A NOTATION TABLE

Table 3: Important notations used in the paper.

Notations	Description
d	input dimension
n	number of per desired label space class samples on the server
t	local updating iterations
C	number of clients in the FL, we index client by $c \in [C]$
J	number of classes in the other label space
K	number of classes in the desired label space
$\boldsymbol{T}$	linear transformation of noisy labels on the server
$\boldsymbol{Q}$	linear transformation between two label spaces
R	aggregation communication rounds
N_c	size of samples on the c-th client
S_c	the set of indices of observations on c-th client
$\mathcal{X}$	input space $\mathcal{X} \subset \mathbb{R}^d$
$\widetilde{\mathcal{Y}}$	the other label space $\widetilde{\mathcal{Y}} = \{\widetilde{Y}^1, \widetilde{Y}^2, \ldots, \widetilde{Y}^J\}$
$\mathcal{Y}$	desired label space $\mathcal{Y} = \{Y^1, Y^2, \ldots, Y^K\}$
$\mathcal{D}^c$	dataset $\mathcal{D}_c = \{(\boldsymbol{x}_i^c, y_i^c) : i \in S_c\}$ on c-th client
$\mathcal{D}^s$	dataset on server $\mathcal{D}^s = \{(\boldsymbol{x}_i^s, y_i^s) : i \in [nK]\}$ where $y_i^s \in \mathcal{Y}_{\text{fine}}$
$\boldsymbol{x}$	inputs variable $\boldsymbol{x} \in \mathcal{X}$
$\boldsymbol{f}(\cdot)$	a classification model that assigns a score to each of the K classes for a given input $\boldsymbol{x}$
$\ell(\cdot, \cdot)$	loss function for the classification task
$\widehat{R}_c(\cdot)$	empirical risk on c-th client
$\widehat{R}_s(\cdot)$	empirical risk on the server
η_{local}	local GD step size
η_{agg}	aggregation step size
ξ	the noise level in $\boldsymbol{T}$

B TECHNICAL DETAILS OF THE NEURAL TANGENT KERNEL ANALYSIS

In this section, we present the technical details of the Neural Tangent Kernel Analysis on the convergence of the algorithm of the label projection approach. We first present the NTK analysis of a simple regression model with weighted loss. We then show how the conclusion be extended to our classification case under FL.

B.1 NTK WITH WEIGHTED MSE LOSS FOR REGRESSION

Let us first consider an over-parameterized one-hidden layer neural network (NN) for regression. Let $f : \mathcal{X} \to \mathbb{R}$ be the output of the NN with the form

$$f(\boldsymbol{u}, \boldsymbol{x}) = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} a_m \phi(\boldsymbol{u}_m^\top \boldsymbol{x})$$

where $\phi(z) = \max\{z, 0\}$ is the ReLU activation function and $\boldsymbol{u} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M] \in \mathbb{R}^{d \times M}$. Let $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) : i \in [n]\}$ be a set of training examples. We consider the learning f under the following weighted MSE loss

$$\mathcal{L}(\boldsymbol{u}) = \frac{1}{2} \sum_{i=1}^{n} w_i (f(\boldsymbol{u}, \boldsymbol{x}_i) - y_i)^2.$$

Let η be the learning rate of gradient descent algorithm, the evolution of the parameters $\boldsymbol{u}$ and the f via continuous time gradient descent satisfies

$$\begin{aligned}\frac{df(\boldsymbol{u}(t),\boldsymbol{x}_i)}{dt} &= \frac{\partial f(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{d\boldsymbol{u}(t)}{dt},\\ \frac{d\boldsymbol{u}(t)}{dt} &= -\eta\sum_{i=1}^{n}\frac{\partial f(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}}\frac{\partial \mathcal{L}}{\partial f(\boldsymbol{u}(t),\boldsymbol{x}_i)}\end{aligned} \tag{8}$$

based on the chain rule.

Let $f(\boldsymbol{u},\mathcal{D}) = (f(\boldsymbol{u},\boldsymbol{x}_1), f(\boldsymbol{u},\boldsymbol{x}_2),\ldots,f(\boldsymbol{u},\boldsymbol{x}_n))^T$. Given the weighted MSE loss, we have

$$\frac{\partial \mathcal{L}}{\partial f(\boldsymbol{u},\mathcal{D})} = (w_1(f(\boldsymbol{u},\boldsymbol{x}_1)-y_1), w_2(f(\boldsymbol{u},\boldsymbol{x}_2)-y_2),\ldots,w_n(f(\boldsymbol{u},\boldsymbol{x}_n)-y_n))^\top$$

and

$$\frac{\partial f(\boldsymbol{u},\mathcal{D})}{\partial \boldsymbol{u}} = \left(\frac{\partial f(\boldsymbol{u},\boldsymbol{x}_1)}{\partial \boldsymbol{u}^\top}, \frac{\partial f(\boldsymbol{u},\boldsymbol{x}_2)}{\partial \boldsymbol{u}^\top},\ldots,\frac{\partial f(\boldsymbol{u},\boldsymbol{x}_n)}{\partial \boldsymbol{u}^\top}\right)^\top.$$

Hence, we get

$$\frac{df(\boldsymbol{u},\boldsymbol{x}_i)}{dt} = -\eta\sum_{j=1}^{n} w_j \frac{\partial f(\boldsymbol{u},\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{\partial f(\boldsymbol{u},\boldsymbol{x}_j)}{\partial \boldsymbol{u}}\{f(\boldsymbol{u},\boldsymbol{x}_j)-y_j\}.$$

Then the full evolution dynamic of learning f is

$$\frac{df(\boldsymbol{u}(t),\mathcal{D})}{dt} = -\eta\boldsymbol{\Lambda}(t)(f(\boldsymbol{u}(t),\mathcal{D})-\boldsymbol{y})$$

where $\boldsymbol{y} = (y_1, y_2,\ldots,y_n)^\top$, the Gram matrix $\boldsymbol{\Lambda}(t)\in\mathbb{R}^{n\times n}$ is

$$\boldsymbol{\Lambda}(t) = \nabla_{\boldsymbol{u}} f(\boldsymbol{u}(t),\mathcal{D}) W \nabla_{\boldsymbol{u}}^\top f(\boldsymbol{u}(t),\mathcal{D}),$$

and $W = \text{diag}(w_1, w_2,\ldots,w_n)$.

B.2 NTK UNDER WEIGHTED MSE LOSS FOR CLASSIFICATION

With the preparation in Section B.1, we now show the NTK kernel under the weighted MSE loss for classification. We first consider the centralized version and then extend the conclusions to the federated learning case.

Centralized Classification To make the presentation easier, we first introduce some notations. Let $\mathcal{D} = \{(\boldsymbol{x}_i,\boldsymbol{y}_i) : i\in[n]\}$ be a set of training examples where $\boldsymbol{y}_i$ is the one-hot representation of the label. Let $\boldsymbol{f}: \mathcal{X}\mapsto\mathbb{R}^K$ be a K-class classifier and $\boldsymbol{g}(\boldsymbol{u},\boldsymbol{x}) = \sigma(\boldsymbol{f}(\boldsymbol{u},\boldsymbol{x}))$. In this section, we show the evolution dynamic of learning $\boldsymbol{g}$ under the following weighted MSE loss

$$\mathcal{L}(\boldsymbol{u}) = \sum_{i=1}^{n}\mathcal{L}_i(\boldsymbol{u}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{J}\sum_{k=1}^{K} Q_{jk}^{-1}\{y_{ik}-g_k(\boldsymbol{u},\boldsymbol{x}_i)\}^2$$

where $\boldsymbol{Q}^{-1}\in\mathbb{R}^{J\times K}$ is a known matrix and Q_{jk}^{-1} is the (j,k)-th element of $\boldsymbol{Q}^{-1}$.

Following (8), the evolution of the parameters $\boldsymbol{u}$ and the output $\boldsymbol{g}$ via continuous time gradient descent satisfies

$$\begin{aligned}\frac{dg_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{dt} &= \frac{\partial g_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{d\boldsymbol{u}(t)}{dt}\\ \frac{d\boldsymbol{u}(t)}{dt} &= -\eta\sum_{l=1}^{n}\sum_{\kappa=1}^{K}\frac{\partial g_\kappa(\boldsymbol{u}(t),\boldsymbol{x}_l)}{\partial \boldsymbol{u}}\frac{\partial \mathcal{L}_l(\boldsymbol{u}(t))}{\partial g_\kappa}\end{aligned} \tag{9}$$

Under the weighted MSE loss, we have

$$\frac{\partial \mathcal{L}_l(\boldsymbol{u}(t))}{\partial g_\kappa} = \sum_{j=1}^{J} Q_{j\kappa}^{-1}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}_l)-y_{l\kappa}\}.$$

Plugin the gradient to (9), we have

$$\frac{dg_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{dt} = -\eta \sum_{l=1}^{n}\sum_{\kappa=1}^{K}\sum_{j=1}^{J} \frac{\partial g_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{\partial g_\kappa(\boldsymbol{u}(t),\boldsymbol{x}_l)}{\partial \boldsymbol{u}} Q^{-1}_{j\kappa}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}_l) - y_{l\kappa}\}$$

Let $\boldsymbol{g}(\boldsymbol{u},\mathcal{D}) = \sigma(\boldsymbol{f}(\boldsymbol{u},\mathcal{D})) = (g_1(\boldsymbol{u},\boldsymbol{x}_1),\ldots,g_1(\boldsymbol{u},\boldsymbol{x}_n),\ldots,g_K(\boldsymbol{u},\boldsymbol{x}_1),\ldots,g_K(\boldsymbol{u},\boldsymbol{x}_n))^\top$, and

$$\boldsymbol{y} = (y_{11},\ldots,y_{n1},\ldots,y_{1K},\ldots,y_{nK})^\top.$$

Then in the matrix form, the full evolution dynamic of learning $\boldsymbol{g}$ becomes

$$\frac{d\boldsymbol{g}(\boldsymbol{u}(t),\mathcal{D})}{dt} = -\eta \boldsymbol{H}(t)(\boldsymbol{g}(\boldsymbol{u}(t),\mathcal{D}) - \boldsymbol{y})$$

where the Gram matrix $\boldsymbol{H}(t) \in \mathbb{R}^{Kn\times Kn}$ is a block matrix with K row partitions and K column partitions. The (p_1,p_2)-th element of the block matrix in the l-th row and m-th column has the following form

$$\boldsymbol{H}^{l,m}_{p_1,p_2}(t) = \sum_{j=1}^{J} Q^{-1}_{jm}\frac{\partial g_l(\boldsymbol{u}(t),\boldsymbol{x}_{p_1})}{\partial \boldsymbol{u}^\top}\frac{\partial g_m(\boldsymbol{u}(t),\boldsymbol{x}_{p_2})}{\partial \boldsymbol{u}}. \tag{10}$$

Federated Classification We can now extend the study of the full evolution dynamic of learning $\boldsymbol{g}$ under centralized version to federated classification. For federated learning case, we have the following datasets: the set of training examples $\mathcal{D}^c = \{(\boldsymbol{x}^c_i,\boldsymbol{y}^c_i) : i \in S_c\}$ on c-th client where $\boldsymbol{y}_i$ is the one-hot representation of the label in dimension K and the training data on the server $\mathcal{D}^s = \{(\boldsymbol{x}^s_i,\boldsymbol{y}^s_i) : i \in [nK]\}$ where $\boldsymbol{y}^s_i$ is the one-hot representation of the label. Let $\boldsymbol{f} : \mathcal{X} \mapsto \mathbb{R}^K$ be a K-class classifier. We consider the learning of $\boldsymbol{f}$ under the mean squared error loss

$$\mathcal{L}_c(\boldsymbol{u}) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{J}\sum_{k=1}^{K} Q^{-1}_{jk}\{y^c_{ik} - g_k(\boldsymbol{u},\boldsymbol{x}^c_i)\}^2,\ \mathcal{L}_s(\boldsymbol{u}) = \frac{1}{2}\sum_{i=1}^{nK}\sum_{k'=1}^{K}\sum_{k=1}^{K} T^{-1}_{k'k}\{y^s_{ik} - g_k(\boldsymbol{u},\boldsymbol{x}^s_i)\}^2.$$

The overall loss is

$$\mathcal{L}(\boldsymbol{u}) = \mathcal{L}_s(\boldsymbol{u}) + \sum_{c=1}^{C}\mathcal{L}_c(\boldsymbol{u}). \tag{11}$$

Same as the centralized case, let us consider the evolution of $\boldsymbol{g}$ via continuous time gradient descent.

$$\frac{dg_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{dt} = \frac{\partial g_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{d\boldsymbol{u}(t)}{dt}$$
$$\frac{d\boldsymbol{u}(t)}{dt} = -\eta\sum_{l}\sum_{\kappa=1}^{K}\frac{\partial g_\kappa(\boldsymbol{u}(t),\boldsymbol{x}_l)}{\partial \boldsymbol{u}}\frac{\partial \mathcal{L}_l(\boldsymbol{u}(t))}{\partial g_\kappa}$$

The difference of our federated version from the centralized version is $\partial\mathcal{L}/\partial g_\kappa$. Under (11), we have

$$\frac{\partial \mathcal{L}_l(\boldsymbol{u}(t))}{\partial g_\kappa} = \begin{cases} \sum_{j=1}^{J} Q^{-1}_{j\kappa}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}_l) - y_{l\kappa}\} & \text{if } \boldsymbol{x}_l \text{ on } c\text{-th client,} \\ \sum_{k'=1}^{K} T^{-1}_{k'\kappa}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}_l) - y_{l\kappa}\} & \text{if } \boldsymbol{x}_l \text{ on the server.} \end{cases}$$

Hence we have

$$\begin{aligned}\frac{dg_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{dt} =& -\eta\sum_{i'=1}^{nK}\sum_{\kappa=1}^{K}\frac{\partial g_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{\partial g_\kappa(\boldsymbol{u}(t),\boldsymbol{x}^s_{i'})}{\partial \boldsymbol{u}}\sum_{k'=1}^{K} T^{-1}_{k'\kappa}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}^s_{i'}) - y^s_{i'\kappa}\} \\ & -\eta\sum_{c=1}^{C}\sum_{i'\in S_c}\sum_{\kappa=1}^{K}\frac{\partial g_k(\boldsymbol{u}(t),\boldsymbol{x}_i)}{\partial \boldsymbol{u}^\top}\frac{\partial g_\kappa(\boldsymbol{u}(t),\boldsymbol{x}^c_{i'})}{\partial \boldsymbol{u}}\sum_{j=1}^{J} Q^{-1}_{j\kappa}\{g_\kappa(\boldsymbol{u},\boldsymbol{x}^c_{i'}) - y^c_{i'\kappa}\}.\end{aligned}$$

To write the evolution of $\boldsymbol{g}$ in matrix form, let us first introduce some notations. Let

$$g_k(\boldsymbol{u},\mathcal{D}) = (g_k(\boldsymbol{u},\boldsymbol{x}^1_1),\ldots,g_k(\boldsymbol{u},\boldsymbol{x}^1_{N_1}),\ldots,g_k(\boldsymbol{u},\boldsymbol{x}^C_1),\ldots,g_k(\boldsymbol{u},\boldsymbol{x}^C_{N_C}),g_k(\boldsymbol{u},\boldsymbol{x}^s_1),\ldots,g_k(\boldsymbol{u},\boldsymbol{x}^s_{nK}))^\top$$

and $\boldsymbol{g}(\boldsymbol{u}, \mathcal{D}) = \left(g_1^\top(\boldsymbol{u}, \mathcal{D}), \ldots, g_K^\top(\boldsymbol{u}, \mathcal{D})\right)^\top$. Note the vector $\boldsymbol{g}(\boldsymbol{u}, \mathcal{D})$ is first grouped by machines and then classes. Similarly, we also denote $\boldsymbol{y}^k = (y_{1k}^1, \ldots, y_{N_1,k}^1, \ldots, y_{1k}^C, \ldots, y_{N_C,k}^C, y_{1k}^s, \ldots, y_{nK,k}^s)$ and $\boldsymbol{y} = (\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^K)$. Then in the matrix form, the full evolution dynamic of learning $\boldsymbol{g}$ becomes

$$\frac{d\boldsymbol{g}(\boldsymbol{u}(t), \mathcal{D})}{dt} = -\eta \boldsymbol{G}(t)(\boldsymbol{g}(\boldsymbol{u}(t), \mathcal{D}) - \boldsymbol{y})$$

where the Gram matrix $\boldsymbol{G}(t) \in \mathbb{R}^{K(nK+N)\times K(nK+N)}$ is a block matrix with K row partitions and K column partitions. The block matrix in the l-th row and m-th column has the following form

$$\boldsymbol{G}^{l,m}(t) = \begin{pmatrix} \mathcal{G}_{1,1}^{l,m}(t) & \mathcal{G}_{1,2}^{l,m}(t) & \ldots & \mathcal{G}_{1,C}^{l,m}(t) & \mathcal{G}_{1,s}^{l,m}(t) \\ \mathcal{G}_{2,1}^{l,m}(t) & \mathcal{G}_{2,2}^{l,m}(t) & \ldots & \mathcal{G}_{2,C}^{l,m}(t) & \mathcal{G}_{2,s}^{l,m}(t) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{G}_{C,1}^{l,m}(t) & \mathcal{G}_{C,2}^{l,m}(t) & \ldots & \mathcal{G}_{C,C}^{l,m}(r) & \mathcal{G}_{C,s}^{l,m}(t) \\ \mathcal{G}_{s,1}^{l,m}(t) & \mathcal{G}_{s,2}^{l,m}(t) & \ldots & \mathcal{G}_{s,C}^{l,m}(t) & \mathcal{G}_{s,s}^{l,m}(t) \end{pmatrix}$$

for $l \in [K]$ and $m \in [K]$. The block matrix $\boldsymbol{G}^{l,m}(r)$ is also consisted of block matrices. The block matrix $\mathcal{G}_{c_1,c_2}^{l,m}(t) \in \mathbb{R}^{N_{c_1}\times N_{c_2}}$, $\mathcal{G}_{s,c}^{l,m}(t) \in \mathbb{R}^{nK\times N_c}$, and $\mathcal{G}_{s,s}^{l,m}(t) \in \mathbb{R}^{nK\times nK}$ for $c_1 \in [C]$, $c_2 \in [C]$, and $c \in [C]$.

We now specify the elements in $\mathcal{G}_{c_1,c_2}^{l,m}(t)$, $\mathcal{G}_{s,c}^{l,m}(t)$, and $\mathcal{G}_{s,s}^{l,m}(t)$ respectively. The derivation is the same as (10). We have

- The (p_1, p_2)-th element of $\mathcal{G}_{c_1,c_2}^{l,m}(t)$ has the following form

$$\sum_{j=1}^{J} Q_{jm}^{-1} \frac{\partial g_l(\boldsymbol{u}, \boldsymbol{x}_{p_1}^{c_1})}{\partial \boldsymbol{u}^\top} \frac{\partial g_m(\boldsymbol{u}, \boldsymbol{x}_{p_2}^{c_2})}{\partial \boldsymbol{u}}.$$

- The (p_1, p_2)-th element of $\mathcal{G}_{s,c}^{l,m}(t)$ has the following form

$$\sum_{j=1}^{J} Q_{jm}^{-1} \frac{\partial g_l(\boldsymbol{u}, \boldsymbol{x}_{p_1}^{s})}{\partial \boldsymbol{u}^\top} \frac{\partial g_m(\boldsymbol{u}, \boldsymbol{x}_{p_2}^{c})}{\partial \boldsymbol{u}}.$$

- The (p_1, p_2)-th element of $\mathcal{G}_{s,s}^{l,m}(t)$ has the following form

$$\sum_{k=1}^{K} T_{km}^{-1} \frac{\partial g_l(\boldsymbol{u}, \boldsymbol{x}_{p_1}^{s})}{\partial \boldsymbol{u}^\top} \frac{\partial g_m(\boldsymbol{u}, \boldsymbol{x}_{p_2}^{s})}{\partial \boldsymbol{u}}.$$

B.3 Proof of Corollaries

In this section, we provide the proof for Corollary 3.5 and Corollary 3.6.

Proof for Corollary 3.6 The key for the proof is to find the form of $\boldsymbol{T}^{-1}$ when $\boldsymbol{T} = (1 - K\xi/(K-1))\mathbb{I}_K + \xi/(K-1)\mathbb{1}_K\mathbb{1}_K^\top$. To find T^{-1}, we make use of the Woodbury matrix identity

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}$$

where $\boldsymbol{A}$, $\boldsymbol{U}$, $\boldsymbol{C}$, and $\boldsymbol{V}$ are matrices of proper sizes.

In our case, $\boldsymbol{A} = \{1 - K\xi/(K-1)\}\mathbb{I}_K$, $\boldsymbol{U} = \mathbb{1}_K$, $\boldsymbol{C} = \xi/(K-1)$, and $\boldsymbol{V} = \mathbb{1}_K^\top$. By applying Woodbury matrix identity, we get

$$\boldsymbol{T}^{-1} = (K-1)/(K-1-K\xi)\mathbb{I}_K - \xi/(K-1-K\xi)\mathbb{1}_K\mathbb{1}_K^\top.$$

Hence

$$\sum_{k=1}^{K} T_{kk'} = \left(\boldsymbol{T}^{-1}\mathbb{1}_K\right)_k = \left((K-1)/(K-1-K\xi)\mathbb{1}_K - K\xi/(K-1-K\xi)\mathbb{1}_K\right)_k = \left(\mathbb{1}_K\right)_k = 1$$

which completes the proof.

Proof for Corollary 3.5 Let $k_1, k_2, \ldots, k_J$ be J non-negative integers such that $K = \sum_{j=1}^{J} k_j$. If the linear transformation from desired label space to the other label space is $\boldsymbol{Q} = \text{diag}(\mathbf{1}_{k_1}, \mathbf{1}_{k_2}, \ldots, \mathbf{1}_{k_J}) \in [0,1]^{K \times J}$ (*i.e.,* the classes in the desired label space are sub-classes of the class in the other label space), then the smaller the value of J, the lower the convergence rate.

We will use the following conclusion (Li et al., 2021) in the proof. For a block positive definite matrix $\mathbf{A} = (\mathbf{A}_{ij})_{n \times n}$, we must have

$$\lambda_{\min}(\mathbf{A}) \leq \min_i \lambda_{\min}(\mathbf{A}_{ii}).$$

By recursively using this conclusion on $\boldsymbol{G}(t)$ given in (7). We have

$$\lambda = \lambda_{\min}(\boldsymbol{G}(0)) \leq \min_{k,c,s} \{\lambda_{\min}(\mathcal{G}_{c,c}^{k,k}(0)), \lambda_{\min}(\mathcal{G}_{s,s}^{k,k}(0))\} \leq \min_{k,c} \lambda_{\min}(\mathcal{G}_{c,c}^{k,k}(0)).$$

Let $\lambda_0^c = \lambda_{\min}(\nabla_{\boldsymbol{u}} g_k(\boldsymbol{u}(0), \mathcal{D}^c) \nabla_{\boldsymbol{u}} g_k^\top(\boldsymbol{u}(0), \mathcal{D}^c))$ for $c \in [C]$. Under the special case where the two label spaces have hierarchical structures, we have $Q^{-1} = \text{diag}(k_1^{-1}\mathbb{1}_{k_1}^\top, k_2^{-1}\mathbb{1}_{k_2}^\top, \ldots, k_J^{-1}\mathbb{1}_{k_J}^\top)$

Since $\mathcal{G}_{c,c}^{k,k}(0) = \left\{\sum_j Q_{jk}^{-1}\right\} \nabla_{\boldsymbol{u}} g_k(\boldsymbol{u}(0), \mathcal{D}^c) \nabla_{\boldsymbol{u}} g_k^\top(\boldsymbol{u}(0), \mathcal{D}^c)$, it is easy to see that

$$\lambda_{\min}(\mathcal{G}_{c,c}^{k,k}(0)) \leq \left\{\min_j k_j^{-1}\right\} \lambda_0^c.$$

where λ_0^c is a constant, the smallest eigen value of $\nabla_{\boldsymbol{u}} g_k(\boldsymbol{u}(0), \mathcal{D}^c) \nabla_{\boldsymbol{u}} g_k^\top(\boldsymbol{u}(0), \mathcal{D}^c)$ (*i.e.,* the original kernel of FL-NTK Huang et al. (2021)), which does not depend on J.

Hence

$$\lambda \leq \min_c \left\{\min_j k_j^{-1}\right\} \lambda_0^c.$$

Since $\sum_j k_j = K$, and in our setting we have $k_1 \approx k_2 \approx \cdots \approx k_J$, the smaller the value of J, the smaller the value of λ and hence the algorithm converges slower.

C Experimental Details

We present our model architectures and training details of the benchmark and synthetic sEMG medical dataset in this section.

C.1 Model Architecture and Implementation Details on Benchmark

The details of the experiments on the benchmark CIFAR100 dataset are presented in this section.

Model architecture For our benchmark experiments on CIFAR100, we use ResNet18 as our backbone. The network details are listed in Tab. 4.

Data split and label Generation Here, we describe how to prepare the CIFAR100 training dataset into sub-class but noisy data on the server and super-class datasets on the clients. We first select n samples from each of the $K = 100$ sub-classes as the datasets on the server. To generate noisy labeled data at noise level $\xi \in [0,1]$, we assume that the noise matrix is known, symmetric, instance independent, and reconstructable, represented as the matrix $\boldsymbol{T}_\xi = \{1 - K/(K-1)\xi\}\mathbb{I}_K + \xi/(K-1)\mathbb{1}_K\mathbb{1}_K^\top$. Then the observed labels are sampled from this given matrix by random flipping, *i.e.,* the label in one class is flipped to the label in another class with probability $\xi/(K-1)$. The higher the value of ξ, the larger the proportion of noise labels. Then for the rest of the observations in the CIFAR100 training set, we randomly split then into C pieces each with N_c observations and each client is assigned N_c samples. Hence in this case $\boldsymbol{Q}$ is a 200×100 matrix, its (j,i)-th element is 1 if the i-th class is in j-th superclass and 0 otherwise.

Table 4: Model architecture of the benchmark experiment on CIFAR100. For convolutional layer (Conv2D), we list parameters with sequence of input and output dimension, kernel size, stride and padding. For max pooling layer (MaxPool2D), we list kernel and stride. For fully connected layer (FC), we list input and output dimension. For BatchNormalization layer (BN), we list the channel dimension.

Layer	Details
1	Conv2D(3, 64, 7, 2, 3), BN(64), ReLU
2	Conv2D(64, 64, 3, 1, 1), BN(64), ReLU
3	Conv2D(64, 64, 3, 1, 1), BN(64)
4	Conv2D(64, 64, 3, 1, 1), BN(64), ReLU
5	Conv2D(64, 64, 3, 1, 1), BN(64)
6	Conv2D(64, 128, 3, 2, 1), BN(128), ReLU
7	Conv2D(128, 128, 3, 1, 1), BN(64)
8	Conv2D(64, 128, 1, 2, 0), BN(128)
9	Conv2D(128, 128, 3, 1, 1), BN(128), ReLU
10	Conv2D(128, 128, 3, 1, 1), BN(64)
11	Conv2D(128, 256, 3, 2, 1), BN(128), ReLU
12	Conv2D(256, 256, 3, 1, 1), BN(64)
13	Conv2D(128, 256, 1, 2, 0), BN(128)
14	Conv2D(256, 256, 3, 1, 1), BN(128), ReLU
15	Conv2D(256, 256, 3, 1, 1), BN(64)
16	Conv2D(256, 512, 3, 2, 1), BN(512), ReLU
17	Conv2D(512, 512, 3, 1, 1), BN(512)
18	Conv2D(256, 512, 1, 2, 0), BN(512)
19	Conv2D(512, 512, 3, 1, 1), BN(512), ReLU
20	Conv2D(512, 512, 3, 1, 1), BN(512)
21	AvgPool2D
22	FC(512, 100)

Training details We implement all the methods by PyTorch and conducted all the experiments on an NVIDIA Tesla V100 GPU. For our `FedMT`, we use SGD optimizer Ruder (2016) with a learning rate of 10^{-2}, momentum 0.9, and weight decay 5×10^{-4}. Each model updates one epoch then aggregates with the others. The total communication iterations is set to be 100. The learning rate is divided by 5 at 20, 30, and 40 iterations. We set the batch size to 16 and the number of training examples on c-th client to $N_c = 4000$ if not specified otherwise. In Tab. 1, we consider various values for the per number of observations n on the server under the noise-free setting. We also consider various values for the noisy level ξ when $n = 10$. For additional results in the Appendix D, we specify the hyper-parameters only when they are different.

We describe details of the baseline methods and corresponding hyper-parameters as follows.

- **Single** For this baseline method, we only use the limited data on the server to train a K-class classifiers. Under the noisy label scenarios, we also apply the label projection and the probability projection on the server and the learning objectives is

$$\widehat{R}(\boldsymbol{f}) = -\frac{1}{nK}\sum_{i=1}^{nK}\sum_{k=1}^{K}\mathbb{1}(y_i^s = Y^k)\log\left\{\sum_{k'=1}^{K}T_{kk'}\sigma(f_{k'}(\boldsymbol{x}_i^s))\right\},$$

  for probability projection and

$$\widehat{R}(\boldsymbol{f}) = -\frac{1}{nK}\sum_{i=1}^{nK}\sum_{k=1}^{K}\left\{\sum_{k'=1}^{K}T_{kk'}^{-1}\mathbb{1}(y_i^s = Y^{k'})\right\}\log\sigma(f_k(\boldsymbol{x}_i^s)),$$

for label projection. We use the same optimizer as our FedMT.

- **FedMatch** We use the semi-supervised FedMatch proposed by (Lu et al., 2021). We treat client samples as unlabeled data, perform pseudo-labeling on the unlabeled sets under the supervision of labeled data on the server. The pseudo-labels are updated heuristically during the training. The data augmentation strategy is also used. For fair comparison, we do not use the pretrained backbone during training. We set learning rate to 0.01, batch size to 16, and the rest of the hyperparameters are the same as the default setting in the implementation.
- **FedRep** (Collins et al., 2021) For this baseline method, all clients train a super-class classifier and the server trains a sub-class classifier that differs in the last layer with different output dimension. The learning objective is

$$\widehat{R}(\boldsymbol{f}) = \frac{1}{C}\sum_{c=1}^{C+1}\left\{\widehat{R}_c(\boldsymbol{f};\mathcal{D}^c) + \widehat{R}_s(\boldsymbol{f};\mathcal{D}^s)\right\},$$

where

$$\widehat{R}_c(\boldsymbol{f};\mathcal{D}^c) = -\frac{1}{N_c}\sum_{i\in S_c}\sum_{j=1}^{J}\mathbb{1}(y_i=\widetilde{Y}^j)\log\sigma(f_j(\boldsymbol{x}_i))$$

and $\widehat{R}_s(\boldsymbol{f};\mathcal{D}^s)$ is the same as the objective function in the Single baseline approach. All classifiers on the server and on the clients use the same ResNet18 backbone, but they have different last layers. For the classifiers on clients, their last layer is FC(512,20); for the classifier on the server, its last layer is FC(512,100). The last layers of classifiers on clients are also not shared during the training. The rest of the parameters of all classifiers are aggregated using FedAvg during training. We use the same optimizer as our FedMT.[6]
- **FedTrans** For this baseline method, we first pretrain a model using the client data to train a J-class classifiers using FedAvg with probability projection or label projection. This pretrained net is then fine-tuned on the server. During the fine-tuning, we add a new linear layer with K classes with random initialization and backbone is initialized with the pretrained model. We use the same optimizer as our FedMTduring the pretrain step and the model is fine-tuned on the server for 100 epochs with SGD optimizer and learning rate 0.01.

C.2 Model Architecture and Implementation Details on Medical Dataset

Model architecture We use a simple MLP as the model for the medical data, whose architecture is listed in Table 5.

Table 5: Model architecture of the experiment on medical dataset sEMG. For fully connected layer (FC), we list input and output dimension. For BatchNormalization layer (BN), we list the channel dimension.

Layer	Details
1	FC(12,128), BN(128), ReLU
2	FC(128, 10)

Data Preprocessing The raw sEMG data is collected from Qin et al. (2020), with the dimensionality of 2048, *i.e.,* 2048 time points per sample, $\boldsymbol{x} = [x_1, \ldots, x_{2048}]$. Following Qin et al. (2019), instead of using the raw time series, we use the extracted features as described in Section 4.The features twelve hand-crafted features, including mean absolute value (MAV), mean square value (MSV), root mean square (RMS), variance (VAR), standard deviation (STD), waveform length (WL), Willison amplitude (WAMP), log detector (LOG), slope sign change (SSC), zero crossing (ZC), mean spectral frequency (MSF) and median frequency (MF). Denote P_j is the sEMG power spectrum at frequency bin j, and f_j is the frequency of the sEMG power spectrum at frequency bin j. Mathematically, these features are defined in Tab. 6.

[6]Note that the data labeling settings, models, and evaluation methods implemented in our work is adjusted to make fair comparison with other methods and different from those in Collins et al. (2021).

Table 6: Mathematical representation of widely used sEMG feature extraction methods. The constant ζ is an user specified threshold and we set $\zeta = 1$.

Feature Extraction	Mathematical Equation
mean absolute value (MAV)	$\frac{1}{N}\sum_{n=1}^{N}\|x_n\|$
mean square value (MSV)	$\frac{1}{N}\sum_{n=1}^{N}x_n^2$
root mean square (RMS)	$\sqrt{\frac{1}{N}\sum_{n=1}^{N}x_n^2}$
variance (VAR)	$\frac{1}{N-1}\sum_{n=1}^{N}x_n^2$
standard deviation (STD)	$\sqrt{\frac{1}{N-1}\sum_{n=1}^{N}x_n^2}$
waveform length (WL)	$\sum_{n=1}^{N-1}\|x_{n+1}-x_n\|$
Willison amplitude (WAMP)	$\sum_{n=1}^{N-1} f(\|x_{n+1}-x_n\|);\ f(x)=\begin{cases}1 & \text{if } x\geq\zeta\\ 0 & \text{otherwise}\end{cases}$
log detector (LOG)	$\exp\left(\frac{1}{N}\sum_{n=1}^{N}\log\|x_n\|\right)$
slope sign change (SSC)	$\sum_{n=2}^{N-1} f[(x_n-x_{n-1})\times(x_n-x_{n+1})];\ f(x)=\begin{cases}1 & \text{if } x\geq\zeta\\ 0 & \text{otherwise}\end{cases}$
zero crossing (ZC)	$\sum_{n=1}^{N-1}[\text{sgn}(x_n\times x_{n+1})\bigcap\|x_n-x_{n+1}\|\geq\zeta];\ \text{sgn}(x)=\begin{cases}1 & \text{if } x\geq\zeta\\ 0 & \text{otherwise}\end{cases}$
mean spectral frequency (MSF)	$\frac{\sum_{j=1}^{M}f_jP_j}{\sum_{j=1}^{M}P_j}$
median frequency (MF)	$\frac{1}{2}\sum_{j=1}^{M}P_j\ \mathcal{X}\subset\mathbb{R}^d$

We randomly sample 9000 samples as training data annotated with label from the other label space, and 1000 samples as the held testing data. We set the number of clients $C = 50$, namely each client has 180 samples. The data are labeling based on the severity value from 0 to 5. We equally partition the values into 3 parts with the same intervals (*i.e.,* [0,5/3), [5/3,10/3), and [10/3,5]) corresponding to $J = 3$ labeled classes on the clients. Such class partition strategy can simulate the overlapped case between two label spaces, where samples in the same Y^k class can belong to different $\widetilde{Y}^j$ classes. From the remaining sEMG samples, we randomly sample n observations per 5 class to represent the data on server. Under the noisy label scenarios, the noisy labels are generated using the same protocol as described in Section C.1.

Training details We implement all the methods by PyTorch and conducted all the experiments on an NVIDIA Tesla V100 GPU. All the methods are trained for 100 round and each round, model update once before aggregation. We use SGD optimizer Ruder (2016) with a learning rate of 10^{-2}, momentum 0.9, and weight decay 5×10^{-4}. The learning rate is divided by 5 at 20, 30, and 40 epochs. We set the batch size to 16. We also consider various values for the noisy level ξ from 0 to 0.4. All the experiments are repeated with independent random seeds.

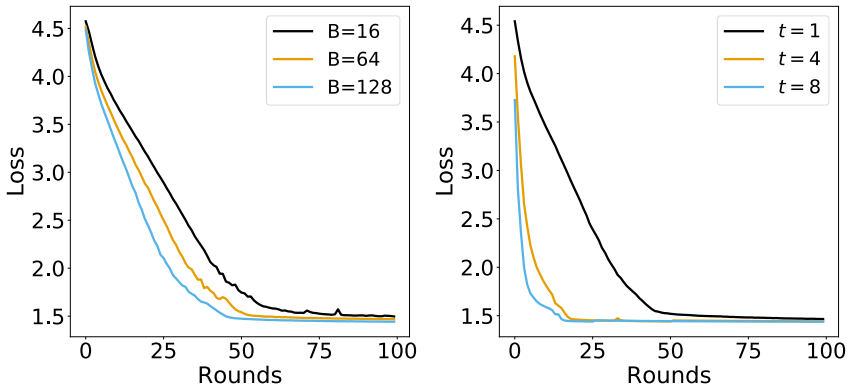
D More Experimental Results on CIFAR100

In this section, we present more experiment results on the benchmark CIFAR100 dataset. In Section D.1, we show the convergence of the loss function as a function of batch sizes and the local rounds. In Section D.2, we show the experiment results of our `FedMT` when different weighting schemes are used for server and clients. In Section D.3, we show the experiment results as the number of per client observation increases. In Section D.4, we study the performance of our proposed method under label heterogeneous across clients. In Section D.5, we show the performance of our method when limited data from desired label space are further split into more pieces.

D.1 Effects of Local Rounds and Batch Size

To demonstrate the robustness of the proposed `FedMT` and validate our developed theoretical results in Theorem 3.4, we investigate the effects of different local rounds t and the batch size B on training losses (2) of `FedMT` with label projection in Section 4.1. For all experiments, the learning rate is set to be 0.0001 and no learning rate scheduler is used during the training. In Fig. 3(a), we explore

loss as a function of the batch size $B \in \{16, 64, 128\}$ with $C = 10$ and without noise on super-class labels. The larger the batch size, the faster the loss function converges. In Fig. 3(b), we explore loss as a function of the local rounds $t \in \{1, 4, 8\}$ with $C = 10$ and without noise on super-class labels. The larger the local rounds, the faster the convergence, which is consistent with the theoretical result of Theorem 3.4. It is worth noting that all the training curves of different variants of the parameters reach convergence.



(a) Loss curve for different batch sizes (b) Loss curve for different local rounds

Figure 3: Effects of (a) the batch size B and (b) the local rounds t on the rate of convergence on the loss function.

D.2 Benchmark Experiments with Weighted FedMT

In our main results, the loss on the each client and the loss on the server are equally weighted with weight $1/(C + 1)$. To see the performance of our FedMT, we also investigate the case when these losses are weighted unequally and report the classification accuracy of the corresponding trained model on the held out test set in Tab. 7. For these experiment results, we consider different number of $n \in \{5, 10, 20, 40, 80\}$ with $\xi = 0.2$. The sample size on c-th client $N_c = 4000$. For the unequal weight case, the weight on the server is 0.5 regardless of the size of n and the weight on each client is $0.5/C$. As shown in the table, the test classification accuracy increases with n for both equally weighted and unequally weighted cases. The performance of the unequally weighted case is not as good as the equally weighted case.

Table 7: Comparison of the accuracy for 100-class classification using different FL aggregation strategies on CIFAR100 benchmark dataset at different number of per K class training examples n when noise level $\xi = 0.2$. We report mean (sd) from three trial runs.

Weights	Methods	5	10	20	40	80
Equal	Our (P)	19.70(0.30)	24.26(0.85)	27.48(0.48)	30.56(0.61)	34.85(0.12)
	Our (L)	18.10(0.41)	21.26(0.32)	24.37(0.26)	28.28(0.18)	32.15(0.86)
Unequal	Our (P)	19.27(0.20)	21.49(0.50)	23.83(0.12)	26.66(0.11)	29.25(0.25)
	Our (L)	13.16(0.62)	14.40(0.21)	16.09(0.22)	19.15(0.25)	21.61(0.33)

D.3 Benchmark Experiments with More Clients

In Tab. 1, we check the performance of our proposed method for varying amount of data on the server. In this section, we present the performance of our proposed method under varying amount of data on the client when $n = 10$ and $\xi = 0.2$ on the server. We also set the per client number of observations to be $N_c = 800$. We change the number of clients $C \in \{10, 20, 30, 40, 50\}$. The larger the number of clients C, more data we have. During the aggregation, we set the weights of the server to be 0.5 and the rest of the client to be $0.5/C$. In comparison, if the equal weight $1/(C + 1)$ is used, the super-class data on the server plays a less importance role as C increases. We therefore use the unequally weighted strategy to avoid. The classification result of the aggregated model on the held

out test set is given in Table 8. As shown in the table, the classification accuracy increases as the number of clients increases.

Table 8: Our FedMT classification accuracy on benchmark CIFAR100 with different number of clients.

Methods	10	20	30	40	50
Our (P)	13.61(0.16)	15.17(0.29)	15.28(0.11)	15.92(0.44)	16.32(0.27)

D.4 Benchmark Experiments under Non-IID Data Split

The heterogeneity of the training data across clients is a major challenge in FL Kairouz et al. (2021). We therefore study the performance of our method under two different FL settings: IID and non-IID. The overall label distribution across clients is the same in the IID setting, whereas clients have different label distributions in the non-IID setting. For the IID setting, we split the sub-class label training data evenly to all clients completely at random. For the non-IID setting, we split the dataset to clients as follows:

- the classes are divided into the majority and minority classes, where the fraction of each minority class is less than 0.08 while the fraction of each majority class is in the range [0.15, 0.25];
- the training data for every client are sampled from 2 majority classes and rest from minority classes;

We compare our method with personalized FL method FedRep both under noise-free and label noise on the server. We make the following comparisons: 1) the performance of our method under IID case and non-IID case. 2) the performance of our method under non-IID case and FedRep under non-IID case. 3) the performance of our method under non-IID case and other baselines under IID case. Since our aggregation is based on FedAvg, it can be seen from the table that there is a 4-5% performance drop of our proposed method when data on the clients are non-IID. However, it is worth noticing that our proposed method under non-IID case works better than other baselines under IID case. Moreover, our proposed method still outperforms FedRep when both approaches are applied on non-IID data split. Therefore, the above experiment results show the robustness of our proposed method compared with other approaches.

Table 9: Comparison of the accuracy for 100-class classification using our methods and FedRep on the CIFAR100 benchmark dataset under IID and non-IID data split. We report mean (sd) from three trial runs.

ξ	0.0	0.1	0.2	0.3	0.4
			IID		
Our (P)	26.44(0.34)	25.88(0.67)	24.26(0.85)	22.25(0.41)	21.04(0.33)
FedRep	21.00(0.45)	19.34(0.57)	17.89(0.37)	16.16(0.08)	14.80(0.89)
			Non-IID		
Our (P)	21.40(0.21)	20.37(0.28)	20.14(0.26)	18.72(0.45)	17.44(0.39)
FedRep	20.71(0.58)	18.45(0.69)	17.11(0.35)	14.73(0.25)	13.74(0.13)

D.5 Different Number of "Servers"

In our problem setting, we assume without loss of generality that there is one 'specialized center' with a small amount of data from desired label space and we call this center as the server. Such a setting is motivated by the real clinical application in Section 1 that "the centers with the most complex labeling criteria...has much less labeled samples due to labeling difficulty or cost". It is technically trivial to generalize our problem to the case with several entities with data from desired

label space by simplify modifying Equation 11 as

$$R_{\text{overall}}(\boldsymbol{f}) = \frac{1}{C+S}\left\{\sum_{s=1}^{S}\widehat{R}_s(\boldsymbol{f};\widetilde{\mathcal{D}}^s) + \sum_{c=1}^{C}\widehat{R}_c(\boldsymbol{f};\widetilde{\mathcal{D}}^c)\right\}$$

where $\widetilde{\mathcal{D}}^s$ is the noisy labeled data on s-th server and S is the total number of such servers. To show the effectiveness of our proposed method under this setting, we run the following experiments on CIFAR100. We have $C = 10$ clients with super-class clean data with each client having $N_c = 4000$ samples. We also have $n = 20$ noisy sub-class data. We consider the following 3 ways to store these 20×100 images: 1) we have one "server" with all noisy sub-class images. 2) we have 2 "server"s with each containing $n = 10$ noisy sub-class images. 3) we have 4 "server"s with each containing $n = 5$ noisy sub-class images. We change the level of noise $\xi \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and show the results for both `FedMT`(P) and `FedMT`(L). It can be seen from the table that an increase in the number of "server"s does not hurt the performance of `FedMT`(P) and there is a slight performance drop for `FedMT`(L).

Table 10: Comparison of the accuracy for 100-class classification using `FedMT`(P) on the CIFAR100 benchmark dataset with different split strategy of the sub-class label at different noise level ξ. We report mean (sd) from three trial runs.

Split	0.1	0.2	0.3	0.4
		Our (P)		
1x20	28.26(0.88)	27.48(0.48)	26.36(0.84)	24.36(1.47)
2x10	29.11(0.21)	27.88(0.62)	26.38(1.09)	24.16(0.27)
4x5	28.13(0.36)	27.26(0.37)	24.64(0.63)	23.37(0.36)
		Our (L)		
1x20	27.53(0.71)	24.37(0.26)	23.78(0.50)	22.45(0.73)
2x10	25.64(0.52)	24.08(0.29)	22.29(0.71)	20.67(0.43)
4x5	23.88(0.30)	22.58(0.50)	21.52(0.18)	20.51(0.20)

D.6 Sensitivity analysis of projection matrix

In this section, we show the experiment on testing the robustness of our proposed method to the perturbation of the transition matrix $\boldsymbol{T}$. Following Lu et al. (2022), we let

$$\widetilde{\boldsymbol{T}} = \boldsymbol{T} \odot \{(2\epsilon - 1)\boldsymbol{P} + 1\}$$

where each element in $\boldsymbol{P}$ are IID random variables from $[0, 1]$ and $\odot$ is the elementwise product between two matrices. We then standard $\boldsymbol{T}$ so that its column sum is 1. We have run the experiments with $\epsilon \in \{0.1, 0.5, 0.9\}$ for different noise levels and the experiment results is given in Tab. 11. The last row of the table is the experiment result with the true $\boldsymbol{T}$. It can be seen from the figure that our proposed method `FedMT` is robust to a perturbation of the projection matrix $\boldsymbol{T}$.

Table 11: Sensitivity analysis when the projection matrix $\boldsymbol{T}$ is perturbed under different noise levels ξ on CIFAR100. We let $C = 10$, $N_c = 4000$ and $n = 10$ in the experiment. The value of ϵ denotes the degree of perturbation, the larger the value of ξ, the higher the degree of perturbation. The last row is the test accuracy with the true $\boldsymbol{T}$.

ϵ	$\xi = 0.0$	$\xi = 0.1$	$\xi = 0.2$	$\xi = 0.3$
0.1	25.44	23.95	22.27	21.72
0.5	24.89	24.37	22.57	21.68
0.9	24.73	24.13	22.52	20.81
True $\boldsymbol{T}$	25.99	25.15	22.95	21.71

E MORE EXPERIMENTAL RESULTS ON TREMOR SEVERITY PREDICTION

For the experiment on tremor severity prediction in Section 4.3, we let $K = 5$ and the $\boldsymbol{Q}$ matrix between two label spaces is

$$\boldsymbol{Q} = \begin{bmatrix} 3/5 & 2/5 & 0 & 0 & 0 \\ 0 & 1/5 & 3/5 & 1/5 & 0 \\ 0 & 0 & 0 & 2/5 & 3/5 \end{bmatrix}.$$

There are 2 out of 5 classes in the desired space overlaps with one of the class in the other space, each with degree of overlapping $1/5 = 20\%$. To investigate the performance of various approaches under different degrees of overlapping, we create the labels on the servers as below. We equally partition the values into 10 parts with the same intervals (*i.e.,* $[0.5k, 0.5k + 0.5)$ for $k \in \{0, 2, \ldots, 9\}$) and keep $J = 3$ and the corresponding labels the same as before. In this case, the $\boldsymbol{Q}$ matrix between two label spaces becomes

$$\boldsymbol{Q} = \begin{bmatrix} 3/10 & 3/10 & 3/10 & 1/10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/5 & 3/10 & 3/10 & 1/5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/10 & 3/10 & 3/10 & 3/10 \end{bmatrix}$$

Under this case, there are 2 out of 10 classes in the desired space overlaps with one of the class in the other space, each with degree of overlapping is $1/10 = 10\%$. As a comparison and additional experiment results, we compare the performance of all methods for different values of K under the case when there are noisy labels on the server. We let $n = 3$ and the rest of the settings the same as before. The experiment results are given in Tab. 12. It is shown in the table that our method can beat the rest of the methods regardless of the noise level.

Table 12: Comparison of the accuracy using our methods and alternative methods on the sEMG dataset with $n = 3$ and $K = 5$ that has smaller degree of class overlapping and $K = 10$ that has larger degree of class overlapping. We conduct the experiment when $C = 50$ at different noise level ξ. We report mean (sd) from three trial runs. The best method is highlighted in boldface.

ξ	Single	FedMatch	FedRep	FedTrans	*Ours*: `FedMT` (P&L)	
larger degree of overlapping						
0.0	22.35(2.31)	13.04(1.79)	29.23(2.31)	28.75(1.21)	28.97(0.32)	32.59(1.81)
0.1	21.12(2.34)	13.14(0.76)	21.12(2.34)	27.73(1.72)	28.86(0.73)	30.60(1.16)
0.2	18.75(1.68)	12.51(2.19)	18.75(1.68)	25.95(0.99)	27.63(1.05)	30.26(1.69)
0.3	17.49(1.50)	10.54(1.07)	17.49(1.50)	21.56(2.04)	27.00(0.42)	28.16(1.53)
0.4	14.25(1.17)	11.64(1.17)	14.25(1.17)	18.00(2.15)	27.03(1.48)	28.96(1.39)
smaller degree of overlapping						
0.0	34.39(2.86)	34.07(1.32)	41.12(1.34)	51.25(2.57)	67.34(0.56)	64.33(0.48)
0.1	31.04(2.69)	33.03(0.99)	39.64(1.62)	45.52(2.15)	61.80(0.24)	66.56(0.57)
0.2	30.77(1.89)	32.37(1.21)	37.66(1.80)	41.07(1.45)	61.34(0.30)	66.00(0.57)
0.3	27.41(3.02)	31.36(0.99)	34.37(3.38)	39.10(1.31)	61.26(0.31)	67.22(0.30)
0.4	24.66(2.59)	27.56(1.26)	27.09(2.03)	37.01(3.15)	66.44(0.41)	63.74(0.27)

F PROPERTIES OF BASELINES FOR COMPARISON

For the empirical experiments in the paper, we compare the performance of our proposed method with several baseline approaches, namely Single, FedRep, FedMatch, and FedTrans. The details of these approaches and their corresponding objective functions are given in Appendix C. As a summary, we highlight their properties in Table 13 in terms of the type of labels they use on the server and clients, their learning strategy, whether they are simple extensions of FedAvg, and whether the theoretical guarantee on the convergence of the algorithm has been established. For the server label, NA means the server label is not used and Desired means datasets are annotated with classes from the desired label space. For the client label, NA means the client label is not used, Other means datasets annotated with classes from the other label space are used, and Desired class prior means only the information of the prior distribution of the desired label space class labels is used. The

Table 13: The conceptual comparison of the baseline approaches with our proposed method in terms of the type of labeled information used on server and clients, the learning strategy, whether the approach is a simple extension of FedAvg, and whether there is a theoretical guarantee on the convergence of the algorithm.

	Server Label	**Client Label**	**Strategy**	**Simple**	**Theory**
Single	Desired	NA	supervised	Yes	Yes
FedMatch	Desired	Yes	semi-supervised	No	No
FedRep	Desired	Other	supervised	Yes	No
FedTrans	Desired	Other	supervised	Yes	No
Ours	Desired	Other	supervised	Yes	Yes

second last column means whether the proposed method is a simple extension of FedAvg and the last column means whether the theoretical convergence of the method has been established under FL. For methods that are simple extensions of FedAvg, the communication cost is in general much cheaper than other more complicated approaches.