$$= err_Q(H) + \tfrac{1}{2} \int_P \inf_{h \in H} \hat{d}_{H\Delta H}(P, P_{new}) dQ(P). \tag{6}$$

Since $err_Q(H)$ is unknown, we use Theorem 2 from [2] to bound the error of the novel client. That yields the bound in the theorem. □

Now we explain this in detail. If we have only one task and one domain, the most common solution is to find $h \in \mathcal{H}$ that minimizes the loss function on a training set sampled from probability distribution P. In general, $\mathcal{H}$ is a hyperparameter defined by the network architecture. [5] shows that in this simple case the generalization error is bounded. The bound depends on the "richness" of H. Choosing a "rich" H (with large VC-dimension), increase the generalization error.

In OD-PFL, each client may use a different hypothesis. Instead of bounding the error of one chosen hypothesis, we bound the error of the chosen hypotheses space that each client chooses from. In this way, we can bound the error of a novel client, without assuming anything about the way it chose from the hypothesis space.

First, we find H that is "rich" enough to contain hypotheses that can fit all data of the clients. Second, for each client, we select the best hypothesis $h \in H$ according to the client data. We define Q as a distribution over P, so, each client sample from Q a distribution P_i. We further define $\mathbb{H}$ as a hypothesis space family, where each $H \in \mathbb{H}$ is a set of functions $h : X \to Y$.

The first goal is to find a hypothesis space $H \in \mathbb{H}$ that minimizes the weighted error of all clients, assuming each client uses the best hypothesis $h \in H$. We define this error using the following loss:

$$err_Q(H) := \int_P \inf_{h \in H} err_P(h) dQ(P) \tag{7}$$

while $err_P(h) := \int_{X \times Y} l(h(x), y) dP(x, y)$. In practice, Q is unknown, so we can only estimate $err_Q(H)$ using the sampled clients and their data.

For each client $i = 1..n$, we sample the training data of the client from $X \times Y \sim P_i$. We denote the sampled training set by $z_i := (x_1, y_1), ..., (x_m, y_m)$, and $z = z_1, ...z_n$. The empirical error of a specific hypothesis is defined by $\hat{er}_z(h) := \frac{1}{m}\sum_{i=1}^{m} l(h(x_i), y_i)$. In training we minimize the empirical loss

$$\hat{er}_z(H) := \frac{1}{n} \sum_{i=1}^{n} \inf_{h \in H} \hat{er}_{z_i}(h) \tag{8}$$

[2] shows that if the number of clients n satisfies $n \geq max\{\frac{256}{\epsilon^2} log(\frac{8C(\frac{32}{\epsilon}, H^*)}{\delta}), \frac{64}{\epsilon^2}\}$, and the number of samples per client m satisfies $m \geq max\{\frac{256}{n\epsilon^2} log(\frac{8C(\frac{32}{\epsilon}, H_l^n)}{\delta}), \frac{64}{\epsilon^2}\}$, then with probability $1 - \delta$ all $H \in \mathcal{H}$ satisfies

$$err_Q(H) \leq \hat{er}_z(H) + \epsilon \tag{9}$$

were $C(\frac{32}{\epsilon}, H_l^n)$ and $C(\frac{32}{\epsilon}, H_l^n)$ are the covering numbers defined in [2], and can be referred to as a way to measure the complexity of H. Note that a very "rich" H makes $\hat{er}_z(H)$ small, but increases the covering number, so for the same amount of data, ϵ increases.

For the PFL setup, this is enough, since we can ensure that for a client that sampled from Q and was a part in the federation, the chosen hypothesis $h \in H$has an error close to the empirical one $\hat{er}_z(H)$. For a novel client, this may not be the case. The novel client may sample from a different distribution over P. In the general case, the novel client may even have a different distribution over $X \times Y$. In the most general case, the error on the novel client cannot be bound. In DA, a common distribution shift is a covariate shift, where $P(x)$ may change but $P(y|x)$ remains constant. This assumption lets us bound the error of the novel client.

[3] shows that for a given $H \in \mathcal{H}$, if S and T are two datasets with m samples, then with probability $1 - \delta$, for every hypothesis $h \in H$:

$$err_T(h) \leq \quad err_S(h) + \tfrac{1}{2}\hat{d}_{H\Delta H}(S, T) \tag{10}$$

$$+4\sqrt{\frac{2d\ log(2m) + log(\frac{2}{\delta})}{m}} + \lambda \tag{11}$$

where $err_D(h) = E_{(x,y)~D}[|h(x) - y|]$ is the error of the hypothesis on the probability distribution of the domain D. $\hat{d}_{H\Delta H}(S,T)$ is a distance measure between the domains S and T, and $\lambda = \arg\max_{h\in H} err_T(h) + err_S(h)$. Note that for over-parametrized models like deep neural networks λ should be very small. To keep the analysis shorter we assume this is the case. We also assume that m is large enough to neglect $4\sqrt{\frac{2d\,log(2m)+log(\frac{2}{\delta})}{m}}$. These assumptions are not mandatory, and the following analysis can be performed without them. This allows us to treat P_{new} as the target domain and P as the source domain in Eq. (4)

B. Additional Details about Training

B.1. Encoding batches of a dataset

We encode the whole dataset by feeding a large batch to the encoder. We also tested an alternative approach that can be applied to large datasets that do not fit in a single batch in memory. In these cases, we randomly split the data into smaller batches, encoded each batch, and used the average over batch descriptors as the final descriptor. When we tested this approach for several batch sizes, we did not observe that the performance was consistently sensitive to batch size. Encoding the full dataset in a single batch did perform better in most scenarios tested.

B.2. Training in two phases

In ODPFL-HN two main components are trained using a federation of labeled clients: A hypernetwork and a client encoder. We train them together end to end. When training the client encoder, batches from the same client yield different representations, and we found that this variability might hurt training end to end. We design 2 approaches to alleviate this issue: (1) Calculating the embedding using all client data, without breaking it into mini-batches. (2) Training in two steps by first learning an embedding of each training client, namely, a mapping from a client identity i to a dense descriptor e_i. This embedding layer was trained in a standard way jointly with the hypernetwork. Then, we trained the client encoder. We tested both methods, and choose between them using cross validation. Practically, for the iNaturalist experiments, the second method was better. For all other datasets, the first method was better. We now explain the second approach in detail.

Training the client encoder and the HN in two phases was done in the following way: The hypernetwork optimizes the L_{HN} loss defined in 12 by updating both its own weights θ and client representations $\{e_k\}_{k=1}^{N}$.

$$L_{HN}(\theta, e_1, ..., e_N) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} l(f_\theta(e_i)(x_j^i), y_j^i) \tag{12}$$

The client encoder trains to predict the representations learned by the hypernetwork from raw client data by minimizing $L_{encoder}$ defined in 13. At inference time, a novel client feeds its data to the client encoder and gets an embedding vector. Then, feeding the embedding vector to the hypernetwork produces a custom model for the client.

$$L_{encoder} = \sum_{i=1}^{n} L_2(g_\gamma(\{x_j^i\}_{j=1}^{m_i}), e_i) \tag{13}$$

In detail, in each communication step: (1) The server selects a random client and feeds its embedding e_i to the HN to create the personal model $h_i = h(\cdot, w_i)$. (2) The servers sends h_i, e_i, and the current encoder g_γ to the client. (3) The client then locally trains this network on its data and communicates the delta between the weights before and after training Δw_i back to the server. Using the chain rule, the server can train the hypernetwork and the embedding layer itself. (4) The client trains the encoder locally using the current given embedding e_i by optimizing $L_{encoder}$, then, the updates of the encoder are sent back to the server for aggregation. Inference is done in the same way as in end-to-end training.

Up to this point, the client encoder trains in parallel to the hypernetwork and has no influence on the hypernetwork weights or the embeddings of the labeled clients. We found that freezing the encoder and fine-tuning the hypernetwork using the trained encoder predictions improve the results of our method. This is done by optimizing the hypernetwork parameters θ using $L_{Fine-tune}$.

$$L_{Fine-tune}(\theta) = \sum_{i=1}^{n}\sum_{j=1}^{m_i} l(f_\theta(g_\gamma(\{x_j^i\}_{j=1}^{m_i}))(x_j^i), y_j^i) \tag{14}$$

However, this fine-tuning step reduces the performance of labeled clients. Note that in a real-world application, the server may save a version of the hypernetwork before fine-tuning it and use it when generating models for the original federation.

C. Experimental Details

For all experiments presented in the main text, we use a fully connected hypernetwork with 3 hidden layers of 100 hidden units each. The size of the embedding dimension is $\frac{N_{clients}}{4}$. Experiments are limited to 500 communication steps. In each step, communication is done with $0.1 \cdot N_{train}$.

Hyperparmeter Tuning We divide the training samples of each training client into 85% / 15% train / validation sets. The validation sets are used for hyperparameter tuning and early stopping of all baselines and datasets. The hyperparameters searched and the corresponding values by method: **FedAVG**: The local momentum μ_{local} is set to 0.5. We search over local learning-rate $\eta_{local} \in \{1e-1, 5e-2, 1e-$

$2, 5e{-}3, 1e{-}3\}$, number of local epochs $K \in \{1, , 2, 5, 10\}$ and batch size $\{16, 32, 64\}$. **FedProx** and **FedMA**: We used the hyperparameters used by [33] in the official code that provided by the authors. **pFEdHN**: We set $\mu_{local} = 0.9$. We search over learning-rates of the hypernetwork, embedding layer and local training: $\eta_{hn}, \eta_{embedding}, \eta_{local} \in \{1e-1, 5e-2, 1e-2, 5e-3, 1e-3\}$, weight decays $wd_{hn}, wd_{embedding}, wd_{local} \in \{1e-3, 1e-4, 1e-5\}$, number of local epochs $K \in \{1, , 2, 5, 10\}$ and batch size $\{32, 64\}$. **ODPFL-HN**: We perform the optimization using the same parameters and values as in pFEdHN. In addition, we search for the learning rate of the client encoder $\eta_{encoder} \in \{1e-1, 5e-2, 1e-2, 5e-3, 1e-3\}$.

CIFAR(Section 6.3) We use a LeNet-based target network with two convolution layers with 16 and 32 filters of size 5 respectively. Following these layers are two fully connected layers of sizes 120 and 84 that output logits vector. The client encoder follows the same architecture with an additional fully connected layer of size 200 followed by Mean-global-pooling for the first 100 units and Max-global-pooling for the other 100 units. Global pooling is done over the samples of a batch.

iNaturalist, Landmarks and Yahoo Answers Data(Sections 6.4-6.6) We use a simple fully-connected network with two Dense layers of size 500 each, followed by a Dropout layer with a dropout probability of 0.2. The client encoder is a fully-connected network with three Dense layers of size 500. The first layer is followed by Mean Global Pooling for the first 250 units and Max Global Pooling for the other 250 units.

D. Differential Privacy

Proof.

$$\begin{aligned}\Delta g \quad &:= \max_{D,D'} ||g(D) - g(D')|| \\ &= \max_{D,D'} ||\psi(\tfrac{1}{|D|}\textstyle\sum_{x\in D}\phi(x)) \\ &\quad -\psi(\tfrac{1}{|D'|}\textstyle\sum_{x\in D'}\phi(x))||.\end{aligned} \tag{15}$$

Denote $d \in D$ and $d' \in D'$ as the only nonidentical instance between D and D', so $D/d = D'/d'$. Then

$$\begin{aligned}\Delta g = \quad &\max_{D,D'} ||\psi\left(\tfrac{1}{|D|}[\textstyle\sum_{x\in D/d}\phi(x) + \phi(d)]\right) \\ &-\psi\left(\tfrac{1}{|D'|}[\textstyle\sum_{x\in D'/d'}\phi(x) + \phi(d')]\right)|| \\ = \quad &max_{d,d'} \tfrac{1}{|D|}||\psi\left(\phi(d) - \phi(d')\right)||\end{aligned} \tag{16}$$

Assume that ϕ is bounded by B_ϕ, so $|\phi(d) - \phi(d')| < 2B_\phi$. Then from the linearity of ψ:

$$\Delta g \le \frac{1}{|D|}L_\psi|\phi(d) - \phi(d')| \le \frac{2}{|D|}L_\psi B_\phi \tag{17}$$

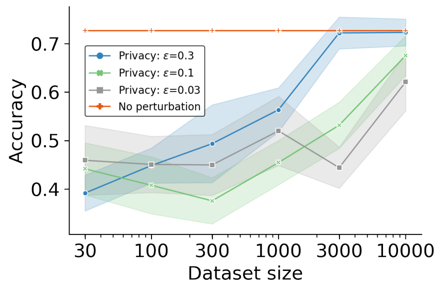


Figure 5. Test accuracy (±SEM) for CIFAR-10 novel client while applying DP. As ϵ decreases, we need more data to preserve the same accuracy of the model.

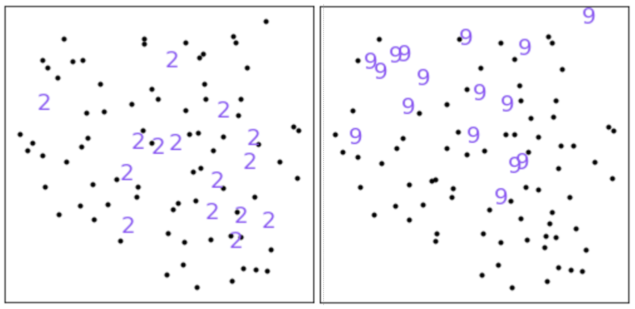


Figure 6. A tSNE plot of the embedding space for the pathological split of CIFAR10. In this setting, each client has a task of categorizing samples from two classes. Each dot and digit correspond to one client. The digits 2 marks clients that had samples from class 2; specifically (2,1), (2,3) (2,4) etc. Same for the digit 9. Similar plots can be made for the remaining classes. Clients involving the same class tend to be closer to each other (2 on the right, 9 at the top).

□

We also compare different levels of privacy, using lower values of ϵ. Figure 5 shows that more privacy (lower ϵ), requires a larger dataset to maintain the same accuracy of the model.

E. Embedding Space Visualization

To get intuition for the way the client encoder captures similarities between clients, we wish to visualize the space of client embeddings $\mathcal{E}$. Since each client involves samples from different classes, it is somewhat hard to visualize which client should be close. To capture some of those similarities, we mark clients that contain samples from some class with that class number. Figure 6 shows all clients that have samples from class #2 (left) or #9 (right). Clients that share a class tend to be located closer. Note that those clients have another class that is not shown, so they are not expected to fully cluster together.