DPFL: Decentralized Personalized Federated Learning

Salma Kharrat
KAUST

Marco Canini
KAUST

Samuel Horvath
MBZUAI

Abstract

This work addresses the challenges of data heterogeneity and communication constraints in decentralized federated learning (FL). We introduce decentralized personalized FL (DPFL), a bi-level optimization framework that enhances personalized FL by leveraging combinatorial relationships among clients, enabling fine-grained and targeted collaborations. By employing a constrained greedy algorithm, DPFL constructs a collaboration graph that guides clients in choosing suitable collaborators, enabling personalized model training tailored to local data while respecting a fixed and predefined communication and resource budget. Our theoretical analysis demonstrates that the proposed objective for constructing the collaboration graph yields superior or equivalent performance compared to any alternative collaboration structures, including pure local training. Extensive experiments across diverse datasets show that DPFL consistently outperforms existing methods, effectively handling non-IID data, reducing communication overhead, and improving resource efficiency in real-world decentralized FL scenarios. The code can be accessed at: https://github.com/salmakh1/DPFL.

1 INTRODUCTION

The ongoing unprecedented growth in data captured and stored on edge devices has led to a flurry of research proposing collaborative learning paradigms that do not necessitate transferring data to a centralized location, thereby enhancing data privacy. One widely adopted approach is federated learning (FL) (McMahan et al., 2017), where clients collaboratively train a global model by sharing only model parameters instead of raw data. In FL, training commonly occurs over several rounds, alternating between local updates on client devices and global aggregation of these updates at a centralized server.

Before the advent of FL, decentralized learning (Lian et al., 2017, 2018; Jiang et al., 2017; Tang et al., 2018; Vanhaesebrouck et al., 2017; Bellet et al., 2018) was developed to address the limitations of centralized systems, including the risk of a single point of failure (Beltrán et al., 2023). These decentralized methods enhance privacy (Cyffers and Bellet, 2022), improve communication efficiency (Lian et al., 2017), and increase system resilience (Neglia et al., 2019). Consequently, they have led to the emergence of decentralized FL approaches, enabling collaborative learning across client devices without a centralized server, and while keeping training data private (Vanhaesebrouck et al., 2017; Koppel et al., 2017; Even et al., 2022).

A key challenge in both centralized and decentralized FL is the presence of statistical heterogeneity, whereby models are trained with clients' data that are not independent and identically distributed (non-IID) (Kairouz et al., 2021). Addressing this issue, personalized FL techniques (e.g., Deng et al. (2020); Li et al. (2021); T Dinh et al. (2020); Hanzely and Richtárik (2020); Collins et al. (2021)) have emerged particularly for centralized settings, focusing on tailoring models to the unique data distributions of individual clients rather than relying on a global model.

However, personalized decentralized FL remains a less-studied problem due to its intrinsic dependence on a decentralized collaboration graph, which defines participant interactions. In non-IID settings, random collaborations can lead to suboptimal learning outcomes, as unsuitable collaborators introduce noise into the learning process. This presents a key challenge: constructing an effective collaboration graph to identify suitable collaborators, as its quality significantly impacts model performance.

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

Moreover, when building the collaboration graph, practical approaches must consider data heterogeneity alongside resource constraints, such as network bandwidth, memory, and computational capacity. Previous works, like Ye et al. (2023); Even et al. (2022), often overlook these factors, focusing primarily on data distribution while neglecting resource limitations. Consequently, we encounter a more challenging problem that requires the mitigation of data heterogeneity while adhering to application-imposed resource limitations.

To address these challenges, we introduce DPFL, a novel bi-level optimization framework that simultaneously optimizes clients' models and the collaboration graph while considering communication and resource constraints. We promote targeted collaboration, where clients collaborate only with others that provide positive returns, quantified through a reward function. Our proposed algorithm leverages the combinatorial effect of clients' collaborations while restricting the number of collaborators and models each client handles, respecting both communication and storage requirements. In our formulation, each client's aggregated model is determined by the inward edges of the collaboration graph, connecting it to its selected collaborators. Crucially, our method infers beneficial collaborators without requiring prior knowledge of clients' data distributions. Unlike prior methods (e.g., Ye et al. (2023); Li et al. (2022a,b)), our approach allows for a directed collaboration graph, where collaborations are not necessarily reciprocal—client A may benefit from collaborating with client B, but not vice versa. This directed nature introduces greater flexibility (Zhang and You, 2019; Nedić et al., 2018) and robustness (Tsianos et al., 2012; Lian et al., 2018), enabling more efficient collaboration strategies.

Contributions.

- We introduce a novel decentralized personalized FL problem, formulated as a constrained discrete combinatorial objective within a bi-level optimization framework that integrates graph generation and model personalization under resource and communication constraints.
- We propose the DPFL algorithm, which alternates between personalized model learning and graph optimization, accounting for combinatorial effects beyond pairwise selection and enabling asymmetric graph solutions for improved collaborator selection and resource efficiency.
- We provide a formal analysis of the bi-level optimization problem, including its combinatorial aspects, and the complexities of the employed approximations. Extensive experiments on diverse datasets compare DPFL against 12 benchmarks, demonstrating its superior performance.

2 METHODOLOGY

We introduce a novel formulation for the optimization problem in personalized FL. First, we provide the underlying intuition behind the problem and present the comprehensive problem formulation in § 2.1. Next, we propose decomposing the problem and employing an alternating minimization approach § 2.2.

2.1 Optimization problem

Generalization in FL aims to fit a single global model with parameters $\mathbf{w}$ to the non-IID data held by individual clients (McMahan et al., 2017). This optimization problem can be represented as follows:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ F^{\mathcal{D}}(\mathbf{w}) \triangleq \sum_{k=1}^{N} p_k F_k(\mathbf{w}) \right\}, \tag{1}$$

where N is the number of devices, $\mathcal{D}$ is a global distribution, $p_k \geq 0$ is the weight of the k-th device s.t. $\sum_{k=1}^{N} p_k = 1$. It is common to select p_k proportional to device k's dataset size. Suppose the k-th device holds data drawn from the distribution $\mathcal{D}_k$. The local objective $F_k(\cdot)$ is defined as: $F_k(\mathbf{w}) \triangleq \mathbb{E}_{x_k\sim\mathcal{D}_k}\ell(\mathbf{w}; x_k)$ where $\ell(\cdot;\cdot)$ is some loss function.

In personalized FL, the objective shifts from seeking a single global model to the pursuit of local models tailored to each client. These local models are designed to perform well on unseen data samples drawn from the same distribution as the client's local training data. Consequently, the original objective function in Eq. (1) becomes the following:

$$\min_{\mathbf{w}=\{\mathbf{w}_i\in\mathbb{R}^d\}_{i=1}^N} \left\{ F(\mathbf{w}) \triangleq \sum_{k=1}^{N} p_k F_k(\mathbf{w}_k) \right\}. \tag{2}$$

Considering significant heterogeneity in the data distributions among the clients, an intuitive solution to the problem outlined in Eq. (2) is that each client independently trains its local model using its local data, obviating the necessity for collaboration with other clients. However, this holds only if each client has access to an infinite number of IID samples from its local distribution $\mathcal{D}_k$ or the local distributions are significantly different.

We focus on collaborative learning in real-world settings, where data access and heterogeneity are major challenges. Clients in such scenarios operate with a finite number of local samples. While we advocate for inter-client collaboration, and while collaboration is encouraged, it must enhance rather than compromise personalized model performance. Collaboration needs to be designed to enhance personalized

model performance within this complex data landscape while respecting the imposed communication and resource constraints, particularly in decentralized settings where coupled all-to-all communication is prohibitive. Therefore, accounting for the application-imposed resource budget becomes crucial.

We reformulate our objective to address these challenges and optimize model performance in this dynamic environment. Thus, we propose the following problem formulation:

$$\min_{\mathbf{w},\mathcal{C}} \left\{ F(\mathbf{w},\mathcal{C}) \triangleq \sum_{k=1}^{N} p_k F_k\left(\mathbf{w}_k,\mathcal{C}_k\right) \right\}, \tag{3}$$

where $\mathcal{C} = \{\mathcal{C}_i \in 2^{[N]\setminus\{i\}}; |\mathcal{C}_i| \leq B_c\}_{i=1}^{N}$ and $\mathbf{w} = \{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^{N}$. In this optimization problem (Eq. (3)), our objective is to minimize the function $F(\mathbf{w},\mathcal{C})$, where $\mathbf{w} = \{\mathbf{w}_1,\ldots,\mathbf{w}_N\}$ represents a set of parameters and $\mathcal{C} = \{\mathcal{C}_1,\ldots,\mathcal{C}_N\}$ denotes collaboration assignments. $\mathcal{C}_i \cup \{i\}$ contains j, if client i receives updates from client j. In our algorithm design, we enforce $\forall k \in [N] : |\mathcal{C}_i| \leq B_c$, where B_c represents the resource budget imposed by communication and resource constraints (hence, it is not a hyperparameter). The function $F_k\left(\mathbf{w}_k,\mathcal{C}_k\right)$ is defined as the average loss over a dataset $\mathcal{D}_k$ for client k, expressed as:

$$F_k(\mathbf{w}_k,\mathcal{C}_k) \triangleq \mathbb{E}_{x_k\sim\mathcal{D}_k}\ell\left(\mathbf{w}_k,\mathcal{C}_k;x_k\right).$$

Here, $\ell\left(\mathbf{w}_k,\mathcal{C}_k,x_k\right)$ represents the loss function applied to the parameters $\mathbf{w}_k$, collaboration assignments $\mathcal{C}_k$, and data point x_k. We can rewrite it as follows: $\ell\left(\mathbf{w}_k,\mathcal{C}_k;x_k\right) = \ell(\hat{\mathbf{w}}_k;x_k)$ where the crucial element in this formulation is $\hat{\mathbf{w}}_k$, computed as the average of individual parameters $\mathbf{w}_i$ within the collaboration set $\mathcal{C}_k$ for client k:

$$\hat{\mathbf{w}}_k \triangleq \frac{1}{\sum_{i\in\tilde{\mathcal{C}}_k} p_i} \sum_{i\in\tilde{\mathcal{C}}_k} p_i\mathbf{w}_i; \text{ where } \tilde{\mathcal{C}}_k \triangleq \mathcal{C}_k \cup \{k\}. \tag{4}$$

2.2 Alternating minimization

Solving Eq. (3) involves simultaneously optimizing for both $\mathbf{w}$ and $\mathcal{C}$. To simplify this complex problem, we propose an alternating minimization approach. In the first sub-problem, we use a given collaboration graph $\mathcal{C}^\star$ to update local parameters $\mathbf{w}_k$, while in the second sub-problem, we focus on identifying the best set of collaborators $\mathcal{C}_k$ given $\mathbf{w}$.

The first sub-problem can be written as:

$$\min_{\mathbf{w}=\{\mathbf{w}_i\in\mathbb{R}^d\}_{i=1}^{N}} \left\{ F(\mathbf{w},\mathcal{C}^\star) \triangleq \sum_{k=1}^{N} p_k F_k\left(\mathbf{w}_k,\mathcal{C}_k^\star\right) \right\}. \tag{5}$$

We propose to solve Eq. (5) in a decentralized manner, where each client first performs τ local updates of their local parameters $\mathbf{w}_k$ based on their local data. This is followed by the aggregation step, in which each client updates their local solution $\mathbf{w}_k$ as defined in Eq. (4).

For finding the collaboration graph, given fixed $\mathbf{w}^\star$, the collaboration assignment $\mathcal{C}_k$ for each client $k \in [N]$ is found as a solution to the following problem:

$$\min_{\mathcal{C}_k\in\Omega_k} F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{C}_k\cup\{k\}} p_i} \sum_{i\in\mathcal{C}_k\cup\{k\}} p_i\mathbf{w}_i^\star \right), \tag{6}$$

where $F_k^{\mathcal{V}}(\cdot)$ represents the validation loss of client k and $\Omega_k \subseteq 2^{[N]\setminus\{k\}}$ such that $|\Omega_k| \leq B_c$ is a set of clients that client k can potentially collaborate with other than itself. Based on this framework, we are now ready to describe our approach in detail.

3 PROPOSED METHOD

We introduce DPFL, which incorporates the identification of beneficial collaborators alongside FL training. To achieve this, we define a combinatorial objective function on the discrete space of client combinations in Eq. (6) and introduce constrained greedy algorithms denoted as GGC (§ 3.2) and BGGC (§ 3.3). These algorithms approximate a solution for the combinatorial objectives by optimizing client combinations to enhance decentralized personalized FL under strict communication and resource constraints.

3.1 Decentralized Personalized FL (DPFL)

We present DPFL, a decentralized personalized FL framework where each client's primary objective is to identify collaborators that can enhance its model personalization while adhering to the imposed resource budget, denoted as B_c. Algorithm 1 lists the pseudocode for DPFL.

As a preprocessing step (lines 1-5), we establish the initial collaboration graph, Ω_k, ensuring adherence to the imposed budget constraint B_c. This process starts by initializing each local model as $\mathbf{w}$, followed by conducting τ_{init} local training epochs. Subsequently, after each client acquires its initial solution $\mathbf{w}_k^{init}$, it proceeds to determine its collaboration set, Ω_k using the BGGC algorithm(§ 3.3, Algorithm 3 in Appendix C).

With the initial graph in place, the main training loop begins, spanning $T-1$ communication rounds (lines 6-13). During each round, clients perform local training for τ_{train} epochs and download the local models $\mathbf{w}_i$ from all clients in their respective collaboration sets Ω_k where Ω_k contains at most B_c clients (lines 8-9). Upon receiving these models, each client identifies the optimal subset of collaborators for the current round, which minimizes local validation loss using the GGC

Algorithm 1 Decentralized Personalized FL (DPFL)

Require: T, $\mathbf{w}$, N, m, B_c, τ_{init}, τ_{train}, LocalOpt
1: **for** device $k \in [N]$ **in parallel do** {Preprocess}
2: $\quad \mathbf{w}_k^{init} \leftarrow \text{LocalOpt}(\mathbf{w}, \tau_{\text{init}})$
3: $\quad \Omega_k \leftarrow \text{BGGC}(k, [N], m, B_c)$ {Explained in § 3.3}
4: $\quad \mathbf{w}_k \leftarrow \frac{1}{\sum_{i \in \Omega_k \cup k} p_i} \sum_{i \in \Omega_k \cup k} p_i \mathbf{w}_i^{init}$
5: **end for**
6: **for** $t = 1, \cdots, T-1$ **do** {Training Loop}
7: $\quad$ **for** device $k \in [N]$ **in parallel do**
8: $\quad\quad \mathbf{w}_k \leftarrow \text{LocalOpt}(\mathbf{w}_k, \tau_{\text{train}})$
9: $\quad\quad$ send $\mathbf{w}_k$, $\forall\, j$ s.t. $k \in \Omega_j$ and receive $\mathbf{w}_j$, $\forall j \in \Omega_k$,
10: $\quad\quad \mathcal{C}_k \leftarrow \text{GGC}(k, \Omega_k, m, B_c)$ {Explained in § 3.2}
11: $\quad\quad \mathbf{w}_k \leftarrow \frac{1}{\sum_{i \in \mathcal{C}_k \cup k} p_i} \sum_{i \in \mathcal{C}_k \cup k} p_i \mathbf{w}_i$
12: $\quad$ **end for**
13: **end for**

algorithm (line 10) (§ 3.2, Algorithm 2 in Appendix B). The resulting collaborator set $\mathcal{C}_k$ for client k satisfies $|\mathcal{C}_k| \leq |\Omega_k|$ and $\mathcal{C}_k \subseteq \Omega_k$.

Notably, the graph Ω_k, constructed in the preprocessing step via BGGC, remains static throughout the process, while the collaboration set $\mathcal{C}_k$ may change each round based on the GGC evaluation. This dynamic allows clients to skip certain collaborators in certain rounds (e.g., due to noisy updates) while still considering them in future rounds. In our experiments, we utilize local SGD as the optimizer, but DPFL is agnostic and, therefore, compatible with any local optimizer. A simplified example of the framework is presented in Appendix F. Additionally, a discussion of the computational and communication complexities of DPFL is provided in Appendix G.

Remark 1. *For simplicity, our algorithm considers a uniform resource and communication budget B_c for all clients, while in practice, clients may have varying resource capabilities, i.e., B_c^i for client i. We note that our approach can be easily extended to consider personalized budgets for each client in lines 3 and 10.*

3.2 Greedy Graph Construction (GGC)

Our primary goal is to solve Eq. (6). In this combinatorial objective, client k seeks a set $\mathcal{C}_k$ wherein collaboration with its elements results in a positive return. In our context, this positive return is quantified by a reward function representing the negative of the validation loss of each client. This approach diverges from prior works, such as Ye et al. (2023), which typically focused on pairs of clients, neglecting the significance of group synergy. For instance, two clients, A and B, collaborating alone might not be ideal, but adding client C to the collaboration set could significantly alter the outcome. In such a scenario, the collaborative efforts of clients A, B, and C contribute to a decrease in the overall loss experienced by client A. We illustrate this in Appendix A, highlighting how reliance on pairwise comparisons can overlook the benefits of group collaboration in improving client outcomes.

Finding the optimal set $\mathcal{C}_k$ for each client k requires extensive computational exploration. Furthermore, notice that our objective function in Eq. (6) is not monotonic,[1] i.e., adding clients to the set $\mathcal{C}_k$ does not always result in a better reward. To address this, we adapt the non-monotone combinatorial bandit algorithm of Fourati et al. (2023), proposing a greedy graph construction (GGC), detailed in Algorithm 2 in Appendix B, which efficiently selects the set $\mathcal{C}_k$.

The complexity of the employed algorithm is $\mathcal{O}(|\Omega_k|)$, which is at most $\mathcal{O}(B_c)$. The discrete combinatorial problem we address is NP-hard, even with well-behaved (submodular) objectives (Buchbinder et al., 2015). However, making greedy decisions based on adding or removing elements has yielded optimal solutions in linear time (Buchbinder et al., 2015).

While our objective is to minimize the local validation loss of clients as defined in Eq. (6), following the literature on combinatorial maximization, it is more common to maximize a reward function. Thus, we define our reward function as follows:

$$\mathcal{R}(\mathcal{S}) \triangleq -F_k^{\mathcal{V}} \left(\frac{1}{\sum_{i \in \mathcal{S} \cup \{k\}} p_i} \sum_{i \in \mathcal{S} \cup \{k\}} p_i \mathbf{w}_i \right), \quad (7)$$

Given the defined reward, GGC operates through a series of iterations, each involving a decision-making process for adding or removing clients from two sets, $\mathcal{X}$ and $\mathcal{Y}$. The set $\mathcal{X}$ represents the set of collaborators and initially contains client k which is the client running GGC, and $\mathcal{Y}$ contains clients in $\Omega_k \cup \{\text{k}\}$. In each step, the algorithm computes two variables, a and b, representing the expected marginal gains from adding and removing a specific client $j \in \mathcal{S}$, respectively, and defined as follows:

$$a = \max(\mathcal{R}(\mathcal{X} \cup \{j\}) - \mathcal{R}(\mathcal{X}), 0)$$
$$b = \max(\mathcal{R}(\mathcal{Y} \setminus \{j\}) - \mathcal{R}(\mathcal{Y}), 0).$$

Following GGC, client k adds client j to the collaborator set $\mathcal{X}$ with a probability $p = \frac{a}{a+b}$, where a represents the marginal gain from adding j and b represents the marginal gain from removing j. A client is more likely to be added if $a \geq b$. If the marginal gain from adding is positive and from removing is negative, $p = 1$ ensures the client is added. Conversely, if the gain from removing is positive and from adding is negative, $p = 0$ leads to removal. When both a and b are zero, $p = 1$ is set by default. This process continues until all clients in Ω_k are considered.

[1] A set function $f : 2^{\Omega} \to \mathbb{R}$ is monotone if for any $A \subseteq B \subseteq \Omega$ we have $f(A) \leq f(B)$.

GGC complexity: The overall complexity of GGC (Algorithm 2) is $\mathcal{O}(B_c)$, which reduces to constant complexity $\mathcal{O}(1)$ since during training B_c is constant. GGC operates by iterating over the received clients as input. It computes two variables a and b, requiring a constant number of forward passes over the network, which translates to $\mathcal{O}(1)$ complexity. Subsequently, it decides whether to add or remove a client, an operation also achieved in $\mathcal{O}(1)$. Therefore, the algorithm's complexity essentially reduces to looping over a fixed-size list of collaborators, resulting in $\mathcal{O}(B_c)$ during training, which does not increase with a growing number of clients as B_c is a predefined constant.

3.3 Batched GGC (BGGC)

As outlined in § 2.2, each client k can only collaborate with a set of clients Ω_k, which is restricted in size (i.e., $|\Omega_k| \leq B_c$) to maintain communication and resource efficiency. Moreover, as depicted in Algorithm 1, the initial graph construction is conducted as a preprocessing step where each client trains a model locally for a sufficient number of epochs using the same model initialization parameters, resulting in the client's local solution denoted as $\mathbf{w}_k^{init}$ (lines 1-2). Then in line 3, each client k determine its set of beneficial collaborators Ω_k by minimizing the following optimization:

$$\min_{\mathcal{S}\subseteq[N]\setminus k} F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{S}\cup k} p_i}\sum_{i\in\mathcal{S}\cup k} p_i \mathbf{w}_i^{init}\right) \text{ s.t. } |\mathcal{S}| \leq B_c. \tag{8}$$

Although our proposed Algorithm 2 in § 3.2 efficiently approximates the solution to our objective Eq. (8) while demonstrating resilience to noisy rewards (c.f. Corollary 2 in (Fourati et al., 2023)), this method relies on reward computations to the objective in Eq. (8), necessitating model updates from potential collaborators. This poses a significant challenge when applying GGC to the initial graph generation of Ω_k, as the number of required models from clients in $\mathcal{S}$ may surpass our budget constraint B_c, dictated by practical limitations such as network bandwidth, compute resources, and storage. To address this challenge, we propose an enhancement to the GGC algorithm called BGGC (Batched Greedy Graph Construction) by streamlining reward computations into efficient batches of communication and computation, each requiring only $\mathcal{O}(B_c)$ resources. Details of this approach are outlined in Algorithm 3; see Appendix C. Note that BGGC is necessary only during the preprocessing step. During the training loop (lines 6-12 in Algorithm 1), the reward computations of GGC will require, at most, B_c models from Ω_k, which falls within the imposed budget constraint.

Consistent with decentralized scenarios (Marfoq et al., 2020; Han et al., 2022; Ranathunga et al., 2022), we consider a cross-silo scenario, as establishing a collaboration graph or links between devices in a cross-device setting is often prohibitively complex due to device heterogeneity and connectivity issues. By focusing on the cross-silo setting, we can manage a reasonable number of devices, potentially scaling to thousands.

BGGC algorithm complexity. The overall computation complexity of BGGC is $\mathcal{O}(N)$. However, the communication and resource complexity is only $\mathcal{O}(B_c)$ for each of the $\lceil N/B_c \rceil$ steps.

Theorem 1. *Assuming seeded randomness, executing algorithms* GGC *and* BGGC *with the same seed produces identical results for a given client k, clients set $\mathcal{S}$, and budget B_c.*

The proof of Theorem 1 is reported in Appendix D.

Practical implications of GGC vs. BGGC. While Theorem 1 establishes that GGC and BGGC yield the same outcomes under identical conditions, they employ different methodologies during execution. Executing GGC during preprocessing may violate the budget constraint, as it requires access to all clients' weights for reward computation. BGGC addresses this by adhering to communication and resource constraints but incurs an additional communication phase. In contrast, during training, GGC avoids this extra phase, as each client naturally considers a subset Ω_k, where $\Omega_k \leq B_c$, thus staying within the budget. This makes GGC more efficient during training by reducing communication while still respecting the budget constraint.

3.4 Properties of DPFL

To establish the effectiveness of our proposed framework, we analyze the properties of the collaboration graph derived from solving Eq. (3), offering insights into its advantages over existing alternatives.

Proposition 1. *The collaboration graph resulting from solving Eq.* (3) *yields superior or the same results compared to any collaboration graph $\mathcal{C} \in \{\mathcal{C}_i \in 2^{[N]\setminus i}; |\mathcal{C}_i| \leq B_c\}_{i=1}^N$ that imposes additional restrictions on the collaboration structure, where N is the total number of clients.*

The proof of Proposition 1 is in Appendix E.

Corollary 1. *It follows from Proposition 1 that our targeted collaboration graph outperforms pure local training (no one collaborates with anyone), any random graph within $\mathcal{C}$, and any graph with further imposed symmetry.*

Remark 2. *In line with previous works (Shalev-Shwartz et al., 2009; Feldman and Vondrak, 2019),*

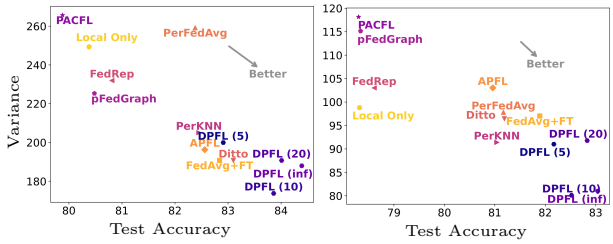


Figure 1: Variance between local models using Dir(0.1) (left) and Patho(3) (right) data splits on CIFAR10.

we assume that the validation loss effectively approximates the objective function $F_k(\mathbf{w}_k)$. Hence, Eq. (6) *serves as a reliable proxy for Eq.* (3)*. Additionally, we use Algorithm 2, which offers approximation guarantees even with noisy reward functions, as demonstrated in (Fourati et al., 2023, Corollary 2). Therefore, it ensures that the returned set is no worse than the empty set with probability* 1*. Consequently, applying this algorithm for graph generation in DPFL is expected to yield a strong solution to Eq.* (3)*.*

4 EVALUATION

We present key experimental settings and results, with additional details in Appendix H. We use standard datasets from personalized FL literature (Li et al., 2021; Collins et al., 2021; Marfoq et al., 2022). For CIFAR10 and CINIC10, we consider two heterogeneous distributions: (1) a pathological split, $Patho(3)$, where each client receives data from three categories (Zhang et al., 2023; McMahan et al., 2017; Colin et al., 2016), and (2) a Dirichlet distribution (Yurochkin et al., 2019; Wang et al., 2020) drawing from $\mathbf{q}_c \sim Dir_k(0.1)$ for each category c and allocating samples to clients accordingly. For FEMNIST, we use the natural non-IID split from Leaf (Caldas et al., 2018), assigning each writer as a client and increasing heterogeneity by omitting some classes per client. We train for 100 rounds on CIFAR10/FEMNIST and 50 on CINIC10, repeating experiments over three seeds (32, 42, 52) and reporting the average local test accuracy and standard deviation. For all methods, the best per-client model based on validation performance is used for testing.

4.1 Quality of personalization results

Table 1 presents the average test accuracy of DPFL under four budget constraints, compared to other personalized FL and baseline methods. For DPFL, we vary the budget constraint with 20% ($B_c = 0.2N$), 10% ($B_c = 0.1N$), and 5% ($B_c = 0.05N$) of the total number of clients, respectively; we also consider the case without a constraint ($B_c = \text{inf}$).

Our method significantly outperforms other personalized methods regarding average test accuracy. We achieve better results than local training by 3 and 5 percentage points (pp), outperform FedAvg by 37 and 33 pp, and surpass other personalized methods by approximately 1 to 6 and 2 to 10 pp on CIFAR10 using $Dir(0.1)$ and $Patho(3)$, respectively. Moreover, on FEMNIST, we improve upon local training and FedAvg by 7 pp. and surpass other personalized methods by approximately 2.8 to 6.5 pp. Finally, on CINIC10, our method demonstrates superior performance, surpassing local training by 3 pp in both $Dir(0.1)$ and $Patho(3)$. Additionally, DPFL outperforms FedAvg by a notable margin of 43 and 41 pp, while surpassing other personalized methods by approximately 1 to 5 and 2 to 7 pp using $Dir(0.1)$ and $Patho(3)$, respectively.

Variance between local models. As our objective is personalization, apart from the overall improvement in average accuracy among clients, it is crucial to assess whether improvements are distributed across most models rather than confined to just a few clients. To evaluate this, we analyze the variance between local models, where lower variance signifies greater parity in their performance. Fig. 1 shows results for CIFAR10 with a $Patho(3)$ distribution. The x-axis represents average test accuracy, and the y-axis represents variance between clients' models. DPFL is consistently in the right-bottom corner, showing higher overall accuracy and lower variance than other methods. Additionally, Appendix I.1 confirms these findings for FEMNIST (Fig. 5), CINIC10 (Fig. 6), and CIFAR10 with a $Dir(0.1)$ distribution (Fig. 7).

4.2 Visualization of collaboration graph

We analyze the initial collaboration graph and its evolution after some rounds. For clarity, we illustrate two cases with budget $B_c = 10$ and $B_c = 5$ (other budget constraints are in Appendix I.2). Fig. 2 depicts the collaboration graph (plotted as the adjacency matrix) in three states, from left to right: constructed as a preprocessing step (line 5 of Algorithm 1), in round 50, and in round 99. The diagonal indicates that every client always "collaborates" with itself. To illustrate the graph evolution, the round 50 and 99 plots display collaborative links in two colors: in red are the clients selected for collaboration in that round; in blue are the clients identified during the preprocessing step but are currently not chosen. The union of red and blue clients corresponds to the initial collaboration graph.

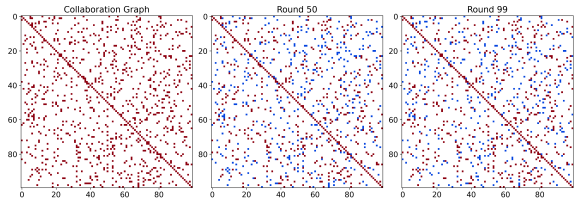
The figures highlight that the initial collaboration graph is denser compared to the actually used clients for aggregation in round 99. This is expected since the initial graph is constructed as a preprocessing step,

	CIFAR10		CINIC10		FEMNIST
	$Dir(0.1)$	$Patho(3)$	$Dir(0.1)$	$Patho(3)$	Natural Split
Local Only	80.38 ± 1.62	78.32 ± 0.49	78.59 ± 1.00	77.53 ± 0.47	87.27 ± 0.27
FedAvg (McMahan et al., 2017)	47.22 ± 1.20	50.93 ± 1.97	38.48 ± 0.37	39.27 ± 1.29	87.41 ± 0.72
FedAvg+FT	82.84 ± 0.88	81.89 ± 0.92	80.31 ± 0.33	79.18 ± 0.46	91.56 ± 0.29
FedProx (Lian et al., 2018)	48.83 ± 1.88	51.14 ± 0.79	38.16 ± 0.82	38.53 ± 0.54	88.14 ± 0.39
FedProx+FT (Lian et al., 2018)	78.86 ± 0.51	73.61 ± 1.29	76.59 ± 1.19	73.04 ± 1.08	93.58 ± 0.29
APFL (Smith et al., 2017)	82.56 ± 0.38	80.96 ± 0.71	80.02 ± 1.07	78.21 ± 0.39	90.80 ± 0.37
PerFedAvg (Fallah et al., 2020a)	82.38 ± 1.43	81.17 ± 0.52	78.88 ± 1.25	78.90 ± 0.37	91.69 ± 0.40
Ditto (Li et al., 2021)	83.10 ± 0.70	81.19 ± 0.63	80.33 ± 1.27	79.52 ± 0.31	90.83 ± 0.46
FedRep (Collins et al., 2021)	80.81 ± 1.09	78.61 ± 1.67	79.56 ± 1.12	78.46 ± 0.69	90.04 ± 0.28
kNN-Per (Marfoq et al., 2022)	82.45 ± 0.69	81.04 ± 0.40	80.42 ± 1.07	79.54 ± 0.32	91.29 ± 0.40
PACFL (Vahidian et al., 2023)	79.88 ± 0.69	78.30 ± 0.54	78.63 ± 1.10	77.83 ± 0.31	90.39 ± 0.64
pFedGraph (Ye et al., 2023)	80.48 ± 0.75	78.34 ± 1.04	78.73 ± 1.04	77.58 ± 0.16	87.70 ± 0.13
DPFL ($B_c = \inf$)	**84.39** ± 0.43	**83.04** ± 0.98	**81.49**± 1.32	**80.32**± 0.51	**94.25** ± 0.18
DPFL ($B_c = 0.2N$)	**84.01** ± 0.44	**82.83** ± 0.91	**81.24**± 1.30	**80.51**± 0.39	**94.31** ± 0.06
DPFL ($B_c = 0.1N$)	**83.86** ± 0.46	**82.52** ± 0.78	**81.32**± 1.20	**80.24**± 0.35	**94.09** ± 0.14
DPFL ($B_c = 0.05N$)	**82.91** ± 0.81	**82.17** ± 0.54	**80.91**± 1.31	**80.11**± 0.30	**93.82** ± 0.17

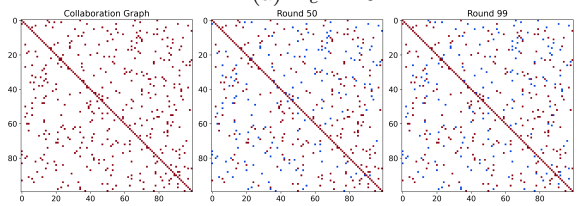
Table 1: Comparison of accuracy across benchmarks is illustrated as $acc \pm std$, where acc represents the average test accuracies of all clients' models, and std denotes the standard deviation over three repetitions.

	Decentralized SGD	DPFL (ours)
$B_c = 0.2N$	78.04 ± 1.55	**84.01** ± 0.44
$B_c = 0.1N$	77.02 ± 1.41	**83.86** ± 0.46
$B_c = 0.01N$	76.21 ± 1.87	**82.91** ± 0.81

Table 2: Comparison of our method DPFL versus decentralized SGD using CIFAR10.



(a) $B_c = 10$



(b) $B_c = 5$

Figure 2: Collaboration graph using CIFAR10.

and at this stage, model weights have not yet converged. Therefore, broader collaboration can be beneficial. However, as training progresses, it is natural to expect each client to benefit primarily from collaborating with clients with similar data distributions, thus leading to a sparser collaboration graph. Another contributing factor is that, in the preprocessing step, the decision to select a specific client for collaboration is made from a pool of 100 clients, making it more challenging than in later rounds where the decision is drawn from the smaller pool Ω_k for client k.

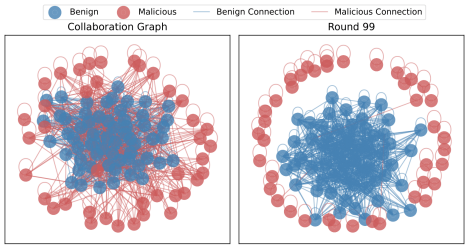
Finally, we analyze the graph sparsity. With $B_c = 10$, the initial sparsity is 80% and increases to 88% at round 99. With $B_c = 5$, the initial sparsity is 95% and is 96% in round 99. We also measure the symmetry of the collaboration graph across different budgets (details in Appendix I.4); we observe around 80-88% symmetry, which decreases as B_c decreases.

4.3 Behavior of DPFL under data flip attack

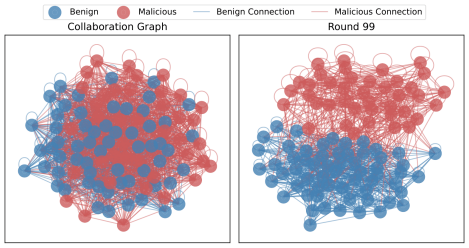
To analyze how DPFL behaves in the presence of distinct groups of clients, we select 40 clients (malicious) out of 100 and flip their labels using the same permutation; the remaining 60 clients (benign) use the true labels (according to CIFAR10). It's important to note that our objective is to delineate the behavior of DPFL, acknowledging that while it may exhibit robustness characteristics, the study of robustness falls outside the scope of this paper.

We then conduct two experiments: the first where malicious clients do not execute GGC (i.e., they only train locally); the second in which they run GGC. In the first case (see Fig. 3a), the collaboration graph initially contains numerous edges involving malicious clients. This is expected due to the inherent randomness in the weights during the preprocessing step, which makes it challenging to identify collaborators. However, as the rounds progress, we observe that benign clients increasingly avoid selecting malicious ones until, ultimately, they cease choosing them altogether. Fig. 12 depicts this evolution every 10 rounds.

In the second case (see Fig. 3b), since malicious clients execute GGC, they initially collaborate with benign clients. Thus, their models become regularized towards the benign ones. Despite this behavior, we observe that as the rounds progress, clients become almost segregated into two groups (red and blue), with



(a) Malicious clients don't execute GGC.



(b) Malicious clients execute GGC.

Figure 3: Collaboration graph when 40% of clients have flipped labels (malicious), while 60% have original labels (benign). For both scenarios, we show the initial collaboration graph (left) and its evolution after 99 rounds (right). Malicious clients appear in red; benign ones are in blue.

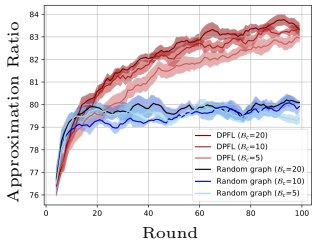


Figure 4: DPFL vs. FL over a randomly generated graph.

very few links between them serving as a form of regularization (full evolution in Fig. 13). On average, benign clients have less than 10% connections with malicious ones (see Fig. 14).

4.4 Comparing with decentralized SGD

Our approach differs from decentralized methods by not assuming a predefined collaboration graph. Instead, we optimize the graph for better personalization. Table 2 compares our method with decentralized SGD (Liu et al., 2024) using the CIFAR10 dataset and $Dir(0.1)$. For decentralized SGD method the collaboration graph is generated randomly as per Definition 1 in Liu et al. (2024).

4.5 Comparing with a random graph

To illustrate the relative importance of GGC, we compare DPFL to a version of our method that replaces GGC with a randomly-constructed collaboration graph. Figure 4 shows that DPFL outperforms the random graph by ≈4 pp, in three budget constraints. This experiment uses CIFAR10 with 100 clients and $Dir(0.1)$ distribution.

4.6 Ablation studies

Sensitivity to τ_{init}. Table 3 reports the results of using CIFAR10 with 100 clients and $Patho(3)$ split and varying the number of local epochs τ_{init}. The results show that the accuracies across different numbers of local epochs are comparable. Performance on average is slightly better with $\tau_{\text{init}} = 10$, but even the case with $\tau_{\text{init}} = 1$ yields good results.

Periodicity of refreshing $\mathcal{C}_k$. To improve efficiency, we consider invoking GGC at line 10 in Algorithm 1 periodically every P rounds. Table 4 reports the effect of varying P while training on CIFAR10 using 100 clients and $Dir(0.1)$. We observe that DPFL maintains good performance across various periodicities, with a slight decrease as P increases. This demonstrates the robustness of our method and the potential for improved efficiency by performing it periodically.

Considering partial participation. To demonstrate the effectiveness of our method, we conduct an experiment in which initially, each client can communicate with only 50% of other clients. During training, we simulate random client dropouts by having each client receive updates from only 80% of its collaborators each round (20% dropout). We use CIFAR-10 dataset and Dir(0.1). All hyperparameters remain as described in Appendix H.

This setup is easily achievable using the DPFL algorithm. Clients periodically "check in" with their collaborators; if a client fails to check in during a round, it does not contribute to the model update, and no weights are exchanged with that client. The GGC selection is agnostic to the number of available clients and will always select the most suitable ones for collaboration. Thus, the algorithm can adapt to this setting by providing a subset of online clients, $\hat{\Omega}_k \subseteq \Omega_k$, in Line 10, instead of all neighbors.

Table 5 shows that the performance using partial participation (p.p.) decreases by about 1% compared to full participation (f.p.). However, these results still outperform all other methods reported in Table 1 (except $B_c = 0.01N$), showing the efficiency of DPFL while saving more communication and allowing clients to drop out during training.

Rebuilding the graph at every round using BGGC. Our algorithm begins with an initial preprocessing step to establish the graph, followed by

τ_{init} \ B_c	inf	20	10	5
1	81.86 ± 1.00	82.33 ± 0.86	82.53 ± 0.72	$\mathbf{82.30} \pm 0.24$
5	83.03 ± 0.73	82.53 ± 1.36	$\mathbf{82.58} \pm 0.97$	80.90 ± 0.96
10	$\mathbf{83.04} \pm 0.98$	$\mathbf{82.83} \pm 0.91$	82.52 ± 0.78	82.17 ± 0.54

Table 3: Effect of local epochs performed in the preprocessing step on the convergence of DPFL.

P \ B_c	inf	20	10	5
1	84.20	83.67	84.20	84.33
5	83.83	84.01	83.31	83.73
10	83.24	84.17	83.31	83.03
20	82.34	82.59	82.08	82.73

Table 4: Effect of the periodicity of invoking GGC on the convergence of DPFL.

	DPFL (p.p.)	DPFL (f.p.)
$B_c = inf$	83.98 ± 0.55	**84.39** ± 0.43
$B_c = 0.2N$	83.10 ± 1.03	**84.01** ± 0.44
$B_c = 0.1N$	82.54 ± 0.85	**83.86** ± 0.46
$B_c = 0.01N$	81.43 ± 0.26	**82.91** ± 0.81

Table 5: Comparison of DPFL performance under partial participation (p.p.) and full participation (f.p.).

	DPFL w/ repeated BGGC	DPFL
$B_c = inf$	**85.46** ± 0.73	84.39 ± 0.43
$B_c = 0.2N$	**84.58** ± 0.70	84.01 ± 0.44
$B_c = 0.1N$	**83.89** ± 0.60	83.86 ± 0.46
$B_c = 0.01N$	**83.83** ± 0.51	82.91 ± 0.81

Table 6: Performance comparison of DPFL and DPFL with repeated BGGC (in every round).

reevaluation of clients' contributions in each communication round using the GGC Algorithm. This ensures that the collaboration graph reflects the most beneficial clients at each round. To test the preprocessing step's sufficiency, we run the BGGC algorithm in every round on a fully connected network, which serves as an upper bound to our method.

The results in Table 6 show improved performance of the DPFL w/ repeated BGGC compared to the original DPFL, but the gain is minimal (about 1 pp). This suggests that the initial collaboration graph, continuously adjusted each round, is robust and effective, with performance not being highly sensitive to the initial graph.

5 RELATED WORK

Personalized FL approaches address data heterogeneity using various strategies. Some methods train both global and personalized models and regularize them with an l_2 term (Li et al., 2021; T Dinh et al., 2020). Others employ multi-task learning techniques (Hanzely and Richtárik, 2020; Khodak et al., 2019; Jiang et al., 2019; Mansour et al., 2020; Gasanov et al., 2022; Hanzely et al., 2020; Marfoq et al., 2022), interpolating between local and global models. Meta-learning-based methods (Fallah et al., 2020b) seek a common initial model that clients can easily adapt to. Other techniques explore splitting model layers (Collins et al., 2021; Liang et al., 2020) or using hypernetworks for personalization (Shamsian et al., 2021; Chen and Chao, 2021). Unlike these approaches, our method eliminates communication overhead by avoiding a global model and achieves fine-grained personalization through dynamic collaborator selection.

In contrast to cosine similarity-based methods (Ye et al., 2023; Li et al., 2022a,b), which focus on pairwise collaborator selection and overlook group synergy; we introduce a novel utility function (Eq. (6)) that considers the collective benefit of collaborators. Additionally, unlike methods that average all clients with weighted schemes (Ye et al., 2023), which limits scalability in constrained environments, our approach incorporates a budget constraint for improved real-world applicability. Furthermore, personalized decentralized methods (Vanhaesebrouck et al., 2017; Koppel et al., 2017; Even et al., 2022) assume a fixed collaboration graph and are often restricted to linear models, relying on parameter tuning to control graph sparsity. In contrast, our approach provides an exact budget without being constrained to linear models.

Moreover, clustering-based approaches (Vahidian et al., 2023; Chen and Chao, 2021; Shamsian et al., 2021; Wang et al., 2023) enhance personalization by grouping clients based on data similarity. However, they often require a predefined number of clusters or similarity thresholds, which may vary across tasks. Our method avoids such hyperparameters, offering greater flexibility and finer-grained collaboration.

Finally, decentralized approaches such as Koloskova et al. (2020); Liu et al. (2024) typically assume a given communication graph; our work fundamentally differs in its approach by dynamically constructing the graph tailored to optimize personalized objectives.

6 CONCLUSION

We addressed the challenge of constructing a collaboration graph in decentralized learning while considering data heterogeneity and adhering to imposed communication and resource constraints. To achieve this, we proposed a bi-level optimization problem and devised a greedy algorithm to efficiently identify the collaboration graph. Experiments on various datasets showed that our method outperforms state-of-the-art baselines.

Acknowledgments

This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No. ORA-CRG2021-4699. We are thankful to the anonymous reviewers for their thoughtful comments and suggestions that helped improving our paper. For computer time, this research used Ibex managed by the Supercomputing Core Laboratory at KAUST.

References

Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 473–481. PMLR, 2018.

Enrique Tomás Martínez Beltrán, Mario Quiles Pérez, Pedro Miguel Sánchez Sánchez, Sergio López Bernal, Gérôme Bovet, Manuel Gil Pérez, Gregorio Martínez Pérez, and Alberto Huertas Celdrán. Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. *IEEE Communications Surveys & Tutorials*, 2023.

Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, pages 1384–1402, 2015.

Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Hong-You Chen and Wei-Lun Chao. On bridging generic and personalized federated learning for image classification. *arXiv preprint arXiv:2107.00778*, 2021.

Igor Colin, Aurélien Bellet, Joseph Salmon, and Stéphan Clémençon. Gossip dual averaging for decentralized optimization of pairwise functions. In *International Conference on Machine Learning*, pages 1388–1396. PMLR, 2016.

Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. In *International conference on machine learning*, pages 2089–2099. PMLR, 2021.

Edwige Cyffers and Aurélien Bellet. Privacy amplification by decentralization. In *International Conference on Artificial Intelligence and Statistics*, pages 5334–5353. PMLR, 2022.

Luke N Darlow, Elliot J Crowley, Antreas Antoniou, and Amos J Storkey. Cinic-10 is not imagenet or cifar-10. *arXiv preprint arXiv:1810.03505*, 2018.

Yuyang Deng, Mohammad Mahdi Kamani, and Mehrdad Mahdavi. Adaptive personalized federated learning. *arXiv preprint arXiv:2003.13461*, 2020.

Mathieu Even, Laurent Massoulié, and Kévin Scaman. Sample optimality and all-for-all strategies in personalized federated and collaborative learning. *arXiv preprint arXiv:2201.13097*, 2022.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020a.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, pages 3557–3568, 2020b.

Vitaly Feldman and Jan Vondrak. High probability generalization bounds for uniformly stable algorithms with nearly optimal rate. In *Conference on Learning Theory*, pages 1270–1279. PMLR, 2019.

Fares Fourati, Vaneet Aggarwal, Christopher Quinn, and Mohamed-Slim Alouini. Randomized greedy learning for non-monotone stochastic submodular maximization under full-bandit feedback. In *International Conference on Artificial Intelligence and Statistics*, pages 7455–7471. PMLR, 2023.

Elnur Gasanov, Ahmed Khaled, Samuel Horváth, and Peter Richtarik. Flix: A simple and communication-efficient alternative to local methods in federated learning. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, pages 11374–11421. PMLR, 2022.

Jialiang Han, Yudong Han, Gang Huang, and Yun Ma. Defl: Decentralized weight aggregation for cross-silo federated learning. *arXiv preprint arXiv:2208.00848*, 2022.

Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516*, 2020.

Filip Hanzely, Slavomír Hanzely, Samuel Horváth, and Peter Richtárik. Lower bounds and optimal algorithms for personalized federated learning. *Advances in Neural Information Processing Systems*, 33:2304–2315, 2020.

Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

Zhanhong Jiang, Aditya Balu, Chinmay Hegde, and Soumik Sarkar. Collaborative deep learning in fixed topology networks. *Advances in Neural Information Processing Systems*, 30, 2017.

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. *Foundations and Trends® in Machine Learning*, pages 1–210, 2021.

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

Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pages 5381–5393. PMLR, 2020.

Alec Koppel, Brian M Sadler, and Alejandro Ribeiro. Proximity without consensus in online multiagent optimization. *IEEE Transactions on Signal Processing*, 65(12):3062–3077, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Shuangtong Li, Tianyi Zhou, Xinmei Tian, and Dacheng Tao. Learning to collaborate in decentralized learning of personalized models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9766–9775, 2022a.

Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021.

Zexi Li, Jiaxun Lu, Shuang Luo, Didi Zhu, Yunfeng Shao, Yinchuan Li, Zhimeng Zhang, Yongheng Wang, and Chao Wu. Towards effective clustered federated learning: A peer-to-peer framework with adaptive neighbor matching. *IEEE Transactions on Big Data*, 2022b.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in neural information processing systems*, 30, 2017.

Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

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.

Yue Liu, Tao Lin, Anastasia Koloskova, and Sebastian U Stich. Decentralized gradient tracking with local steps. *Optimization Methods and Software*, pages 1–28, 2024.

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

Othmane Marfoq, Chuan Xu, Giovanni Neglia, and Richard Vidal. Throughput-optimal topology design for cross-silo federated learning. *Advances in Neural Information Processing Systems*, pages 19478–19487, 2020.

Othmane Marfoq, Giovanni Neglia, Richard Vidal, and Laetitia Kameni. Personalized federated learning through local memorization. In *International Conference on Machine Learning*, pages 15070–15092. PMLR, 2022.

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

Angelia Nedić, Alex Olshevsky, and Michael G Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, pages 953–976, 2018.

Giovanni Neglia, Gianmarco Calbi, Don Towsley, and Gayane Vardoyan. The role of network topology for distributed machine learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2350–2358. IEEE, 2019.

Tharindu Ranathunga, Alan McGibney, Susan Rea, and Sourabh Bharti. Blockchain-based decentralized model aggregation for cross-silo federated learning in industry 4.0. *IEEE Internet of Things Journal*, pages 4449–4461, 2022.

Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Stochastic convex optimization. In *Conference on Learning Theory (COLT)*, 2009.

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

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, pages 21394–21405, 2020.

Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. D^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pages 4848–4856. PMLR, 2018.

Konstantinos I Tsianos, Sean Lawlor, and Michael G Rabbat. Consensus-based distributed optimization: Practical issues and applications in large-scale machine learning. In *2012 50th annual allerton conference on communication, control, and computing (allerton)*, pages 1543–1550. IEEE, 2012.

Saeed Vahidian, Mahdi Morafah, Weijia Wang, Vyacheslav Kungurtsev, Chen Chen, Mubarak Shah, and Bill Lin. Efficient distribution similarity identification in clustered federated learning via principal angles between client data subspaces. In *Proceedings of the AAAI conference on artificial intelligence*, pages 10043–10052, 2023.

Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*, pages 509–517. PMLR, 2017.

Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*, 2020.

Jiaqi Wang, Xingyi Yang, Suhan Cui, Liwei Che, Lingjuan Lyu, Dongkuan DK Xu, and Fenglong Ma. Towards personalized federated learning via heterogeneous model reassembly. *Advances in Neural Information Processing Systems*, pages 29515–29531, 2023.

Rui Ye, Zhenyang Ni, Fangzhao Wu, Siheng Chen, and Yanfeng Wang. Personalized federated learning with inferred collaboration graphs. In *International Conference on Machine Learning*, pages 39801–39817. PMLR, 2023.

Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning*, pages 7252–7261. PMLR, 2019.

Jianqing Zhang, Yang Hua, Hao Wang, Tao Song, Zhengui Xue, Ruhui Ma, and Haibing Guan. Fedala: Adaptive local aggregation for personalized federated learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 11237–11244, 2023.

Jiaqi Zhang and Keyou You. Fully asynchronous distributed optimization with linear convergence in directed networks. *arXiv preprint arXiv:1901.08215*, 2019.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, Section 2 and Section 3]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes, Section 3 and Appendix B, C]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes, supplementary material https://github.com/salmakh1/DPFL]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes, Section 3]
   (b) Complete proofs of all theoretical results. [Yes, Appendix D]
   (c) Clear explanations of any assumptions. [Yes, Section 3]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes, Appendix H and supplementary material https://github.com/salmakh1/DPFL]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, Appendix H]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes, experimental section main paper and Appendix H]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes, Appendix H]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes, Appendix H]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes, supplementary material]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A MOTIVATION OF OUR COMBINATORIAL PROPOSITION

In this section, we will show how our utility function based on group synergy by considering the combinatorial effect of clienwts is more interesting than pairwise comparisons.

We conducted an experiment utilizing the CIFAR10 dataset. We allocated three clients as follows: Client 1 has access to four classes (0, 4, 6, 8) with 50, 50, 50, and 50 data points per class, respectively. Client 2 is assigned four classes (0, 6, 1, 3) with 300, 300, 200, and 200 data points per class, respectively. Lastly, client 3 has access to four classes (4, 8, 5, 7) with 300, 300, 200, and 200 data points per class, respectively. In other words, clients 2 and 3 have two classes in common each with client 1. We then train three models using FedAvg-fine-tuned-like training (E local steps and aggregation): one where clients 1 and 2 collaborate, one where clients 1 and 3 collaborate, and one where all three clients collaborate. Remarkably, our findings indicate that collaboration between clients 1 and 2, or 1 and 3 alone, leads to a decrease in performance by approximately 11% in both cases when compared to client 1 training a model independently on its own private dataset alone.

However, the collaborative effort involving clients 1, 2, and 3 collectively enhances accuracy by approximately 6%.

The effectiveness of GGC in building the collaboration graph could also be shown in the case of two clients each of them contributed positively to client 1, however the collaboration of 1, 2, 3 leads to negative gain.

Moreover, we would like to clarify that these two cases are the extreme ones where the algorithms that compare pairwise performance fail to address, however sometimes adding client 1 alone is good and adding client 2 alone is good, but the marginal gain of adding them together is significantly higher, and this could play an important role when we have communication constraints where we are restricted to collaborate only with a subset of clients and choosing them wisely is important.

We believe that scenarios, where the synergy between a group of clients outweighs the pairwise comparison of collaboration effectiveness, are common in real-world settings, particularly when dealing with highly heterogeneous data. In such cases, a specific client might adversely affect the training of another client. However, when collaborating with a "complementary client", the combined effort can significantly enhance performance.

B GREEDY GRAPH CONSTRUCTION (GGC)

Algorithm 2 Greedy Graph Construction (GGC)

Require: client k, clients $\mathcal{S}$
$\mathcal{X} \leftarrow \{k\}, \mathcal{Y} \leftarrow \mathcal{S} \cup \{k\}$,
for client $j \in \text{Shuffle}(\mathcal{S})$ **do**
  $\mathcal{R}(\mathcal{X}) \leftarrow -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{X}} p_i}\sum_{i\in\mathcal{X}} p_i \mathbf{w}_i\right)$
  $\mathcal{R}(\mathcal{X}\cup\{j\}) \leftarrow -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{X}\cup\{j\}} p_i}\sum_{i\in\mathcal{X}\cup\{j\}} p_i \mathbf{w}_i\right)$
  $\mathcal{R}(\mathcal{Y}) \leftarrow -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{Y}} p_i}\sum_{i\in\mathcal{Y}} p_i \mathbf{w}_i\right)$
  $\mathcal{R}(\mathcal{Y}\setminus\{j\}) \leftarrow -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{Y}\setminus\{k\}} p_i}\sum_{i\in\mathcal{Y}\setminus\{j\}} p_i \mathbf{w}_i\right)$
  $a \leftarrow \max(\mathcal{R}(\mathcal{X}\cup\{j\}) - \mathcal{R}(\mathcal{X}), 0)$
  $b \leftarrow \max(\mathcal{R}(\mathcal{Y}\setminus\{j\}) - \mathcal{R}(\mathcal{Y}), 0)$
  with probability $p = \frac{a}{a+b}$ **do**
    $\mathcal{X} \leftarrow \mathcal{X}\cup\{j\}$ and $\mathcal{Y} \leftarrow \mathcal{Y}$
  else
    $\mathcal{Y} \leftarrow \mathcal{Y}\setminus\{j\}$ and $\mathcal{X} \leftarrow \mathcal{X}$
end for
if $|\mathcal{X}| = B_c$ **then**
  break
end if
Return $\mathcal{X}$

Additional explanation: We would like to note that as our objective function is not monotone, meaning

that adding won't always be the best choice, we necessitate the computation of both marginal gains of adding and removing a client. To this end, four cases will appear, three of them will happen or do not happen with probability 1.

- If the marginal gain of adding is positive (a is positive) and the marginal gain of removing is negative (b is zero) then p = 1.
- If the marginal gain of adding is negative (a is zero) and the marginal gain of removing is positive (b is positive) then p = 0 (does not happen with probability 1).
- If the marginal gain of adding is negative (a is zero) and the marginal gain of removing is negative (b is zero) then p = 1.
- If the marginal gain of adding is positive (a is positive) and the marginal gain of removing is positive (b is positive) only in this case we flip a biased coin that decides probability p based on the values of a and b for either adding or removing that client. In our experiments, we noticed that this case happens less than 1% of the time.

Furthermore, it has been shown in (Buchbinder et al., 2015) that if you remove the randomness in the agnostic case i.e., when the marginal gains of adding and removing are both positive (a and b are positive) then the algorithm guarantees decreases from achieving $\frac{1}{2}$ of the optimal solution to the $\frac{1}{3}$ of the optimal solution.

C EFFICIENT REWARD COMPUTATION

Looking at Algorithm 2, we notice that it requires reward computations to the function $\mathcal{R}(S)$, which is defined in our objective according to Eq. (7). This reward computation requires access to $\mathbf{w}_i$ in $\mathcal{S}$, where $|\mathcal{S}|$ may exceed our budget constraint B_c (see Eq. (8)). Therefore, to ensure that we do not violate our constraints during the preprocessing step outlined in Algorithm 1, we propose amortizing the communication complexity needed to evaluate the objective function. This involves dividing the necessary downloads into $O(\frac{n}{B_c})$ steps. Each step entails only $O(B_c)$ computation, storage, and communication complexity, as opposed to a single step with $O(n)$ complexity, which breaches the budget constraint. To elaborate, during each communication step, a client k receives at most B_c model updates and monitors the necessary averages for algorithmic decision-making, without retaining all individual models.

To execute this, the process commences by preparing the call for variable b, requiring the average of all models in $\mathcal{Y}$, beginning with a full set of clients. We envisage $\lceil \frac{n}{B_c} \rceil$ communication steps (line 2), where in each step, B_c clients transmit their model updates (line 3). We compute the average, as depicted in lines 4-6 of Algorithm 3. In line 5, $\mathbf{w}^Y$ denotes storing the weighted-sum of received models in batches. Upon completing these steps, $\mathbf{w}^Y$ is computed. The second phase of communications initiates. Client k begins receiving batches of models without replacement from B_c clients, storing their indices in S_b. Similar to GGC, client k computes the marginal gains of adding and removing a client from $\mathcal{S}_b$ by computing a and b, respectively. A decision is then made based on probability p in line 16. A crucial step is to keep track of the values of $\mathbf{w}^X$ and $\mathbf{w}^Y$ and update them as in lines 17 and 19, for future use even for the next received batch of models B_c.

It's important to note that this expanded window of communications is solely required for the preprocessing step; for graph verification at line 10 of Algorithm 1, the algorithm takes as input $|\Omega_k| \leq B_c$.

Algorithm 3 Batched Greedy Graph Construction (BGGC)

Require: client k, clients $\mathcal{S}$, budget B_c.
$\mathbf{w}^Y \leftarrow p_k \mathbf{w}_k$
for s in range($\lceil \frac{n}{B_c} \rceil$) **do**
  Client k receives a batch $\mathcal{B}$ of at most B_c models without replacement
  for model $\mathbf{w}_s \in \mathcal{B}$ **do**
    $\mathbf{w}^Y \leftarrow \mathbf{w}^Y + p_s * \mathbf{w}_s$
  end for
end for
$\mathcal{X} \leftarrow \{k\}, \mathcal{Y} \leftarrow \mathcal{S} \cup \{k\}$, $\mathbf{w}^X \leftarrow p_k \mathbf{w}^k, \mathbf{w}^Y \leftarrow \mathbf{w}^Y$,
$\mathcal{S} \leftarrow$ Shuffle($\mathcal{S}$)
for s in range($\lceil \frac{n}{B_c} \rceil$) **do**
  Client k receives a batch $\mathcal{B}$ of at most B_c models in order from $\mathcal{S}$ without replacement and indices in $\mathcal{S}_b$
  for j in $\mathcal{S}_b$ **do**
    $\mathcal{R}(\mathcal{X}) \leftarrow -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^X}{\sum_{i \in \mathcal{X}} p_i} \right)$
    $\mathcal{R}(\mathcal{X} \cup \{j\}) \leftarrow -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^X + p_j \mathbf{w}^j}{(p_j + \sum_{i \in \mathcal{X}} p_i)} \right)$
    $\mathcal{R}(\mathcal{Y} \setminus \{j\}) \leftarrow -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^Y - p_j \mathbf{w}^j}{(-p_j + \sum_{i \in \mathcal{Y}} p_i)} \right)$
    $\mathcal{R}(\mathcal{Y}) \leftarrow -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^Y}{\sum_{i \in \mathcal{Y}} p_i} \right)$
    $a \leftarrow \max(\mathcal{R}(\mathcal{X} \cup \{j\}) - \mathcal{R}(\mathcal{X}), 0)$ {marginal gain of adding a client}
    $b \leftarrow \max(\mathcal{R}(\mathcal{Y} \setminus \{j\}) - \mathcal{R}(\mathcal{Y}), 0)$ {marginal gain of removing a client}
    with probability $p = \frac{a}{a+b}$ **do**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{j\}$; $\mathcal{Y} \leftarrow \mathcal{Y}$; $\mathbf{w}^X \leftarrow \mathbf{w}^X + p_j \mathbf{w}^j$
    else
      $\mathcal{Y} \leftarrow \mathcal{Y} \setminus \{j\}$; $\mathcal{X} \leftarrow \mathcal{X}$; $\mathbf{w}^Y \leftarrow \mathbf{w}^Y - p_j \mathbf{w}^j$
    if $|\mathcal{X}| = B_c$ **then**
      break
    end if
  end for
end for
Return $\mathcal{X}$

D PROOF OF Theorem 1

Both algorithms, GGC and BGGC, for a given client k, for a given set of clients $\mathcal{S}$, aim to find a set of collaborators $\mathcal{X} \subseteq \mathcal{S} \cup k$ that maximizes the reward function $\mathcal{R}(\mathcal{X}) \triangleq -F_k^{\mathcal{V}} \left(\frac{1}{\sum_{i \in \mathcal{X}} p_i} \sum_{i \in \mathcal{X}} p_i \mathbf{w}_i \right)$. Both algorithms use the marginal gains a and b computed from the reward function $\mathcal{R}(\cdot)$ and add a client with the same probability function $p = \frac{a}{a+b}$. Both BGGC and GGC initialize $\mathcal{X}$ as $\{k\}$ and $\mathcal{Y}$ as $\mathcal{S} \cup \{k\}$. Both algorithms go through the potential collaborators in $\mathcal{S}$ and decide to add or remove sequentially.

Assuming both GGC and BGGC apply some seeded sorting function to set $\mathcal{S}$ and follow the shuffled order, to show that both methods yield exactly the same output with seeded randomness, it only needs to be verified that the computed p is the same for GGC and BGGC for each client. This requires showing that the reward computation yields exactly the same outcome.

In the following, we show that GGC and BGGC compute the same reward function of the four considered sets in every decision round, which are $\mathcal{X}$, $\mathcal{X} \cup \{k\}$, $\mathcal{Y}$ and $\mathcal{Y} \setminus \{k\}$, leading to the same value for the probability function p, hence the same decision.

Reward of $\mathcal{Y}$ computation

GGC computes $\mathcal{R}(\mathcal{Y})$ as follows:

$$\mathcal{R}(\mathcal{Y}) = -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{Y}} p_i}\sum_{i\in\mathcal{Y}} p_i\mathbf{w}_i\right), \tag{9}$$

which implicitly assumes having access to all the weights of clients in $\mathcal{Y}$, possibly requiring communication with all these clients as well as the storage of their models.

In contrast, BGGC, does not assume the possibility of communicating or storing more models than the expected budget, hence in its first loop (lines 2-6) iterates through batches of clients, summing their weighted models into $\mathbf{w}^Y$. This effectively pre-computes the sum for the entire client set $\mathcal{Y} = \mathcal{S} \cup \{k\}$:

$$\mathbf{w}^Y = p_k\mathbf{w}_k + \sum_{s\in\lceil\frac{n}{\mathcal{B}_c}\rceil}\sum_{j\in\mathcal{B}_s} p_j\mathbf{w}_j \qquad = p_k\mathbf{w}_k + \sum_{j\in\mathcal{S}} p_j\mathbf{w}_j = \sum_{j\in\mathcal{S}\cup\{k\}} p_j\mathbf{w}_j \qquad = \sum_{j\in\mathcal{Y}} p_j\mathbf{w}_j, \tag{10}$$

where $\mathcal{B}_s$ denotes a batch of clients received in the s-th iteration. This weighted summation clearly ends up summing all the weights of all the clients in $\mathcal{S}$.

This summation is adaptive to the change in $\mathcal{Y}$, as shown in line 23, where whenever a client j is removed, its weighted weights ($p_j\mathbf{w}^j$) are removed from the weighted sum ($\mathbf{w}^Y - p_j\mathbf{w}^j$). Therefore, $\mathbf{w}^Y$ always represents the weighted sum of the model's weights in the set $\mathcal{Y}$. Hence, in every decision step, it is always the case that:

$$\mathbf{w}^Y = \sum_{j\in\mathcal{Y}} p_j\mathbf{w}_j. \tag{11}$$

BGGC computes $\mathcal{R}(\mathcal{Y})$, in line 17, as follows:

$$\mathcal{R}(\mathcal{Y}) = -F_k^{\mathcal{V}}\left(\frac{\mathbf{w}^Y}{\sum_{i\in\mathcal{Y}} p_i}\right).$$

Replacing by $\mathbf{w}^Y$ recovers the same reward computation of GGC.

Reward of $\mathcal{Y}\setminus\{k\}$ computation

GGC computes $\mathcal{R}(\mathcal{Y}\setminus\{j\})$ as follows:

$$\mathcal{R}(\mathcal{Y}\setminus\{j\}) = -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{Y}\setminus\{j\}} p_i}\sum_{i\in\mathcal{Y}\setminus\{j\}} p_i\mathbf{w}_i\right),$$

which implicitly assumes having access to all the weights of clients in $\mathcal{Y}$.

BGGC computes $\mathcal{R}(\mathcal{Y})$, in line 16, as follows:

$$\mathcal{R}(\mathcal{Y}\setminus\{j\}) = -F_k^{\mathcal{V}}\left(\frac{\mathbf{w}^Y - p_j\mathbf{w}^j}{-p_j + \sum_{i\in\mathcal{Y}} p_i}\right) = -F_k^{\mathcal{V}}\left(\frac{\mathbf{w}^Y - p_j\mathbf{w}^j}{\sum_{i\in\mathcal{Y}\setminus\{j\}} p_i}\right).$$

Replacing by $\mathbf{w}^Y$, using Eq. (11) recovers the same reward computation of GGC.

Reward of $\mathcal{X}$ computation

GGC computes $\mathcal{R}(\mathcal{X})$ as follows:

$$\mathcal{R}(\mathcal{X}) = -F_k^{\mathcal{V}}\left(\frac{1}{\sum_{i\in\mathcal{X}} p_i}\sum_{i\in\mathcal{X}} p_i\mathbf{w}_i\right).$$

BGGC initializes $\mathcal{X} = \{k\}$ in the same way as GGC. Moreover, initialized $\mathbf{w}^X \leftarrow p_k\mathbf{w}^k$. Therefore, in the first iteration, $\mathbf{w}^X$ represents exactly the weighted weight of client k, representing the sum of that single element.

This summation is adaptive to the change in $\mathcal{X}$, as shown in line 21, where whenever a client j is added, its weighted weights ($p_j \mathbf{w}^j$) are added to the weighted sum ($\mathbf{w}^X + p_j \mathbf{w}^j$). Therefore, $\mathbf{w}^X$ always represents the weighted sum of the model's weights in the set $\mathcal{X}$. Hence, in every decision step, it is always the case that:

$$\mathbf{w}^X = \sum_{j \in \mathcal{X}} p_j \mathbf{w}_j. \tag{12}$$

BGGC computes $\mathcal{R}(\mathcal{Y})$, in line 14, as follows:

$$\mathcal{R}(\mathcal{X}) = -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^X}{\sum_{i \in \mathcal{X}} p_i} \right).$$

Replacing by $\mathbf{w}^X$ recovers the same reward computation of GGC.

Reward of $\mathcal{X} \cup \{k\}$ computation

GGC computes $\mathcal{R}(\mathcal{X} \cup \{j\})$ as follows:

$$\mathcal{R}(\mathcal{X} \cup \{j\}) = -F_k^{\mathcal{V}} \left(\frac{1}{\sum_{i \in \mathcal{X} \cup \{j\}} p_i} \sum_{i \in \mathcal{X} \cup \{j\}} p_i \mathbf{w}_i \right)$$

BGGC computes $\mathcal{R}(\mathcal{Y})$, in line 15, as follows:

$$\mathcal{R}(\mathcal{X} \cup \{j\}) = -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^X + p_j \mathbf{w}^j}{p_j + \sum_{i \in \mathcal{X}} p_i} \right) = -F_k^{\mathcal{V}} \left(\frac{\mathbf{w}^X + p_j \mathbf{w}^j}{\sum_{i \in \mathcal{X} \cup \{j\}} p_i} \right)$$

Replacing by $\mathbf{w}^X$ using Eq. (12) recovers the same reward computation of GGC.

E PROOF OF Proposition 1

Assume there exists a collaboration graph $\mathcal{C}_c$, which is optimal within a restricting subset $\mathcal{P}$, i.e.,

$$\mathcal{C}_c \in \underset{\substack{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N \\ \mathcal{C}=\{\mathcal{C}_i \in 2^{[N]\setminus\{i\}};\ |\mathcal{C}_i| \leq B_c\}_{i=1}^N \\ \mathcal{C} \in \mathcal{P}}}{\arg\min} \left\{ F(\mathbf{w}, \mathcal{C}) \triangleq \sum_{k=1}^N p_k F_k \left(\mathbf{w}_k, \mathcal{C}_k \right) \right\}$$

Solving Eq. (3) yields a collaboration graph $\mathcal{C}^\star$, where

$$\mathcal{C}^\star \in \underset{\substack{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N \\ \mathcal{C}=\{\mathcal{C}_i \in 2^{[N]\setminus\{i\}};\ |\mathcal{C}_i| \leq B_c\}_{i=1}^N,}}{\arg\min} \left\{ F(\mathbf{w}, \mathcal{C}) \triangleq \sum_{k=1}^N p_k F_k \left(\mathbf{w}_k, \mathcal{C}_k \right) \right\}$$

Therefore, it follows that

$$\begin{aligned}
\min_{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N} F(\mathbf{w}, \mathcal{C}^\star) &= \min_{\substack{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N \\ \mathcal{C}=\{\mathcal{C}_i \in 2^{[N]\setminus\{i\}};\ |\mathcal{C}_i| \leq B_c\}_{i=1}^N,}} \left\{ F(\mathbf{w}, \mathcal{C}) \triangleq \sum_{k=1}^N p_k F_k \left(\mathbf{w}_k, \mathcal{C}_k \right) \right\} \\
&\leq \min_{\substack{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N \\ \mathcal{C}=\{\mathcal{C}_i \in 2^{[N]\setminus\{i\}};\ |\mathcal{C}_i| \leq B_c\}_{i=1}^N \\ \mathcal{C} \in \mathcal{P}}} \left\{ F(\mathbf{w}, \mathcal{C}) \triangleq \sum_{k=1}^N p_k F_k \left(\mathbf{w}_k, \mathcal{C}_k \right) \right\} \\
&= \min_{\mathbf{w}=\{\mathbf{w}_i \in \mathbb{R}^d\}_{i=1}^N} F(\mathbf{w}, \mathcal{C}_\mathbf{c}),
\end{aligned}$$

which concludes the proof.

F EXPLANATION OF THE FRAMEWORK WITH A SIMPLE EXAMPLE

To clarify the formulation of the DPFL algorithm, we consider a simple example with 3 devices (clients) labeled $\{k = 1, 2, 3\}$, each having its own local data. The goal is to optimize a personalized model for each device.

- **Step 1:** Local Optimization and Initial Collaboration Graphs:* Each device performs local updates for a certain number of steps τ_{init} using its own data. After the local update and communication of the updates, each device computes its collaboration graph using BGGC which can result for example in: $\Omega_1 = \{1, 2\}$ (i.e., device 1 will collaborate with devices 2), $\Omega_2 = \{2, 3\}$, $\Omega_3 = \{3, 1\}$.
- **Step 2:** Training Loop
  - **Local Optimization:** Each device performs local updates for τ_{train} steps using its data.
  - **Collaboration Assignment:** After local optimization, each device reevaluates its collaboration graph. For example: Device 1 might decide to collaborate only with itself, $\mathcal{C}_1 = \{1\}$, Device 2 might select $\mathcal{C}_2 = \{2, 3\}$, Device 3 might select $\mathcal{C}_3 = \{3, 1\}$.
  - **Parameter Averaging:** Each device updates its local model by averaging parameters from its collaborators ($\mathcal{C}_i$ for client i). For instance, if device 1 collaborates with device 2, it updates its model as:
$$\mathbf{w}_2 = \frac{1}{p2 + p_3}(p_2\mathbf{w}_2 + p_3\mathbf{w}_3)$$
where p_i is proportional to the size of the data on the device i. This process is repeated for all devices.
- **Step 3:** Repeat for multiple rounds the process of local optimization, collaboration selection, and parameter averaging repeats for T rounds, refining the models and evolving the collaboration graphs.

Greedy Collaboration Selection For the greedy algorithm. Consider the same 3 devices, where device 1 decides whom to collaborate with.

- **Initialization:** Device 1 starts with $\mathcal{X} = \{1\}$ and $\mathcal{Y} = \{1, 2, 3\}$.
- **Marginal Gain Calculation:** Device 1 computes the marginal gain for adding or removing each collaborator. For example, adding device 2 gives a gain of:
$$a = \mathcal{R}(\{1, 2\}) - \mathcal{R}(\{1\}),$$
and removing device 2 gives:
$$b = \mathcal{R}(\{1, 3\}) - \mathcal{R}(\{1, 2, 3\}).$$
- **Selection:** If the gain of adding device 2 is greater than removing it, device 1 will collaborate with device 2. If adding device 3 results in a smaller gain, device 1 will be excluded it from its collaboration set.
- **Final Collaboration Set:** Based on the marginal gains, device 1 will collaborate with device 2, resulting in the final collaboration set $\mathcal{C}_1 = \{1, 2\}$.

G COMMUNICATION AND COMPUTATION COSTS OF DPFL AND COMPARISION WITH OTHER METHODS

G.1 Communication Cost of DPFL

In DPFL, devices communicate with their collaborators, which are selected based on a greedy algorithm to balance the tradeoff between personalization and global model aggregation. The key factors influencing the communication cost are:

- Number of devices (N): The total number of devices participating in the learning process.
- Set of collaborators size $|\mathcal{C}_k|$: Each device k communicates with a subset of collaborating devices $\mathcal{C}_k$, where $|\mathcal{C}_k| \leq B_c$ is the number of devices in the collaboration set.

- Model size (d): The number of model parameters (weights) that each device is updating and exchanging during communication.

- Communication rounds (T): The number of iterations or rounds of communication.

In each round of communication, the cost for device k is proportional to the model size d and the number of collaborators it communicates with. Specifically, if each device communicates with B_c collaborators, the communication cost per round is at most $B_c d$. Over T iterations, the total communication cost becomes at most $TB_c d$, where $B_c << N$ in real-world applications (i.e., each device typically communicates with only a small subset of neighbors). Accounting for the preprocessing step, the total communication cost of our method is O($TB_c d+Nd$). Therefore, for $T > N$, the total communication cost of our method is O($TB_c d$). For decentralized federated learning with a fully connected network (e.g., extended FedAvg), the communication cost scales with the total number of devices N and the model size d, yielding a cost of O(TNd). Thus, DPFL significantly reduces the communication cost compared to extended centralized methods with fully connected networks, as it avoids the need for all-to-all communication across devices.

G.2 Computation Cost of DPFL

The computational cost in DPFL is primarily determined by three factors: the greedy algorithm for neighborhood selection, the local optimization process, and the aggregation of updates. Below, we break down the computation cost for each component:

- **Greedy Collaborators Selection:** In DPFL, each device runs a greedy algorithm to select its collaborators. This step involves computing the set of collaborators for each device, which requires $2 \times B_c \times F$ FLOPs (2 follows from computing a and b as shown in lines 7 and 8 Algorithm 2), where B_c is the size of the neighborhood and F denotes the FLOPs required for a forward pass of the model.

- **Local Optimization:** Each device performs local optimization (e.g., gradient descent) for E steps. The computational cost for each local update involves both a forward and a backward pass of the model, requiring $F+B$ FLOPs per step, where $F+B$ denotes the FLOPs required for a forward and backward pass. Therefore, the total computation for local optimization over E steps is $E \times (F + B)$ FLOPs.

- Aggregation: The aggregation step in DPFL, where model updates are averaged across a device's collaborations, requires $B_c \times d$ FLOPs, where d is the size of the model (i.e., the number of parameters). Thus, the total computational cost for each device in DPFL per round is the sum of these three terms: Total training FLOPs per round (DPFL) $= 2 \times B_c \times F + E \times (F + B) + B_c \times d$

G.3 Comparison with Baseline Methods

- **Decentralized FedAvg:** In decentralized federated learning methods like FedAvg, each device performs local optimization for E steps, requiring $E \times (F + B)$ FLOPs. The aggregation step on each device requires $N \times d$ FLOPs, where N is the total number of devices. Thus, the total computational cost per device in FedAvg is:

  Total FLOPs (FedAvg) per round $= E \times (F + B) + N \times d$ Compared to DPFL, FedAvg does not include the additional greedy neighborhood selection step, but it does require communication with all N devices for aggregation, which increases the computational burden.

- **DITTO and APFL:** These methods (used as baselines in the main table) require training two models per device for personalized federated learning. The computational cost per device in these methods is therefore higher, as each device performs local optimization for two models. Specifically, for E local steps, the total computational cost becomes $2 \times E \times (F + B) + N \times d$ due to the need to train two models and aggregate them. This makes DITTO and APFL more computationally expensive than DPFL.Total FLOPs (DITTO/APFL) $= 2 \times E \times (F + B) + N \times d$

H EXPERIMENTAL DETAILS

All the experiments reported in Tables 3, and 1 represent the average of three different repetitions across three different seeds. For the experiment in Table 4, we report the results for seed equals to 42. In all experiments, we preserve the best model based on the validation dataset, and the reported test results are obtained by performing inference on this best validation model.

H.1 Datasets

In our experiments, we utilize various datasets following existing literature on personalized FL, as highlighted in (Li et al., 2021; Collins et al., 2021; Marfoq et al., 2022). We conduct experiments with CIFAR10 (Krizhevsky et al., 2009), Federated Extended MNIST (FEMNIST) (Caldas et al., 2018), and the CINIC10 dataset (Darlow et al., 2018).

H.2 Data heterogeneity

We explore varying degrees of data heterogeneity, particularly employing two distinct distribution strategies for the CIFAR10 and CINIC10 datasets. The first approach involves a pathological distribution split (Zhang et al., 2023; McMahan et al., 2017; Colin et al., 2016), wherein each client exclusively receives data from three specified categories. The second approach utilizes the Dirichlet distribution (Yurochkin et al., 2019; Wang et al., 2020), where a distribution vector $\mathbf{q}_c$ is drawn from $Dir_k(\alpha)$ for each category c. Subsequently, the proportion $\mathbf{q}_{c,i}$ of data samples from category c is allocated to client i. Moreover, for the FEMNIST dataset, we consider the natural Non-IID split provided in the Leaf framework (Caldas et al., 2018) where each writer corresponds to a client and we add more degrees of heterogeneity by having each client missing some classes.

H.3 CIFAR10 benchmark

H.3.1 Distribution

CIFAR10 is a vision dataset comprising 50,000 training images and 10,000 testing images. We split the training data into 20% of the validation dataset and 80% of the training dataset. Furthermore, we split the testing data among clients in such a way the local test data follows the distribution of the training data. During the training, we save the best local models on the validation dataset, and we make inferences afterward using the local test data and the best saved model. To simulate real-world heterogeneity, we consider two types of data heterogeneity. The first approach involves a pathological distribution split (Zhang et al., 2023; McMahan et al., 2017; Colin et al., 2016), wherein each client exclusively receives data from three specified categories. In the case of the CIFAR10 dataset, we use $Patho(3)$ to denote that each client has access to only three classes out of ten. The second approach utilizes the Dirichlet distribution (Yurochkin et al., 2019; Wang et al., 2020), where a distribution vector $\mathbf{q}_c$ is drawn from $Dir_k(\alpha)$ for each category c. Subsequently, the proportion $\mathbf{q}_{c,i}$ of data samples from category c is allocated to client i. In CIFAR10 dataset we use $Dir(0.1)$.

H.3.2 Model

The employed model is a simple CNN network comprising three convolutional layers and two fully connected layers. The first convolutional layer has three input channels, six output channels, and a kernel size of 5. The ReLU activation function and a 2D Maxpool Layer with a kernel size of 2 follow it. The second convolutional layer transforms an input of six channels to sixteen channels with a kernel size of 5, followed by the ReLU activation function. The first fully connected layer takes an input of size 400 and produces an output of size 120. The second layer produces an output of size 84, and the last layer has a size equal to the number of classes, which is 10.

H.3.3 Hyperparameters

For the preprocessing step for each client, we train 10 local epochs ($\tau_{init} = 10$). During the training the number of local epochs $\tau_{train} = 5$, the number of rounds $T = 100$, however as the preprocessing step corresponds to 2 rounds of training, for fairness with other methods we use only 98 rounds instead of 100 for our method. The

number of clients $|\mathcal{S}_t| = 100$. The learning rate $\eta = 0.01$. For the training, we use SGD optimizer with $1e-3$ decay, 0.9 momentum, and batch size of 16.

H.4 FEMNIST benshmark

H.4.1 Distribution

We utilize the FEMNIST dataset within the LEAF framework (Caldas et al., 2018). This dataset consists of training and testing sets accompanied by a client-data mapping file that partitions the data in a non-IID (non-identically distributed) manner among the clients. The dataset exhibits inherent heterogeneity due to variations in the writing styles of individual contributors. We first downloaded the full FEMNIST data from (Caldas et al., 2018). Additionally, we employed a file obtained from (Li et al., 2021) to further increase the heterogeneity in the dataset. The data files will be provided along with our code.

H.4.2 Model

The employed model is a simple CNN network comprising three convolutional layers and two fully connected layers. The first convolutional layer has one input channel, 4 output channels, and a kernel size of 5. It is followed by the ReLU activation function and a 2D Maxpool Layer with a kernel size of 2. The second convolutional layer transforms an input of 4 channels to 12 channels with a kernel size of 5, followed by the ReLU activation function. The first fully connected layer takes an input of size 192 and produces an output of size 120. The second layer produces an output of size 100, and the last layer has a size equal to the number of classes, which is 10.

H.4.3 Hyperparameters

For the preprocessing step for each client, we train 4 local epochs ($\tau_{init} = 4$). During the training the number of local epochs $\tau_{train} = 2$, the number of rounds $T = 50$, however as the preprocessing step corresponds to 2 rounds of training, for fairness with other methods we use only 48 rounds instead of 50 for our method. The number of clients $|\mathcal{S}_t| = 100$. The learning rate $\eta = 0.001$. For the training, we use SGD optimizer with $1e-3$ decay, 0.9 momentum, and batch size of 10.

H.5 CINIC10 benshmark

H.5.1 Distribution

CINIC10 serves as an extension to CIFAR10, encompassing 90,000 training images, 90,000 validation images, and 90,000 test images. Initially, the training and validation sets are combined, and the data is then distributed among clients in a heterogeneous manner. Subsequently, 20% of each partition is allocated for validation, with the remaining portion designated for training. Similar to CIFAR10, we split the test data among clients in such a way that it follows the train distribution for each client. The training data is distributed either following a Dirichlet distribution $Dir(0.1)$ or the pathological distribution $Patho(3)$, where each client has access to only three classes among the ten available. After assigning to each client which classes it will get, we distribute the class data points among clients that share the same class, following a $Dir(0.5)$ distribution. This additional step adds more degrees of heterogeneity and simulates a real-world scenario more realistically.

H.5.2 Model

The same model used in CIFAR10 is employed.

H.5.3 Hyperparameters

For the preprocessing step for each client, we train 10 local epochs ($\tau_{init} = 10$). During the training the number of local epochs $\tau_{train} = 5$, the number of rounds $T = 50$, however as the preprocessing step corresponds to 2 rounds of training, for fairness with other methods we use only 48 rounds instead of 50 for our method. The number of clients $|\mathcal{S}_t| = 200$. The learning rate $\eta = 0.01$. For the training, we use SGD optimizer with $1e-3$ decay, 0.9 momentum, and batch size of 16.

H.6 Baselines

We compared our method against eleven baselines:

- Local: local training for T rounds and we report the average local test accuracies of the clients in every round.
- FedAvg (McMahan et al., 2017)
- FedAvg+FT: which is fedavg fine-tuning version, where we save best models on validation dataset when training FedAvg, then starting from that model we perform $2 * \tau$ local epochs and report the average test accuracies.
- FedProx (Lian et al., 2018): For FedProx we use $\lambda = 0.1$ for all datastes (CIFAR10, CINIC10 and FEMNIST)
- FedProx+FT (Lian et al., 2018): It is FedProx fine-tuning version, where we save the best models on the validation dataset when training FedProx, then starting from that model we perform $2 * \tau$ local epochs and report the average test accuracies.
- APFL (Smith et al., 2017): We chose the hyperparameter which they call τ in their paper to be $\tau = 1$ which means we synchronise the models every round.
- PerFedAvg (Fallah et al., 2020a): We use the hyperparameter $\alpha = 0.01$ and it is fixed for all runs and all datasets
- Ditto (Li et al., 2021): We set the hyperparameter λ that represents the tradeoff between local and global models to 0.75, and keep it fixed for all datasets.
- FedRep (Collins et al., 2021)
- PACFL (Vahidian et al., 2023): We use the threshold for clustering to be 1, following their optimal value in CIFAR10 and we use it for all other datasets.
- kNN-Per (Marfoq et al., 2022): we set the hyperparameter k_knn to 10 and the interpolation hyperparameter to 0.5
- pFedGraph (Ye et al., 2023)

H.7 Compute resources

We use an internal SLURM cluster for running our experiments. The experiments were done on an ASUS ESC N4A-E11 server. The node has 4 A100 GPUs, an AMD EPYC 7003 series 64 core @ 3.5GHz CPU and 512GB of RAM. We used one A100, with 2 cores, and required at most 100GB of memory for the experiments.

I ADDITIONAL EXPERIMENTS

I.1 Variance between accuracies of local models

As our objective is personalization, we believe that, apart from improving the overall average accuracy among clients, it is crucial to assess whether improvements are distributed across most models rather than being confined to just a few clients. To evaluate this, we examine the variance between local models. Lower variance indicates that the accuracies of local models are closer. Therefore, we consider both accuracy and variance as important metrics. In Figures 1, 5 and 6, the x-axis represents the average test accuracy, while the y-axis represents the variance between clients' models. For better visualization, in Fig. 1 we choose to report only methods that the average test accuracy is higher than 80% and 78% for $Dir(0.1)$ and $Patho(3)$ respectively, while for Fig. 6 we show the methods having higher than 78% and 77% for $Dir(0.1)$ and $Patho(3)$ respectively. Furthermore, in Fig. 5 we were able to show all methods. All figures show that our method (DPFL) is situated in the right-bottom corner across all variants of budget constraints. This positioning signifies that, compared to other methods, our approach achieves superior average test accuracy and lower variance between local models.

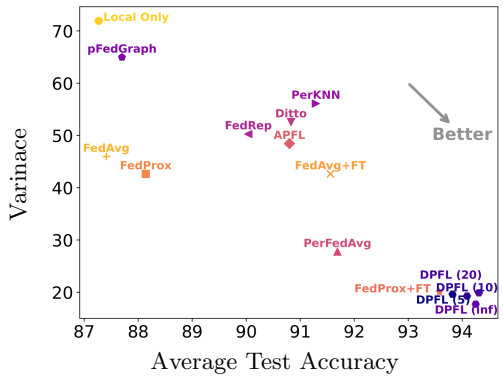


Figure 5: Comparison of our method with other personalized methods on the FEMNIST dataset in terms of variance between local models.

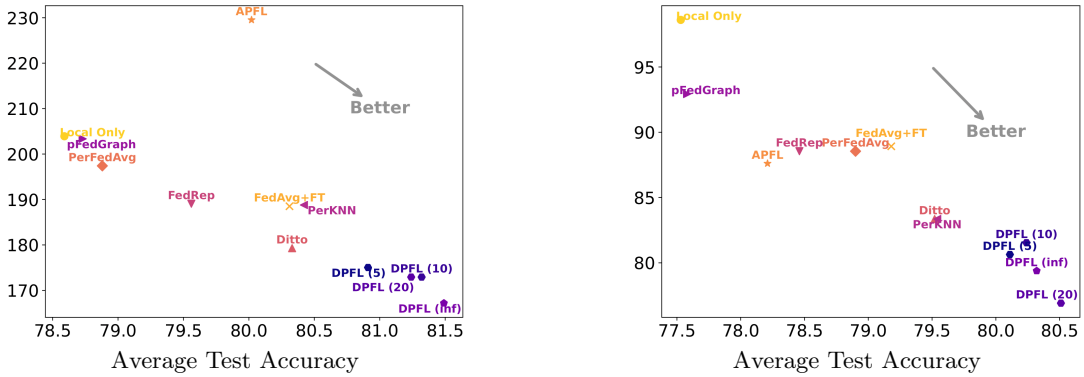


Figure 6: Comparison of our method with other personalized methods on the CINIC10 dataset in terms of variance between local models.

I.2 Visualization of the collaboration graph

We present Figures 8, 9, 10 and 11, showing our initial collaboration graph on the top left and the clients considered for collaboration every 10 round from 0 to 80 for cases without budget constraint, with $B_c = 20$, $B_c = 10$, and with $B_c = 5$ respectively. The diagonal indicates that every client always "collaborates" with itself. To illustrate the graph evolution, the plots display collaborative links in two colors: in pink are the clients that are selected for collaboration in that round; in yellow are the clients that were identified during the preprocessing step but are currently not chosen. The union of pink and yellow clients corresponds to the initial collaboration graph. The figures highlight that the initial collaboration graph is denser compared to the actually used clients for aggregation in later rounds. This is expected since the graph is constructed as a preprocessing step, and at this stage, model weights have not yet converged. Therefore, broader collaboration can be beneficial. However, as training progresses, it is natural to expect that each client will benefit primarily from collaborating with clients having similar data distributions, thus leading to a sparser collaboration graph. Another contributing factor is that, in the preprocessing step, the decision to select a specific client for collaboration is made from a pool of 100 clients, making it more challenging than in later rounds where the decision is drawn from a smaller pool denoted as Ω_k for client k. Another observation is that in all cases, from round 60 onward, the graph remains almost unchanged. This is attributed to the weights nearing convergence, resulting in a less random selection of collaborators. The diversity of the graph at early rounds proves that we shouldn't remove an edge between two nodes if that node hasn't been selected, as that decision could change in later rounds.

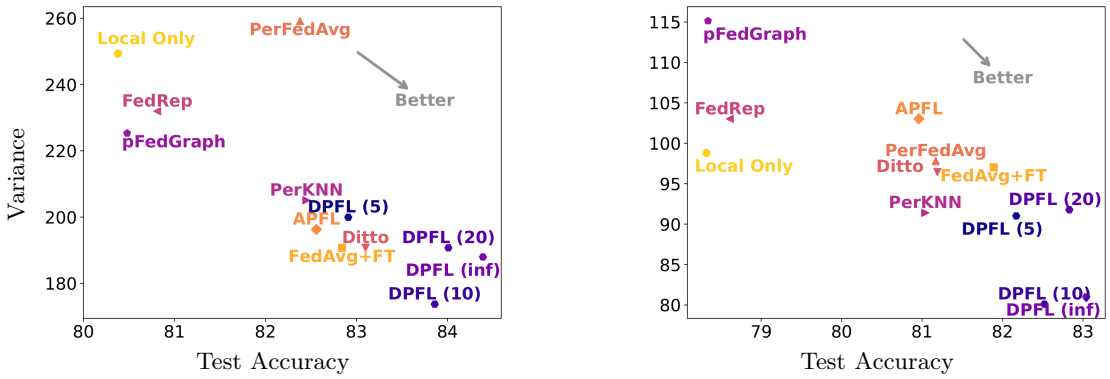


Figure 7: Variance between local models using Dir(0.1) (left) and Patho(3) (right) data splits on CIFAR10.

I.3 Behavior of the collaboration graph under two groups of clients

To better visualize and have more explainability of the behavior of our algorithm, we ran an experiment using CIFAR10 dataset and 100 clients, where 40 among them had their label flipped using the same permutation and 60 had their true labels. It's important to note that our objective wasn't to create an attack; rather, we aimed to delineate the behavior of our algorithm, acknowledging that while it may exhibit robustness characteristics, the study of robustness falls outside the scope of this paper. In this experiment, we effectively created two distinct groups of clients. The first experiment, illustrated in Fig. 3a, involved flipped clients who did not execute the greedy algorithm (Algorithm 2). Instead, they consistently maintained their models (a stronger attack strategy). As expected, the collaboration graph displayed numerous edges with these flipped clients initially. This outcome aligns with the inherent randomness in the weights during the preprocessing step, making it challenging to select between clients. However, as the rounds progressed, we observed a notable trend: black clients increasingly avoided selecting the red ones until they ultimately ceased choosing them altogether. This evolution is depicted in Fig. 12, where the complete graph evolution is visualized every 10 rounds. In the second experiment (Fig. 3b), even the red clients executed the greedy algorithm, resulting in their selection from the black clients initially. Consequently, their models became regularized towards the black ones. Despite this behavior in the collaboration graph, we observed that as the rounds progressed, the clients were almost segregated into two subgroups (red and black), with very few links between them serving as a form of regularization (full graph is in Fig. 13). We visualized the percentage of connection with malicious clients for a benign client, as presented in Fig. 14, and found it to be very small.

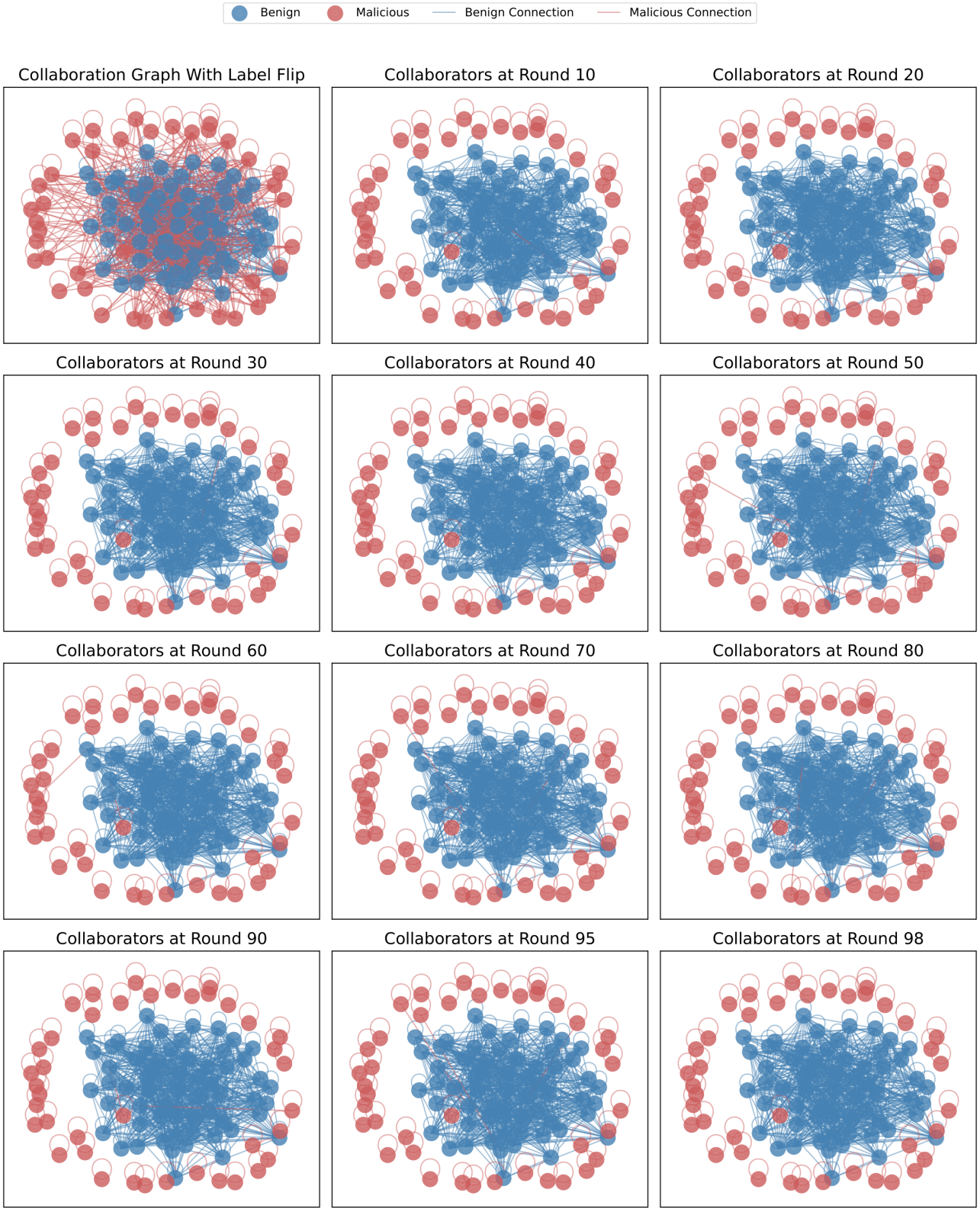


Figure 12: Our collaboration graph without Greedy. 40% of clients have flipped labels (Malicious), while 60% have original labels (Benign). Malicious clients don't execute Algorithm 2; instead, they consistently send their local model to Benign clients.

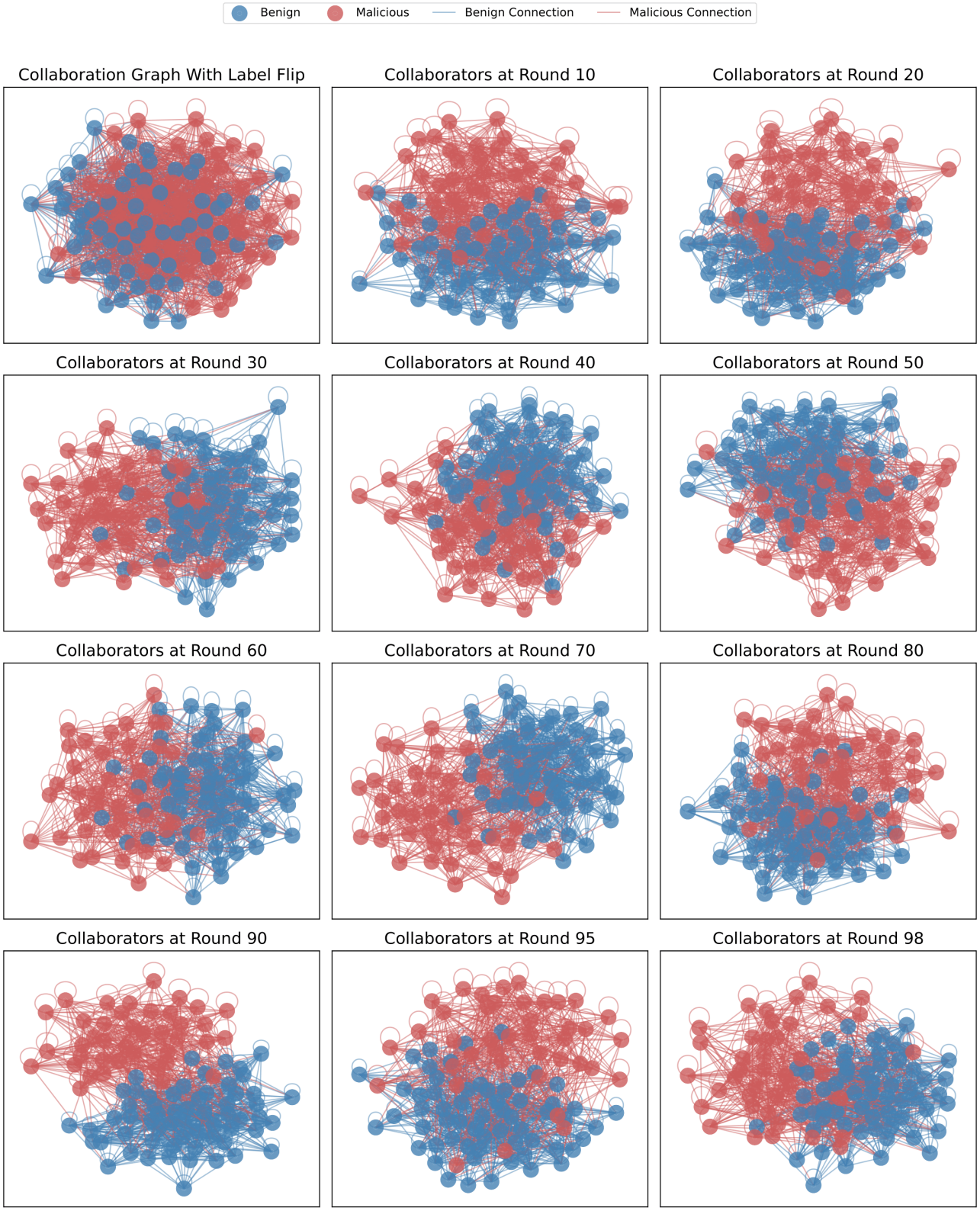


Figure 13: Our collaboration graph with Greedy. 40% of clients have flipped labels (Malicious), while 60% have original labels (Benign). Malicious execute Algorithm 2.

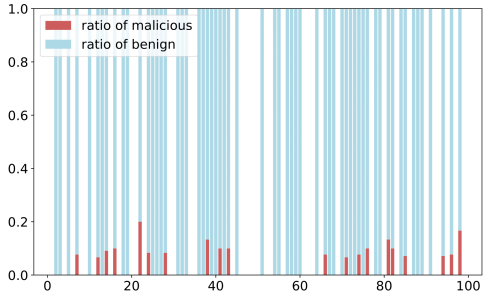


Figure 14: Visualization of the ratio of benign clients collaborating with malicious vs with benign, in the case that malicious runs the greedy algorithm

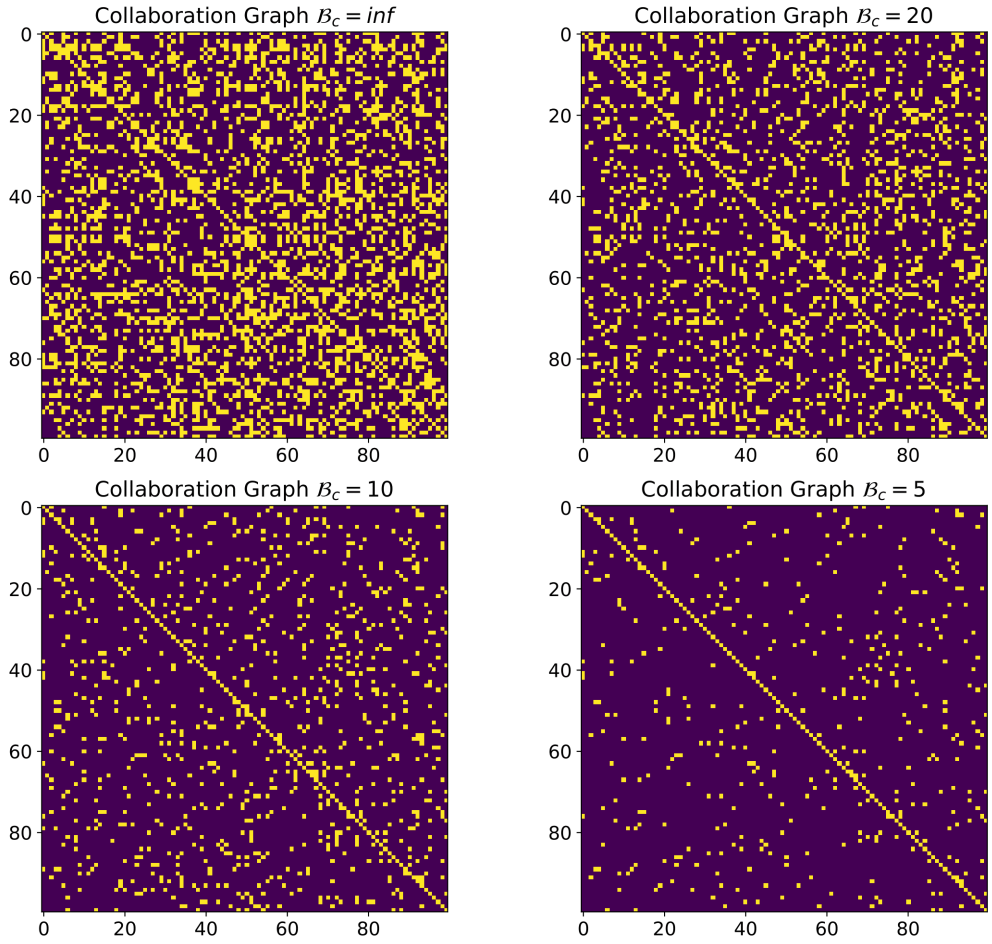


Figure 15: Our collaboration graph on CIFAR10 dataset with 100 clients

I.4 Measuring Asymmetry of the collaboration graph

For the experiments conducted using the CIFAR10 dataset, we analyzed the percentage of asymmetry in the graph across different rounds for three budget settings $\mathcal{B}_c = inf$, $\mathcal{B}_c = 20$ and $\mathcal{B}_c = 10$ respectively.We observed that the asymmetry percentage is consistently higher in round 0 compared to subsequent rounds. This aligns with the expectation that the greedy decision-making process during preprocessing exhibits more randomness due to two factors, first, the model weights still didn't capture enough of the data structure, and second as the decision of choosing or removing a client is made from bigger pool, it makes it harder to decide. Additionally, across all budget settings, the asymmetry percentage appears to stabilize around a specific value, exhibiting fluctuations, starting from round 40 onwards.

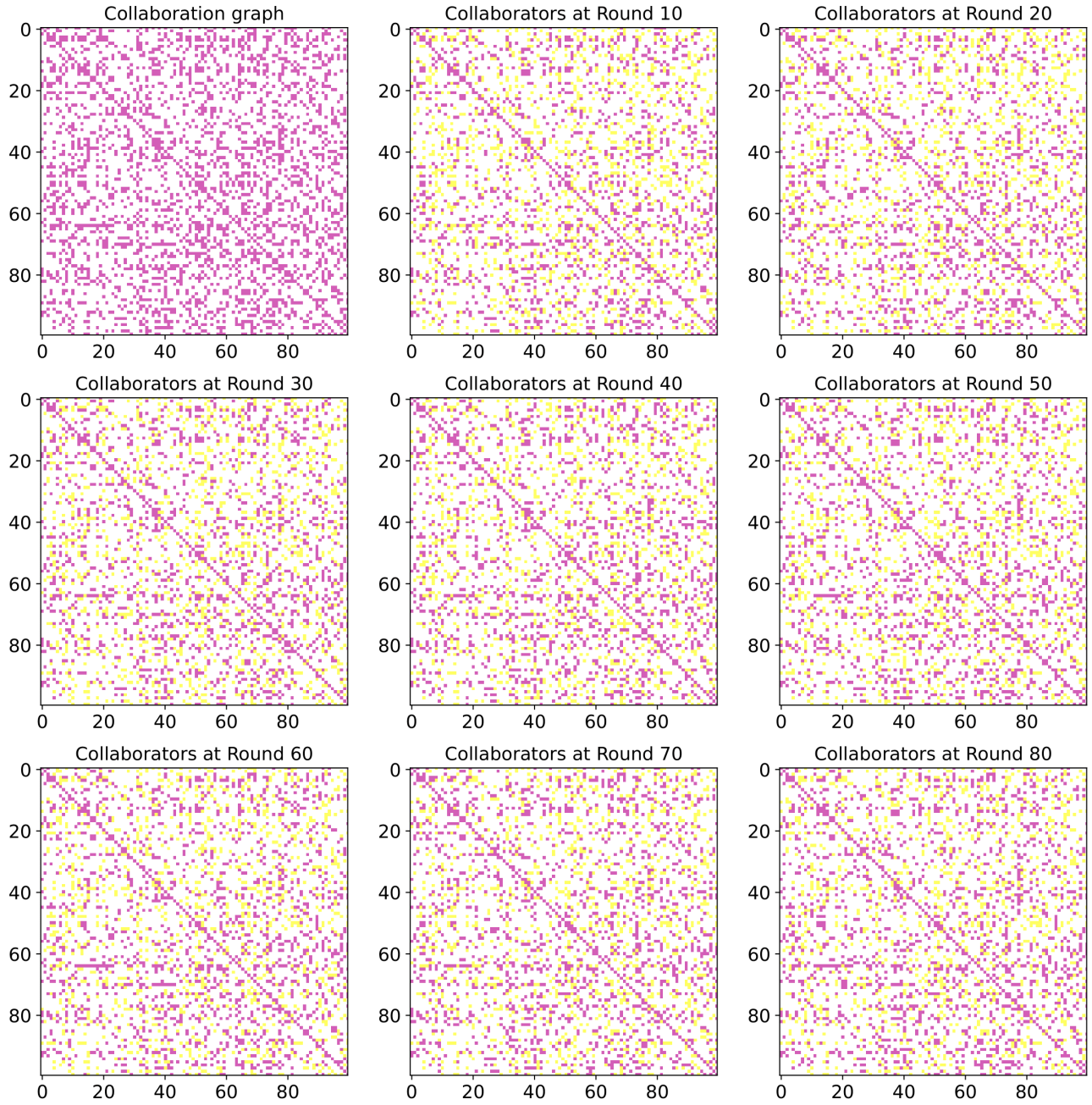


Figure 8: Our collaboration graph without constraint on CIFAR10 dataset with 100 clients

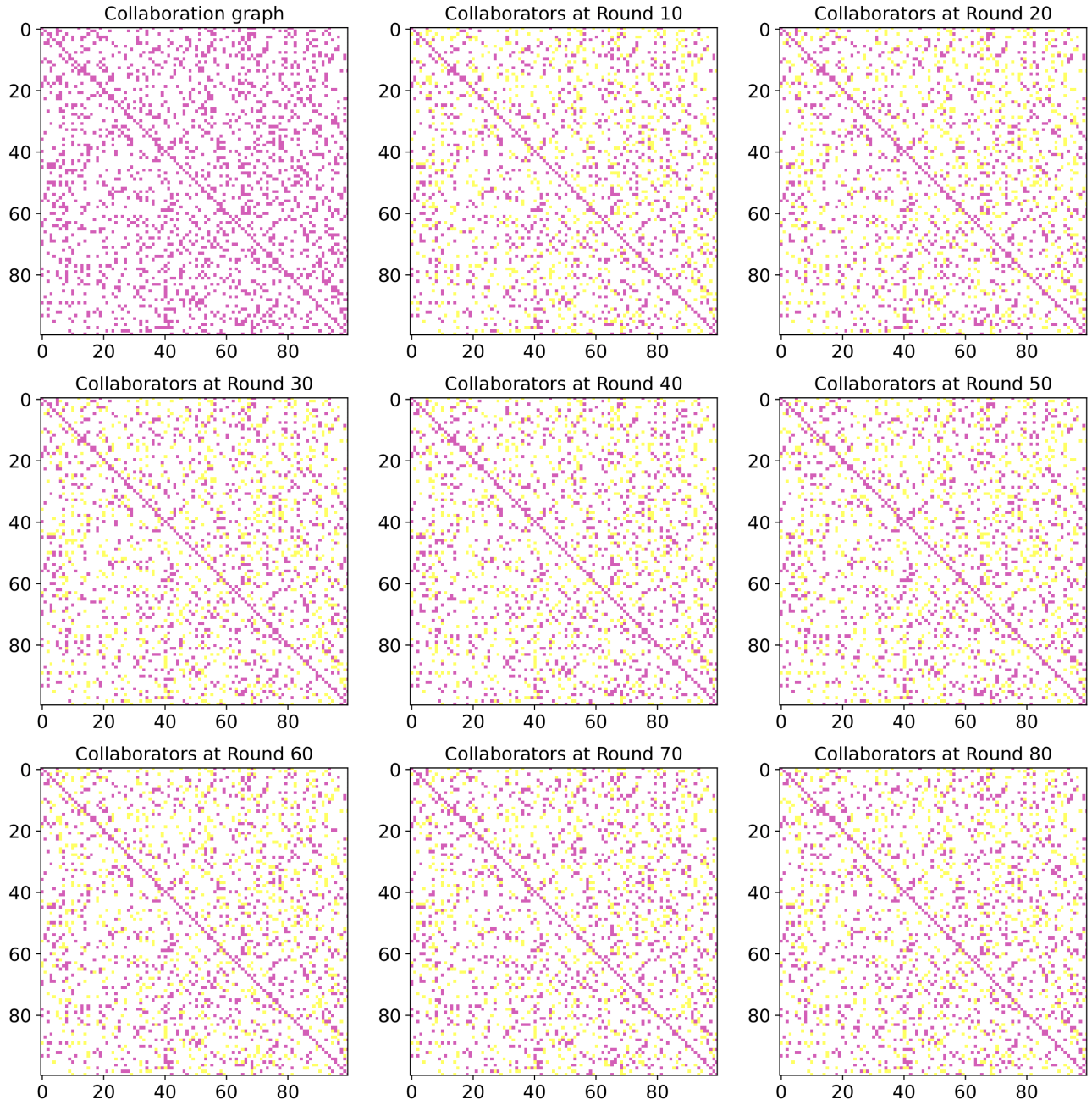


Figure 9: Our collaboration graph with constraint B_c=20 on CIFAR10 dataset with 100 clients.

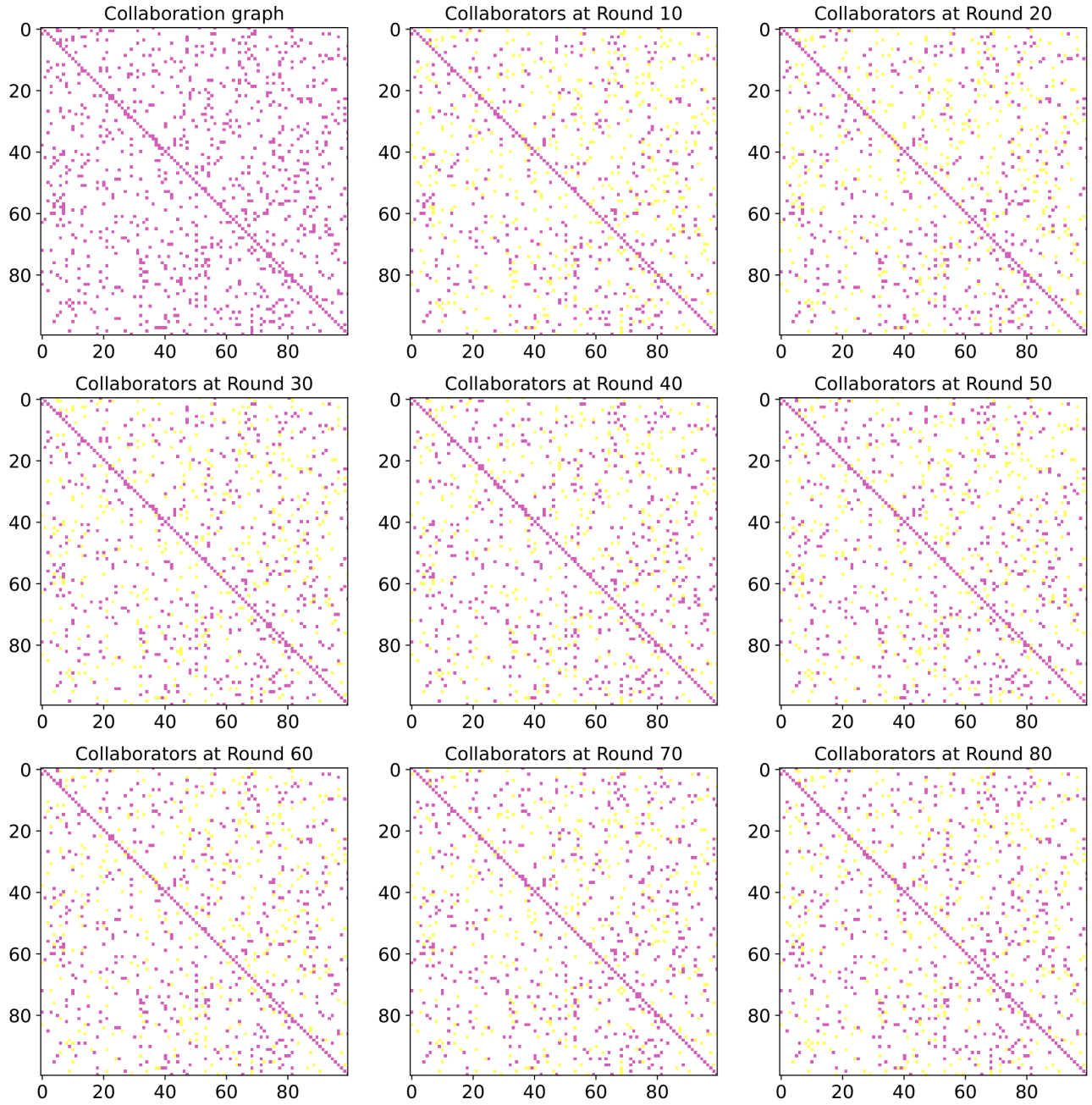


Figure 10: Our collaboration graph with constraint B_c=10 on CIFAR10 dataset with 100 clients.

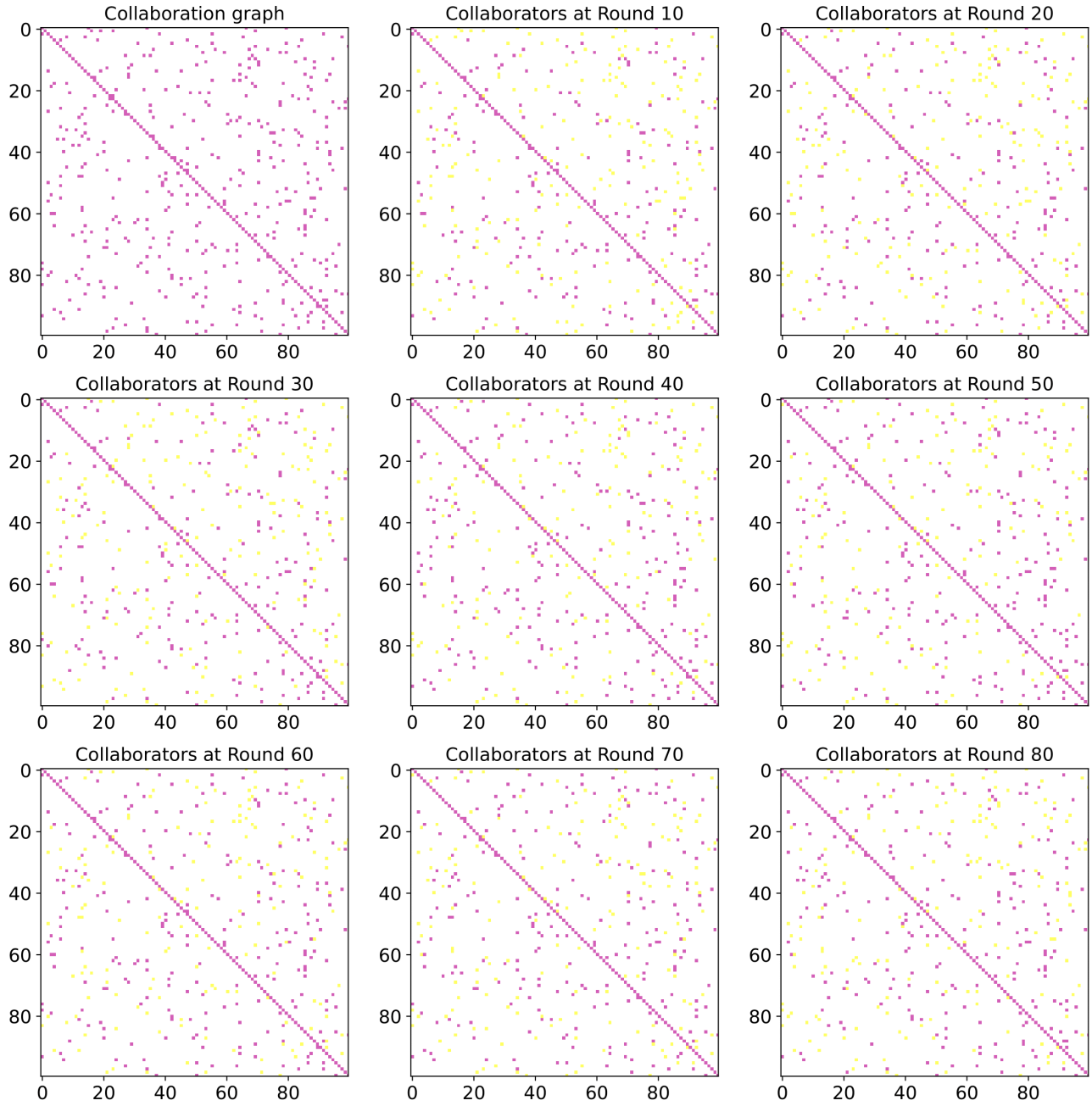


Figure 11: Our collaboration graph with constraint B_c=5 on CIFAR10 dataset with 100 clients.