DYNAFed: Tackling Client Data Heterogeneity with Global Dynamics

Renjie Pi\textsuperscript{1*} Weizhong Zhang\textsuperscript{2†} Yueqi Xie\textsuperscript{1*} Jiahui Gao\textsuperscript{3} Xiaoyu Wang\textsuperscript{1}
Sunghun Kim\textsuperscript{1} Qifeng Chen\textsuperscript{1†}
\textsuperscript{1}The Hong Kong University of Science and Technology
\textsuperscript{2}Fudan University \textsuperscript{3}The University of Hong Kong

Abstract

The Federated Learning (FL) paradigm is known to face challenges under heterogeneous client data. Local training on non-iid distributed data results in deflected local optimum, which causes the client models drift further away from each other and degrades the aggregated global model’s performance. A natural solution is to gather all client data onto the server, such that the server has a global view of the entire data distribution. Unfortunately, this reduces to regular training, which compromises clients’ privacy and conflicts with the purpose of FL. In this paper, we put forth an idea to collect and leverage global knowledge on the server without hindering data privacy. We unearth such knowledge from the dynamics of the global model’s trajectory. Specifically, we first reserve a short trajectory of global model snapshots on the server. Then, we synthesize a small pseudo dataset such that the model trained on it mimics the dynamics of the reserved global model trajectory. Afterward, the synthesized data is adopted to help aggregate the deflected clients into the global model.

1. Introduction

Federated learning (FL) has become a popular distributed training paradigm to alleviate the server’s computational bur-

\*Joint first authors. Code is available at this link.
\dagger Correspondence to Weizhong Zhang (weizhongzhang@fudan.edu.cn) and Qifeng Chen (chenqifeng22@gmail.com)

Figure 1. Illustration of DYNAFed. Firstly, we run the standard FedAvg for \(L\) communication rounds and save the checkpoints to form a trajectory of the global model at the server. Then, we synthesize a pseudo dataset \(D_{syn}\), with which the network can be trained to mimic the dynamics of the global trajectory. In this way, the knowledge that captures the essential information about the global data distribution is transferred from the global dynamics to \(D_{syn}\). Afterward, \(D_{syn}\) is adopted to help aggregate the deflected clients into the global model at the server.

den and preserve clients’ data privacy [3,23,31,45]. In the FL paradigm, the clients only have access to their private datasets, while the server is responsible for aggregating the clients’ updates into a global model. The most prevalent approaches in FL are based on local-SGD [34] (also referred to as FedAvg), where the client model is updated locally for multiple steps before being sent and merged on the server. Such approaches save communication costs and perform well given the client data are iid-distributed. However, in real-world applications such as healthcare [22,23,36,37] and bio-metrics [2], the client data usually demonstrates heterogeneity (highly non-iid), which deflects the local optimum from the global optimum [24,32,52] and makes the locally
trained clients biased. Therefore, naively averaging the client models results in slow convergence and performance drop.

To alleviate the difficulty of training under heterogeneity, a few lines of work have been frequently discussed. The first line attempts to modify the local training process, including imposing regularization on the client models [1, 24, 29, 31] and data sharing or augmentation [35, 38, 48, 52]. However, these solutions have a high requirement for the server’s control of local clients. An orthogonal line focuses on refining the global model in the server aggregation process [6, 33, 40, 44, 46]. The majority of these methods typically require a large external dataset on the server, then use it to align the outputs of the global model with that of the client ensemble [6, 15, 34, 44]. Unfortunately, such a large-scale task-related dataset is often hard to acquire in reality. To circumvent this limitation, a few data-free knowledge distillation (KD) approaches are recently proposed [50, 53]. These methods attempt to transfer the knowledge contained in the global model to a generator, which is subsequently leveraged to produce pseudo data to either help local training at the clients [53] or finetune the global model at the server [50]. It is clear that these methods require a global model with reasonable performance to ensure the generation of helpful pseudo data. However, such requirement is hard to achieve in practice since the global model often performs poorly under heterogeneity, especially in the early rounds of training. Other solutions include personalized FL [16, 27, 30, 39], similarity clustering [4, 10, 15], meta learning [21, 28], etc.

Even though the above-mentioned works propose techniques to alleviate the challenges posed by heterogeneity to some extent, they do not tackle the issue from its root cause: the data is unevenly scattered at different clients and is kept locally due to privacy concerns, which is also the main obstacle for training an accurate global model. Ideally, imagine if we can collect all the client data to the server, then training can be directly conducted at the server, and the heterogeneity issue no longer exists. However, this reduces to regular training and conflicts with the original purpose of FL to protect client privacy. We then raise a natural question: *is it possible to derive the essential information about the global data distribution on the server to help training without compromising client privacy?*

Despite the global model typically performing poorly due to heterogeneity, the changes in its parameters are steered jointly by the data scattered at different clients. Therefore, the update dynamics of the global model contain knowledge about global data distribution. Driven by this intuition, we propose DYNAFed to explicitly unearth such knowledge hidden in the global dynamics and transfer it to a pseudo dataset $D_{syn}$ at the server. $D_{syn}$ can then approximate the global data distribution on the server to aid aggregation. More specifically, inspired by recent works in dataset condensation [5, 41, 51], we formulate the data synthesis process into a learning problem, which minimizes the distance between the trajectory trained with $D_{syn}$ and the global model trajectory derived with $D$. Fine-tuning the aggregated global model with $D_{syn}$ effectively alleviates the performance degradation caused by deflected clients. An appealing feature of our DYNAFed is that the data can be synthesized using just the global model’s trajectory of the first few rounds, which enables $D_{syn}$ to take effect and help aggregation from early rounds. In addition, the synthesizing process only needs to be conducted once in practice, after which the derived $D_{syn}$ can be directly applied in subsequent rounds to help aggregate the deviated client models.

Notably, our framework can be readily applied to the majority of FL approaches, since we rely on only the history of the global model’s parameters for synthesizing $D_{syn}$, which is available in the conventional setting of FedAvg-based methods. Furthermore, because we extract global knowledge using the global dynamics, rather than any client-specific information as in [12, 17, 18, 20, 47], the derived $D_{syn}$ comprises of information mixed with the entire global data distribution, thus prevents leakage of client privacy.

Our DYNAFed possesses the following advantages compared with previous approaches: 1) It leverages the knowledge of the global data distribution to alleviate the aggregation bias of the global model without depending on any external datasets; 2) DYNAFed is able to generate informative data in the early rounds of federated learning, which significantly helps convergence and stabilizes training in subsequent rounds; 3) Compared with [50, 53], which need to keep updating the generator throughout all communication rounds, the data synthesis process in our method only needs to be done once, which reduces the computational overhead. In summary, we make the following contributions:

- We propose a practical approach named DYNAFed for tackling the heterogeneity problem, which extracts and exploits the hidden information from the global model’s trajectory. In this way, the server can access the essential knowledge of the global data distribution to reinforce aggregation;
- We experimentally show the synthesized dataset helps stabilize training, boost convergence and achieve significant performance improvement under heterogeneity;
- We provide insights and detailed analysis into the working mechanisms of the proposed DYNAFed both experimentally and theoretically.

### 2. Related Work

**Regularization-based Methods** FedAvg [34] is the most widely used technique in FL, which periodically aggregates the local models to the global model in each communication round. FedProx [31] proposes to impose a proximal term during local training, such that the local model does not
drift too far from its global initialization; Scaffold [24] introduces a control variate and variance reduction to alleviate the drift of local training. Moon [29] proposes to leverage the similarity between model representations to regularize the client local training. FedDyn [1] introduces the linear and quadratic penalty terms to correct the clients’ objective during local training. These methods are orthogonal to our approach and can be jointly used.

**Data-Dependent Knowledge Distillation Methods** This line of work attempts to distill client ensemble knowledge into the global model. [33] proposes to use an unlabeled external dataset on the server to match the global model’s outputs with that of the client ensemble. On top of this, [6] further proposes to sample and combine higher-quality client models via a Bayesian model ensemble. Subsequently, some advanced techniques, such as pre-training [15] and weighted consensus distillation scheme [8] are proposed. These methods typically require a large amount of data following similar distribution as the task data, which is usually hard to acquire in practice. Besides, these methods need to conduct KD with a large dataset in every communication round, which introduces prohibitive computational overhead.

**Data-free Knowledge Distillation Methods** Recently, a few works have proposed to perform KD in a data-free manner by synthesizing data with generative models [50,53]. [53] proposes to train a lightweight generator on the server, which produces a feature embedding conditioned on the class index. The generator is then sent to the clients to regularize local training. [50] trains a class conditional GAN [14] on the server, where the global model acts as the discriminator. The pseudo data is then used to finetune the global model. These methods all depend on the global model for training the generator. Unfortunately, the model performance is often poor under high data heterogeneity, which makes the quality of the pseudo data questionable. On the other hand, due to the use of update dynamics of the global model rather than an individual model, our method can synthesize high-quality data containing rich global information even if the global model performs poorly.

### 3. Preliminary

**Federated Learning.** Suppose we have $M$ clients in a federated learning system. For each client $m \in [M]$, a private dataset $D_m = \{(x_i, y_i)\}_{i=1}^{|D_m|}$ is kept locally. The overall optimization goal of the FL system is to jointly train a global model which performs well on the combination of local data, denoted as $D = \bigcup_{m=1}^{M} D_m$, where $D$ is from a global distribution. Let $\alpha_m = \frac{|D_m|}{|D|}$ denote the portion of data samples on client $m$. Let $w \in \mathbb{R}^d$ denote the model parameter to optimize, and $L_m(w, D_m) = \frac{1}{|D_m|} \sum_{\xi \in D_m} \ell(w, \xi)$ denote the empirical risk with the loss function $\ell(\cdot, \cdot)$. The optimization problem of a generic FL system can be formulated as follows:

$$
\min_w \mathcal{L}(w, D) = \sum_{m=1}^{M} \alpha_m L_m(w, D_m).
$$

**FedAvg.** The main-stream solutions of Eqn. 1 rely on local-SGD to reduce the communication cost of transferring gradients. FedAvg [34] is the most prevalent approach, which uses a weighted average to aggregate the locally trained models into a global model in each communication round. Typically, not all the clients participate in every communication round. Suppose $M_e \subset [M]$ is the set of the participated clients in the $c$-th round, and $P_e = \sum_{m \in M_e} \alpha_m$. The aggregation process for the global model $w^{c+1}$ at the end of $c$-th communication round can be formulated as:

$$
w^{c+1} = \frac{1}{P_e} \sum_{m \in M_e} \alpha_m w^c_m,
$$

where $w^c_m$ denotes the client $m$’s locally trained model. After the aggregation, the updated global model $w^{c+1}$ is then distributed to and client and utilized to initiate the $c+1$-th round of the training.

### 4. Proposed Method

In this section, we introduce our proposed method DYNAFED. The overall framework is illustrated in Figure 1. Firstly, we collect a trajectory of the global model’s updates in the early phase of federated training, with which we construct a synthetic dataset $D_{syn} = \{X, y\}$ on the server side. Then, we utilize $D_{syn}$ to aid the server-side aggregation, which effectively helps recover the performance drop caused by deflected local models.

#### 4.1. Acquiring Global Knowledge by Data Synthesis

Our goal is to construct a pseudo dataset $D_{syn}$, which achieves a similar effect during training as the global dataset $D$. In other words, the trajectory of a network trained with $D_{syn}$ should have similar dynamics with the trajectory trained with $D$. To be precise, we denote the trajectory trained on $D$ as a sequence $\{w^t\}_{t=0}^{L}$, where $L$ is the length of the trajectory. In order to reduce the number of unrolling steps during optimization, we align the trajectory in a segment-by-segment manner. Without loss of generality, we consider a segment from $w^t$ to $w^{t+s}$, then the problem becomes the following: starting from $w^t$, the network should arrive at a place close to $w^{t+s}$ after being trained for $s$ steps on $D_{syn}$. We further formulate the data synthesis task into a learning problem as follows:

$$
\min_{\tilde{w}, s} \mathbb{E}_t \sum_{L-s} [d(\tilde{w}, w^{t+s})]
$$

subject to

$$
\tilde{w} = \mathcal{A}(X, y, w^t, s)
$$

In the inner loop expressed by Eqn. 4, we run the trainer $\mathcal{A}()$ that trains a neural network initialized from $w^t$ on the synthetic
dataset $D_{\text{syn}} = \{X, y\}$ for $s$ steps, which arrives at $\hat{w}$. In the outer loop, we minimize the distance between $w^{\text{syn}}$ and $\hat{w}$, denoted as $d(\hat{w}, w^{\text{syn}})$, by optimizing over $(X, y)$. The expectation of the uniform distribution $U$ is adopted to take into account all the segments along the trajectory. $d(\cdot, \cdot)$ is a general distance measure, which can take the form of euclidean distance or cosine distance, etc. During the optimization, we treat both $(X, y)$ to be learnable variables, where $X$ is initialized with random noise, and $y$ is initialized with equal probability over all labels. The detailed algorithm is shown in Algorithm 1.

Algorithm 1 DataSyn

**Require:** Global trajectory $\{w^s\}_1^T$, learning rate $\eta$, training rounds $N$, inner steps $s'$.

1. Randomly initialize $X_0$ and pair them with $y$ to form the synthetic dataset.
2. for training iteration $n = 1, 2 \ldots N$ do
3. Sample $t \sim U(1, L - s)$, then take $w^t$ and $w^{t+s}$ from the trajectory.
4. Get the trained parameters $\hat{w} = A(X_n, y_n, w^t, s')$
5. Calculate the distance $d(\hat{w}, w^{t+s})$ and obtain gradient w.r.t $X_n$ and $y_n$ as $\nabla X_n d(\hat{w}, w^{t+s})$ and $\nabla y_n d(\hat{w}, w^{t+s})$.
6. Update $X_n$ and $y_n$ by gradient descent $X_{n+1} = X_n - \eta \nabla X_n d(\hat{w}, w^{t+s}), y_{n+1} = y_n - \eta \nabla y_n d(\hat{w}, w^{t+s})$
7. end for

**Output:** Optimized synthetic data $D_{\text{syn}}$.

Note that since the size of $D_{\text{syn}}$ is much smaller than $D$, the effective step size should have a different scale. Therefore, in Equation 4, we may increase the number of steps from $s$ to $s'$ to account for this scale mismatch.

In this way, the knowledge hidden in the dynamics of the global model can be transferred to $D_{\text{syn}}$, which then acts as an approximation of the global data distribution to aid the server-side aggregation.

### 4.2. Overall Algorithm of DynaFed

In this section, we present our **DynaFed** that integrates the data synthesis process into the federated learning framework.

Firstly, to construct the global trajectory on the server, we collect the global model’s checkpoints from the first few communication rounds of FedAvg mainly considering the following two factors: 1) collecting a long trajectory induces prohibitive cost due to the expensive global communication, 2) the change in global model’s parameters becomes insignificant during late rounds, which makes it difficult to extract knowledge from the dynamics.

Note that although $D_{\text{syn}}$ contains rich global knowledge, it is not sufficient to replace the global data distribution $D$ due to the following reasons: 1) the objective in Eqn.3 can not be ideally solved due to the two-level optimization procedure; 2) to make the scale of the optimization problem acceptable, the size of $D_{\text{syn}}$ can not be as large as $D$; 3) there exists an inconsistency between the trainer $A(\cdot)$ and the one that produces the global trajectory, i.e., FedAvg. Therefore, instead of simply conduct regular training with $D_{\text{syn}}$, we leverage such $D_{\text{syn}}$ to help aggregation by finetuning the global model on the server side.

The rundown of the entire algorithm is presented in Algorithm 2. Firstly, the global model’s trajectory in the earliest $L$ rounds is collected and stored on the server. Then, the server executes the data synthesis procedure to generate the pseudo data $D_{\text{syn}}$. In all subsequent rounds, $D_{\text{syn}}$ is leveraged on the server to help reduce the negative impact of deflected client models by finetuning the global model.

Our method enjoys the following appealing properties:

- In contrast to data-dependent KD methods [6, 33], **DynaFed** extracts the knowledge of the global data distribution from the model trajectory, which does not depend on any external datasets;
- Compared with data-free KD methods [50, 53], **DynaFed** is able to synthesize informative pseudo data in the early rounds of federated learning without requiring the global model to be well trained, which significantly helps convergence and stabilizes training;
- The data synthesis only needs to be conducted once. Besides, since only a few samples are synthesized, refining the global model on the server requires negligible time.

**Remark 1.** We emphasize that our **DynaFed** does not raise privacy concerns given following reasons: 1) we rely on only the trajectory of the global model’s parameters, which is available in the conventional setting of all FedAvg-based methods; 2) we keep $D_{\text{syn}}$ at the server rather than sending it to the clients; 3) we extract global knowledge using the global dynamics, rather than any client-specific information as in [12, 17, 18, 20, 47], the derived $D_{\text{syn}}$ comprises of information mixed with the entire global data distribution, thus prevents leakage of client privacy; 4) our **DynaFed** shares a similar flavor as dataset condensation (DC) methods, which aims to generate informative pseudo data containing global knowledge, rather than real-looking data. The privacy-preserving ability of DC was also discussed in previous work [11].
Further, we assume

where $J$ denotes the time index for aggregation. The detailed steps of performing local training, while when $I \notin \mathcal{J}$, we conduct finetuning. In this case, the following lemma about the continuous dependence of the NTK theory [19]. Note that the right-hand sides of the two network functions trained on $D$ using $f$ are two continuous functions in a region $x \in \mathcal{D}_f$, and $D_f$ is a strongly convex loss function. During training.

Lemma 1 (Continuous Dependence [42]). Suppose $\tilde{F}(t, f)$ and $F(t, f)$ are two continuous functions in a region $G$ satisfying

$$|\tilde{F}(t, f) - F(t, f)| \leq \epsilon, \forall (t, f) \in G.$$ Further, we assume $F(t, f)$ satisfy the $L_F$-Lipschitz condition w.r.t., $f$. Let $\tilde{f}(t)$ and $f(t)$ be the solutions of initial problems,

$$\frac{df}{dt} = \nabla_L f(x, w^t) \nabla L(w^t, D_{syn}),$$

$$\frac{d\tilde{f}}{dt} = \tilde{F}(t, \tilde{f}) \quad \text{and} \quad \frac{df}{dt} = F(t, f),$$

with $\tilde{f}(t_0) = f_0$ and $f(t_0) = f_0$. Then, in a common region $[t_0, t] \leq \alpha$, we have the following estimation:

$$|\tilde{f}(t) - f(t)| \leq \frac{\epsilon}{L_F} \left( \alpha L_F - 1 \right).$$

To analyze the convergence of DYNAPES, we need to define some additional notations and rewrite our method as follows. Suppose with the $D_{syn}$ generated from the data synthesis process, in DYNAPES we run SGD for $\tau_1$ and $\tau_2$ iterations in each local training round and finetuning process, respectively. Let the sets $\mathcal{I}$ and $\mathcal{J}$ be

$$\mathcal{I} = \{ t | t = k(\tau_1 + \tau_2) + c, k = 0, 1, 2, \ldots, c \in [\tau_1] \},$$

$$\mathcal{J} = \{ t | t = k(\tau_1 + \tau_2) + \tau_1, k = 0, 1, 2, \ldots \},$$

where $[\tau_1] = \{ 0, 1, \ldots, \tau_1 - 1 \}$. Therefore, when $t \in \mathcal{I}$, we perform local training, while when $t \notin \mathcal{I}$, we conduct finetuning. $\mathcal{J}$ denotes the time index for aggregation. The detailed steps of DYNAPES can be rewritten as

$$w_{m+1}^{t+1} = \begin{cases} w_m^{t+1} - \eta \nabla L(w_m^{t+1}, \xi_m^t), & \text{if } t \in \mathcal{I} \\ w_m^{t+1} - \eta \nabla L(w_m^{t+1}, D_{syn}), & \text{if } t \notin \mathcal{I} \end{cases},$$

$$w_m^{t+1} = \begin{cases} w_m^{t+1} - \eta \nabla L(w_m^{t+1}, \xi_m^t), & \text{if } t \in \mathcal{I} \\ w_m^{t+1} - \eta \nabla L(w_m^{t+1}, D_{syn}), & \text{if } t \notin \mathcal{I} \end{cases},$$

where $\xi_m^t \sim D_m$. Based on the notations, we define a sequence:

$$w_t^\tau = \sum_{m=1}^M \alpha_m w_m^t.$$

Hence, our algorithm is an integration of FedAvg and a biased GD. For $w^\tau$, note that $w^\tau = w_1^\tau = \cdots = w_M^\tau$ in the finetuning process and we have the following convergence results:

Theorem 1 (Convergence). For $\tilde{L}$-smooth, $\mu$-strongly convex loss functions $\tilde{L}(\cdot, \cdot)$. We assume $\|\nabla L(w, D_{syn}) - \nabla L(w, D)\| \leq \delta \|\nabla L(w, D)\| + \epsilon$ holds with two small non-negative scalars $\delta$ and $\epsilon$. Let $\eta_t = \frac{\eta}{t}$ for a proper constant $\eta$. Then, DYNAPES satisfies

$$\mathbb{E}L(w^\tau, D) - L(w^*, D) \leq \frac{C}{T},$$

where $w^*$ is the minimum of $L(w, D)$ and $C$ is a constant, whose detailed formula is given in the appendix.

More detailed discussions about the convergence result are given in the Appendix.

6. Experiments

Benchmark Datasets and Experimental Settings. We conduct experiments over four commonly used datasets: Fashion-MNIST [43], CIFAR10 [25], CINIC10 [9] and CIFAR100 [25]. Among them, FashionMNIST is a dataset containing grayscale images of fashion products. CIFAR10 is an image classification dataset containing daily objects. CINIC10 is a dataset combining CIFAR10 and samples from similar classes that are downloaded from ImageNet [26]. These three datasets contain 10 classes. CIFAR100 contains the same data as CIFAR10, but categorizes the data into 100 classes. For each dataset, we mainly conduct experiments with heterogeneous client data distribution. We follow prior work [6, 33] to use Dirichlet distribution for simulating the non-IID data distribution, where the degree of heterogeneity is defined by $\alpha$, smaller $\alpha$ value corresponds to more severe heterogeneity.

Baseline Methods. We consider various state-of-the-art solutions against non-IID data distribution in the context of federated learning. Specifically, we compare with the following approaches 1) the vanilla aggregation strategy FedAVG [34]; 2) regularization-based strategies FedProx [31], Scaffold [24]; 3) data-dependent knowledge distillation strategies that need external dataset FedDF [33] and FedBE [6], ABAVG [44]; (4) data sharing [49] or data-free knowledge distillation [53] methods. Note that we do not compare with [50] since the code is not published. Please refer to Appendix for detailed settings of the baseline methods.

Configurations. Unless specified otherwise, we follow [7, 7, 15] and adopt the following default configurations throughout the experiments: we run 200 global communication rounds with local epoch set to 1. There are 80 clients in total, and the participation ratio in each round is set to 40%. Experiments using other participation ratios are in the Appendix. We report the global model’s average performance in the last five rounds evaluated using the test split of the datasets. For the construction of global trajectory, we first run
### Table 1. Comparison of test performances achieved by different FL methods with different degrees of data heterogeneity $\alpha$ across multiple datasets.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\alpha = 0.01$</th>
<th>$\alpha = 0.04$</th>
<th>$\alpha = 0.08$</th>
<th>$\alpha = 0.16$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FMNIST</td>
<td>CIFAR10</td>
<td>CINIC10</td>
<td>FMNIST</td>
</tr>
<tr>
<td>FedAVG</td>
<td>74.50 ± 1.32</td>
<td>39.30 ± 3.42</td>
<td>31.60 ± 5.50</td>
<td>81.74 ± 1.98</td>
</tr>
<tr>
<td>FedProx</td>
<td>76.84 ± 1.83</td>
<td>42.13 ± 3.64</td>
<td>32.56 ± 4.59</td>
<td>83.06 ± 2.53</td>
</tr>
<tr>
<td>Scaffold</td>
<td>77.95 ± 0.87</td>
<td>42.04 ± 2.26</td>
<td>34.90 ± 3.34</td>
<td>82.25 ± 1.35</td>
</tr>
<tr>
<td>FedDF$_{\text{F}}$</td>
<td>72.36 ± 2.08</td>
<td>39.73 ± 3.98</td>
<td>31.97 ± 4.31</td>
<td>81.65 ± 0.97</td>
</tr>
<tr>
<td>FedBE$_{\text{F}}$</td>
<td>72.33 ± 1.79</td>
<td>38.36 ± 3.74</td>
<td>32.04 ± 3.73</td>
<td>81.31 ± 1.25</td>
</tr>
<tr>
<td>ABAVG$_{\text{F}}$</td>
<td>75.98 ± 1.99</td>
<td>39.95 ± 1.37</td>
<td>32.75 ± 4.18</td>
<td>84.88 ± 1.84</td>
</tr>
<tr>
<td>FedGen$^\dagger$</td>
<td>75.59 ± 1.12</td>
<td>40.19 ± 2.14</td>
<td>32.59 ± 3.25</td>
<td>81.46 ± 1.08</td>
</tr>
<tr>
<td>FedMix$^\dagger$</td>
<td>81.34 ± 0.68</td>
<td>50.48 ± 1.23</td>
<td>37.15 ± 1.81</td>
<td>84.23 ± 0.50</td>
</tr>
<tr>
<td>DynaFed$^\dagger$</td>
<td>87.52 ± 0.15</td>
<td>65.53 ± 0.34</td>
<td>48.04 ± 0.70</td>
<td>89.45 ± 0.11</td>
</tr>
</tbody>
</table>

$^\dagger$ Methods are based on data sharing or generation. We observe that our approach outperforms other methods by a large margin, and its advantage is more prominent on more challenging datasets with higher heterogeneity. Specifically, DynaFed demonstrates relative improvement over the FedAvg baseline by 17.5%, 64.5%, 52.0%, and 82.2% on FMNIST, CIFAR10, CINIC10, and CIFAR100, respectively.

### 6.1 Main Experiments with Data Heterogeneity

We demonstrate the superior performance of our DynaFed by conducting experiments on heterogeneous client data across comprehensive datasets and various heterogeneity values $\alpha$. Specifically, we use three datasets with 10 classes (shown in Table 1): FashionMNIST [43], CIFAR10 [25] and CINIC10 [9], heterogeneity degree $\alpha$ set to 0.01, 0.04 and 0.16; and CIFAR100 containing 100 classes, with $\alpha$ values 0.01, 0.04, 0.08 and 0.16 (shown in Table 2). DynaFed significantly boosts the convergence, stabilizes training, and brings considerable performance improvement compared with previous approaches. Specifically, with heterogeneity value $\alpha = 0.01$, DynaFed demonstrates relative improvement over the FedAvg baseline by 17.5%, 64.5%, 52.0%, and 82.2% on FMNIST, CIFAR10, CINIC10, and CIFAR100, respectively.

As demonstrated in Figure 2, the performance of DynaFed is rapidly boosted as soon as the synthesized data starts refining the global model on the server. This verifies that DynaFed does not depend on the global model’s performance in data synthesis,
We conduct experiments with CIFAR10 and set $\alpha = 0.01$. By leveraging the dynamics of the global model’s trajectory in the first few rounds, e.g., $L \in \{10, 15, 20, 15\}$, the derived $D_{\text{syn}}$ already helps achieve faster convergence and stable training. In contrast, the FedGen approach only brings slight performance gain during the later phase of training due to the dependence on the global model’s performance. Right: We show the performance of DYNAFED with different sizes of $D_{\text{syn}}$ for various $\alpha$. The performance gain is significant with just 150 synthesized samples.

Figure 5. Left: The test accuracy curves for different choices of trajectory length L on the CIFAR10 dataset with $\alpha = 0.01$. By leveraging the dynamics of the global model’s trajectory in the first few rounds, e.g., $L \in \{10, 15, 20, 15\}$, the derived $D_{\text{syn}}$ already helps achieve faster convergence and stable training. In contrast, the FedGen approach only brings slight performance gain during the later phase of training due to the dependence on the global model’s performance. Right: We show the performance of DYNAFED with different sizes of $D_{\text{syn}}$ for various $\alpha$. The performance gain is significant with just 150 synthesized samples.

**$D_{\text{syn}}$ Can be Learned with Early Trajectory.** We conduct experiments with different choices of trajectory length $L$ in left of Figure 5. We observe that even if with $L = 10$, the result is comparable with performance obtained with longer trajectory $L = 25$. Compared with the baseline methods, DYNAFED achieves significant convergence speedup and performance boost. These results support our claim in Section 4 that our method can take effect early during training. By contrast, the data-free KD method FedGen [53] that trains a generator to produce pseudo data starts to show a slight improvement only in the late stage of training due to the dependence on the global model’s performance when training the generator.

**How the Size of $D_{\text{syn}}$ Impacts the Performance.** As shown in the right of Figure 5, we conduct experiments with different sizes of $D_{\text{syn}}$ and various heterogeneity degrees $\alpha$. We find that a small $D_{\text{syn}}$ suffices for good performance, while larger $D_{\text{syn}}$ brings only marginal performance boost. This property not only saves the cost for synthesizing $D_{\text{syn}}$, but also makes the finetuning of the global model more efficient.

**$D_{\text{syn}}$ is Able to Mimic the Global Dynamics.** In the left of Figure 6, we calculate the cosine distance between the target checkpoint $w_{t+s}$' and the parameters $w_t$ trained from $w_{s'}$ for $s'$ steps with $D_{\text{syn}}$, a randomly sampled real dataset of the same size as $D_{\text{syn}}$, and a dataset consisted of noisy pixels, respectively. We can see that in terms of mimicking the global trajectory, $D_{\text{syn}}$ not only significantly outperforms the noise dataset, but also achieves only half of the distance obtained with real dataset, which is not
with local training epochs of 5 and 10, respectively. As can be seen, therefore, $D$ consistently outperforms other methods by a large margin even.

### Table 3. Performance on CIFAR10 achieved by different FL algorithms under various degrees of data heterogeneity and local training epochs. Total communication rounds of 100 and 50 are set with local training epochs of 5 and 10, respectively. As can be seen, DyNaFed significantly surpasses other methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>CIFAR10</th>
<th>CINIC10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha = 0.01$</td>
<td>$\alpha = 0.04$</td>
</tr>
<tr>
<td></td>
<td>Acc $\pm 0.55$</td>
<td>Acc $\pm 0.45$</td>
</tr>
<tr>
<td>FedAVG</td>
<td>132.0 $\pm$ 15.0</td>
<td>117.0 $\pm$ 8.0</td>
</tr>
<tr>
<td>FedProx</td>
<td>113.3 $\pm$ 16.4</td>
<td>102.0 $\pm$ 4.7</td>
</tr>
<tr>
<td>Scaffold</td>
<td>105.0 $\pm$ 10.4</td>
<td>100.3 $\pm$ 3.5</td>
</tr>
<tr>
<td>FedDF*</td>
<td>145.7 $\pm$ 13.1</td>
<td>117.3 $\pm$ 5.8</td>
</tr>
<tr>
<td>FedBE*</td>
<td>165.0 $\pm$ 12.7</td>
<td>122.7 $\pm$ 4.5</td>
</tr>
<tr>
<td>ABAVG†</td>
<td>109.7 $\pm$ 5.4</td>
<td>110.7 $\pm$ 5.0</td>
</tr>
<tr>
<td>FedGen†</td>
<td>115.7 $\pm$ 10.4</td>
<td>110.3 $\pm$ 5.7</td>
</tr>
<tr>
<td>FedMix†</td>
<td>77.3 $\pm$ 3.7</td>
<td>89.3 $\pm$ 3.5</td>
</tr>
<tr>
<td>DyNaFed†</td>
<td>22.3 $\pm$ 0.6</td>
<td>22.0 $\pm$ 1.0</td>
</tr>
</tbody>
</table>

DyNaFed is Robust to Longer Local Training. Longer local training is generally required in FL to reduce the total number of global communication rounds. Under different heterogeneity degrees, we conduct experiments to evaluate the impact of longer local training epochs on DyNaFed. Specifically, we conduct experiments with total communication rounds of 100 and 50 with local training epochs of 5 and 10, respectively. The results are presented in Table 3 from which we observe the following: 1) DyNaFed consistently outperforms other methods by a large margin even with longer local training; 2) the performance of DyNaFed is less sensitive to the length of local training, which benefits from the $D_{syn}$ containing information about the global data distribution. Therefore, DyNaFed is able to achieve similar performance with less global communication rounds, which is the major bottleneck in the efficiency of FL.

We further conduct experiments on varying local epochs to measure the quality of $D_{syn}$. Specifically, we use it to train a network from scratch and evaluate its test performance. Shown in right of Figure 6, the quality of $D_{syn}$ stays similar with longer local training and more severe heterogeneity. This further explains the superior performance of DyNaFed with longer local training.

### 6.3. Architecture Generalization and Efficiency

To showcase the generalization ability of our approach over different network architecture choices, we conduct experiments on.

Table 5. Performance comparison across different network architectures. We conduct the experiment on FMNIST dataset using MLP and ConvNet to demonstrate the generalization of DyNaFed for different network architectures.

<table>
<thead>
<tr>
<th>Method</th>
<th>MLP</th>
<th>ConvNet</th>
<th>MLP</th>
<th>ConvNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha = 0.01$</td>
<td>$\alpha = 0.04$</td>
<td>$\alpha = 0.01$</td>
<td>$\alpha = 0.04$</td>
</tr>
<tr>
<td>FedAVG</td>
<td>65.64 $\pm$ 1.69</td>
<td>74.51 $\pm$ 1.32</td>
<td>73.26 $\pm$ 1.49</td>
<td>81.74 $\pm$ 1.98</td>
</tr>
<tr>
<td>FedProx</td>
<td>68.09 $\pm$ 1.47</td>
<td>76.88 $\pm$ 1.83</td>
<td>79.83 $\pm$ 1.70</td>
<td>83.06 $\pm$ 2.53</td>
</tr>
<tr>
<td>Scaffold</td>
<td>67.60 $\pm$ 1.53</td>
<td>77.92 $\pm$ 0.87</td>
<td>78.09 $\pm$ 1.35</td>
<td>82.25 $\pm$ 1.35</td>
</tr>
<tr>
<td>FedDF*</td>
<td>64.59 $\pm$ 1.70</td>
<td>72.36 $\pm$ 2.08</td>
<td>77.20 $\pm$ 1.58</td>
<td>81.65 $\pm$ 0.97</td>
</tr>
<tr>
<td>FedBE*</td>
<td>65.97 $\pm$ 1.64</td>
<td>72.33 $\pm$ 1.79</td>
<td>75.42 $\pm$ 1.35</td>
<td>81.31 $\pm$ 1.25</td>
</tr>
<tr>
<td>ABAVG†</td>
<td>69.19 $\pm$ 1.50</td>
<td>75.98 $\pm$ 1.99</td>
<td>81.64 $\pm$ 1.20</td>
<td>84.44 $\pm$ 1.84</td>
</tr>
<tr>
<td>FedGen†</td>
<td>68.67 $\pm$ 1.45</td>
<td>75.59 $\pm$ 1.12</td>
<td>77.94 $\pm$ 1.38</td>
<td>56.60 $\pm$ 1.08</td>
</tr>
<tr>
<td>FedMix†</td>
<td>70.30 $\pm$ 0.92</td>
<td>81.34 $\pm$ 0.68</td>
<td>81.95 $\pm$ 0.64</td>
<td>84.23 $\pm$ 0.50</td>
</tr>
<tr>
<td>DyNaFed†</td>
<td>73.89 $\pm$ 0.24</td>
<td>87.52 $\pm$ 0.15</td>
<td>83.54 $\pm$ 0.42</td>
<td>89.45 $\pm$ 0.11</td>
</tr>
</tbody>
</table>

Figure 6. Left: the cosine distance between the target checkpoint $w_t^{*}$ and the parameters $\hat{w}$ obtained by training with different datasets from $w_s$ for $s$ steps. We can see that the distance derived using $D_{syn}$ is constantly smaller compared with other data. Right: Test performances of a model trained from scratch using only $D_{syn}$ generated under various local epochs and client heterogeneity. The two plots together verify the quality of the generated $D_{syn}$. FMNIST using different network architectures. Specifically, we choose ConvNet and MLP following [34, 53]. As shown in Table 5, under various client data heterogeneity, our method demonstrates superior performances with both network architectures.

Thanks to the rich knowledge of global data distribution contained in $D_{syn}$, using it to refine the global model greatly boosts the convergence speed of training. As shown in Figure 2, as soon as $D_{syn}$ is used to refine the global model, its performance rapidly increases to a reasonable accuracy, which reduces many rounds of communication. In Table 4, we also quantitively compare the convergence speed of different FL algorithms by showing the number of communication rounds needed to reach the highest test accuracy achievable by the baselines. As can be observed, our DyNaFed requires only less than 20% communication rounds to reach a target accuracy comparable to other methods.

### 7. Conclusion

We propose DyNaFed to tackle the data heterogeneity issue, which synthesizes a pseudo dataset to extract the knowledge of the global data distribution from the dynamics of the global model’s trajectory. Extensive experiments show that DyNaFed demonstrates relative improvement over the FedAvg baseline up to 82.2% on CIFAR100. Further, we believe our work is able to provide insights for extracting global information on the server side, which goes beyond tackling the data heterogeneity issue.
References


