DIURNAL OR NOCTURNAL? FEDERATED LEARNING OF MULTI-BRANCH NETWORKS FROM PERIODICALLY SHIFTING DISTRIBUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning has been deployed to train machine learning models from decentralized client data on mobile devices in practice. The clients available for training are observed to have periodically shifting distributions changing with the time of day, which can cause instability in training and degrade the model performance. In this paper, instead of modeling the distribution shift with a block-cyclic pattern as previous works, we model it with a mixture of distributions that gradually changes between daytime modes and nighttime modes, and find this intuitive model to better match the observations in practical federated learning systems. We propose a Federated Expectation-Maximization algorithm enhanced by Temporal priors of the shifting distribution (FedTEM), which jointly learns a mixture model to infer the mode of each client, while training a network with multiple light-weight branches specializing at different modes. Experiments for image classification on EMNIST and CIFAR datasets, and next word prediction on the Stack Overflow dataset show that the proposed algorithm can effectively mitigate the impact of the distribution shift and significantly improve the final model performance.

1 INTRODUCTION

In Federated Learning (FL), many clients collaboratively train a machine learning model with decentralized data under the orchestration of a central server (Kairouz et al., 2019). FL is designed for privacy protection: the private data of local clients will never be directly transferred to the server or shared with other clients, which follows the principle of data minimization and keeps the attack surface of the system small (Wang et al., 2021). Initially introduced for decentralized training on mobile devices (McMahan et al., 2017), FL has been widely applied for various different applications including finance, health, digital assistance and personalized recommendations (see a few recent surveys (Yang et al., 2019; Kairouz et al., 2019; Li et al., 2020a; Lim et al., 2020; Wang et al., 2021)). Specifically, cross-device FL has been used in practice to boost utility and privacy for applications such as next word prediction (Hard et al., 2018), emoji suggestion (Ramaswamy et al., 2019), query suggestion (Yang et al., 2018), out-of-vocabulary word discovery (Chen et al., 2019), and keyword trigger models (Granqvist et al., 2020; Hard et al., 2020).

A typical communication round of FL starts with a server broadcasting a global model to clients. Clients then perform local computation on private data and only send back aggregated model updates. Finally, the server will aggregate the client updates and apply them to the global model before beginning the next round. In cross-device FL, only a small fraction of clients (from dozens to thousands) out of a large population (possibly in the millions) will participate in a training round. Moreover, in practical FL systems (Bonawitz et al., 2019; Paulik et al., 2021), clients can only participate when local criteria are met, such as when mobile devices are plugged into power, idle, and connected to an unmetered network. For this reason, the central server has limited control over the "sampling" process. In addition, clients that satisfy the local criteria and participate at a specific time of a day can have intra-domain features compared to inter-domain clients from another time, possibly due to user mobile usage preference and time zone difference, which cause a periodically shifting distribution of clients that may degrade the training stability and final model performance (Yang et al., 2018; Eichner et al., 2019).

Conventional centralized training often assumes independent identically distributed (IID) data. FL must deal with non-IID (heterogeneous) client data because the orchestrator (server) is unable to random sample or shuffle the decentralized private data. In cross-device FL, the training objective is the combination of client objectives based on an unknown population distribution, and even if we considered the empirical risk minimization formulation of the training set of clients, it is challenging because of the heterogeneity and lack of sampling control due to privacy and system constraints (Wang et al., 2021). Transfer learning, multi-task learning, and meta learning are introduced in FL to explicitly handle distribution shifts among clients assuming heterogeneous clients are from different domains or tasks (Smith et al., 2017; Khodak et al., 2019). Continual learning is introduced to handle distribution shift due to streaming tasks on each client (Yoon et al., 2021). More recently, distribution shift in clusters of clients instead of each individual client are studied in Mansour et al. (2020); Ghosh et al. (2020), while still assuming the clients can be uniformly accessed during the training process. We provide further discussions of related works in Appendix A.4.

To our knowledge, there are very few previous works (Eichner et al., 2019; Ding et al., 2020) discussed periodical distribution shift of client population in federated learning. Both works assume a block-cyclic structure where daytime clients and nighttime clients will alternatively participate in training. Eichner et al. (2019) proposed the semi-cyclic SGD approach, where clients participating training at different time slots will contribute to and use the corresponding model of the group. However, there are several caveats of semi-cyclic SGD that makes it difficult to apply in practice: (1) It assigns models to clients based on their participation time, but not all clients will participate in federated learning, hence it is hard to decide the correct group for these clients. (2) It maintains a version of the full model for each clients group, which potentially increases the communication cost or privacy risk. (3) The assumption of the abrupt switch from the daytime group to the nighttime group at a specific time of a day is unintuitive in practice. (Ding et al., 2020) is a variant of semi-cyclic SGD inherits these issues.

In this paper, we study periodical distribution shift of clients, and make the following contributions:

1. We revisit the periodical distribution shift. Instead of adoopting the block-cyclic structure (Eichner et al., 2019), we assume a smooth transition between the day mode and night mode, and empirically verify through simulation that its impact on training better matches the observation in practical FL systems.

2. We propose to jointly train a multi-branch network and a Gaussian mixture model to select the branch that better fits the client's distribution based on the feature representations. The lightweight branches for the day and night modes only slightly increase the communication cost, but significantly improve the model performance. Unlike (Mansour et al., 2020; Ghosh et al., 2020; Marfoq et al., 2021), the feature-based mixture model does not rely on labelled data, and can be easily applied to test clients.

3. We propose a Federated Expectation-Maximization algorithm enhanced by Temporal priors of the shifting distribution (FEDTEM) to train the multi-branch networks. We assume participating clients per communication round is a mixture of daytime and nighttime clients, and the number of participating clients from the daytime group will gradually increase as time passes from the night mode to day mode, and vice versa. We use such temporal prior to guide the EM algorithm in the M step to learn parameters that better distinguish the two modes. By exploiting such temporal prior, we can train models that are even more accurate than models trained with uniformly sampled clients.

4. We provide simulations of the distribution shift on three benchmark datasets (EMNIST, CIFAR and Stack Overflow) to evaluate the empirical performance of FL algorithms under the periodic distribution shift with smooth transition. We perform extensive experiments, where the multi-branch networks trained by FEDTEM outperform the distribution-oblivious baselines by a large margin: 3-5% on EMNIST, 2-14% on CIFAR, and 0.4-1.35% on the challenging Stack Overflow dataset under various degrees of distribution shifts. By leveraging the temporal priors, the proposed method can take advantage of the periodic distribution shift and beat the strong baselines of training with uniformly sampled clients by 4%, 4% and 0.45%, respectively.

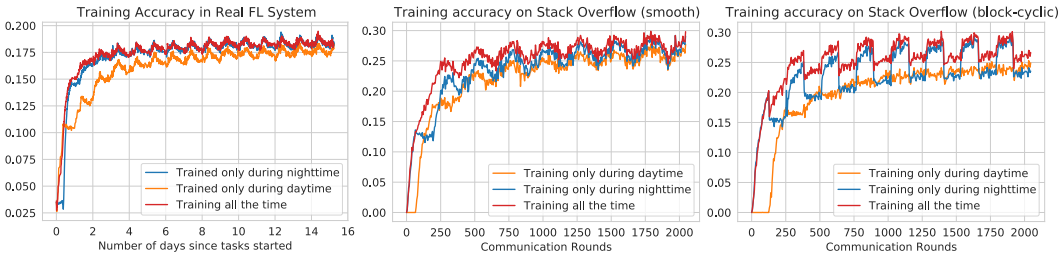

Figure 1: Training accuracy of next word prediction models. (Left): on-device training in a FL system. (Middle): simulation of the smooth distribution shift with $q_{L,1}(t)$ on Stack Overflow ($T = 256$). (Right): simulation of the block-cyclic shift on Stack Overflow ($T = 256$).

2 Modeling Periodical Distribution Shift

FL setting. We consider the federated learning algorithm for a set of clients $\mathcal{I}$, and the i-th client has data samples $\mathcal{D}_i$. The clients are heterogeneous, while the data on each client are IID (independent and identically distributed) sampled from its own distribution. We minimize the expected loss on all clients,

$$\underset{\boldsymbol{w}}{\text{minimize}}\ L(\boldsymbol{w}) = \sum_{i\in\mathcal{I}} p_i L_i(\boldsymbol{w}),\ \text{where}\ L_i(\boldsymbol{w}) = \frac{1}{|\mathcal{D}_i|}\sum_{\boldsymbol{\xi}\in\mathcal{D}_i} \ell(\boldsymbol{w},\boldsymbol{\xi}),\ \text{and}\ \sum_{i\in\mathcal{I}} p_i = 1, \tag{1}$$

where p_i is the weight (probability) of client i, and $\boldsymbol{\xi} = (\boldsymbol{x}, \boldsymbol{y})$ is a training sample pair of data and label on a client. By setting $p_i = |\mathcal{D}_i|/\sum_{j\in\mathcal{I}}|\mathcal{D}_j|$, the federated training loss recovers the empirical risk minimization (ERM) objective on all client samples. For simplicity, we abuse notation and use $\boldsymbol{x} \in \mathcal{D}_i$ to denote a training sample (without label) for client i.

Periodic distribution shift. A subset of clients $\tilde{\mathcal{I}}(t) \subset \mathcal{I}(t)$ are available for training in a communication round t. $\mathcal{I}(t)$ periodically changes with most of the clients from a daytime (nighttime) client group in midday (midnight). Figure 1 (left) shows the federated training curves in a cross-device FL system have daily oscillation. Such oscillation suggests domain difference of daytime clients and nighttime clients was also observed in (Yang et al., 2018), and Yang et al. (2018) conjecture it is due to the time zone difference of clients. Eichner et al. (2019) study the block-cyclic structure, where training takes place over T rounds each day, and clients are alternatively from the daytime group and the nighttime group for a continuous $T/2$ rounds. We simulate the training curves of block-cyclic structure in Figure 1 (right), and observe it is different from the (left) curves from real FL systems.

Smooth transition. Unlike (Eichner et al., 2019), we assume clients at round t are a mixture of daytime clients and nighttime clients, denoted as $\mathcal{I}_1(t)$ and $\mathcal{I}_2(t)$, respectively. Intuitively, since the available population is usually large around the clock, the population distribution of available clients will gradually shift, rather than abruptly jumping from one mode to another as in the block-cyclic structure. To better approximate the behavior in practice, we focus on periodic distribution shifts in which clients come from the day mode $\mathcal{I}_1(t)$ with a probability $q(t)$ that varies *smoothly* from the day mode to night mode. Specifically, we simulate $\mathcal{I}_1(t)$ and $\mathcal{I}_2(t)$ with two disjoint sets of clients with different data distributions, and define $q : \mathbb{R}^+ \to [0, 1]$ to be periodic function with a period of T. At each round t, we sample the clients from the following distribution

$$P(i \in \mathcal{I}_1(t)) = q(t),\ P(i \in \mathcal{I}_2(t)) = 1 - q(t). \tag{2}$$

We consider two types of functions for $q(t)$: *Linear (L)* and *Cosine (C)*, each further parameterized by an exponent factor $p > 0$ to control the transition,

$$q_{L,p}(t) = \left|2\frac{t \bmod T}{T} - 1\right|^p, q_{C,p}(t) = \left[\frac{1}{2}(\cos(2\pi t/T) + 1)\right]^p. \tag{3}$$

We visualize the transition probability $q(t)$ in Figure 6 of the appendix. When $p < 1$, more daytime clients are available in T rounds, and when $p > 1$, more nighttime clients are available. This simulates the difference in the number of completed training rounds during daytime and nighttime observed in practice.

Observation and insight. Figure 1 (middle) simulates the training curve with $q_{L,1}(t)$ to control the probability for sampling from $\mathcal{I}_1(t)$, which more accurately approximate the curves in real FL system. All three curves show that training a single model around the clock achieves reasonable performance on both modes (much better than random) despite the domain differences, which motivates us to train a model with large amount of shared weights. Semi-cyclic SGD (Eichner et al., 2019) provides the worst-case guarantees when the domains of each block are unrelated, which does not fully exploit the data in practice. Another important observation is that the training accuracy reaches its minima (maxima) when the clients are most biased towards the day mode or night mode, e.g., round 1024 and 1152 in the middle figure, from which we can infer the peak momentum of daytime clients and nighttime clients in practice and use it to define a strong prior in the proposed FedTEM algorithm.

3 Mitigating Periodical Distribution Shift

In this section, we propose FedTEM to tackle the periodical distribution shift, in which we joint train a multi-branch network and a Gaussian mixture model (GMM). The GMM can infer latent variables that determine a participating client is from daytime mode or nighttime mode. The performance of the inference is enhanced by priors about the probability of daytime (nighttime) clients at the current round. A multi-branch network is simultaneously trained on the participating clients based on the inferred latent variables, with each branch corresponding to a mode. We will discuss how FedTEM is applied in cross-device FL in detail, where server and client will only exchange information once in a communication round, and clients do not maintain local states.

Multi-branch network. We consider applications in which clients are training a model with common input and output space, despite the distribution shift of clients. Though there exists distribution shift, training a model that shares common features by leveraging data from all clients can enable representation learning with better data efficiency. For vision tasks, for example, a common model can use data across domains to learn common low-level features, while for language tasks it helps with learning common embedding and grammatical rules from the context. To alleviate the communication overhead, we adopt the weight-sharing strategy from multi-task learning (Caruana, 1997) to train a multi-branch network with shared feature representation layers $f(\boldsymbol{w}_f, \boldsymbol{x})$ and specialized prediction branches $g_k(\boldsymbol{w}_k, f(\boldsymbol{w}_f, \boldsymbol{x}))$ for clients from mode k. In this paper, we set each of the prediction branches to be a single linear layer. A k-branch network is potentially more communication efficient and data efficient than having k copies of models.

3.1 Learning the Mixture Model with Temporal Priors

Algorithm 1 FedTEM: Federated EM with Temporal Prior (Training)

Input: A stream of clients $\mathcal{I}(t)$ with periodical distribution shift; Number of communication rounds N; Number of rounds per day T.
Output: Network parameters $\boldsymbol{w}^{N+1} = (\boldsymbol{w}_f^{N+1}, \boldsymbol{w}_1^{N+1}, \boldsymbol{w}_2^{N+1})$, mixture model parameters $\boldsymbol{\theta}^{N+1} = (\boldsymbol{\mu}^{N+1}, \boldsymbol{\sigma}^{N+1}, \boldsymbol{\pi}^{N+1})$.
for $t = 1$ **to** N **do**
  A set of m clients $\mathcal{I}'(t) \subset \mathcal{I}(t)$ is available for training;
  Server broadcasts parameters of the network $\boldsymbol{w}^t$ and the GMM $(\boldsymbol{\mu}^t, \boldsymbol{\sigma}^t, \boldsymbol{\pi}^t)$ to $\mathcal{I}'(t)$;
  for clients $i \in \mathcal{I}'(t)$ **in parallel do**
    Estimate $\boldsymbol{\pi}_i$ on $\mathcal{D}_i$, and choose the branch $k_i^* = \arg\max_k \pi_{ik}^*$; ▷ see Eq. 5
    Given k_i^*, run local updates for the network by optimizing Eq. 6 and get $\tilde{\boldsymbol{w}}_i^{t+1}$;
    On $\mathcal{D}_i$, compute the MLE $\bar{\boldsymbol{\pi}}_i^*$ and all feasible Gaussian parameters $(\tilde{\boldsymbol{\mu}}_i^{t+1}, \tilde{\boldsymbol{\sigma}}_i^{t+1})$; ▷ see Eq. 8,11
  Server aggregates the network and GMM params $\{\tilde{\boldsymbol{w}}_i^{t+1}, \bar{\boldsymbol{\pi}}_i^*, \tilde{\boldsymbol{\mu}}_i^{t+1}, \tilde{\boldsymbol{\sigma}}_i^{t+1} | i \in \mathcal{I}'(t)\}$ from the clients;
  Update the GMM parameters to $(\boldsymbol{\mu}^{t+1}, \boldsymbol{\sigma}^{t+1}, \boldsymbol{\pi}^{t+1})$ with the temporal prior on $\bar{\boldsymbol{\pi}}_i^*$; ▷ see Eq. 12
  Update network parameters to $\boldsymbol{w}^{t+1}$ with the prescribed server optimizer.

We learn a mixture model together with the multi-branch network to infer the mode and select the corresponding branch for prediction of each client. Similar joint learning strategies are applied in (Ghosh et al., 2020; Marfoq et al., 2021) to learn a clustering or mixture model for selection from

multiple networks. Our approach has three key difference: (1) our k-branch network is more communication and data efficient compared to previous k networks, (2) our model selects the branches based on the features instead of the loss, and therefore does not need labeled data to select branches for new clients; (3) we propose the temporal prior to tackle periodic distribution shift when learning the mixture model.

We define discrete latent variables z and ζ to represent which mode a sample and a client comes from, respectively, which will be inferred by our model. Our model also assumes all samples from the same client are from the same mode during training, so the prior $P(\zeta) = P(z)$, but our model can still be used to estimate the posterior for each sample during inference. For convenience, we also introduce a vector $\boldsymbol{\pi}$ with each entry $\pi_k = P(z = k)$ to denote the prior distribution of z. We model $P(\boldsymbol{x}|z = k)$ as a Gaussian distribution $\mathcal{N}(f(\boldsymbol{w}, \boldsymbol{x})|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2)$ on the feature representation $f(\boldsymbol{w}, \boldsymbol{x})$. For efficiency, we assume and learn a diagonal covariance $\boldsymbol{\sigma}_k^2$ for the Gaussian models. We use $\boldsymbol{\theta}$ to denote the parameter of the GMM for convenience. Algorithm 1 summarizes the training process, and the details of each step are provided in the following sections.

3.1.1 Modeling the Client Distributions and Selecting Branches for Training

During training, on each client, before proceeding to the local update steps, we estimate the probabilities of client i coming from each mode k given its training set $\mathcal{D}_i$, and then select the corresponding branch. We assume all samples on one client are from the same mode, so $P(\zeta) = P(z)$. There are two options for inferring the modes

1. Given the current mixture model, estimate $P(\zeta|\mathcal{D}_i)$ from $P(\boldsymbol{x}|z)$ of all $\boldsymbol{x} \in \mathcal{D}_i$;
2. Given the training samples on the client, find the Maximum Likelihood Estimation (MLE) of $P(z)$ for the client and use it for $P(\zeta)$.

Option 1 exploits the current parameters of the mixture model to infer the mode of each client. With the assumption that samples in $\mathcal{D}_i$ are independent, from Bayes' theorem, we can get $P(z|\mathcal{D}_i)$ as

$$P(\zeta = k|\mathcal{D}_i) = \frac{\pi_k \prod_{\boldsymbol{x}\in\mathcal{D}_i} P(\boldsymbol{x}|z = k)}{\sum_{j=1}^{K} \pi_j \prod_{\boldsymbol{x}\in\mathcal{D}_i} P(\boldsymbol{x}|z = j)}. \tag{4}$$

Intuitively, since the client distribution is constantly shifting, we should not expect the prior $P(\zeta)$ to be unchanged from the last round. Option 2 re-estimates the prior $P(z)$ of the mixture model on the current client through MLE and use it as $P(\zeta|\mathcal{D}_i)$. As shown in Figure 5, we find this achieves better empirical results than Option 1. Specifically, we use $\boldsymbol{\pi}_i^*$, the MLE of $P(z)$ on the data of client i, to select the branch for training:

$$\pi_{ik}^* = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{x}\in\mathcal{D}_i} P(z = k|\boldsymbol{x}) = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{x}\in\mathcal{D}_i} \frac{\pi_k \mathcal{N}(f(\boldsymbol{w}^t, \boldsymbol{x})|\boldsymbol{\mu}_k^t, \boldsymbol{\sigma}_k^t)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(f(\boldsymbol{w}^t, \boldsymbol{x})|\boldsymbol{\mu}_j^t, \boldsymbol{\sigma}_j^t)}, \tag{5}$$

where π_{ik}^* is the k-th dimension of $\boldsymbol{\pi}_i^*$, corresponding to the MLE of $P(z = k)$. For completeness, we give the derivation of Eq. 5 in Appendix A.1.

After estimating the distribution of ζ, we can either greedily select the branch k_i^* corresponding to the maximum probability, or we can sample from this distribution. Our experiments in Figure 5 show that the greedy approach is more effective. Since our experiments only consider the case of two modes, let $\bar{k}_i^*$ be the other, less-likely branch. On each client, we train the model to optimize the following objective

$$\underset{\boldsymbol{w}}{\text{minimize}}\ L_i(\boldsymbol{w}) = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{\xi}\in\mathcal{D}_i} \ell_{CE}(g_{k_i^*}(\boldsymbol{w}, \boldsymbol{x}), \boldsymbol{y}) + \lambda \ell_{CE}(g_{\bar{k}_i^*}(\boldsymbol{w}, \boldsymbol{x}), s(\epsilon, \boldsymbol{y})), \tag{6}$$

where $\lambda > 0$ is the regularization strength, $s(\epsilon, \boldsymbol{y}) = \epsilon\frac{1}{n} + (1 - \epsilon)\boldsymbol{y}$ is the label smoothing function with $0 \leq \epsilon \leq 1$ to determine the amount of label smoothing. The regularization enables the other branch to be updated jointly with the feature extractor to prevent mismatch. Meanwhile, it encourages learning distinctive features for the two clusters through label smoothing: the other branch treats samples from the current mode as outliers and is trained to become less certain on these samples.

3.1.2 UPDATING THE MIXTURE MODEL

After the local updates of the network parameters on client i with the chosen branch as described in Section 3.1.1, we obtain the updated network parameters $\boldsymbol{w}_i^{t+1}$, and update the parameters of the GMM based on $\boldsymbol{w}_i^{t+1}$. In this way, the GMM is synchronized with the latest network parameters and ready for inference immediately after training. The GMM parameters are usually estimated with the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). In the federated setting, this can be achieved privately by first running EM on each client i to get the locally updated GMM parameters $(\tilde{\boldsymbol{\mu}}_i^{t+1}, \tilde{\boldsymbol{\sigma}}_i^{t+1}, \tilde{\boldsymbol{\pi}}_i^{t+1})$, and then aggregating on the server.

Further, to target the temporal distribution shift, we integrate our priors $\tilde{q}(t)$ into the aggregation step on the server, so that a ratio of $\tilde{q}(t)$ clients will be used to update $(\boldsymbol{\mu}_1^{t+1}, \boldsymbol{\sigma}_1^{t+1}, \pi_1^{t+1})$ for mode 1, while the remaining clients will only update $(\boldsymbol{\mu}_2^{t+1}, \boldsymbol{\sigma}_2^{t+1}, \pi_2^{t+1})$ for mode 2. The server decides which clients are from mode 1 and which are from mode 2 by their $\tilde{\boldsymbol{\pi}}_i^{t+1}$, since $\tilde{\boldsymbol{\pi}}_i^{t+1}$ estimates the probability of an arbitrary sample from client i to come from each mode. We give details of the steps below.

E step. For each client $i \in \mathcal{I}(t)$, compute the posterior $\gamma_{ik}(\boldsymbol{x})$ for each sample $\boldsymbol{x}$ to infer the probability of $\boldsymbol{x}$ coming from mode k

$$\gamma_{ik}(\boldsymbol{x}) = P(z = k|\boldsymbol{x}) = \frac{\pi_k^t \mathcal{N}(f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x})|\boldsymbol{\mu}_k^t, \boldsymbol{\sigma}_k^t)}{\sum_{j=1}^{K} \pi_j^t \mathcal{N}(f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x})|\boldsymbol{\mu}_j^t, \boldsymbol{\sigma}_j^t)}. \tag{7}$$

Temporal prior. We integrate the temporal prior into the M step, so that we can estimate GMM parameters that better capture the differences in the modes behind the shifting distributions. The temporal prior $\tilde{q}(t)$ is a rough estimate for the ratio of clients coming from the day mode. From the observations in Section 2, we can locate the time when $q(t)$ is most likely to be 0 or 1 by observing when the minima and maxima occur from the train curve. In between these minima and maxima, we consider three types of $\tilde{q}(t)$ in our current experiments: 1) Linear: $\tilde{q}(t) = q_{L,1}(t)$; 2) Cosine: $\tilde{q}(t) = q_{C,1}(t)$; 3) Soft: see Appendix A.2.

Conditioned on $\tilde{q}(t)$, our goal is to estimate the posterior $P(\zeta|\mathcal{D}_i, \boldsymbol{\theta}^t, \tilde{q}(t))$ for each client to decide which mode they are from. To achieve this, similar as in Section 3.1.1, we first obtain the MLE of the prior on $\mathcal{D}_i$ using the updated network parameters $\boldsymbol{w}_i^{t+1}$ as

$$\bar{\pi}_{ik}^* = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{x} \in \mathcal{D}_i} \gamma_{ik}(\boldsymbol{x}), \tag{8}$$

where $\gamma_{ik}(\boldsymbol{x})$ comes from the E step in Eq. 7. Then, we sort the prior probability $\bar{\pi}_{i1}^*$ of mode 1 for all sampled clients $i \in \mathcal{I}'(t)$, and assign $\lfloor \tilde{q}(t) \cdot |\mathcal{I}'(t)| + \frac{1}{2} \rfloor$ clients with the highest $\bar{\pi}_{i1}^*$ to mode 1, denoted as $\tilde{\mathcal{I}}_1'(t)$. The remaining clients, denoted as $\tilde{\mathcal{I}}_2'(t)$, are assigned to mode 2. As a result, for $i_1 \in \tilde{\mathcal{I}}_1'(t)$, $P(\zeta = 1|\mathcal{D}_i, \boldsymbol{\theta}^t, \tilde{q}(t)) = 1$, while for $i_2 \in \tilde{\mathcal{I}}_2'(t)$, $P(\zeta = 2|\mathcal{D}_i, \boldsymbol{\theta}^t, \tilde{q}(t)) = 1$. Note it is possible to design soft assignments where the posteriors are not one-hot, but this one-hot assignment has smaller communication overhead, as we will discuss in the M steps below.

M step (client). In our federated setting, the M step happens on both the server and the clients. It is first executed locally on each client i, where its objective is to find the GMM parameters to maximize the expectation of the log likelihood of data, weighted by the time-varying client-level posterior $P(\zeta = k|\mathcal{D}_i, \boldsymbol{\theta}^t, \tilde{q}(t))$

$$\tilde{\boldsymbol{\theta}}_i^{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{k=1}^{K} P(\zeta = k|\mathcal{D}_i, \boldsymbol{\theta}^t, \tilde{q}(t)) \log P(\mathcal{D}_i, \zeta = k|\boldsymbol{\theta}, \tilde{q}(t)). \tag{9}$$

For each client, $\tilde{q}(t)$ affects the posterior distribution $P(\zeta|\mathcal{D}_i, \tilde{q}(t))$, and the effect depends on other clients. Therefore, the distributions in Eq. 9 should be estimated on the server. However, we cannot send client data to the server, and it is almost impossible in practice to have multiple communication rounds between the server and client (Bonawitz et al., 2019; Wang et al., 2021) so that the server can send the re-estimated prior to the clients. A viable approach is to restrict $P(\zeta|\mathcal{D}_i, \tilde{q}(t))$ to have a finite number of possible values, so that the client can solve Eq. 9 for all possible $P(\zeta|\mathcal{D}_i, \tilde{q}(t))$

and send all feasible solutions to the server. To make $\tilde{q}(t)$ effective, we should set a minimum of K different values for $P(\zeta|\mathcal{D}_i, \tilde{q}(t))$ so that each mode corresponds to a different value. To learn more distinctive representations for the modes, we let $P(\zeta = k|\mathcal{D}_i, \tilde{q}(t)) = 1$ if the client is estimated to come from mode k, so $P(\zeta = j|\mathcal{D}_i, \tilde{q}(t)) = 0$ for all $j \neq k$, and Eq. 9 becomes

$$\begin{aligned}
\tilde{\boldsymbol{\theta}}_i^{t+1} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \log P(\mathcal{D}_i, \zeta = k|\boldsymbol{\theta}, \tilde{q}(t)) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \log P(\zeta = k|\boldsymbol{\theta}, \tilde{q}(t)) + \sum_{\boldsymbol{x}\in\mathcal{D}_i} \log P(\boldsymbol{x}|\zeta = k, \boldsymbol{\theta}, \tilde{q}(t)) \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \pi_k - \sum_{\boldsymbol{x}\in\mathcal{D}_i} \left[\sum_{j=1}^{d} \frac{([f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x})]_j - \mu_{kj})^2}{2\sigma_{kj}^2} + \log \sigma_{kj}\sqrt{2\pi} \right]
\end{aligned} \tag{10}$$

where d is the dimension of the features, μ_{kj}, σ_{kj} denotes the j-th dimension of $\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k$, and we use similar notation for π_k and $[f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x})]_j$. The last equation comes from our modeling assumption that all $\boldsymbol{x} \in \mathcal{D}_i$ are from the same mode. From Eq. 10, we can see the optimal solution is to keep parameters for other modes unchanged, while only updating them for the k-th mode as

$$\tilde{\boldsymbol{\mu}}_{ik}^{t+1} = \frac{1}{|D_i|} \sum_{\boldsymbol{x}\in\mathcal{D}_i} f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x}), \ \ [\tilde{\boldsymbol{\sigma}}_{ik}^{t+1}]^2 = \frac{1}{|\mathcal{D}_i|} \sum_{\boldsymbol{x}\sim\mathcal{D}_i} (f(\boldsymbol{w}_i^{t+1}, \boldsymbol{x}) - \tilde{\boldsymbol{\mu}}_{ik}^{t+1})^2, \tag{11}$$

In this way, we can pack the all the feasible $\tilde{\boldsymbol{\mu}}_{ik}^{t+1}, [\tilde{\boldsymbol{\sigma}}_{ik}^{t+1}]^2$ into matrices that have the same size as one set of GMM parameters, which does not increase the communication cost.

M step (server). The server collects the MLE of the priors $\{\bar{\boldsymbol{\pi}}_i^*|i \in \mathcal{I}'(t)\}$ (Eq. 8) and all the feasible solutions $\{(\tilde{\boldsymbol{\mu}}_i^{t+1}, \tilde{\boldsymbol{\sigma}}_i^{t+1})|i \in \mathcal{I}'(t)\}$ from the clients. Then, with the temporal prior, it divides the clients into two sets $\mathcal{I}_1'(t)$ and $\mathcal{I}_2'(t)$ as described before. Similar as FedAvg (McMahan et al., 2017), it updates the parameters for each mode k weighted by the number of samples from each client as

$$\boldsymbol{\mu}_k^{t+1} = \sum_{i\in\tilde{\mathcal{I}}_k'(t)} \frac{|\mathcal{D}_i|}{M_k^t} \tilde{\boldsymbol{\mu}}_{ik}^{t+1}, \ [\boldsymbol{\sigma}_k^{t+1}]^2 = \sum_{i\in\tilde{\mathcal{I}}_k'(t)} \frac{|\mathcal{D}_i|}{M_k^t} [\tilde{\boldsymbol{\sigma}}_{ik}^{t+1}]^2, \ \pi_k^{t+1} = \frac{M_k^t}{\sum_{j=1}^K M_j^t}, \tag{12}$$

where $M_k^t = \sum_{i\in\mathcal{I}_k'(t)} |\mathcal{D}_i|$ is the total number of samples of clients assigned to mode k. Note the resulting $\boldsymbol{\sigma}^{t+1}$ is not guaranteed to be an unbiased estimate, but we found this process to give satisfying results in practice.

Setting the prior. During test, to compare the model performance with established baselines and ensure fairness, we consider a static test set where the number of clients and samples from both modes are roughly the same. Therefore, we use a uniform distribution for the priors on the test set. This does not make the GMM worse if the underlying $P(\boldsymbol{x}|z)$ is the same for the training and test sets, and our estimate of $P(\boldsymbol{x}|z)$ is accurate. During training, we estimate the prior $\boldsymbol{\pi}^t$ in every step to match the shifting distribution. However, we find it beneficial to use a running average of $\pi_k^{t+1} = \beta\pi_k^t + (1-\beta)\pi_k M_k^t / (\sum_{j=1}^K M_j^t)$ during training, which achieves better results than setting $\beta = 0$ or using uniform prior in our preliminary experiments.

4 Experiments

Dataset. We consider two image classification datasets, EMNIST and CIFAR, and one next word prediction task on Stack Overflow (SO). The split of the day mode ($\mathcal{I}_1$) and night mode ($\mathcal{I}_2$), and other statistics, are shown in Table 1. On Stack Overflow, we use a vocabulary size of 10K and report the test accuracy without special tokens. We truncate each sentence to have no more than 20 tokens.

Architecture of the multi-branch networks. For image classification, on EMNIST, we train LeNet with 2 Conv layers and 2 FC layers, while on CIFAR, we train a ResNet-18 with the Batch Normalization replaced by Group Normalization (Wu & He, 2018) for stability in federated learning. We use the last FC layer of these convolutional networks as the multi-branch part and share

Table 1: Statistics of the datasets. On Stack Overflow, # Classes refers to vocabulary size. We use $|\mathcal{D}_\cdot|$ to denote number of samples in each subset. On Stack Overflow, $\mathcal{I}_1$ or $\mathcal{I}_2$ clients only have the question or answer sentences respectively.

Dataset	# Classes	$\mathcal{I}_1$	$\lvert\mathcal{I}_1\rvert$	$\lvert\mathcal{D}_{\mathcal{I}_1}\rvert$	$\mathcal{I}_2$	$\lvert\mathcal{I}_2\rvert$	$\lvert\mathcal{D}_{\mathcal{I}_2}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$
EMNIST	62	Digits	3383	341,873	Characters	3400	329,712	77,483
CIFAR	110	CIFAR10	500	50,000	CIFAR100	500	50,000	20,000
Stack Overflow	10K	Questions	171K	67M	Answers	171K	67M	16M

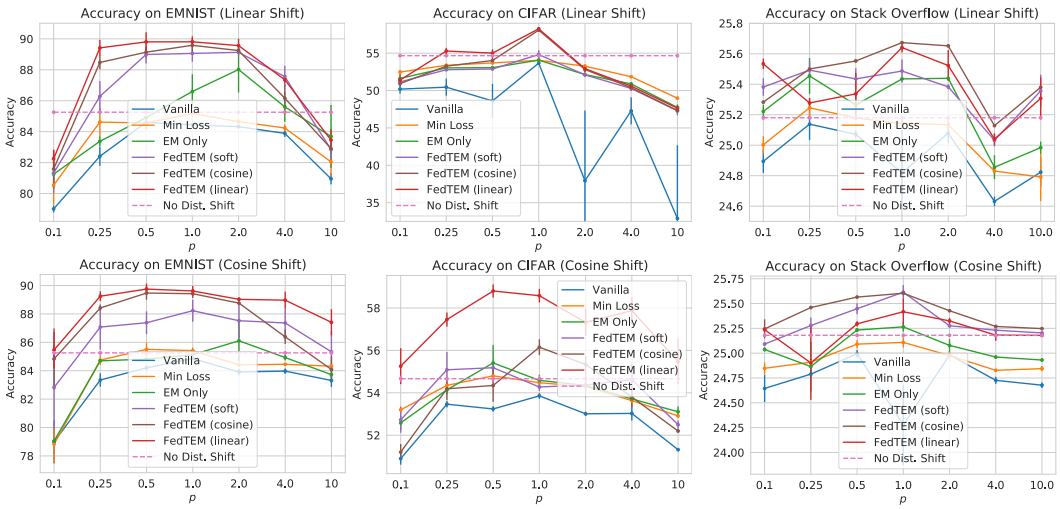

Figure 2: Comparing the methods on EMNIST, CIFAR and Stack Overflow under linear (top row) and cosine (bottom row) distribution shifts with different p's.

all the remaining layers for all modes. We use the output from the shared feature extractor for GMM, which are 128 and 512 dimensional respectively. On Stack Overflow, we train a single-layer LSTM (Hochreiter & Schmidhuber, 1997) with a hidden size of 670 and embedding size of 96. To alleviate the communication overhead and prevent overfitting, we define the branches to be the last projection layer before the final prediction layer and share the remaining weights for both modes, which is 15x smaller than the final prediction layer. We concatenate the average of all tokens' features from both branches for GMM, which has dimensionality 192.

Methods to compare. The comparison with baselines are shown in Figure 2. For fair comparison, we maintain capacity of the network unchanged. For the baselines, we train the same multi-branch network without any branch-selection, and take the averaged output from both branches for prediction. We consider a baseline trained under the same distribution shift (denoted as Vanilla), and a strong baseline trained without any distribution shift where the clients are sampled uniformly from all clients (denoted as "No Dist. Shift" or NDS). For our method, FEDTEM, we compare the effect of the three different temporal priors (Linear, Cosine and Soft). We also compare with the version without temporal prior ("EM Only") to demonstrate the efficacy of the prior, where the only difference from FEDTEM is that the server, in the M step, greedily uses client i to update parameters of mode $k_i^* = \arg\max_k \bar{\pi}_{ik}^*$ (see Eq. 8 and Eq. 12). We also compare with an approach ("Min Loss") similar to IFCA (Ghosh et al., 2020), where same as IFCA, we choose the branch with minimum loss on the local training set of each client during training, but different from IFCA, we choose branches with highest certainty during test when labels are unavailable, which is achieved by evaluating the KLD between the prediction and a uniform distribution on the labels. We have also applied the same label smoothing regularization (Eq. 6) to "Min Loss" and found it improves the baseline results.

Hyperparameters. We use FedAdam (Reddi et al., 2021) as the optimizer. Table 2 shows the training hyperparameters. We use $T = 256$ for the majority of our results, and we evaluate both linear and cosine distribution shifts under $p \in \{0.1, 0.25, 0.5, 1, 2, 4, 10\}$. We use $\beta = 0.99$ for the moving average of $\boldsymbol{\pi}_k$ in all experiments. For FEDTEM and Min Loss, we also do a grid search on the label smoothing parameters ϵ and λ. For each set of hyperparameters, we run 3 experiments with different random seeds and report their mean and standard error.

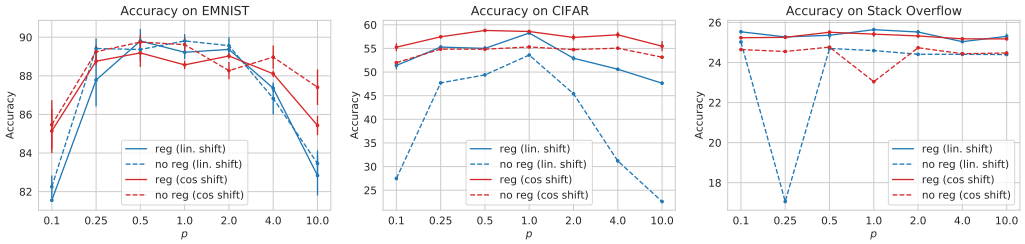

Figure 3: Compare the effect of the label smoothing regularization on FEDTEM under linear and cosine distribution shifts. We use the linear prior in all cases.

Main Results The best results of all compared methods are shown in Figure 2. By comparing results of Vanilla and NDS, we find the temporal distribution shift indeed causes issues for normal training, causing the worst-case decrease in accuracy by more than 7%, 20% and 1% respectively on EMNIST, CIFAR and Stack Overflow. The Min Loss approach alleviates the drop and sometimes catches up with NDS. Surprisingly, with FEDTEM, the accuracy can be even higher than models trained in the NDS setting in most cases, indicating FEDTEM obtains more distinctive feature representations by learning to separate the two modes in the feature space and learning specialized prediction branches. The improvement on the strong NDS baseline can be as high as 4%, 4% and 0.45% on the three datasets. FEDTEM is not better than NDS when p deviates too much from 1, but the data distribution is extremely skewed in such settings, where FEDTEM still improves over other methods in most cases. Among different temporal priors for FEDTEM, we find the linear prior to perform the best in the two image classification tasks, while the smoother cosine prior performs the best on SO.

Effect of label smoothing regularization. Shown in Figure 3. On EMNIST where the task is relatively easy and network is simple, the smoothing regularization does not show significant improvements. However, it is critical to the success on CIFAR and SO. Without the regularization, one of the branches will not be updated simultaneously with the feature extractor, causing instability and resulting in low test accuracy.

MLE for Evaluation. During evaluation, for image classification, we find it beneficial in most cases to use the MLE in Eq. 5 on the minibatches (typically of size no larger than 64) to select the best branch, compared with the sample-level inference. We show its benefit in Figure 4. In practice, similar approach can be realized through caching. We find Min Loss does not show much difference when a similar approach is applied, where branches are chosen by the average of per-sample certainty. The baseline model even gets worse performance with this approach, as shown in Figure 4, indicating our model is much better at distinguishing two modes in expectation. However, this approach does not show improvements for FEDTEM on SO.

Effect of T. As shown in Figure 7, for FEDTEM with linear prior on EMNIST, a smaller T tends to decrease the performance when p deviates too much from 1. In such scenarios, the distribution changes frequently and abruptly. However, the performance of FEDTEM is maintained when the distribution changes in a frequent but more balanced way.

5 CONCLUSIONS

In this paper, we demonstrated that the smooth transition model for simulating the periodical distribution shift better matches practical FL systems, and developed FEDTEM algorithm to train multi-branch networks to tackle the distribution shift. FEDTEM incorporated priors of the temporal distribution shifts into learning a GMM to guide the network to select corresponding branches for clients of different modes. The GMM is defined in the feature space and does not require labelled data for inference, hence is ready to use for new test clients. The branches are light-weight with little communication overhead, and the model demonstrates significant improvements in test accuracy on the benchmark datasets. FEDTEM also satisfies many other real-world constraints like single round communication and stateless clients (Bonawitz et al., 2019; Paulik et al., 2021; Wang et al., 2021), which makes it ready to be deployed in practical FL systems.

6 REPRODUCIBILITY STATEMENT

For reproducibility, we have reported the mean and standard error for 3 experiments with different random seeds for all settings. The dataset we used for simulations are all available in Tensorflow Federated, and our implementation is largely based on the code from Adaptive Federated Optimization.[1] We have listed our hyperparameters in Table 2. We have given the pseudo code in Algorithm 1 and the detailed formulas and derivations of our algorithm in Section 3.1. We are committed to open-source our implementation upon publication.

REFERENCES

Maruan Al-Shedivat, Jennifer Gillenwater, Eric Xing, and Afshin Rostamizadeh. Federated learning via posterior averaging: A new perspective and practical algorithms. In *International Conference on Learning Representations (ICLR)*, 2021.

Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *SysML*, 2019.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635*, 2019.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1): 1–22, 1977.

Yucheng Ding, Chaoyue Niu, Yikai Yan, Zhenzhe Zheng, Fan Wu, Guihai Chen, Shaojie Tang, and Rongfei Jia. Distributed optimization over block-cyclic data. *arXiv:2002.07454*, 2020.

Hubert Eichner, Tomer Koren, Brendan McMahan, Nathan Srebro, and Kunal Talwar. Semi-cyclic stochastic gradient descent. In *ICML*, 2019.

Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. *NeurIPS*, 33, 2020.

Filip Granqvist, Matt Seigel, Rogier van Dalen, Áine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy. *arXiv preprint arXiv:2008.02651*, 2020.

Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

Andrew Hard, Kurt Partridge, Cameron Nguyen, Niranjan Subrahmanya, Aishanee Shah, Pai Zhu, Ignacio Lopez Moreno, and Rajiv Mathews. Training keyword spotting models on non-iid data with federated learning. *arXiv preprint arXiv:2005.10406*, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[1] https://github.com/google-research/federated/tree/780767fdf68f2f11814d41bbbfe708274eb6d8b3/optimization

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for on-device federated learning. *International Conference on Machine Learning (ICML)*, 2020.

Mikhail Khodak, Maria-Florina F Balcan, and Ameet S Talwalkar. Adaptive gradient-based meta-learning methods. In *Advances in Neural Information Processing Systems*, 2019.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020a.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems*, 2020b.

Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B. Allen, Randy P. Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 2020.

Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv:2002.10619*, 2020.

Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. *arXiv:2108.10252*, 2021.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pp. 1273–1282. PMLR, 2017.

Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503*, 2021.

Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *ICLR*, 2021.

Amirhossein Reisizadeh, Farzan Farnia, Ramtin Pedarsani, and Ali Jadbabaie. Robust federated learning: The case of affine distribution shifts. In *NeurIPS*, 2020.

Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, 2020.

Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, Keith Rush, and Sushant Prakash. Federated reconstruction: Partially local federated learning. *arXiv preprint arXiv:2102.03448*, 2021.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, 2017.

Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Blaise Aguera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, et al. A field guide to federated optimization. *arXiv:2107.06917*, 2021.

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

Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

Jaehong Yoon, Wonyong Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. Federated continual learning with weighted inter-client transfer. In *International Conference on Machine Learning*, pp. 12073–12086. PMLR, 2021.

Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning. *ICML*, 2021.

A APPENDIX

A.1 DERIVATION OF OPTION 2

We consider maximizing the expectation of the complete-data log likelihood $\log P(\mathcal{D}_i, \boldsymbol{Z}|\boldsymbol{\theta}^{t+1})$ under the posterior distribution $P(\boldsymbol{Z}|\mathcal{D}_i, \boldsymbol{\theta}^t)$, where $\boldsymbol{Z}$ denotes the collection of latent variables associated with each of the sample $\boldsymbol{x} \in \mathcal{D}_i$. Formally, it is solving the following constrained optimization problem

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{maximize}}\ \mathbb{E}_{\boldsymbol{Z}\sim P(\boldsymbol{Z}|\mathcal{D}_i,\boldsymbol{\theta}^t)}\left[\log P(\mathcal{D}_i, \boldsymbol{Z}|\boldsymbol{\theta})\right], \\ &\text{subject to } \sum_{j=1}^{K} \pi_j = 1, \pi_j \geq 0, \text{for all } 1 \leq j \leq K. \end{aligned} \tag{13}$$

We introduce an indicator function $\mathbb{1}_{[z=k]}$ which is 1 if $z = k$ or 0 otherwise. In this way, the log likelihood can be represented as

$$\begin{aligned} \log P(\mathcal{D}_i, \boldsymbol{Z}|\boldsymbol{\theta}) &= \log \prod_{n=1}^{|\mathcal{D}_i|} \prod_{k=1}^{K} [P(\boldsymbol{x}_n, z_n|\boldsymbol{\theta})]^{\mathbb{1}_{[z=k]}} \\ &= \sum_{n=1}^{|\mathcal{D}_i|} \sum_{k=1}^{K} \mathbb{1}_{[z_n=k]} \left[\log \pi_k + \log P(\boldsymbol{x}_n|z_n, \boldsymbol{\theta})\right]. \end{aligned} \tag{14}$$

Meanwhile,

$$\mathbb{E}_{\boldsymbol{Z}\sim P(\boldsymbol{Z}|\mathcal{D}_i,\boldsymbol{\theta}^t)}\left[\mathbb{1}_{[z_n=k]}\right] = P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) = \frac{\pi_k^t P(\boldsymbol{x}_n|z_n = k, \boldsymbol{\theta}^t)}{\sum_{j=1}^{K} \pi_j^t P(\boldsymbol{x}_n|z_n = j, \boldsymbol{\theta}^t)}. \tag{15}$$

Plug Eq. 14 and Eq. 15 back into Eq. 13, and ignore terms that do not depend on $\boldsymbol{\pi}^{t+1}$, we find we are solving the following problem for $\boldsymbol{\pi}^{t+1}$

$$\begin{aligned} &\underset{\boldsymbol{\pi}}{\text{maximize}} \sum_{n=1}^{|\mathcal{D}_i|} \sum_{k=1}^{K} P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) \log \pi_k, \\ &\text{subject to } \sum_{j=1}^{K} \pi_j = 1, \pi_j \geq 0, \text{for all } 1 \leq j \leq K. \end{aligned} \tag{16}$$

The Lagrangian multiplier of this problem is

$$h(\boldsymbol{\pi}, \lambda) = \sum_{n=1}^{|\mathcal{D}_i|} \sum_{k=1}^{K} P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) \log \pi_k + \lambda(1 - \sum_{j=1}^{K} \pi_j). \tag{17}$$

Let the derivative w.r.t. π_k be 0,

$$\frac{\partial h}{\partial \pi_k} = \sum_{n=1}^{|\mathcal{D}_i|} P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) \frac{1}{\pi_k} - \lambda = 0. \tag{18}$$

Multiply both sides of Eq. 18 with π_k, sum over $1 \leq k \leq K$, and apply the constraint that $\sum_{k=1}^{K} \pi_k = 1$, we have

$$\lambda = \sum_{n=1}^{|\mathcal{D}_i|} \sum_{k=1}^{K} P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) = |\mathcal{D}_i|. \tag{19}$$

Plug this back into Eq. 18, we get the equation as desired.

$$\pi_k^t = \frac{1}{|\mathcal{D}_i|} \sum_{n=1}^{|\mathcal{D}_i|} P(z_n = k|\boldsymbol{x}_n, \boldsymbol{\theta}^t) \tag{20}$$

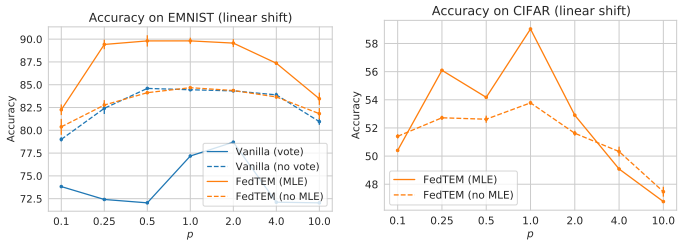

Figure 4: Comparing the effect of MLE on EMNIST and CIFAR.

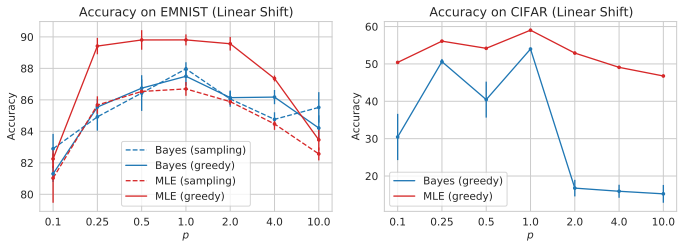

Figure 5: Comparing the Bayesian and MLE based branch selection on EMNIST and CIFAR under linear distribution shifts. We also compare the greedy branch selection vs. sampling based branch selection on EMNIST.

A.2 The Soft Prior

The soft prior $\tilde{q}_S(t)$ gives strong signals only at time steps where we believe $q(t) = 0$ or $q(t) = 1$. In between, it only requires the ratio to be non-increasing or non-decreasing, where the ratio is estimated using the posterior from Eq. 7 as

$$\tilde{q}'_S(t) = \frac{|\{\boldsymbol{x} | \boldsymbol{x} \in \mathcal{D}_i, \gamma_{i1}(\boldsymbol{x}) > \gamma_{i2}(\boldsymbol{x})\}|}{|\mathcal{D}_i|}, \tag{21}$$

i.e., the ratio of samples whose posterior has higher probability on mode 1. In this way, the model can still learn to distinguish the modes but we do not enforce a strong prior at every time step. Specifically, it is defined as

$$\tilde{q}_S(t) = \begin{cases} 1, & \text{if } t \bmod T = 1 \\ 0, & \text{if } t \bmod T = \frac{T}{2} \\ \min(\tilde{q}_S(t-1), \tilde{q}'_S(t)), & \text{if } 1 < t \bmod T < \frac{T}{2} \\ \max(\tilde{q}_S(t-1), \tilde{q}'_S(t)). & \text{if } t \bmod T > \frac{T}{2} \end{cases} \tag{22}$$

A.3 Verifying the effect of MLE for evaluation

To quantize the effect the batch-level MLE, we compare the results of our method with or without MLE on EMNIST and CIFAR. As shown in Figure 5, this only decreases the accuracy of the baseline model, indicating our model is much better at distinguishing the two modes in expectation.

Table 2: Training hyperparameters on the datasets. The network parameters are trained for one epoch on each client. In addition, on Stack Overflow, we also limit each client to use no more than 512 samples during training.

Dataset	Server Opt	Server LR	ϵ	Client Opt	Client LR	$\lvert\tilde{I}(t)\rvert$	Batch Size	Total Rounds
EMNIST	Adam	$10^{-2.5}$	10^{-4}	SGD	$10^{-1.5}$	10	20	2049
CIFAR	Adam	1	10^{-1}	SGD	$10^{-1.5}$	10	20	8196
Stack Overflow	Adam	0.01	10^{-5}	SGD	$10^{-0.5}$	50	16	2048

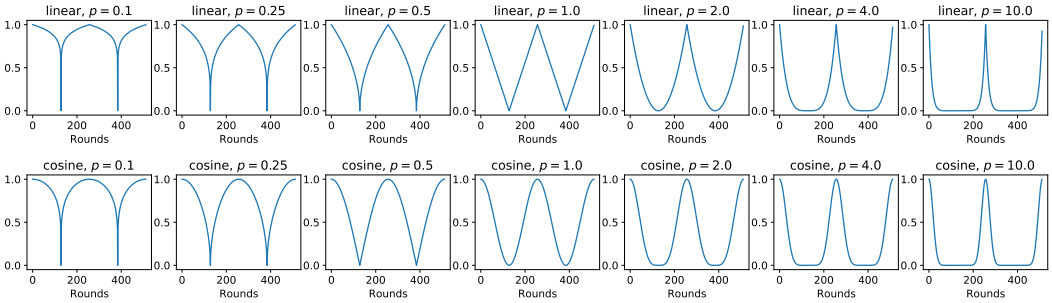

Figure 6: Probability of sampling of clients from $\mathcal{I}_1(t)$ in each round, with $T = 256$, under the linear and cosine transit functions and different values of p.

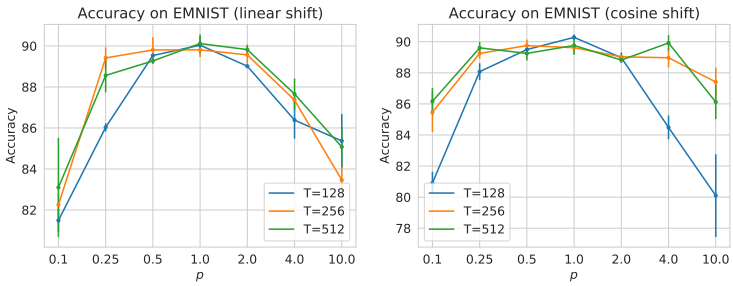

Figure 7: Evaluating the effect of the underlying T on FEDTEM with linear prior.

A.4 ADDITIONAL RELATED WORKS

As mentioned above, semi-cyclic SGD (Eichner et al., 2019) is the only previous work we are aware of that explicitly consider periodical distribution shift in FL. We now review other related works that either consider other types of distribution shift, or have (weak) similarity as the proposed FEDTEM method.

Heterogeneity and client distribution shift. Client heterogeneity is an active topic in FL, and various methods have been proposed to address the distribution difference among clients. Notably, FedProx (Li et al., 2020b) applies proximal regularizer when perform client updates; SCAFFOLD (Karimireddy et al., 2020) uses control variates as local states on clients for variance reduction; FedPA (Al-Shedivat et al., 2021) provides a Bayesian view and adopts posterior sampling; FedGen (Zhu et al., 2021) learns a generator that can be shared among clients; FedRobust (Reisizadeh et al., 2020) applies gradient descent ascent to tackle distribution shifts in the form of affine transforms in the image domain. We kindly ask interested readers to find more papers on heterogeneity in a few recent surveys (Kairouz et al., 2019; Li et al., 2020a; Wang et al., 2021). All these methods consider the distributions shift among different clients, while the proposed FEDTEM considers periodic distribution shift of client population.

Clustering and mixture models. Several methods have proposed to study the distribution shift under clustering or mixture models of clients in FL. Clustered Federated Learning (Sattler et al., 2020) applies an one shot clustering based on trained global model, and then train a personalized model for each cluster. Sattler et al. (2020) implicitly assumes strong control over population and clients sampling. Ghosh et al. (2020) and Mansour et al. (2020) propose a similar algorithm that alternatively perform clustering and updating the personalized models for corresponding clusters. During the preparation of this draft, we notice a preprint (Marfoq et al., 2021) proposes a Federated EM algorithm to learn a mixture model for each client and weigh the predictions from multiple models. By enforcing the probability in mixture models of a client to be an one-hot vector, (Marfoq et al., 2021) can recover the clustering algorithm in (Ghosh et al., 2020; Mansour et al., 2020).

However, these methods do not consider the periodic distribution shift. Moreover, these methods are designed for personalization when labelled data can be accessed during inference time to calculate loss and select the corresponding personalized models.

Multi-branch networks in FL. Multi-branch networks have been explored in FL when a personalized model is preferred for each client: the branchs are either locally stored on clients (Arivazhagan et al., 2019; Liang et al., 2020), or reconstructed based on client data (Singhal et al., 2021). In this paper, we design multi-branch networks to mitigate periodical distribution shift by training each branch with corresponding clients.