FEDERATED SEMI-SUPERVISED LEARNING WITH INTER-CLIENT CONSISTENCY & DISJOINT LEARNING

Anonymous authors
Paper under double-blind review

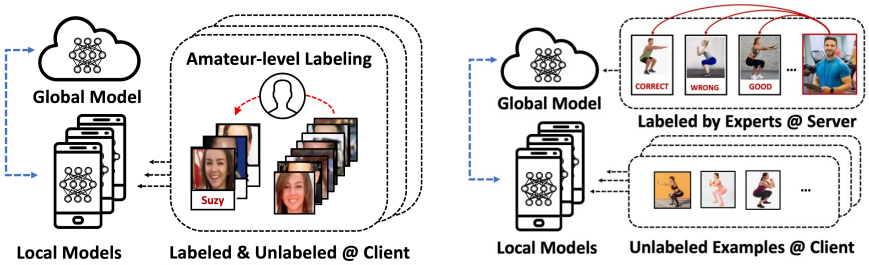
ABSTRACT

While existing federated learning approaches mostly require that clients have fully-labeled data to train on, in realistic settings, data obtained at the client-side often comes without any accompanying labels. Such deficiency of labels may result from either high labeling cost, or difficulty of annotation due to the requirement of expert knowledge. Thus the private data at each client may be either partly labeled, or completely unlabeled with labeled data being available only at the server, which leads us to a new practical federated learning problem, namely *Federated Semi-Supervised Learning* (FSSL). In this work, we study two essential scenarios of FSSL based on the location of the labeled data. The first scenario considers a conventional case where clients have both labeled and unlabeled data (labels-at-client), and the second scenario considers a more challenging case, where the labeled data is only available at the server (labels-at-server). We then propose a novel method to tackle the problems, which we refer to as *Federated Matching* (FedMatch). FedMatch improves upon naive combinations of federated learning and semi-supervised learning approaches with a new inter-client consistency loss and decomposition of the parameters for disjoint learning on labeled and unlabeled data. Through extensive experimental validation of our method in the two different scenarios, we show that our method outperforms both local semi-supervised learning and baselines which naively combine federated learning with semi-supervised learning.

1 INTRODUCTION

Federated Learning (FL) (Zhao et al., 2018; Li et al., 2018; Chen et al., 2019a;b), in which multiple clients collaboratively learn a global model via coordinated communication, has been an active topic of research over the past few years. The most distinctive difference of federated learning from distributed learning is that the data is only *privately accessible* at each local client, without inter-client data sharing. Such decentralized learning brings us numerous advantages in addressing real-world issues such as data privacy, security, and access rights. For example, for on-device learning of mobile devices, the service provider may not directly access local data since they may contain privacy-sensitive information. In healthcare domains, the hospitals may want to improve their clinical diagnosis systems without sharing the patient records.

Existing federated learning approaches handle these problems by aggregating the locally learned model parameters. A common limitation is that they only consider supervised learning settings, where the local private data is fully labeled. Yet, the assumption that all of the data examples may include sophisticate annotations is not realistic for real-world applications. Suppose that we perform on-device federated learning, the users may not want to spend their time and efforts in annotating the data, and the participation rate across the users may largely differ. Even in the case of enthusiastic users may not be able to fully label all the data in the device, which will leave the majority of the data as unlabeled **(See Figure 1 (a))**. Moreover, in some scenarios, the users may not have sufficient expertise to correctly label the data. For instance, suppose that we have a workout app that automatically evaluates and corrects one's body posture. In this case, the end users may not be able to evaluate his/her own body posture at all **(See Figure 1 (b))**. Thus, in many realistic scenarios for federated learning, local data will be mostly *unlabeled*. This leads us to practical problems of federated learning with deficiency of labels, namely *Federated Semi-Supervised Learning* (FSSL).



(a) Labels-at-Client Scenario (b) Labels-at-Server Scenario

Figure 1: **Illustrations of Two Practical Scenarios in Federated Semi-Supervised Learning** (a) Labels-at-Client scenario: both labeled and unlabeled data are available at local clients. (b) Labels-at-Server scenario: labeled instances are available only at server, while unlabeled data are available at local clients.

A naive solution to these scenarios is to simply perform Semi-Supervised Learning (SSL) using any off-the-shelf methods (e.g. FixMatch (Sohn et al., 2020), UDA (Xie et al., 2019)), while using federated learning algorithms to aggregate the learned weights. Yet, this does not fully exploit the knowledge of the multiple models trained on heterogeneous data distributions. To address this problem, we present a novel framework which we refer to as *Federated Matching (FedMatch)*, which enforces the consistency between the predictions made across multiple models. Further, conventional semi-supervised learning approaches are not applicable for scenarios where labeled data is only available at the server (Figure 1 (b)), which is a unique SSL setting for federated learning. Also, even when the labeled data is available at the client (Figure 1 (a)), learning from the unlabeled data may lead to forgetting of what the model learned from the labeled data. To tackle these issues, we *decompose* the model parameters into two, a dense parameter for supervised and a sparse parameter for unsupervised learning. This sparse additive parameter decomposition ensures that training on labeled and unlabeled data are effectively separable, thus minimizing interference between the two tasks. We further reduce the communication costs with both the decomposed parameters by sending only the difference of the parameters across the communication rounds. We validate FedMatch on both scenarios (Figure 1 (a) and (b)) and show that our models significantly outperform baselines, including a naive combination of federated learning with semi-supervised learning, on the training data which are both non-i.i.d. and i.i.d. data.

The main contributions of this work are as follows:

- We introduce a practical problem of federated learning with deficiency of supervision, namely **Federated Semi-Supervised Learning (FSSL)**, and study two different scenarios, where the local data is partly labeled (*Labels-at-Client*) or completely unlabeled (*Labels-at-Server*).
- We propose a **novel method**, **Federated Matching (FedMatch)**, which learns inter-client consistency between multiple clients, and decomposes model parameters to *reduce* both interference between supervised and unsupervised tasks, and communication cost.
- We show that our method, FedMatch, **significantly outperforms** both local SSL and the naive combination of FL with SSL algorithms under the conventional labels-at-client and the **novel labels-at-server** scenario, across multiple clients with both non-i.i.d. and i.i.d. data.

2 Problem Definition

We begin with formal definition of Federated Learning (FL) and Semi-Supervised Learning (SSL). Then, we define Federated Semi-Supervised Learning (FSSL) and introduce two essential scenarios.

2.1 Preliminaries

Federated Learning Federated Learning (FL) aims to collaboratively learn a global model via coordinated communication with multiple clients. Let G be a global model and $\mathcal{L} = \{l_k\}_{k=1}^{K}$ be a set of local models for K clients. Let $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N}$ be a given dataset, where $\mathbf{x}_i$ is an arbitrary training instance with a corresponding one-hot label $\mathbf{y}_i \in \{1, \dots, C\}$ for the C-way multi-class classification problem and N is the number of instances. $\mathcal{D}$ is composed of K sub-datasets $\mathcal{D}^{l_k} = \{\mathbf{x}_i^{l_k}, \mathbf{y}_i^{l_k}\}_{i=1}^{N^{l_k}}$

privately collected at each client or local model l_k. At each communication round r during training, G first randomly selects A local models that are available for training $\mathcal{L}^r \subset \mathcal{L}$ and $|\mathcal{L}^r| = A$. The global model G then initializes $\mathcal{L}^r$ with global weights $\boldsymbol{\theta}^G$, and the active local models $l_a \in \mathcal{L}^r$ perform supervised learning to minimize loss $\ell_s(\boldsymbol{\theta}^{l_a})$ on the corresponding sub-dataset $\mathcal{D}^{l_a}$. G then aggregates the learned weights $\boldsymbol{\theta}^G \leftarrow \frac{N^{l_a}}{N} \sum_a^A \boldsymbol{\theta}^{l_a}$ and broadcasts newly aggregated weights to local models that would be available at the next round $r+1$, and repeat the learning procedure until the final round R.

Semi-Supervised Learning Semi-supervised learning (SSL) refers to the problem of learning with partially labeled data, where the ratio of unlabeled data is usually much larger than that of the labeled data (e.g. $1:10$). For SSL, $\mathcal{D}$ is further split into labeled and unlabeled data. Let $\mathcal{S} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^S$ be a set of S labeled data instances and $\mathcal{U} = \{\mathbf{u}_i\}_{i=1}^U$ be a set of U unlabeled samples without corresponding label. Here, in general, $|\mathcal{S}| \ll |\mathcal{U}|$. With these two datasets, $\mathcal{S}$ and $\mathcal{U}$, we now perform semi-supervised learning. Let $p_\theta(\mathbf{y}|\mathbf{x})$ be a neural network that is parameterized by weights $\boldsymbol{\theta}$ and predicts softmax outputs $\hat{\mathbf{y}}$ with given input $\mathbf{x}$. Our objective is to minimize loss function $\ell_{final}(\boldsymbol{\theta}) = \ell_s(\boldsymbol{\theta}) + \ell_u(\boldsymbol{\theta})$, where $\ell_s(\boldsymbol{\theta})$ is loss term for supervised learning on $\mathcal{S}$ and $\ell_u(\boldsymbol{\theta})$ is loss term for unsupervised learning on $\mathcal{U}$

2.2 Federated Semi-Supervised Learning

Now we further describe a practical problem of deficiency of labels in federated learning, which we refer to as Federated Semi-Supervised Learning (FSSL), in which the data obtained at the clients may or may not come with accompanying labels. Given a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, $\mathcal{D}$ is split into a labeled set $\mathcal{S} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^S$ and a unlabeled set $\mathcal{U} = \{\mathbf{u}_i\}_{i=1}^U$ as in the standard semi-supervised learning. Under the federated learning framework, we have a global model G and a set of local models $\mathcal{L}$ where the unlabeled dataset $\mathcal{U}$ is privately spread over K clients hence $\mathcal{U}^{l_k} = \{\mathbf{u}_i^{l_k}\}_{i=1}^{U^{l_k}}$. For a labeled set $\mathcal{S}$, on the other hand, we consider two different scenarios depending on the availability of labeled data at clients, namely the *Labels-at-Client* and the *Labels-at-Server* scenario, of which problem settings and learning procedures will be discussed later.

3 Federated Matching

We now describe our *Federated Matching (FedMatch)* algorithm to tackle the federated semi-supervised learning problem. We describe its core components in detail in the following subsections.

3.1 Inter-Client Consistency Loss

Consistency regularization (Xie et al., 2019; Sohn et al., 2020; Berthelot et al., 2019b;a) is one of most popular approaches to learn from unlabeled examples in a semi-supervised learning setting. Conventional consistency-regularization methods enforce the predictions from the augmented examples and original (or weakly augmented) instances to output the same class label, $||p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{u}) - p_{\boldsymbol{\theta}}(\mathbf{y}|\pi(\mathbf{u}))||_2^2$, where $\pi(\cdot)$ is a stochastic transformation function. Based on the assumption that class semantics are unaffected by small input perturbations, these methods basically ensures consistency of the prediction across the multiple perturbations of *same input*. For our federated semi-supervised learning method, we additionally propose a novel consistency loss that regularizes the *models* learned at multiple clients to output the same prediction. This novel consistency loss for FSSL, *inter-client consistency* loss, is defined as follows:

$$\frac{1}{H} \sum_{j=1}^{H} \mathrm{KL}[p^*_{\boldsymbol{\theta}^{\mathrm{h}_j}}(\mathbf{y}|\mathbf{u})||p_{\boldsymbol{\theta}^l}(\mathbf{y}|\mathbf{u})] \tag{1}$$

where $p^*_{\boldsymbol{\theta}^h}(\mathbf{y}|\mathbf{x})$ are helper agents that are selected from the server based on model similarity to the client (which we describe later), that are not trained at the client (* denotes that we freeze the parameters). The server selects and broadcasts H helper agents at each communication round. We also use data-level consistency regularization at each local client similarly to FixMatch (Sohn et al., 2020). Our final consistency regularization term $\Phi(\cdot)$ can be written as follows:

$$\Phi(\cdot) = \mathrm{CrossEntropy}(\hat{\mathbf{y}}, p_{\boldsymbol{\theta}^l}(\mathbf{y}|\pi(\mathbf{u}))) + \frac{1}{H} \sum_{j=1}^{H} \mathrm{KL}[p^*_{\boldsymbol{\theta}^{\mathrm{h}_j}}(\mathbf{y}|\mathbf{u})||p_{\boldsymbol{\theta}^l}(\mathbf{y}|\mathbf{u})] \tag{2}$$

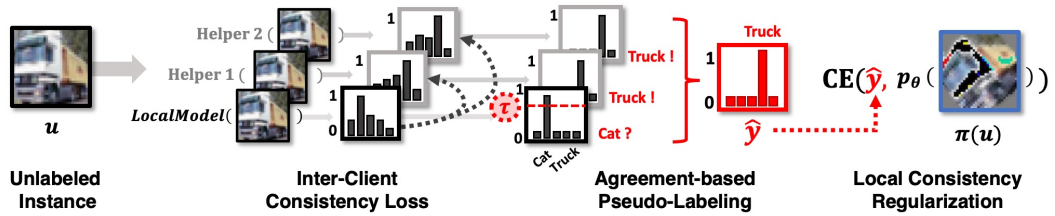


Figure 2: **Illustration of Inter-Client Consistency Loss.** This figure illustrates each step for Inter-Client Consistency regularization. We provide the detailed explanations in Section 3.1.

where $\pi(\mathbf{u})$ performs RandAugment (Cubuk et al., 2019) on unlabeld instance $\mathbf{u}$. $\hat{\mathbf{y}}$ is our novel pseudo-labeling technique, which we refer to as the *agreement-based* pseudo label, defined as follows:

$$\hat{\mathbf{y}} = \text{Max}(\mathbb{1}(p^*_{\boldsymbol{\theta}^l}(\mathbf{y}|\mathbf{u})) + \sum_{j=1}^{H} \mathbb{1}(p^*_{\boldsymbol{\theta}^{h_j}}(\mathbf{y}|\mathbf{u}))) \tag{3}$$

where $\mathbb{1}(\cdot)$ produces one-hot labels with given softmax values , and $\text{Max}(\cdot)$ outputs one-hot labels on the class that has the maximum agreements. We discard instances with low-confident predictions below confidence threshold τ when generating pseudo-labels. We then perform standard cross-entropy minimization with the pseudo-label $\hat{\mathbf{y}}$.

We select the H helper agents $p^*_{\boldsymbol{\theta}^{\mathrm{h}_{j:H}}}(\mathbf{y}|\mathbf{u})$ for each client as the most relevant models from other clients. Specifically, we represent each model by its prediction $\mathbf{m}$ on the same arbitrary input $\mathbf{a}$ located at server (we use random Gaussian noise), such that $\mathbf{m}^l = p_{\boldsymbol{\theta}^l}(\mathbf{m}|\mathbf{a})$. The server tries to keep and update all model embeddings $\mathbf{m}^{1:K}$ from clients once each client updates its weights to server, and creates *K-Dimensional Tree (KD Tree)* on $\mathbf{m}$ in the current round r for nearest neighbor search to rapidly select the H helper agents for each client in the next round $r+1$. If a certain client has not yet updated its weights to server in the previous step, then server simply sends random helpers to the client.

3.2 Parameter Decomposition for Disjoint Learning

In the standard semi-supervised learning approaches, learning on labeled and unlabeled data is simultaneously done with a shared set of parameters. However, since this is inapplicable to the disjoint learning scenario (Figure 1 (b)) and may result in forgetting of knowledge of labeled data (see Figure 6 (c)), we separate the supervised and unsupervised learning via the decomposition of model parameters into two sets of parameters, one for supervised learning and another for unsupervised learning. To this end, we decompose our model parameters θ into two variables, σ for supervised learning and ψ for unsupervised learning, such that $\theta = \sigma + \psi$. We perform standard supervised learning on σ, while keeping ψ fixed during training, by minimizing the loss term as follows:

$$\text{minimize } \mathcal{L}_s(\sigma) = \lambda_s \text{CrossEntropy}(\mathbf{y}, p_{\sigma+\psi^*}(\mathbf{y}|\mathbf{x})) \tag{4}$$

where $\mathbf{x}$ and $\mathbf{y}$ are from labeled set $\mathcal{S}$. For learning on unlabeled data, we perform unsupervised learning conversely on ψ, while keeping σ fixed for the learning phase, by minimizing the consistency loss terms as follows:

$$\text{minimize } \mathcal{L}_u(\psi) = \lambda_{\text{ICCS}} \Phi_{\sigma^*+\psi}(\cdot) + \lambda_{L_2}||\sigma^* - \psi||_2^2 + \lambda_{L_1}||\psi||_1 \tag{5}$$

where all λs are hyper-parameters to control the learning ratio between the terms. We additionally add L_2- and L_1-Regularization on ψ such that ψ is sparse, while not drifting far from knowledge that σ has learned. To sum up, our decomposition technique allows us:

- **Preservation Reliable Knowledge from Labeled Data**: We empirically find that learning on both labeled and unlabeled data with a single set of parameters may result in the model to forget about what it learned from the labeled data (see Figure 6 (c)). Our method can effectively prevent the *inter-task interference* via utilizing disjoint parameters only for supervised learning.
- **Reduction of Communication Costs**: Sparsity on the unsupervised parameter ψ allows to reduce communication cost. In addition, we further minimize the cost by subtracting the learned knowledge for each parameter, such that $\Delta\psi = \psi_r^l - \psi_r^G$ and $\Delta\sigma = \sigma_r^l - \sigma_r^G$, and transmit only the differences $\Delta\psi$ and $\Delta\sigma$ as sparse matrices for both client-to-server and server-to-client costs.

Algorithm 1 **Labels-at-Client Scenario**

RunServer()
initialize σ^0 and ψ^0
for each round $r = 1, 2, ..., R$ **do**
  $\mathcal{L}^r \leftarrow$ (select random A clients from $\mathcal{L}$)
  for each client $l_a^r \in \mathcal{L}^r$ **in parallel do**
    $\psi_{1:H}^r \leftarrow$ GetNearestNeighbors(ψ^r)
    $\sigma_a^r, \psi_a^r \leftarrow$ RunClient($\sigma^r, \psi^r, \psi_{1:H}^r$)
    EmbedLocalModel(σ_a^r, ψ_a^r)
  end for
  $\sigma^{r+1} \leftarrow \frac{N^{l_a}}{N} \sum_{a=1}^{A} (\sigma_{l_a}^r)$
  $\psi^{r+1} \leftarrow \frac{N^{l_a}}{N} \sum_{a=1}^{A} (\psi_{l_a}^r)$
end for
RunClient($\sigma, \psi, \psi_{1:H}$)
$\theta_{l_a} \leftarrow \sigma + \psi, \theta_{h_{1:H}} \leftarrow \sigma + \psi_{1:H}$
for each local epoch e from 1 to E_L **do**
  for minibatch $s \in \mathcal{S}_{l_a}$ and $u \in \mathcal{U}_{l_a}$ **do**
    $\theta_{\sigma+\psi^*} \leftarrow \theta_{\sigma+\psi^*} - \eta \nabla \ell_s(\theta_{\sigma+\psi^*}; \theta_{h_{1:H}}, s)$
    $\theta_{\sigma^*+\psi} \leftarrow \theta_{\sigma^*+\psi} - \eta \nabla \ell_u(\theta_{\sigma^*+\psi}; \theta_{h_{1:H}}, u)$
  end for
end for

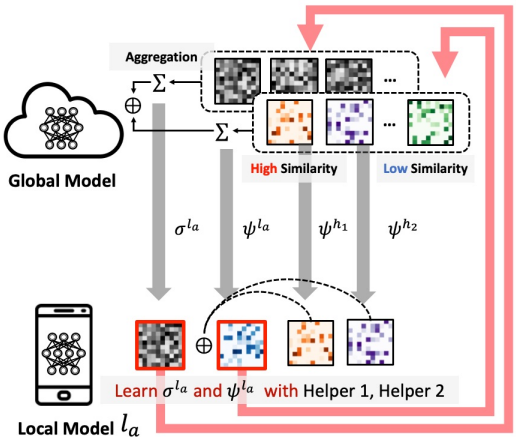


Figure 3: **Illustrative Running Example of Labels-at-Client Scenario** We describe training and communication procedure between local and global model under Labels-at-Client scenario corresponding to the Algorithm 1. More details are described in Section 4.2.

- **Disjoint Learning**: In federated semi-supervised learning, labeled data can be located at either client or server, which requires the model's learning procedure to be flexible. Our decomposition technique allows the model for the supervised training to be done separately elsewhere.

4 Labels-at-Client Scenario

4.1 Problem Definition

The *Labels-at-Client* scenario posits that the end-users intermittently annotate a small portion of their local data (i.e., 5% of the entire data), leaving the rest of data instances unlabeled as illustrated in Figure 1 (a). This is a common scenario for user-generated personal data, where the end-users can easily annotate the data but may not have time or motivation to label all the data (e.g. annotating faces in pictures for photo albums or social networking). We assume that clients train on both labeled and unlabeled data, while the server only aggregates the updates from the clients and redistributes the aggregated parameters back to the clients. In this scenario, labeled data $\mathcal{S}$ is a set of individual sub-datasets $\mathcal{S}^{l_k} = \{\mathbf{x}_i^{l_k}, \mathbf{y}_i^{l_k}\}_{i=1}^{S^{l_k}}$, yielding K sub-datasets for K local models $l_{1:K}$. The overall learning procedure of the global model is the same as that of conventional federated learning (global model G aggregates updates from the selected subset of clients and broadcasts them), except that active local models $l_{1:A}$ perform semi-supervised learning by minimizing the loss $\ell_{final}(\boldsymbol{\theta}^{l_a}) = \ell_s(\boldsymbol{\theta}^{l_a}) + \ell_u(\boldsymbol{\theta}^{l_a})$ respectively on $\mathcal{S}^{l_a}$ and $\mathcal{U}^{l_a}$.

4.2 FedMatch Algorithm for Labels-at-Client Scenario

Now we introduce our *FedMatch* algorithm for the labels-at-client scenario. As shown in Figure 3, which illustrates an example case of the labels-at-client scenario, active local models $l_{1:A}$ at the current round r learn both $\sigma_{1:A}^{l_r}$ and $\psi_{1:A}^{l_r}$ on both the labeled data $\mathcal{S}^{l_{1:A}}$ and unlabeled data $\mathcal{U}^{l_{1:A}}$ at each local environment. After the completion of local training, the clients update both their learned knowledge $\sigma_{1:A}^{l_r}$ and $\psi_{1:A}^{l_r}$ to the server. The server then aggregates $\sigma^{l_{1:A}}$ and $\psi^{l_{1:A}}$, *respectively*, after embedding local models based on model similarity as well as create KD-Tree to rapidly retrieve the top-H nearest neighbors $\psi^{h_{1:H}}$ for each client. At the next round, the server transmits the aggregated σ^{r+1} and ψ^{r+1}, as well as the H selected $\psi^{h_{1:H}}$ to each client connected at the next round. More details of the training procedures for FedMatch, for the labels-at-client scenario, is described in Algorithm 1.

Algorithm 2 **Labels-at-Server Scenario**

RunServer()
initialize σ^0,ψ^0
for each round $r = 1, 2, ..., R$ **do**
  for each server epoch e from 1 to E_G **do**
    for minibatch $s \in \mathcal{S}_G$ **do**
      $\theta_{\sigma+\psi^*} \leftarrow \theta_{\sigma+\psi^*} - \eta \nabla \ell_s(\theta_{\sigma+\psi^*}; s)$
    end for
  end for
  $\mathcal{L}^r \leftarrow$ (select random A clients from $\mathcal{L}$)
  for each client $l_a^r \in \mathcal{L}^r$ **in parallel do**
    $\psi_{1:H}^r \leftarrow$ GetNearestNeighbors(ψ^r)
    $\psi_a^r \leftarrow$ RunClient(σ^{r+1}, ψ^r, $\psi_{1:H}^r$)
    EmbedLocalModel(σ^{r+1}, ψ_a^r)
  end for
  $\psi^{r+1} \leftarrow \frac{N^{l_a}}{N} \sum_{a=1}^{A} (\psi_{l_a}^r)$
end for
RunClient($\sigma, \psi, \psi_{1:H}$)
$\theta_l \leftarrow \sigma^* + \psi$, $\theta_{h_{1:H}} \leftarrow \sigma^* + \psi_{1:H}$
for each local epoch e from 1 to E_L **do**
  for minibatch $u \in \mathcal{U}_{l_a}$ **do**
    $\theta_{\sigma^*+\psi} \leftarrow \theta_{\sigma^*+\psi} - \eta \nabla \ell_u(\theta_{\sigma^*+\psi}; \theta_{h_{1:H}}, u)$
  end for
end for

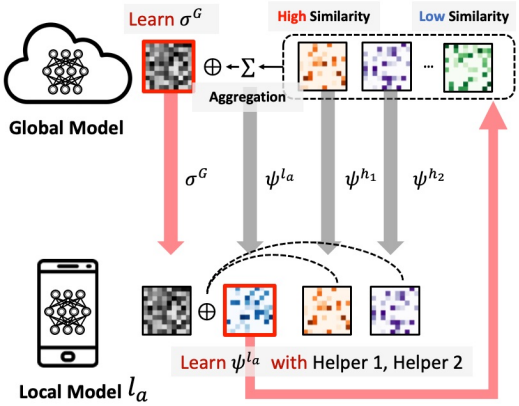


Figure 4: **Illustrative Running Example of Labels-at-Server Scenario** We depict learning and transmitting procedure between a client and the global server under Labels-at-Server scenario corresponding to the Algorithm 2. Further details are explained in Section 5.2

5 Labels-at-Server Scenario

5.1 Problem Definition

We now describe another realistic setting, which is the labels-at-server scenario. This scenario assumes that the supervised labels are only available at the server, while local clients work with unlabeled data as described in Figure 1 (b). This is a common case of real-world applications where labeling requires expert knowledge (e.g. annotating medical images, evaluating body postures for exercises), but the data cannot be shared due to privacy concerns. In this scenario, $\mathcal{S}^G$ is identical to $\mathcal{S}$ and is located at server. The overall learning procedure is the same as that of conventional federated learning, except the global model G performs supervised learning on $\mathcal{S}^G$ by minimizing the loss $\ell_s(\boldsymbol{\theta}^G)$ before broadcasting $\boldsymbol{\theta}^G$ to local clients. Then, the active local clients $l_{1:A}$ at communication round r perform unsupervised learning which solely minimizes $\ell_u(\boldsymbol{\theta}^{l_a})$ on the unlabeled data $\mathcal{U}^{l_a}$.

5.2 FedMatch Algorithms for Labels-at-Server Scenario

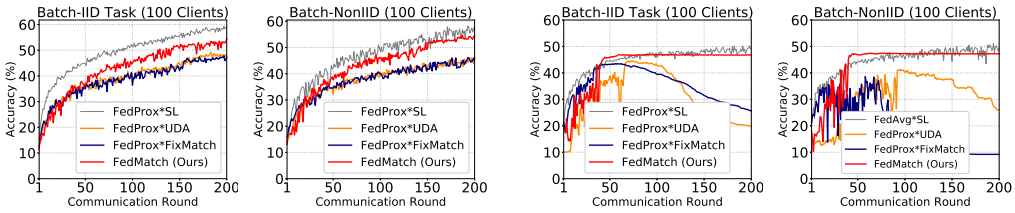
We now describe our *FedMatch* algorithm for the labels-at-server scenario. As depicted in Figure 4, which describes an illustrative running example for labels-at-server scenario, the global model G learns σ on labeled data $\mathcal{S}^G$ at the server and the active local clients $l_{1:A}$ at the current round r learn $\psi^{l_{1:A}}$ on unlabeled data $\mathcal{U}^{1:A}$ at each local environment. After the completion of local training, clients update their learned knowledge $\psi^{l_{1:A}}$ to the server. The server then embeds local models based on model similarity and create a KD-Tree for rapid nearest neighbor search for the top-H most similar $\psi^{h_{1:H}}$ models for each client. At the next round, server transmits its learned σ^{r+1} and the aggregated ψ^{r+1}, as well as top-H similar $\psi^{h_{1:H}}$ to each client connected at the next communication round. Further training details of FedMatch for the labels-at-server scenario is described in Algorithm 2.

6 Experiments

We now experimentally validate our method, *FedMatch*, on three tasks, such as Batch-IID, Batch-NonIID, and Streaming-NonIID, under both scenarios, *Labels-at-Client* and *Labels-at-Server*.

Table 1: **Model Performance on Batch-IID & NonIID Tasks** We use 100 clients (F=0.05) for 200 rounds. We measure global model accuracy and averaged communication costs (both client-to-server and server-to-client). Note that the SL (Supervised Learning) models learn on both $\mathcal{S}$ and $\mathcal{U}$ with full labels, and are utilized as the upper bounds for each experiment.

CIFAR-10, Batch-IID Task with 100 Clients (K=100, F=0.05, H=2)						
	Labels-at-Client Scenario			*Labels-at-Server* Scenario		
Methods	**Acc.(%)**	**C2S Cost**	**S2C Cost**	**Acc.(%)**	**C2S Cost**	**S2C Cost**
FedAvg-SL	58.60 ± 0.42	100 %	100 %	52.45 ± 0.23	100 %	100 %
FedProx-SL	59.30 ± 0.31	100 %	100 %	49.11 ± 0.38	100 %	100 %
FedAvg-UDA	46.35 ± 0.29	100 %	100 %	24.81 ± 0.73	100 %	100 %
FedProx-UDA	47.45 ± 0.21	100 %	100 %	19.91 ± 0.31	100 %	100 %
FedAvg-FixMatch	47.01 ± 0.43	100 %	100 %	11.95 ± 0.60	100 %	100 %
FedProx-FixMatch	47.20 ± 0.12	100 %	100 %	25.61 ± 0.32	100 %	100 %
FedMatch (Ours)	**53.51 ± 0.38**	**29 %**	**142 %**	**46.81 ± 0.24**	**49 %**	**43 %**
CIFAR-10, Batch-NonIID Task with 100 Clients (K=100, F=0.05, H=2)						
FedAvg-SL	55.15 ± 0.21	100 %	100 %	51.50 ± 0.51	100 %	100 %
FedProx-SL	57.75 ± 0.15	100 %	100 %	49.31 ± 0.18	100 %	100 %
FedAvg-UDA	44.35 ± 0.39	100 %	100 %	27.61 ± 0.71	100 %	100 %
FedProx-UDA	46.31 ± 0.63	100 %	100 %	26.01 ± 0.78	100 %	100 %
FedAvg-FixMatch	46.20 ± 0.52	100 %	100 %	09.45 ± 0.34	100 %	100 %
FedProx-FixMatch	45.55 ± 0.63	100 %	100 %	09.21 ± 0.24	100 %	100 %
FedMatch (Ours)	**54.26 ± 0.17**	**28 %**	**134 %**	**47.11 ± 0.28**	**53 %**	**67 %**



(a) Labels-at-Client Scenario (b) Labels-at-Server Scenario

Figure 5: **Test Accuracy Curves on Batch-IID & NonIID Tasks** We visualize test accuracy curves of model performance corresponding to the Table 1. Note that the SL (Supervised Learning) models learn on both $\mathcal{S}$ and $\mathcal{U}$ with full labels, and are utilized as the upper bounds for each experiment.

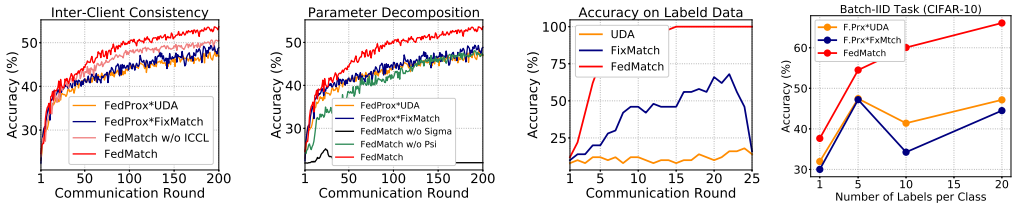
6.1 Experimental Setup

Tasks **1) Batch-IID**: We use CIFAR-10 for this task and split $60,000$ instances into training $(54,000)$, valid $(3,000)$, and test $(3,000)$ sets. We extract 5 labeled instances per class (C=10) for each client (K=100) as labeled set $\mathcal{S}$, and the rest of instances $(49,000)$ are used as unlabeled data $\mathcal{U}$, so that we can evenly split $\mathcal{S}$ and $\mathcal{U}$ into $\mathcal{S}^{l_{1:100}}$ and $\mathcal{U}^{l_{1:100}}$, such that local models $l_{1:100}$ learn on corresponding labeled and unlabeled data during training. **2) Batch-NonIID (class-imbalanced)**: The setting of this task is mostly the same with the Batch-IID task, except we arbitrarily control the distribution of the number of instances per class for each client to simulate *class-imbalanced* environments. **3) Streaming-NonIID (class-imbalanced)**: In this task, data streams into each client from class-imbalanced distributions. We use Fashion-MNIST dataset for this task, and split $70,000$ instances into training $(63,000)$, valid $(3,500)$, and test $(3,500)$ sets. From train set, we extract 5 labeled instances per class (C=5) for each client (K=10) for a labeled set $\mathcal{S}$. We discard labels for the rest of instances to construct an unlabeled set $\mathcal{U}$ $(62,000)$. Then, we split $\mathcal{S}$ and $\mathcal{U}$ into $\mathcal{S}^{l_{1:100}}$ and $\mathcal{U}^{l_{1:100}}$ based on a class-imbalanced distribution. For individual local unlabeled data $\mathcal{U}^{l_k}$, we again split all instances into $\mathcal{U}_t^{l_k}$, $t \in \{1, 2, ..., T\}$, where T is the number of total streaming steps (we set T=10). We train each streaming step for 10 rounds. We describe above tasks under Labels-at-Client scenario. For Labels-at-Server scenario, S is simply located at server without any partition. Please see Figure 7 in Appendix, which we visualize the concepts of dataset configuration.

Baselines and Training Details Our baselines are: **1) Local-SL**: local supervised learning (SL) with full labels $(\mathcal{S} + \mathcal{U})$ without sharing locally learned knowledge. **2) Local-UDA** and **3) Local-FixMatch**: local semi-supervised learning, including UDA and FixMatch, without sharing local knowledge. **4) FedAVG-SL** and **5) FedProx-SL**: supervised learning with full labels $(\mathcal{S} + \mathcal{U})$ while

Table 2: **Averaged Local Performance on Streaming-NonIID Task** We use 10 clients (F=1.0) for 10 rounds per streaming step (100 rounds in total). We measure averaged local model accuracy and communication costs. Note that the SL (Supervised Learning) models learn on both $\mathcal{S}$ and $\mathcal{U}$ with full labels, and are utilized as the upper bounds for each experiment.

Fashion-MNIST, Streaming-NonIID Task with 10 Clients (K=10, F=1.0, H=2)						
	Labels-at-Cleint Scenario			*Labels-at-Server* Scenario		
Methods	**Acc.(%)**	**C2S Cost**	**S2C Cost**	**Acc.(%)**	**C2S Cost**	**S2C Cost**
Local-SL	87.19 $\pm$ 0.36	N/A	N/A	N/A	N/A	N/A
Local-UDA	70.70 $\pm$ 0.28	N/A	N/A	N/A	N/A	N/A
Local-FixMatch	62.62 $\pm$ 0.32	N/A	N/A	N/A	N/A	N/A
FedProx-SL	82.06 $\pm$ 0.26	100 %	100 %	77.43 $\pm$ 0.42	100 %	100 %
FedProx-UDA	73.71 $\pm$ 0.17	100 %	100 %	83.34 $\pm$ 0.21	100 %	100 %
FedProx-FixMatch	62.40 $\pm$ 0.43	100 %	100 %	73.71 $\pm$ 0.32	100 %	100 %
FedMatch (Ours)	**75.85** $\pm$ **0.14**	**32 %**	**15 %**	**85.15** $\pm$ **0.21**	**65 %**	**25 %**



(a) Inter-Client Consistency (b) Sigma & Psi (c) Inter-Task Interference (d) Number of Labels

Figure 6: **Ablation Study and Additional Analysis on FedMatch Algorithm** We study effectiveness of each components of our method, (a) *inter-client consistency loss* and (b) *parameter decomposition*. (c) We effectively tackle the *inter-task interference*. (d) Performance improvement of our method when labeled data is increased.

sharing local knowledge via FedAvg and FedProx frameworks. **6) FedAvg-UDA** and **7) FedProx-UDA**: naive combinations of FedAvg/Prox with UDA. **8) FedAvg-FixMatch** and **9) FedProx-FixMatch**: naive combination of with FixMatch with FedAvg/Prox. For training, we use SGD with adaptive-learning rate decay introduced in (Serra et al., 2018) with the initial learning rate $1e{-}3$. We use **ResNet-9** networks as the backbone architecture for all baselines and our methods. We ensure that all hyper-parameters are set equally for all base models and ours to perform fair evaluation and comparison. Please see the **Section B** in the Appendix for further details. For all experiments, we report the mean and the standard deviation over 3 runs.

6.2 Experimental Results

Results on Batch-IID & NonIID Tasks Table 1 shows performance comparison of our models and naive Fed-SSL algorithms on Batch-IID and NonIID tasks under the two different scenarios. We observe that our model outperforms all naive Fed-SSL baselines for all tasks and scenarios. In particular, under labels-at-server scenario, which is more challenging than labels-at-client scenario, we observe that the naive combination models significantly suffer from the forgetting issue and their performances keeps deteriorating after a certain communication round. This phenomenon is mainly caused by the base models failing to properly perform disjoint learning, in which case the learned knowledge from the labeled and unlabeled data causes inter-task interference. Contrarily, our methods show consistent and robust performance regardless where the labeled data exists, which shows that our decomposition techniques effectively handles the challenging disjoint learning scenario. In addition, when the class-wise distribution is imbalanced for each client (Non-IID task), we observe that the base models' performance slightly drops by 1-3$\%p$, while our methods show consistent or slightly better performance. This shows that our inter-client consistency effectively enhances consistency with the helper agents selected from server based on model similarity, which is good at, in particular, class-imbalanced tasks. We also visualize the test accuracy curve for our models and naive Fed-SSL in Fig. 5. Our method (Red line) trains faster and consistently outperforms the base models, and is most robustness against inter-task interference in both scenarios. For analysis on the averaged communication costs, please see Section C.3 in the Appendix.

Results on Streaming-NonIID Task Table 2 shows averaged model performance on Streaming-NonIID tasks with 10 synchronized clients. In the labels-at-client scenario, there is no big differences

of performance between local SSL and Fed-SSL models, while our model outperforms all the base models with large margins in a range of 2-10$\%p$ (except SL models). This implies that our method effectively utilizes inter-client and reliable knowledge than naive Fed-SSL models do. In the labels-at-server scenario, interestingly, the performance of FedProx-SL decreases by around $5\%p$, while Fed-SSL models obtain improved performance. We conjecture that this is because for streaming situation the model may not sufficiently train on the new data, while Fed-SSL models overcomes it by utilizing only the consistent pseudo-labels. Even on this task, FedMatch outperforms all baselines with significantly smaller communication cost on average (see Section C.3).

Effectiveness of Inter-Client Consistency To show the effectiveness of our inter-client consistency loss, we eliminate the loss, while learning on Batch-IID task with 100 clients (F=0.05). In Figure 6 (a), when we remove our inter-client consistency loss, we observe that the performance has slightly dropped (Pink line) from one with the loss term (Red line). This gap clearly tells us that our inter-client consistency loss improves model consistency across multiple models while keeping reliable knowledge. Interestingly, our model without inter-client consistency loss still outperforms base models. This additionally implies that our another proposed method, parameter decomposition for disjoint learning, also effectively enhances model performance.

Effectiveness of Parameter Decomposition Our model with parameter decomposition alone, without inter-client consistency loss, outperforms base models. For further analysis, we show the effect of each decomposed variables, σ and ψ, in Figure 6 (b). Removing either of σ and ψ results in substantial drop in the performance, with larger performance degeneration when dropping σ, which captures much more essential knowledge from labeled data (Green line). Such decomposition is effective since there exists knowledge interference between supervised learning and unsupervised learning. We show this with an experiment where we perform semi-supervised learning with 5 labels per class and $1,000$ unlabeled instances for 100 rounds. We measure accuracy on the labeled set at each training steps. As shown in Figure 6 (c), our method effectively preserves learned knowledge from labeled set, while other base models suffer from knowledge interference. This effective separation of supervised and unsupervised learning tasks enhances the overall performance of our methods even without inter-client consistency loss as shown in Figure 6 (a) (shown in pink).

Number of Labels per Class We increase the number of labels per class for each client in a range of 1, 5, 10, and 20 on Batch-IID task (CIFAR-10). Our method shows consistent performance improvement as the number of labels increases. Interestingly, we observe that baseline models, FedProx-UDA/FixMatch, show performance degradation even when the labeled data increases ($5 \rightarrow 10$). These results show that our method effectively utilize knowledge from labeled and unlabeled data in federated semi-supervised learning settings, while other naive combinations of FSSL could fail to learn properly from labeled and unlabeled data in federated learning framework.

7 Related Work

Federated Learning A variety of approaches for averaging local weights at server have been introduced in the past few years. FedAvg (McMahan et al., 2017) performs weighted-averaging on local weights according to the local train size. FedProx (Li et al., 2018) uniformly averages the local updates while clients perform proximal regularization against the global weights, while FedMA (Wang et al., 2020) matches the hidden elements with similar feature extraction signatures in layer-wise manner when averaging local weights. PFNM (Yurochkin et al., 2019) introduces aggregation policy which leverages Bayesian non-parametric methods. Beyond focusing on averaging local knowledge, there are various efforts to extend FL to the other areas, such as continual learning under federated learning frameworks (Yoon et al., 2020a) inspired by parameter decomposition techniques proposed by (Yoon et al., 2020b). Recently, interests of tackling scarcity of labeled data in FL are emerging and discussed in (Jin et al., 2020; Guha et al., 2019; Albaseer et al., 2020).

Semi-Supervised Learning While there exist numerous work on SSL, we mainly discuss consistency regularization approaches. Consistency regularization techniques(Sajjadi et al., 2016) assume that the class semantics will not be affected by transformations of the input instances, and enforce the model output to be the same across different input perturbations. Some extensions to this technique perturb inputs adversarially (Miyato et al., 2018), through dropout (Srivastava et al., 2014), or

through data augmentation (French et al., 2018). UDA (Xie et al., 2019) and ReMixMatch (Berthelot et al., 2019a) use two sets of augmentations, weak and strong, and enforce consistency between the weakly and strongly augmented examples. Recently, in addition to enforcing consistency between weak-strong augmented pairs, FixMatch (Sohn et al., 2020) performs pseudo-label refinement on model predictions via thresholding. Entropy minimization (Grandvalet & Bengio, 2004) which enforces the classifier to predict low-entropy on unlabeled data, is another popular technique for SSL. Pseudo-Label (Lee, 2013) constructs one-hot labels from highly confident predictions on unlabeled data and uses these as training targets inn a standard cross-entropy loss. MixMatch (Berthelot et al., 2019c) performs "sharpening" on target distribution on unlabeled data, to further refine the generated pseudo-label.

8 Conclusion

In this work, we introduced two practical scenarios of *Federated Semi-Supervised Learning* (FSSL) where each client learns with only partly labeled data (*Labels-at-Client* scenario), or supervised labels are only available at the server, while clients work with completely unlabeled data (*Labels-at-Server* scenario). To tackle the problem, we propose a novel method, *Federated Matching* (FedMatch), which introduces the *inter-client consistency loss* that aims to maximize the agreement between the models trained at different clients, and the *parameter decomposition for disjoint learning* which decomposes the parameters into one for labeled data and the other for unlabeled data for preservation of reliable knowledge, reduction of communication costs, and disjoint learning. Through extensive experimental validation, we show that FedMatch significantly outperforms both local semi-supervised learning methods and naive combinations of federated learning algorithms with semi-supervised learning on diverse and realistic scenarios. As future work, we plan to further improve our model to tackle the scenario where pretrained models deployed at each client adapts to a completely unlabeled data stream (e.g. on-device learning of smart speakers).

References

Abdullatif Albaseer, Bekir Ciftler, Mohamed Abdallah, and Ala Al-Fuqaha. Exploiting unlabeled data in smart cities using federated learning. 01 2020.

David Berthelot, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019a.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019b.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems 32*, pp. 5049–5059. Curran Associates, Inc., 2019c.

Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation. *arXiv preprint arXiv:1903.07424*, 2019a.

Yujing Chen, Yue Ning, and Huzefa Rangwala. Asynchronous online federated learning for edge devices. *arXiv preprint arXiv:1911.02134*, 2019b.

Muhammad EH Chowdhury, Tawsifur Rahman, Amith Khandakar, Rashid Mazhar, Muhammad Abdul Kadir, Zaid Bin Mahbub, Khandaker Reajul Islam, Muhammad Salman Khan, Atif Iqbal, Nasser Al-Emadi, et al. Can ai help in screening viral and covid-19 pneumonia? *arXiv preprint arXiv:2003.13145*, 2020.

Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019. URL http://arxiv.org/abs/1909.13719.

Geoff French, Michal Mackiewicz, and Mark Fisher. Self-ensembling for visual domain adaptation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkpoTaxA-.

Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pp. 529–536, Cambridge, MA, USA, 2004. MIT Press.

Neel Guha, Ameet Talwlkar, and Virginia Smith. One-shot federated learning. 02 2019.

Yilun Jin, Xiguang Wei, Yang Liu, and Qiang Yang. A survey towards federated semi-supervised learning. 02 2020.

Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

Takeru Miyato, Shin ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. PAMI, 2018.

Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1163–1171. Curran Associates, Inc., 2016.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4548–4557, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/serra18a.html.

Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkluqlSFDS.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training. 2019.

Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *arXiv preprint arXiv:2003.03196*, 2020a.

Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=r1gdj2EKPB.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. 2019.

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.

A APPENDIX

Organization The appendix is organized as follows: We further explain detailed experimental setups, such as our baselines, model architecture utilized for all experiments, and the training configurations, in Section B. We also provide additional experimental results and analysis in Section C.

B EXPERIMENTAL DETAILS

We now further describe our experimental setups in detail, such as our baseline models, network architecture that is used for all base models and our method, and the detailed training setups.

B.1 BASELINE MODELS

We consider UDA (Xie et al., 2019) and FixMatch (Sohn et al., 2020) as our baselines, since they are state-of-the-art SSL models and are based on the consistency-based mechanisms that are conceptually similar to our inter-client consistency loss. We reimplement UDA with the Training Signal Annealing (TSA) and exponential scheduling for its best performance as reported in their paper (we use RandAugment (Cubuk et al., 2019) for consistency regularization with random magnitude). We also reimplement FixMatch algorithms with strong augmentation as RandAugment (Cubuk et al., 2019). For weak augmentation (filp-and-shift), however, as the performance has significantly dropped when we apply the weak augmentation, we use original images rather than weakly augmenting the images. We fix confidence threshold τ=0.85 for all FixMatch and our model experiments. For federated learning frameworks, we use FedAvg (McMahan et al., 2017) and FedProx (Li et al., 2018) algorithms since they are the standard baselines for federated learning and can be easily combined with the SSL baselines. Detailed hyper-parameter settings are described in Table 4.

B.2 NETWORK ARCHITECTURE

We build ResNet-9 networks as our base architecture for all base model and our method. In the architecture, the first two convolutional neural layers have 64 and 128 filters and the same 3×3 kernel sizes followed by 2×2 max-pooling layer. Then we have a skip connection between the subsequent two convolution layers with 128 filters. We then double the filter size from 128 to 256 with the next conv layer and down-sample via the following 2×2 max-pooling layer. We repeat the previous step, such that we have 512 filter size and 4×4 kernel size. Then, we perform another skip connection through the two subsequent conv layers with 512 filters. As a final step, we down-sample the kernel size from 4×4 to 1×1, then perform softmax classifier with the last fully connected layer. All layers are equally initialized based on the varaiance scalining method. The model architecture is described in Table 3.

Table 3: **Model Architecture of ResNet-9 Networks**

Layer	Filter Shape	Stride	Output
Input	N/A	N/A	$32 \times 32 \times 3$
Conv 1	$3 \times 3 \times 3 \times 64$	1	$32 \times 32 \times 64$
Conv 2	$3 \times 3 \times 64 \times 128$	1	$32 \times 32 \times 128$
Pool 1	2×2	2	$16 \times 16 \times 128$
Conv 3	$3 \times 3 \times 128 \times 128$	1	$16 \times 16 \times 128$
Conv 4	$3 \times 3 \times 128 \times 128$	1	$16 \times 16 \times 128$
Conv 5	$3 \times 3 \times 128 \times 256$	1	$16 \times 16 \times 256$
Pool 2	2×2	2	$8 \times 8 \times 256$
Conv 6	$3 \times 3 \times 256 \times 512$	1	$8 \times 8 \times 512$
Pool 3	2×2	2	$4 \times 4 \times 512$
Conv 7	$3 \times 3 \times 512 \times 512$	1	$4 \times 4 \times 512$
Conv 8	$3 \times 3 \times 512 \times 512$	1	$4 \times 4 \times 512$
Pool 4	4×4	4	$1 \times 1 \times 512$
Softmax	512×10	N/A	$1 \times 1 \times 10$

B.3 TRAINING DETAILS

We use Stochastic Gradient Descent (SGD) to optimize our model with initial learning rate 1e-3. We also adopt adaptive learning rate decay which is introduced by (Serra et al., 2018). The learning rate strategy gradually reduces the learning rate by a factor of 3 for every 3 epochs that validation loss does not consecutively decreases. We use L2 weight decay regularization on the base architecture with L2 factor to be 1e-4. All hyper-parameters and other training setups are equally set for fair comparison as shown in Table 4. In the table, we denote LPC as number of labels per class for each client (or at server). $B^{\mathcal{S}}$ and $B^{\mathcal{U}}$ denote batch-size of labeled set $\mathcal{S}$ and unlabeled set $\mathcal{U}$. μ is a

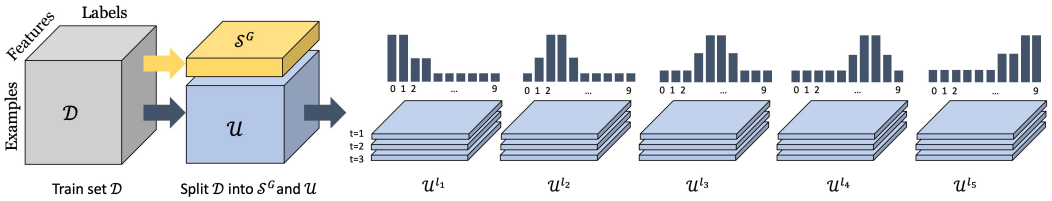
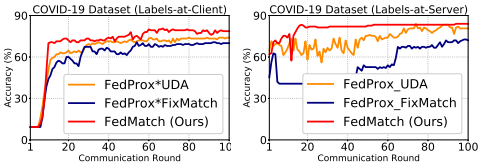


Figure 7: **Illustration of Dataset Partition for Experimental Tasks** We split the dataset $\mathcal{D}$ into a set of labeled data $\mathcal{S}$ and a set of unlabeled data $\mathcal{U}$. U is divided into K subsets which are distributed to K clients (Batch Task). For *streaming* tasks, we further split all instances in each subset into T subsets for T streaming steps. For *class-imbalanced* tasks, we additionally control the number of instances per class for each client.

Table 4: **Hyper-Parameters & Training Setups** We provide all hyper-parameters and training setups for all baseline models and our method. For λ_s, we set 10 for CIFAR-10, while we set 1 for Fashion-MNIST dataset.

Labels-at-Client Scenario												
Methods	lr	wd	λ_s	λ_u	λ_{ICCS}	λ_{L_1}	λ_{L_2}	LPC	$B^{\mathcal{S}}_{\text{client}}$	$B^{\mathcal{U}}_{\text{client}}$	$B^{\mathcal{S}}_{\text{server}}$	μ
SL	1e-3	1e-4	10	-	-	-	-	5	10	100	-	1e-2
UDA	1e-3	1e-4	10	1	-	-	-	5	10	100	-	1e-2
FixMatch	1e-3	1e-4	10	1	-	-	-	5	10	100	-	1e-2
FedMatch	1e-3	1e-4	10	-	1e-2	1e-4	10	5	10	100	-	1e-2
Labels-at-Server Scenario												
SL	1e-3	1e-4	10	-	-	-	-	100	100	100	100	1e-2
UDA	1e-3	1e-4	10	1	-	-	-	100	100	100	100	1e-2
FixMatch	1e-3	1e-4	10	1	-	-	-	100	100	100	100	1e-2
FedMatch	1e-3	1e-4	10	-	1e-2	1e-4	10	100	100	100	100	1e-2

COVID-19 Radiography Dataset		
	Labels-at-Client	Labels-at-Server
Methods	**Acc.(%)**	**Acc.(%)**
F.Prx-UDA	74.24 $\pm$ 0.25	80.11 $\pm$ 0.18
F.Prx-FixMtch	70.02 $\pm$ 0.28	72.15 $\pm$ 0.14
FedMatch	**78.67** $\pm$ **0.23**	**84.32** $\pm$ **0.11**



(a) Labels-at-Client (b) Labels-at-Server

Figure 8: **Experimental Results on COVID-19 Radiography Dataset. Left**: Performance comparison of our method (FedMatch) with the naive federated semi-supervised learning algorithms (FedProx-UDA/FixMatch). **Right**: Test accuracy curves corresponding to the left performance table. Our method trains stably and consistently outperforms all base models.

hyper-parameter for FedProx framework. We additionally provide visual illustration of our dataset configuration. Please see Figure 7.

C Additional Experiments & Analysis

In this section, we additionally provide more experimental results and analysis, such as the additional experiment on real-world dataset, the analysis of the communication costs of FedMatch algorithms, experiments with the different backbone networks, and the analysis on the fraction of available clients per communication round.

C.1 Experiments on Real-world Dataset

Even though the majority of federated learning algorithms have been validated on synthetic datasets due to limited accessibility toward real-world or industrial datasets, validation on such datasets is indeed very important for the potential practical usages in real-world applications. To show our method consistently work with real-world dataset, particularly, in the medical field, we have decided to experiment on COVID-19 Radiography Dataset (Chowdhury et al., 2020). The COVID-19 dataset contains 219 instances for *COVID-19*, 1341 instances for *Normal*, and 1341 instances for *Viral Pneumonia* images. We use 10 clients with fraction 1.0. We use 5 labels per class for each client. We

Table 6: **Model Performance Comaprison utilizing AlexNet-Like architecture** We use 100 clients for 100 rounds for streaming task and 200 rounds for batch tasks. We measure global model accuracy, while varying experimental settings (i.e. fraction of available clients and the accessibility of labeled data).

Experiments based on AlexNet-Like Architecture					
	Streaming-NonIID (F=1.0)		**Batch-IID** (Labels-at-Client)		
	Labels-at-Client	Labels-at-Server	F=0.05	F=0.10	F=0.20
Methods	**Acc.(%)**	**Acc.(%)**	**Acc.(%)**	**Acc.(%)**	**Acc.(%)**
FedAvg-SL	68.20 $\pm$ 0.29	70.51 $\pm$ 0.11	47.23 $\pm$ 0.31	47.87 $\pm$ 0.73	48.73 $\pm$ 0.15
FedProx-SL	68.47 $\pm$ 0.13	70.55 $\pm$ 0.72	47.54 $\pm$ 0.28	48.01 $\pm$ 0.17	49.20 $\pm$ 0.64
FedAvg-UDA	52.25 $\pm$ 0.04	46.28 $\pm$ 0.32	35.27 $\pm$ 0.29	35.20 $\pm$ 0.53	36.21 $\pm$ 0.12
FedProx-UDA	52.84 $\pm$ 0.15	46.35 $\pm$ 0.31	34.94 $\pm$ 0.46	36.67 $\pm$ 0.73	35.80 $\pm$ 0.43
FedAvg-FixMatch	57.09 $\pm$ 0.89	52.67 $\pm$ 0.78	32.33 $\pm$ 0.51	36.27 $\pm$ 0.33	37.61 $\pm$ 0.05
FedProx-FixMatch	57.12 $\pm$ 0.41	51.51 $\pm$ 0.32	36.83 $\pm$ 0.23	36.37 $\pm$ 0.39	37.40 $\pm$ 0.18
FedMatch (Ours)	**63.84** $\pm$ **0.18**	**59.12** $\pm$ **0.35**	**41.67** $\pm$ **0.32**	**41.97** $\pm$ **0.14**	**42.18** $\pm$ **0.27**

compare our method with naive combinations of semi-supervised learning and federated learning models during training 100 rounds. As shown in the left table of Figure 8, our method consistently outperforms all base models with large margins (around $4\%p$-$10\%p$) in both scenarios. The test accuracy curves in Figure 8, we can see that our method trains faster than the base models and shows more stability during training. Theses results are clear evidence that our method has strong potential toward a variety of practical usages for real-world applications.

C.2 Further Analysis of the Number of Labels per Class

Although we explained our analysis for the Number of Labels per Class under Section 6.2, we further provide additional experimental evidence. We conducted experiments of our method without the decomposition technique. As shown in Table 5, our method without decomposition technique (indicated as FedMatch (w/o)) shows not much performance improvement when the number of labels per class increases from 5 to 10 (around 3.x$\%p$) than from 1 to 5 (around 9.x$\%p$) or from 10 to 20 (10.x$\%p$), which are the similar tendency with the baseline models. However, our method with the decomposition technique shows consistent performance improvement. These results can be clear evidence that our decomposition technique has the effectiveness to preserve reliable knowledge in federated semi-supervised learning scenarios.

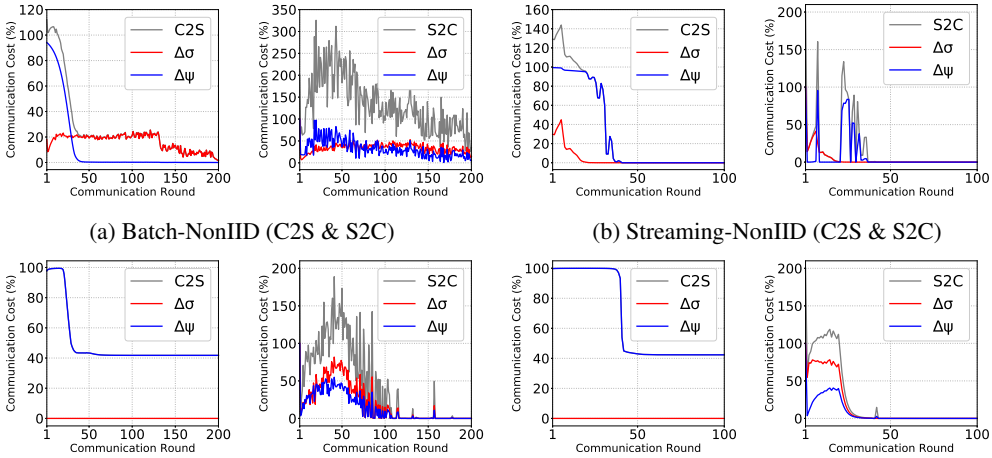
Table 5: **Analysis of the Number of Labels per Class**

	Number of Labeled Examples per Class			
-	**1**	**5**	**10**	**20**
Methods	**Acc.(%)**	**Acc.(%)**	**Acc.(%)**	**Acc.(%)**
FedPrx*UDA	31.95	47.45	41.4	47.15
FedPrx*FxMtch	30.01	47.2	34.25	44.5
FedMatch (w/o)	37.7	47.51	51.15	62.7
FedMatch	**37.65**	**54.5**	**60.65**	**66.1**

C.3 The Efficient Communication of FedMatch

Since the actual bit-level compression techniques are rather implementation issues, which are beyond our research scope, we only consider the reduction of *the amount of information* that needs to be transmitted between the server and the client. To minimize the communication costs, we not only learn ψ to be sparse, but also subtract the parameters between server and client, such that $\Delta\psi = \psi_r^l - \psi_r^G$ and $\Delta\sigma = \sigma_r^l - \sigma_r^G$, then send only the difference, $\Delta\psi$ and $\Delta\sigma$, as sparse matrices from both directions of server-to-client (S2C) and client-to-server (C2S). As shown in Figure 9, we observe that both the C2S and S2C costs are gradually decreased during the learning phases. For $\Delta\sigma$, σ could be in general rapidly converged in earlier steps, as both the amount of the labeled data is relatively small (i.e. 5 instances per class) and the knowledge from the labeled data can be effectively captured by σ without inter-task interference. Even though we learn σ to be dense, the fast convergence of σ makes the difference $\Delta\sigma$ significantly small as training steps proceed in both two different direction (C2S and S2C). In particular, we do not even need to send $\Delta\sigma$ to server under the labels-at-server scenario (Figure 9 C2S cost of $\Delta\sigma$ in (c) and (d)).

For $\Delta\psi$, ψ learns knowledge from unlabeled data, and thus it captures relatively more information at client when learning unlabeled data alone (labels-at-server) than learning both labeled and unlabeled data (labels-at-client), as shown in Figure 9 C2S cost of $\Delta\psi$ in (a) and (c), and Figure 9 C2S cost



(a) Batch-NonIID (C2S & S2C)

(b) Streaming-NonIID (C2S & S2C)

(c) Batch-NonIID (C2S & S2C)

(d) Streaming-NonIID (C2S & S2C)

Figure 9: **Communication Cost Curves of FedMatch (ResNet-9) Corresponding to the Table 1 and 2**. We measure the communication costs for each parameters, $\Delta\sigma$ and $\Delta\psi$, during training phase. The communication costs under the labels-at-client scenario are visualized in (a) and (b) on the upper row. (c) and (d) on the lower row represent the communication costs under labels-at-server scenario.

of $\Delta\psi$ in (b) and (d). However, even if the ψ captures more information in the labels-at-server scenario, the C2S cost is still small since we do not send σ to server. Minimizing the cost of helper agents $\psi_{1:H}$ are also important, since the helper agents can be multiple, which makes S2C costs heavy. We assume that the parameters also could be effectively reduced by subtracting with the relevant ψ_l, which is supposed to be similar to the $\psi_{1:H}$, such that $\Delta\psi_{1:H} = \psi_{1:H} - \psi_l$. As shown in Figure 9 S2C costs in (a)-(d), we effectively minimize the helper agents costs, since S2C costs $\text{S2C} = |\Delta\sigma| + |\Delta\psi| + |\Delta\psi_{1:H}|$ is in general less than $|\Delta\sigma| + |\Delta\psi| + H \times |\Delta\psi|$.

C.4 Backbone Architecture

Most existing works on federated learning considers smaller networks since the focus is on-device learning of low-resource devices, and thus we utilize a smaller backbone networks than ResNet-9. To verify that our method also successfully works on the smaller & different architecture, we adopt AlexNet-Like (Serra et al., 2018), of which the first three layers are convolutional neural layers with 64, 128, and 256 filters with the $4, 3$, and 2 kernel sizes followed by the two fully-connected layers of 2048 units, while 2×2 max-pooling layers are followed after each convolutional layer. In Table 6, for both Streaming-NonIID and Batch-IID tasks, our methods still outperforms all naive Fed-SSL models with the similar tendency with that of the results based on ResNet-9. This shows that our methods can be applied to the smaller and different base networks, and still effectively utilize inter-client and reliable knowledge across multiple clients than naive algorithms.

C.5 Fraction of Available Clients per Communication Round

To see the effect of participation rate of clients, we increase the fraction of available clients per communication round in a range of 0.05, 0.10, and 0.2. This means, for every round, server can connect to the arbitrary 5, 10, and 20 available clients out of 100 clients, and perform distributed learning on each individual local data through each client and updates global knowledge by aggregating the locally-learned knowledge. The experimental results are shown in Table 6 Batch-IID. We observe that the performances of all models are slightly improved when the faction increases. This is natural that the more knowledge the client updates, the more the global performance is improved. We are not able to find any extraordinary phenomenon on the fraction of the number of clients per round.