
PEPPER: EMPOWERING USER-CENTRIC RECOMMENDER SYSTEMS OVER GOSSIP LEARNING

A PREPRINT

✉ **Yacine Belal**
 INSA Lyon
 LIRIS
 Lyon, France
 yacine.belal@insa-lyon.fr

✉ **Aurélien Bellet**
 Univ. Lille
 INRIA
 CNRS
 Lille, France
 aurelien.bellet@inria.fr

✉ **Sonia Ben Mokhtar**
 INSA Lyon
 LIRIS
 CNRS
 Lyon, France
 sonia.benmokhtar@insa-lyon.fr

✉ **Vlad Nitu**
 INSA Lyon
 LIRIS
 CNRS
 Lyon, France
 vlad.nitu@cnrs.fr

August 11, 2022

ABSTRACT

Recommender systems are proving to be an invaluable tool for extracting user-relevant content helping users in their daily activities (e.g., finding relevant places to visit, content to consume, items to purchase). However, to be effective, these systems need to collect and analyze large volumes of personal data (e.g., location check-ins, movie ratings, click rates .. etc.), which exposes users to numerous privacy threats. In this context, recommender systems based on Federated Learning (FL) appear to be a promising solution for enforcing privacy as they compute accurate recommendations while keeping personal data on the users’ devices. However, FL, and therefore FL-based recommender systems, rely on a central server that can experience scalability issues besides being vulnerable to attacks. To remedy this, we propose PEPPER, a decentralized recommender system based on gossip learning principles. In PEPPER, users gossip model updates and aggregate them asynchronously. At the heart of PEPPER reside two key components: a personalized peer-sampling protocol that keeps in the neighborhood of each node, a proportion of nodes that have similar interests to the former and a simple yet effective model aggregation function that builds a model that is better suited to each user. Through experiments on three real datasets implementing two use cases: a location check-in recommendation and a movie recommendation, we demonstrate that our solution converges up to 42% faster than with other decentralized solutions providing up to 9% improvement on average performance metric such as hit ratio and up to 21% improvement on long tail performance compared to decentralized competitors.

Keywords Decentralized Federated Learning, Gossip Learning, model aggregation, Point-of-Interest recommendation, recommender systems

1 Introduction

Recommender systems are at the heart of many popular online services used by billions of users on a daily basis to get insights on what could be a good restaurant to eat in, a nice museum to visit or the next TV series to watch. For instance, around 53 million US users perform smartphone searches on a daily basis about local businesses/services

in their immediate surroundings and visit these places [8]. To be effective, recommender systems generally require the collection of large corpuses of personal data (e.g., check-ins, clicks, ratings) and substantial computing power in order to be trained. Therefore, they are traditionally centralized and often hosted in the data center of the service provider. However, this approach poses serious privacy concerns due to the never-ending list of attacks and privacy scandals [9, 57, 65], which continue to unfold. These scandals generally harm the image of the service provider and cause strong reactions in the public opinion. Furthermore, surveyed users generally consider that (centralized) recommender systems violate their privacy [44] and would prefer not to be profiled [2].

In the past decade, there have been various solutions investigating privacy-preserving recommender systems [4, 17, 21, 42, 56, 63]. Among these solutions, Federated Recommender Systems, i.e., recommender systems leveraging the Federated Learning principles (FL), are considered promising solutions [22, 45, 54, 64] as they provide privacy-by-design guarantees. Indeed, instead of centralizing users' personal data and training recommendation models on remote cloud infrastructures, FL proposes to train the models right where the data is generated, i.e., on the edge / mobile devices. However, this solution has its own drawbacks due to the inherent limitations of FL, one such limitation being its centralized master-slave architecture. Even if FL allows users to keep their data local, they still send ML model updates to a (logically) central server, which drives the model convergence process. This centralized server poses not only fault tolerance but also scalability problems since it restricts Federated Recommender Systems only to those companies which are able to scale the centralized server up to the necessary number of clients [39]. To address this issue, we investigate in this paper decentralized recommender systems over Gossip Learning [28, 30]. In Gossip Learning, nodes exchange model updates with their neighbors and asynchronously aggregate the received models using a model aggregation function (e.g., federated averaging [41]). There exist preliminary solutions for decentralized recommender systems over Gossip Learning [28]. However, these solutions suffer from poor average, and even worse tail performance compared to their centralized counterpart. Said differently, existing solutions tend to improve the average user satisfaction at the cost of having extremely unsatisfied users at the long and short tail.

To address this issue, we present PEPPER, a novel solution that aims at maximizing average user performance without penalizing users at the tail. PEPPER is composed of a performance-based model aggregation function and a personalized peer-sampling protocol. While the former compares the received models with respect to their performance on a local test set and gives more weight to those models that perform better from the point of view of each user, the latter keeps a proportion of seemingly similar neighbors in the view of each user, i.e., neighbors that previously sent models, which performed well for this user. We evaluate PEPPER with extensive simulations involving up to one thousand nodes and by relying on two different machine learning models trained on three real world datasets: Foursquare-NYC, Gowalla-NYC [37] and MovieLens ML-100k [25] and compare its performance against six state-of-the-art federated and decentralized solutions. Our results show that PEPPER systematically improves the average performance of decentralized recommender systems (up to 9%, 6% and 13% improvement on performance metrics such as Hit Ratio, NDCG, F1 score, respectively) and in some cases outperforms the average performance of its centralized counterparts. In addition to improving average performance, PEPPER substantially improves long tail performance compared to both federated and decentralized competitors. Finally, thanks to its personalized peer-sampling protocol, PEPPER converges up to 42% faster than other decentralized competitors.

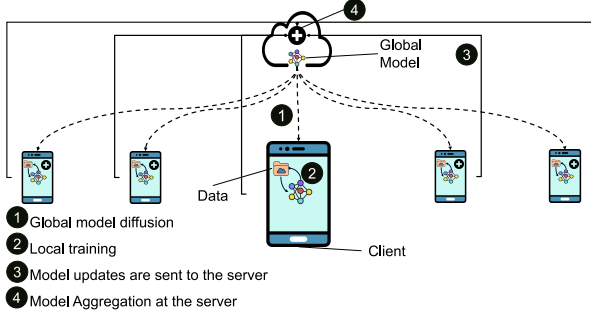
The rest of the paper is organised as follows. In Section 2, we present a background on Federated Learning, Gossip Learning and the recommendation models we rely on in this work. In Sections 3 and 4, we present an overview of PEPPER and its detailed description. We then present the performance evaluation of PEPPER in Section 5 and discuss its limitations and possible mitigations in Section 6. Finally, we present the related research works in Section 7 and conclude the paper in Section 8.

2 Background

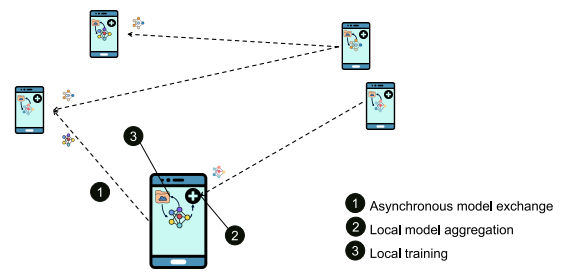
We present in this section preliminary background on key concepts related to PEPPER, namely Federated Learning (Section 2.1), Gossip Learning (Section 2.2) and the used recommendation models (Section 2.3).

2.1 Federated Learning

Federated Learning (FL), depicted in Figure 1a, is a computing paradigm that instantiates the "bring code close to the data" principle in the context of Machine Learning. It was first proposed by Google [41] to allow hundreds of millions of mobile devices (clients) to collaboratively train and build a global model while preserving the privacy of their data. To this end, FL assumes the presence of a coordinator (i.e., a logically centralized server) and N clients C_i , where $i \in [1, N]$. Each client C_i creates and stores data locally, say D_i . Given a learning task, let us denote by M_{global} the global model held by the coordinator and by M_i the model held by each client C_i . Following an initialization



(a) Federated Learning Architecture [41]



(b) Gossip Learning Architecture

Figure 1: Federated Learning vs Gossip Learning Architectures.

phase where the coordinator communicates the model architecture and the optimization algorithm to clients, the former continuously initiates training rounds until the convergence of the global model is considered as satisfactory. Each round is conducted as follows: (i) the coordinator randomly selects a fraction p of clients of the original client set and sends them the current state of the global model M_{global} (see step 1 in Figure 1a); (ii) each client C_i initializes its model M_i with the model parameters received from the coordinator. M_i is then trained on the local dataset D_i before being sent back to the coordinator (see steps 2 and 3 in Figure 1a); (iii) the coordinator aggregates the different models' parameters received and updates the global model (see step 4 in Figure 1a). A classical way of aggregating models in FL is by using the *FedAvg* [41] algorithm, which relies on Formula 1. *FedAvg* computes a weighted average of the received models, parameter wise. Specifically, each model is weighted by the number of samples it was trained on, that is, the dataset size of the owner of the model (*cf.*, equation (1)).

$$M_{global} = \frac{1}{\sum_{i=1}^p |D_i|} \sum_{i=1}^p |D_i| M_i \quad (1)$$

While it enhances data privacy, FL is known to have convergence properties that slightly deviate from classical centralized ML [66]. Additionally, the FL architecture heavily relies on the presence of a central server, which plays a key role in the orchestration of the learning process, both at the start of each round and during the aggregation phase. The necessity of this server has been often criticized as it makes FL vulnerable to failures, scalability issues [33] and attacks [58, 62]. This has quite naturally created within research works a tendency towards approaches that relax the single server assumption. In the next section, we present a gossip-based approach that we adopted in this work.

2.2 Gossip Learning

Gossip Learning is an asynchronous learning protocol that was first proposed by Ormándi et al. [48] to solve the problem of learning over fully distributed data using a peer-to-peer communication protocol called gossip. Its flexibility, privacy-preserving nature and scalability properties give it the potential to be a first-class citizen in the field of distributed machine learning. Gossip Learning relies on a key protocol called *peer-sampling* that provides every node with a set of peers, called a view, to gossip with. The view of each node is periodically shuffled so that nodes have the opportunity to interact with new nodes in the system. This protocol plays a crucial role in the dissemination speed of messages. In gossip learning, it can even impact the performance of the learned models as further illustrated in Section 5. Indeed, the peer-sampling protocol enables each node to periodically change partially or totally its view. This change can be done randomly, creating an unstructured network. It can also follow a well defined logic that can speed-up the convergence time, reduce the communication overhead or improve the performance of the overall system [3].

In a classical Gossip Learning algorithm, a node periodically sends its local model to its view P . Then, upon the reception of a model m from one of its neighbors, it aggregates this model according to a predefined *model aggregation function*. Later on, the obtained model is updated by performing a local training using a local dataset D . The resulting model is thereafter considered as the new current model. In the above algorithm, the model aggregation function is a key component that makes it possible for nodes to learn from others nodes' data without having to actually train on it. This makes it one of the most crucial components in Gossip Learning. Model aggregation functions generally perform a weighted averaging of the models' parameters or a subset of these parameters. Algorithm 1 illustrates two common examples of model aggregation functions, namely Decentralized FedAvg [38] and Model-Aged-Based [30]. While the

former applies the FedAvg algorithm defined in FL to pairs of models, the latter is more advanced as it is based on a notion of model age, which reflects how much a model has circulated in the network. The intuition behind it is to give more weight to older models as they are likely to hold more knowledge.

Algorithm 1: Aggregation Function : Implementations

Data: train data sizes D_1, D_2 ; models to be aggregated : m_1, m_2 ; models' number of updated times : age_1, age_2 ;

```

1 Procedure DECENTRALIZED FEDAVG( $m_1, m_2$ ):
2   return  $\frac{1}{|D_1|+|D_2|}(|D_1| \times m_1) + (|D_2| \times m_2)$ 
3
4 Procedure MODELAGEBASED( $m_1, m_2$ ):
5   return  $\frac{1}{|age_1|+|age_2|}(|age_1| \times m_1) + (|age_2| \times m_2)$ 
6
    
```

2.3 The Recommendation Models

While PEPPER is agnostic to the underlying machine learning model, we use two models in our experiments: the Generalized Matrix Factorization (GMF) model proposed by He et al. [27] and the Personalized Ranking Metric Embedding Method (PRME-G) proposed by Feng et al. [15]. This choice is motivated by two criteria: (i) the effectiveness of these models, which have a sufficiently accurate performance to allow us to quantify the contribution of our system and compare it with different competitors and (ii) their lightness. Indeed, compared to other benchmark recommendation models, GMF and PRME-G are light enough to be considered in a Gossip Learning context where nodes can have constrained resources. Concretely, GMF [27] is a neural network inspired by matrix factorisation (MF). As input, this model takes both user and item identifiers. It is followed by an embedding layer that projects them into a latent vector space. These vectors are equivalent to the latent feature vectors of MF. A multiplication layer does a point-wise product on these two vectors, to combine the features describing the item and those describing the user. The product is then fed to at least one fully connected layer before passing through a sigmoid activation function that outputs the degree of relevance of the item to the user. By comparing this output with the actual relevance of the item, GMF computes an error function (*i.e.*, binary cross entropy) and optimizes the embedding features, until it finds features that best describe the user's preferences and the item's characteristics.

As opposed to GMF, PRME-G [15] is specific to the point-of-interest recommendation task. By using the metric embedding method, it puts locations in a latent space and minimizes the Euclidean distance between each pair of locations (x, y) the more likely is y to be visited after x . Similarly, it minimizes in another latent space the distance between users and locations, based on their preferences. Finally, it incorporates spatial influence by giving more weight to closer POIs. These three pieces of information are combined into a three dimensional matrix D where each cell (u, l^s, l_i) is the likelihood of user u visiting location l_i starting from location l^s . To optimize this matrix, for each training observation (u, l^s, l_i) a random POI l_j that is not observed for (u, l^s) is generated and the following update rule is executed:

$$\Theta \leftarrow \Theta + \gamma \frac{\partial}{\partial \Theta} (\log(\sigma(z))) - \lambda \|\Theta\|^2 \quad (2)$$

where $z = D(u, l^s, l_j) - D(u, l^s, l_i)$ and γ, σ and Θ are respectively the learning rate, the sigmoid function, and the matrices representing locations' and/or users' latent representations.

3 PEPPER in a Nutshell

In this section, we start by defining the objectives of PEPPER (Section 3.1) and our assumptions (Section 3.2) before presenting an overview of our system (Section 3.3).

3.1 PEPPER Objectives

Existing decentralized solutions have a centralized perspective of the learning process. Indeed, they aim at building N models that best approximate the global data distribution, that is, N models converging to a similar minimum as a global FL model. They do so by aggregating models generically and evaluating them on global metrics, which quantify the ability of models to perform well for any given user. While building such models seems to be the natural way to go for decentralizing FL, it may lead to poor performance because of the following reasons: (i) users can have different

distributions which can lead to a phenomenon called client drift [33] that can hurt convergence and (ii) targeting the same minima in a decentralized system is costly and requires aggregating a large number of models, hence, a lot of communication messages. In this work, we argue that these limitations can be circumvented in use cases where approximating a global distribution is not necessary to achieve user satisfaction. For instance, in a recommendation system, users are mainly interested in having a model that recommends items which are highly relevant to them and may not care much about the model’s ability to recommend items for other users. This can be achieved by approximating a local yet more relevant distribution for each model. In this context, the main objective of PEPPER is to train N decentralized recommendation models that are each tailored to their respective user. More specifically, we target a better average personalized-data performance, which is described in [55] as the ability of a model to derive the same decisions as specified by the individual, even if these decisions do not hold with respect to the performance metrics evaluated on a global dataset. As a positive side effect, this can also improve the performance of tail users, that are often neglected in a global metric optimization approach.

3.2 PEPPER Assumptions

In PEPPER, we make several assumptions, some of them being common in decentralized ML systems. First, we are only concerned with the performance aspects of gossip-based recommender systems. Therefore, we do not quantify its robustness to attacks nor do we consider the presence of any malicious or curious users. All users are assumed to be honest clients aiming to train a model that has the best possible local performance. We assume that nodes have heterogeneous bandwidth connectivity and may experience networking delays. However, we assume reliable communication links (i.e., if a message is lost it is retransmitted sufficiently enough to be received) and a stable participation of nodes in the system (i.e., no churn). Then, as commonly considered in decentralized ML systems, we also assume the presence of a central bootstrap service in charge of broadcasting to all nodes the optimization algorithm, the model architecture and the hyper-parameters. However, this only occurs at the beginning of the learning process.

3.3 PEPPER Overview

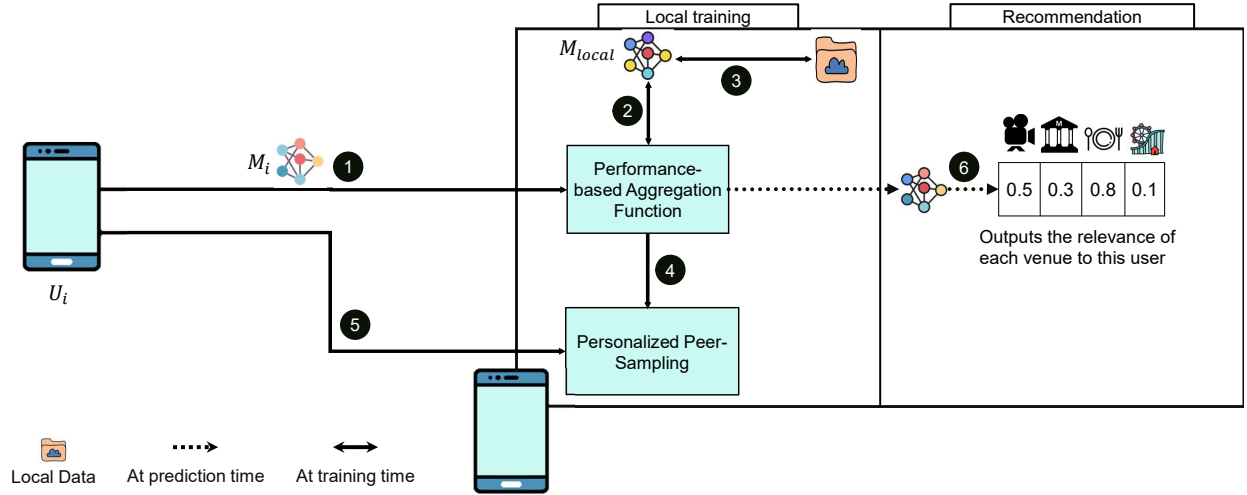


Figure 2: PEPPER System Main Components.

Figure 2 presents an overview of PEPPER, which operates in two main phases: a training phase and a recommendation phase. The training phase makes use of two main components depicted in the left part of the figure: a performance-based model aggregation function and a personalized peer-sampling protocol. Upon receiving a model, say M_i from one of its neighbors, say U_i (step ①), the performance-based model aggregation function assesses the local performance of M_i and aggregates the latter with its latest version of the local model M_{local} (step ②). Periodically, this model is further trained with new local data (e.g., new ratings) as depicted in step ③ of the figure. The information about the performance of M_i is not only used to adjust the way the latter is aggregated with M_{local} , it is also sent to the personalized peer-sampling protocol (step ④). The objective of this protocol is to periodically update nodes’ views (step ⑤). At the very beginning, view updates are performed randomly to leverage previous theoretical and empirical works [7, 32] on how to ensure a logarithmic time dissemination of models. Later, when the received models start to exhibit good local performance, the personalized peer-sampling service remembers good neighbors and uses this information to improve the quality of the view. In PEPPER, both the performance-based model aggregation function

and the personalized peer sampling protocol are totally decentralized protocols that do not rely on any central entity in the system.

4 PEPPER Detailed Description

In this section, we present the two key components of PEPPER, *i.e.*, the personalized peer-sampling protocol and the performance-based model aggregation function.

4.1 Performance-based Aggregation Function

Algorithm 2: Performance-based Aggregation Function

Data: Local Dataset D_i , currentModel M_i , $WeightingSize$, $TestSize$

```

1 Procedure INIT:
2    $D_i^{Weighting} = RandomlySamplesSet(D_i, WeightingSize)$            ▷ Randomly samples the weighting set
3    $D_i^{test} = RandomlySamplesSet(D_i \setminus D_i^{Weighting}, TestSize)$        ▷ Randomly samples the test set
4    $D_i^{train} = D_i \setminus (D_i^{Weighting} \cup D_i^{test})$ 
5
6 Procedure PERFORMANCEBASEDAGGREGATIONFUNCTION( $M_x$ ):
7    $P_x = PredictAndEvaluate(M_x, D_i^{Weighting})$            ▷ Make prediction with  $M_x$  and evaluate its performance
8    $P_i = PredictAndEvaluate(M_i, D_i^{Weighting})$            ▷ Make prediction with  $M_i$  and evaluate its performance
9    $M_i = \frac{1}{P_i + P_x}(P_i \times M_i + P_x \times M_x)$            ▷ Aggregate  $M_i$  and  $M_x$  w.r.t their performance
10   $Update(M_i, D_i^{train})$                                    ▷ Train the new  $M_i$  on local data
11
```

As opposed to existing model aggregation functions, which aim at optimizing global performance, we propose a method that aims at maximizing the local performance of nodes. By doing so, nodes are more likely to obtain satisfactory recommendations, which can improve both average and tail performance. To reach this objective, a node needs to (i) assess the quality of a model received from one of its neighbors and (ii) use this information during the model aggregation phase.

Let us consider the gossip learning setup described in section 2, where each node C_i has its own local data D_i and maintains a local model, say M_i . In order to assess the quality of received models, C_i uses a random subset $D_i^{Weighting}$ taken from its local data set D_i , which we refer to as the weighting set of C_i (Algorithm 2, line 2). $D_i^{Weighting}$ can be seen as a validation set so it cannot be used for model training. However, in scenarios where the data points are produced in a continuous stream, $D_i^{Weighting}$ can be refreshed with new data points while some older points from $D_i^{Weighting}$ can be appended to the training set. Upon reception of a model M_x from one of its neighbors C_x , C_i evaluates it on $D_i^{Weighting}$. More specifically, M_x is required to provide C_i with a top-K recommendation list for each item $\in D_i^{Weighting}$ among 100 random items (see Section 5.3). By quantifying how high items of $D_i^{Weighting}$ are ranked, a measure of performance of M_x , say P_x is obtained (Algorithm 2, line 7). Intuitively, this measure quantifies how similar the data distribution M_x was trained on, is to the distribution of D_i . In the context of recommender systems, a node's best-received models are more likely to come from nodes who have similar tastes. Then, in order to use this information during the model aggregation phase, the node proceeds by computing the performance of its local model M_i on $D_i^{Weighting}$, say P_i (Algorithm 2, line 8). This will allow C_i to compare the two models and assess the potential contribution of each. As illustrated in line 9 of Algorithm 2, P_i and P_x are used as weights by the model aggregation function. By doing so, nodes give more substance to the models that perform better on their data. This pairwise aggregation principle is classical in gossip learning works [28, 30] where nodes can have substantially different model pushing periods, that is, a node rarely receives multiple models at the same time. Therefore, creating synchronicity in the network by waiting for a set of models can hinder convergence. Hence, by following the pairwise principle, PEPPER reduces the synchronicity, enabling models to be aggregated as soon as they are received.

Following this, a traditional model update is done on the train data, that is, D_i^{train} (Algorithm 2, line 10). At last, C_i stores P_x as the last model performance received from C_x . This information will be used by the personalized peer-sampling protocol as further discussed in the following section.

4.2 Personalized Peer-Sampling Algorithm

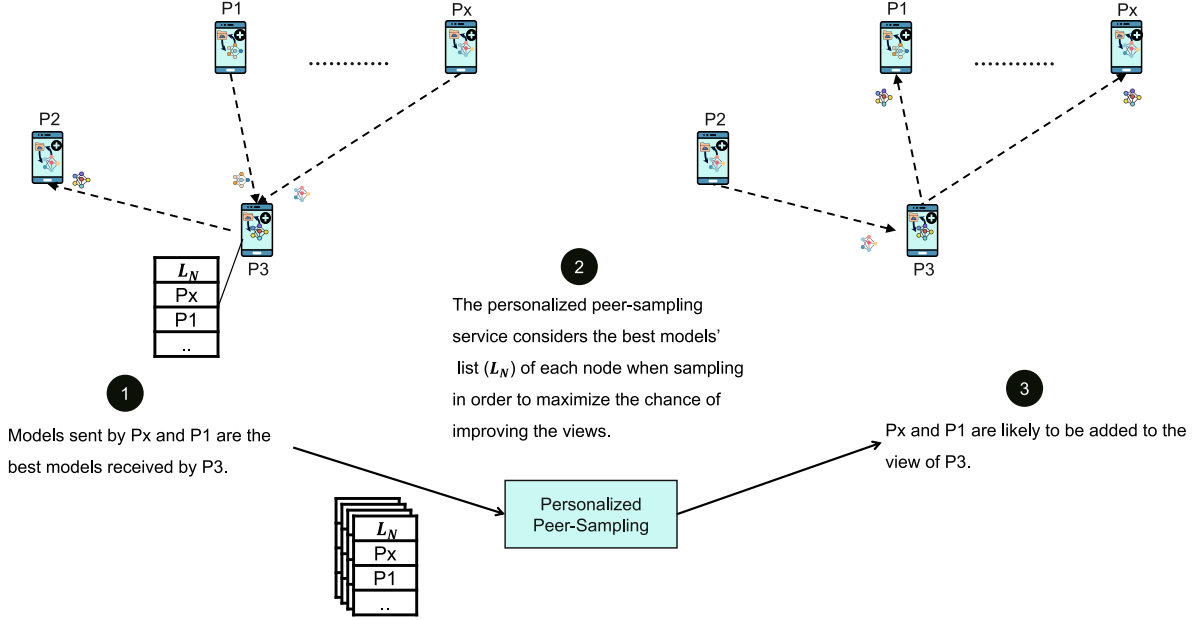


Figure 3: Personalized Peer-Sampling Execution Scenario.

As discussed in Section 2, peer-sampling can have a significant impact on the convergence speed of Gossip Learning algorithms. This protocol is even more important in use cases where a node might be more interested in models coming from similar nodes in terms of data distribution. To take this into account, we extend the random peer-sampling protocol by considering the quality of the current nodes in the view, that is, their ability of sending quality models, as illustrated in Figure 3. To this end, the personalized peer-sampling service collects the recommendation performance of the received models and the identity of their respective nodes (steps ① and ② in the figure). These performances come from the evaluation step of the performance-based aggregation function. As a result, nodes that sent quality models to a given node are likely to join his view in later rounds (step ③ in the figure).

More formally, let us consider a node C_i during a peer-sampling phase. To update its view V_{C_i} , C_i performs two main steps, as illustrated in Algorithm 3. First, it relies on past information collected from the performance-based model aggregation function regarding the performance of models received from other nodes in the system. Specifically, C_i maintain a list L_N of nodes from which it received models in the past, sorted by the performances of these models (lines 2-3). To update its view, C_i considers the top T elements of L_N (lines 6-7). In addition to this first step, C_i randomly collects R nodes from its peers' views to form the new view (lines 8 -10). T and R depend on an exploration/exploitation ratio α . Values of α close to zero translate to an exploitation-dominant approach that maximizes the performance of the view at the expense of discovering the network (*i.e.*, $P' \approx T$) whereas on the contrary $\alpha = 1$ is an exploration-dominant approach and is equivalent to the traditional random peer-sampling (*i.e.*, $P' = R$). The optimal value of α cannot really be determined as it is highly dependent on the underlying data. However, we observed empirically that an $\alpha = 0.4$, equivalent to 40% of nodes coming from L_N and 60% of random nodes, gives the best results in our experiments.

5 Performance Evaluation

In this section, we evaluate PEPPER on three real world datasets and compare its performance against centralized and decentralized solutions. More specifically, we aim at answering the following research questions:

- **RQ1:** PEPPER aggregates models based on their local performance. How efficient is PEPPER compared to decentralized solutions (that aggregate models to target average performance) and can it be competitive with federated baselines?

Algorithm 3: Personalized Peer-Sampling Algorithm

Data: Nodes set N , currentModel M_i , exploitation/exploration ratio α , list of received models and their best performance L_N, V_{C_i}

```

1 Procedure ONMODELRECEIVED( $M_x$ ):
2    $P_x = \text{PERFORMANCEBASEDAGGREGATIONFUNCTION}(M_x)$ 
3    $L_N.\text{ADDSORTED}(C_x, P_x)$   $\triangleright$  Add received model's performance and its owner to the sorted list
4
5 Procedure UPDATEVIEW():
6    $T = (1 - \alpha) \times |V_{C_i}|$ 
7    $\text{ExploitationPeers} = L_N.\text{GET}(T)$   $\triangleright$  Get the top T elements of  $L_N$  performance wise
8    $R = |V_{C_i}| - T$ 
9    $\text{ExplorationPeers} = \text{RandomPeers}(N, R)$   $\triangleright$  Get R random peers from N
10   $V_{C_i} = \text{Concat}(\text{ExplorationPeers}, \text{ExploitationPeers})$   $\triangleright$  Update view with exploitation and exploration peers
11
```

- **RQ2:** Does considering the local performance of models make them more tailored to their users and how does that affect tail performance?
- **RQ3:** What role does the amount of sparsity play in the context of an increased personalization?
- **RQ4:** PEPPER extends the random-peer-sampling protocol by considering the local model performance. How does this affect the learning process?
- **RQ5:** How sensitive is PEPPER to the peer set size parameter?
- **RQ6:** What level of overhead does PEPPER incur on the recommendation pipeline?

The rest of this section is structured as follows. We first introduce the experimental environment we used to evaluate PEPPER (Section 5.1) before describing the used datasets (Section 5.2). Further, we explain our evaluation methodology and the employed evaluation metrics (Section 5.3). Finally, we present our competitors (Section 5.4) and analyze our experimental results answering the six questions introduced above (Section 5.5).

5.1 Experimental Environment

PEPPER is built on top of a decentralized network of nodes that is simulated using *OMNetPy* [43], a python interface for *OMNet++* [61], a popular simulation platform for communication networks. Associated with the well known *TensorFlow/Keras* [1], it allows training models at the level of each node and exchanging these models over a decentralized network. It is worth noting that since *OMNet++* is an event-based simulator and our experiments are based on round-based behaviors, we use timed events to simulate these actions. Note that in our simulations initial nodes' neighbors were generated randomly. Nevertheless, to ensure reliability and put all decentralized competitors in the same setup, experiments were repeated several times using the same seed when generating the initial setup.

5.2 Datasets

Dataset	Type	Users	Locations/Movies	Records	Sparsity %
Foursquare-NYC	Points of interest	1083	38333	227,428	0.997
Gowalla-NYC	Points of interest	718	32924	185,932	0.986
MovieLens-100k	Movies Recommendation	943	1682	100,000	0.936

Table 1: Statistics of Datasets.

As presented in Table 1, for our experiments we chose three different datasets that are commonly used to evaluate recommendation systems. MovieLens-100K [25] is a dataset containing 100k movie ratings from 943 users on 1682 movies. Similarly to previous works [45, 64, 64] we adapted this dataset to the Generalized Matrix Factorization (GMF) as follows: (i) all user ratings are converted to positive ratings (*i.e.*, 1) and (ii) items not rated by a user are labeled with zero. This pre-processing is closely related to the fact that the GMF model is a classifier whose output is subject

to a probabilistic activation function (*i.e.*, the sigmoid function) representing the probability of relevance of an item to a user [27]. Furthermore, the work of Koren [36] details how incorporating such binary data, which normalizes the interpretation of ratings by users, can improve prediction accuracy. Finally, we note that our competitors [45, 64] adopted the same approach.

The two other datasets are Foursquare-NYC and Gowalla-NYC Check-ins. The former consists of 227,428 check-ins collected by 1083 users from 38333 venues and the latter is composed of 82197 check-ins and 718 users, both in New York city. Each check-in is associated with its time stamp, its GPS coordinates and the venue category. In the pre-processing phase, we removed locations with less than 10 visitors, and users with less than 10 check-ins, as it is common for Point-of-Interest recommender systems [11, 22]. In addition, similarly to MovieLens, when evaluating GMF, we transform the check-ins to binary ratings in order to build a top-K ranking recommender system. This step is not necessary for evaluating the PRME-G model that is specifically designed for sequential check-in data. We have also computed the sparsity of each dataset as in Equation (3). It can be observed that MovieLens has significantly less sparsity than the other two datasets.

$$sparsity = 1 - \frac{avg. rated items}{all items} \quad (3)$$

5.3 Evaluation Methodology

To evaluate PEPPER, we split the local datasets of each user using the 85-15 rule: 85% of the data is used to train the model, which will be evaluated on the remaining 15%. Specifically, for each client C_i , we split its data into a training set D_i^{train} and a test set D_i^{test} . As explained in Section 4.1, for PEPPER, a weighting set $D_i^{weighting}$ with the same size as the test set is furthermore extracted from D_i^{train} . The model M_i is trained on D_i^{train} and locally evaluated on D_i^{test} as opposed to other works which evaluate models on the union of test sets (*i.e.*, a global test set). By doing so, we measure the local performance of each model and quantify the user’s satisfaction. More specifically, for a given user, we require each algorithm to provide the top-K recommendation list for each test item among 100 randomly sampled and unvisited/unrated items. This is a common strategy to avoid ranking test items among all unseen items [27, 53], which can be considerably time-consuming. The value of K is set to 5, 10 and 20 respectively.

For the GMF model, we use two classic metrics to evaluate the ranked lists, namely the hit ratio at rank r (*i.e.*, $HR@r$), which is the performance metric we use to weight models in our aggregation function, and the normalized discounted cumulative gain at rank r (*i.e.*, $NDCG@r$). Intuitively, $HR@r$ of the item t takes the value 1 if the latter is present in the top r elements of the ranked list and 0 otherwise. $NDCG@r$ measures how good is the position of t in the list, assigning higher scores to hits at the top of the ranked list. More specifically, $HR@r$ of a client C_i will be computed as in Equation (4), while its $NDCG@r$ will be the average of $NDCGs@r$ of each of its items, computed each as in Equation (5).

$$HR@r = \frac{\sum_{j=0}^I HR@r_j}{|D_i^{test}|} \quad (4)$$

where $HR@r_j$ is the hit ratio for item j and I the set of items of the user C_i .

$$NDCG@r_i = \frac{\log(2)}{\log(p+2)} \quad (5)$$

with i being the corresponding item and p its position in the ranked list.

For the PRME-G model, we resort to the widely known metrics used in the original paper of Feng et al. [15], namely Precision@ r and Recall@ r , which we use to compute the F1-Score at rank r . These metrics are well suited for a next POI recommendation use case, where the ability of a model to find the known relevant POIs for a user (*i.e.*, recall) and its capacity to distinguish between relevant and non-relevant locations (*i.e.*, precision) is very important. More precisely, Precision@ r is the proportion of recommended items in the top- r set that are relevant. Hence, it is computed by taking the ratio of relevant items found in the top- r divided by r (see Equation (6)). Similarly, Recall@ r is the proportion of relevant items found in the top- r recommendations. It is computed by taking the ratio of relevant items found in the top r divided by the total number of relevant items for a particular user (see Equation (7)). To combine these two metrics, we compute the F1-Score@ r and use it both for evaluation and aggregation purposes. F1-Score@ r metric takes into account both Precision@ r and Recall@ r , as showed in Equation (8).

$$Precision@r_u = \frac{|Relevant items for user u@r|}{r} \quad (6)$$

$$Recall@r_u = \frac{|\text{Relevant items for user } u@r|}{\text{All relevant items for user } u} \quad (7)$$

$$F1-Score@r_u = 2 \times \frac{Precision@r_u \times Recall@r_u}{Precision@r_u + Recall@r_u} \quad (8)$$

5.4 Baselines

We compare our aggregation techniques against six baselines. These baselines are of two kinds: federated baselines, *i.e.*, baselines that follow the classical FL architecture and decentralized baselines, *i.e.*, baselines that rely on Gossip Learning.

5.4.1 Federated Baselines

- *FedAvg* [41] is one of the most popular FL protocols, based on the traditional master-slave FL architecture. Instead of centralizing the raw data as in Centralized GMF, FedAvg centralizes model updates from the FL users and aggregates them based on the number of samples they were trained on.
- *FedFast* [45] is the closest work to ours, albeit being centralized. FedFast is an extension over the FedAvg algorithm that clusters users at each round using a k-means algorithm. Subsequently, it performs cross-cluster aggregation for the user embeddings of the GMF model while aggregating the item embeddings and other model parameters in the same manner as FedAvg. At the beginning of each FL round, it samples users equally from each cluster. We adapt this technique for the PRME-G model in order to compare it with PEPPER.
- *Reptile* [64] is a meta-learning approach, which uses the weights learned by the FedAvg algorithm to initialize a local meta-model to each user. Afterwards, this meta-model is fine-tuned on local data by updating the initial parameters towards the final parameters learned locally. This approach has been shown to improve personalization.

5.4.2 Decentralized Baselines

- *Model-Age-Based* is a decentralized gossip-based approach introduced in [28, 30]. Models received by a user are weighted proportionally to the number of times they have been updated (*i.e.*, the model’s age). The rationale behind this aggregation algorithm is that a model which was updated by a larger number of nodes has more information so it should weight more in the aggregation function.
- *Decentralized FedAvg* [38] is a decentralized version of FedAvg. Models circulate freely through the network and nodes weight them in the aggregation function depending on the number of samples they were trained on.
- *Decentralized Reptile* is a decentralized version of Reptile [64] that we implemented in which each node fine-tunes its local model based on the parameters learned in a decentralized collaborative way.

5.5 Experimental Results and Discussions

Based on the evaluation setup described above, we conduct experiments to evaluate the performance of PEPPER against the state-of-the-art solutions introduced in the previous section. Our evaluation aims at answering the questions introduced at the beginning of Section 5.

5.5.1 PEPPER Average Performance Comparison

In this experiment, we compare the performance of all models on PEPPER and the aforementioned baselines. Firstly, we evaluate the *average* top-K recommendation quality where K is 5, 10 and 20. Table 2 and Table 4 show the average top-20 recommendation hit ratio comparison on Foursquare and MovieLens, while Table 3 and Table 5 show the average NDCG. In Tables 6 and 7 we report the F1 score of PRME-G on the Foursquare and Gowalla datasets. From these results, we can observe that PEPPER outperforms its decentralized competitors on all datasets. Compared to Decentralized FedAvg, PEPPER has a better HR@20 and NDCG@20 by a margin of 8.93% and 4.35%, respectively. The results for different values of K and two different metrics confirm that PEPPER does not privilege recall to the ranking quality of tested items, even though HR is used during the aggregation. This is essential to show that our system is not only optimizing a specific metric (*i.e.*, Goodhart’s law [40]) but using it to optimize the true quality of the model.

Algorithm	Average Hit Ratio %		
	K = 20	K = 10	K = 5
FedAvg	32.42	24.0	18.30
Reptile	31.59	23.25	18.10
FedFast	37.12	26.27	18.94
Decentralized FedAvg	37.00	28.10	20.44
Model-Age-Based	38.54	26.87	20.16
Decentralized Reptile	38.58	28.99	22.70
PEPPER	45.93	32.59	25.90

Table 2: Average Hit Ratio % on Foursquare-NYC (GMF).

Algorithm	Average NDCG %		
	K = 20	K = 10	K = 5
FedAvg	16.61	14.37	12.66
Reptile	17.12	14.36	13.38
FedFast	18.41	15.69	13.30
Model-Age-Based	19.07	16.81	14.92
Decentralized FedAvg	18.75	17.39	15.37
Decentralized Reptile	22.04	19.61	17.63
PEPPER	23.10	20.44	19.24

Table 3: Average NDCG % on Foursquare-NYC (GMF).

Algorithm	Average Hit Ratio %		
	K = 20	K = 10	K = 5
FedAvg	79.69	64.4	47.59
Reptile	79.49	62.6	44.1
FedFast	82.79	67.80	50.00
Decentralized FedAvg	74.9	55.0	40.7
Model-Age-Based	73.69	55.5	42.4
Decentralized Reptile	70.19	51.60	36.29
PEPPER	79.29	61.20	47.59

Table 4: Average Hit Ratio % on MovieLens-100k (GMF).

Algorithm	Average NDCG %		
	K = 20	K = 10	K = 5
FedAvg	41.03	37.35	32.07
Reptile	40.07	35.80	29.83
FedFast	44.14	40.34	34.50
Decentralized FedAvg	36.17	31.79	27.56
Model-Age-Based	37.47	33.06	29.81
Decentralized Reptile	34.0	29.05	24.14
PEPPER	40.09	35.60	32.53

Table 5: Average NDCG % on MovieLens-100k (GMF).

Algorithm	Average F1-Score %		
	K = 20	K = 10	K = 5
FedAvg	32.87	28.62	21.50
Reptile	35.00	31.53	23.86
FedFast	33.35	30.56	23.18
Decentralized FedAvg	21.82	18.70	16.03
Model-Age-Based	20.48	18.60	17.19
Decentralized Reptile	27.28	25.46	21.19
PEPPER	34.05	32.44	26.38

Table 6: Average F1-Score % on Foursquare-NYC (PRME-G).

Algorithm	Average F1-Score %		
	K = 20	K = 10	K = 5
FedAvg	21.35	18.53	15.37
Reptile	20.90	17.07	13.46
FedFast	21.80	18.28	15.09
Decentralized FedAvg	11.30	10.18	9.13
Model-Age-Based	9.92	8.59	7.64
Decentralized Reptile	9.10	7.66	6.05
PEPPER	23.63	20.66	17.15

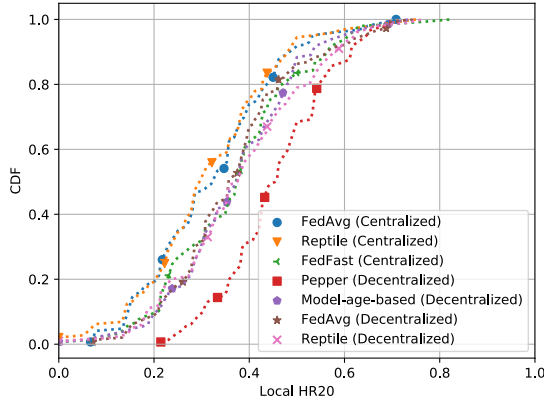
Table 7: Average F1-Score % on Gowalla-NYC (PRME-G).

Foursquare is the most sparse dataset out of the three (see Table 1) and therefore has more heterogeneous user profiles so one global model may not be able to fit all user preferences (see §5.5.3). Therefore, our solution which builds personalized models outperforms centralized solutions in most cases except for the F1@20 on Foursquare (see Table 6). The sparsity observation is further supported by the results obtained on Gowalla (see Table 7), which is similar in sparsity to Foursquare and where PEPPER outperforms all the other solutions, for all values of K. The Reptile algorithm has a fairly high performance with the PRME-G model and is the best algorithm for K=20 on Foursquare. However, since it is outperformed by PEPPER for other values of K, we can conclude that it finds many of the relevant items but does not rank them as accurately as PEPPER. On MovieLens, which is less sparse (see Tables 4 and 5), PEPPER is usually overpassed by centralized solutions (*i.e.*, FedFast, FedAvg and Reptile). Among the latter three, FedFast performs the best since it personalizes based on a view over all users so it reaches a better personalization-generalization tradeoff compared to PEPPER. Nevertheless, PEPPER still substantially overpasses all the decentralised algorithms.

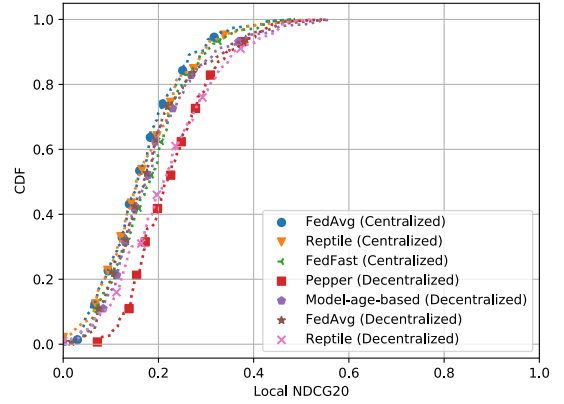
Based on these results, we can observe that **PEPPER always outperforms its decentralized competitors on average performance** and can even be competitive with centralized solutions, especially in the case of sparse recommendation matrices (**RQ1**).

5.5.2 PEPPER Tail Performance Comparison

In this experiment, we are interested in evaluating the individual satisfaction of each user. Figures 4a and 5a illustrate the cumulative distribution functions of the HR in two datasets: Foursquare-NYC and MovieLens.

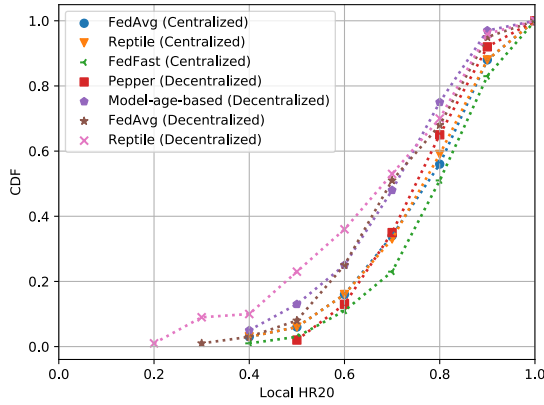


(a) Local Hit Ratio@20 cumulative distribution function.

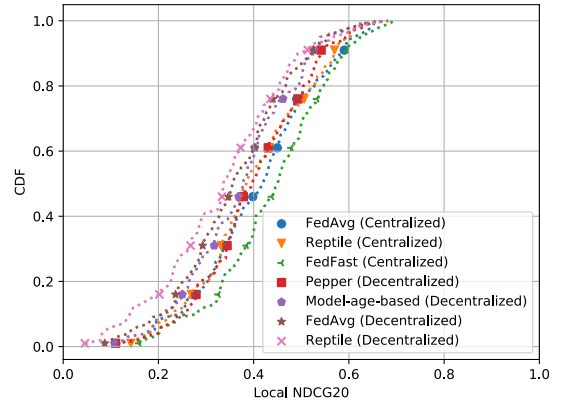


(b) Local NDCG@20 cumulative distribution function.

Figure 4: Top-K recommendation quality distribution comparison on Foursquare-NYC (GMF, K = 20).



(a) Local Hit Ratio@20 cumulative distribution function.



(b) Local NDCG@20 cumulative distribution function.

Figure 5: Top-K recommendation quality distribution comparison on MovieLens-100K (GMF, K = 20).

Similarly, Figures 4b and 5b illustrate that for NDCG. In these figures a point $(x = HR, y = CDF(x))$, respectively $(x = NDCG, y = CDF(x))$, represents the proportion of users y having a hit ratio at most equal to HR, respectively an NDCG value at most equal to NDCG. Hence for a fixed value of HR or NDCG, the best curve is the one furthest to the right. Centralized and generically aggregated models are often optimized with respect to a global objective and are, thus, less tailored to each individual user. Therefore, there is often a large gap between the tail and the average performance of these solutions. PEPPER aims to reduce this gap. For example, the 99.9th percentile in PEPPER achieves about three times the performance of its best competitor (*i.e.*, centralized FedAvg) on Foursquare (see Figure 4a). We mention that the 99th percentile represents the values corresponding to a CDF of 0.1 on Figures 4, 5, and 6. Concerning MovieLens, PEPPER is generally overpassed by centralized solutions on both short and long tail performance. However, it always achieves better tail performance than its decentralized competitors (see Figure 5). The aforementioned experiments are performed with a Generalized Matrix Factorization (GMF) model. Figures 6a and 6b illustrate the percentiles' F1 score results for the PRME-G model. Similarly to the previous use case, The FedFast protocol and Reptile have the best short tail score. Nevertheless, we can observe that PEPPER has a performance close to the best performing solution and even outperforms them on the long tail (*i.e.*, 60th to 85th percentile). For the short tail, around 2.5% of nodes don't seem to have enough datapoints to converge PRME-G in a decentralized learning setup so the 99.9th percentile F1 score of all decentralized solutions tends towards zero. In summary, the evaluation results show that **PEPPER outperforms state-of-the-art decentralized solutions on long and short tail performance**. Moreover, PEPPER slightly improves

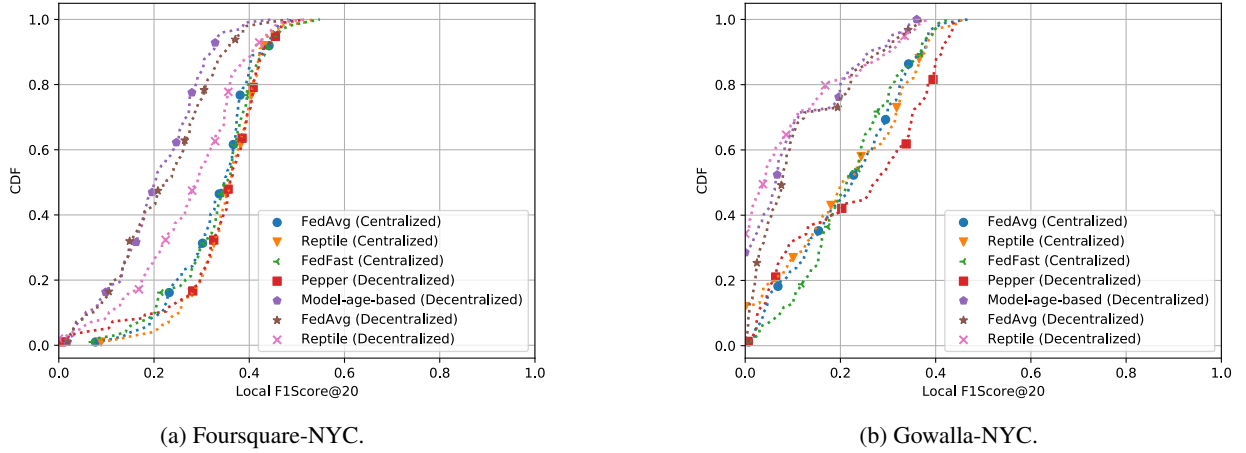


Figure 6: Top-K F1-Score cumulative distribution function (PRME-G, K = 20).

the short tail (*i.e.*, 99.9th percentile) on MovieLens over centralized competitors, while significantly improving the long tail on the other datasets (**RQ2**).

5.5.3 Effects of Sparsity

Sparsity is a very important aspect that often negatively impacts the performance of recommendation systems. Therefore, to evaluate its impact on PEPPER, we have generated two additional datasets (Dense ML-100k and Sparse ML-100k) by clustering the users of the original MovieLens-100k using k-means. Dense ML-100k is composed of the users which come from the most dense cluster while Sparse ML-100k is composed of all of the outliers.

Higher sparsity levels can often lead to users having less overlap in terms of preferences (*i.e.*, rated movies, visited locations), which makes it challenging for global models, and especially those based on learning the user-item relationships, to fit all users' preferences. Therefore, the centralized solutions (*e.g.*, FedAvg), which train a unique model will experience more difficulties to provide satisfactory results for all users. However, as shown in Figure 7, **the personalized models built by PEPPER, through their ability to capture the preferences of each user, seem to be more robust to high degrees of sparsity**. This result further corroborates the Section 5.5.1 results, where PEPPER was more competitive with the centralized approaches on sparsest datasets (**RQ3**).

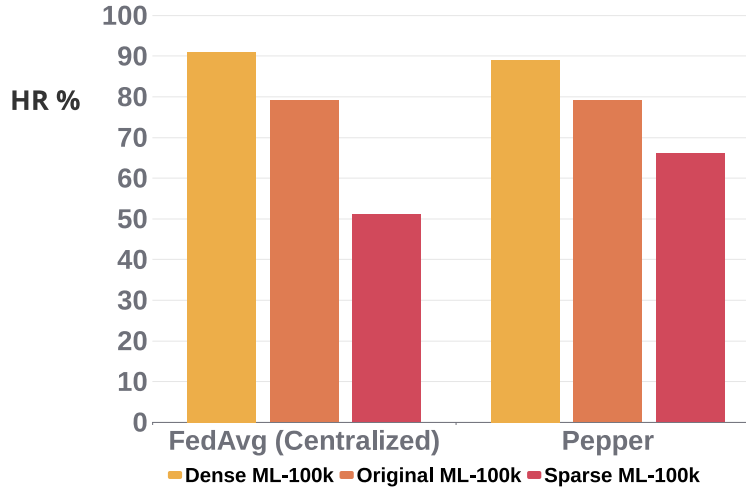
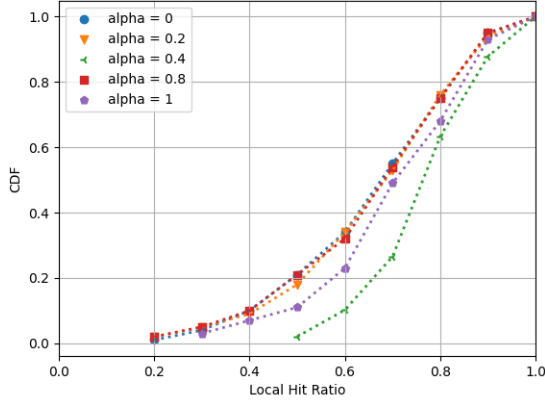


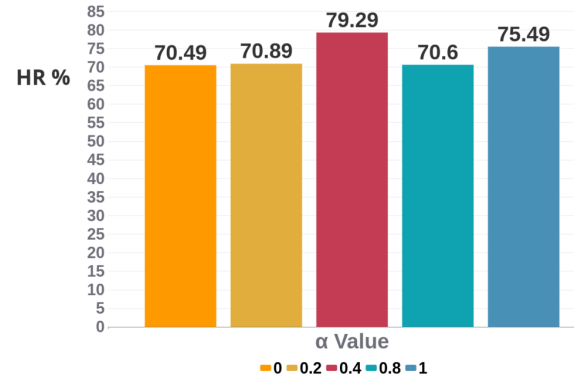
Figure 7: The robustness of PEPPER to sparsity in comparison with Federated Averaging, on three different sparsity levels: Dense ML-100k (sparsity = 0.64, users = 579), Original ML-100k (sparsity = 0.93, users = 943) and Sparse ML-100k (sparsity = 0.99, users = 201).

5.5.4 Sensitivity of PEPPER to the exploitation-exploration ratio alpha

Peer-sampling is a key component of the system as the search for similar neighbors helps model personalization and hence improves the local performance. In this context, the alpha parameter in PEPPER, which fixes the exploration versus the exploitation ratio, plays a crucial role. We have evaluated the impact of this parameter on the performance of PEPPER in an equal number of training rounds by measuring the CDF of the local hit ratio with various values of alpha, as well as computing the average hit ratio. Results depicted in Figure 8a show that a value of $\alpha = 0.4$ yields the best average hit ratio. By looking closer at Figure 8b, we observe that for a value of $\alpha = 0$ where nodes never change their neighbors, there are two consequences: i) the rate of model dissemination is reduced and ii) the initial placement of nodes becomes very significant, as a node without similar first or second degree neighbors will not be able to personalize its model and will have to settle for the models it receives, finding itself forced to build a more global model. On the other hand, for an $\alpha = 1$, where at each peer-sampling period, nodes completely change their view, which does not give them the time to make the most of similar nodes' model, it seems that personalization is significantly diminished but not completely annihilated. This leads to better models than the first case. Inevitably, the average performance suffers in both of these extreme cases (70.49% in the former and 75.49% in the latter). This also shows, as expected, that an aggregation maximizing local performance cannot have a significant impact if the model evaluation step is not exploited to identify similar nodes and take advantage of their models. Simultaneously, PEPPER cannot entirely rely on a personalized peer-sampling as randomization is important to disseminate models and to find better neighbours (**RQ4**).



(a) Cumulative Distribution Function comparison between different alpha values on MovieLens-100k (GMF).



(b) Average performance w.r.t different values of Alpha on MovieLens-100k (GMF).

Figure 8: Sensitivity of PEPPER to the exploitation-exploration ratio alpha.

5.5.5 Sensitivity of PEPPER to the peer set size

In order to assess the impact of the peer set size on the performance of PEPPER, we performed an experiment where we doubled the peer set size used in the other experiments of this paper (*i.e.*, peer set size = 6 instead of peer set size = 3). We performed this experiment on the MovieLens-100k dataset using the GMF model. Results are depicted in Table 8. From these results, we observe that increasing the peer set size positively impacts convergence as the models need less rounds to converge. Moreover, performance is faintly improved. However, this comes at the price of an increased computational overhead as nodes receive more models, hence, computing more SGD and aggregation steps. To choose our peer set size, we complied to the theoretical bounds proved by previous works on gossip [7]. The latter state that a logarithmic peer set size in the total number of nodes provides the best tradeoff between all of these metrics under the condition that each new node v is connected to another one w with a probability proportional to the number of neighbours of w (**RQ5**).

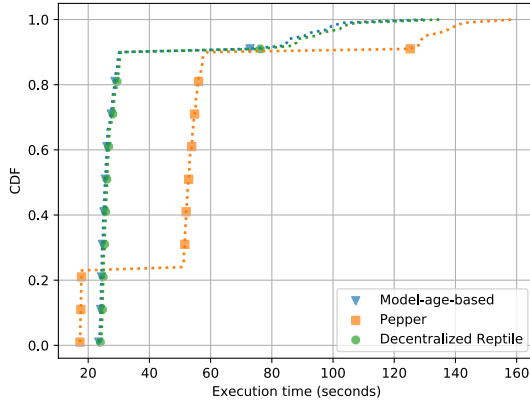
	Average Performance (HR %)	Average Communication Rounds	Average Overhead (seconds)
Peer Set Size = 3	0.79	283	53.4
Peer Set Size = 6	0.80	136	119.8

Table 8: Impact of the Peer Set Size on PEPPER.

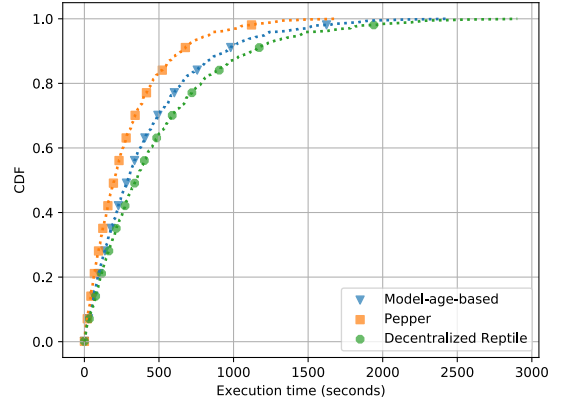
5.5.6 Overhead evaluation

We also evaluated the overhead of PEPPER compared with its decentralized competitors. In Figure 9a we represented the CDF of the total computation time for each individual node. On each node, we sum up the time spent to compute the SGD algorithm on local data and the aggregation time over all of the learning process and then, we represent these values as a CDF. In this latter, we did not consider the Decentralized FedAvg algorithm due to it having the same cost as Model-age-based. We can observe that 90% of the nodes in both Decentralized Reptile and Model-age-based have fairly homogeneous execution times, with 90% of them having around 25 seconds. **Concerning PEPPER, 21% of the nodes have a total execution time inferior to the other decentralized solutions and 69% of the nodes have a longer execution time of 54 seconds (RQ6).** These two clusters of users are formed because of the following phenomenon. The nodes with less data will usually send less-accurate models to their neighbours so, because of our Personalized Peer-sampling, they will be chosen with less probability by the other nodes. Therefore, they will receive less models to be evaluated by PEPPER and therefore their execution time will be shorter.

Even if the execution time is higher on some nodes, PEPPER personalizes the models to the preferences of each user so it needs less communication rounds to converge (see Figure 9b). **On average, the nodes of PEPPER reach the model convergence after 283 communication rounds comparing with 410 for Model-age-based and 490 for Decentralized Reptile.** When analysing the high percentiles, we can observe that the gap increases even more. For example, the 99th percentile of PEPPER is characterized by 799 and 1247 fewer rounds than Model-age-based and Decentralized Reptile, respectively (RQ6').



(a) The CDF of the total computing time for each individual node until the model convergence is reached.



(b) The average number of communication rounds executed by each individual user.

Figure 9: Computation vs Communication costs.

6 Discussion

In this section we discuss the limitations of PEPPER and their possible mitigations.

On the use of simulations to evaluate PEPPER: Decentralized systems imply the existence of a large number of nodes interconnected in a peer-to-peer network. However, few organizations have access to such a system in the physical world. Therefore, many research works rely on simulators to evaluate their decentralized algorithms [28, 30]. In PEPPER, we rely on Omnet++ [61], a popular network simulator allowing to evaluate settings involving up to 1000 nodes. Practical studies have further assessed the accuracy of using this simulator compared to real testbeds [10].

On the privacy of PEPPER: We assume in this paper that participants are trusted. However, despite the fact that nodes keep their data in their premises, there exist attacks that have been devised in the context of Federated Learning and that can leak private information from the exchanged model updates [20, 47]. These attacks could very well take place in a fully decentralized setting as in PEPPER, but considering such attacks is out of the scope of this paper. Nevertheless, there exist research works focusing on this issue (e.g., solutions relying on differential privacy [46]). Specifically, Hegedűs and Jelasity [29] have investigated the use of differential privacy in the context of Gossip Learning [29]. In the more generic context of Federated Learning other solutions to protect users against inference attacks are heavily investigated by the research community as surveyed in [13]. A combination of existing solutions with PEPPER shall

be investigated in future work.

On the resilience of PEPPER to poisoning attacks: Similarly to inference attacks, there exist poisoning attacks that have been proposed in the context of Federated Learning and that could easily be launched by PEPPER participants (e.g., [31, 60, 67]). For instance, malicious participants could send poisonous model updates to their neighbors. While we considered this issue as being out of the scope of this paper, the fact that nodes in PEPPER locally assess the relevance of received models before aggregating them could naturally protect them against such attacks. We plan to assess in our future work the resilience of PEPPER to poisoning attacks. In case these attacks are still possible, there exist model aggregation functions that have been devised in the context of FL in order to identify and exclude poisonous gradients and that could be integrated in PEPPER (e.g., [5, 59, 67]).

On the impact of churn, network dynamics and node heterogeneity: Churn and more generally network dynamics and node heterogeneity (e.g., in terms of computing and networking capabilities) have an impact on the performance of gossip protocols and thus on the applications running on top of them such as PEPPER. While studying the impact of churn, network dynamics and node heterogeneity is important, we considered it out of the scope of this paper. In practice, we expect churn, network dynamics and node heterogeneity to have a similar impact on PEPPER and on its decentralized competitors, who did not either look at these issues yet. Nevertheless, the distributed systems community has investigated many gossiping flavors that increase the robustness of gossip to the above issues. For instance, Frey et al. [16] showed that while the gossip protocol is by design robust to churn in a uniform and highly capable distribution, it can be adapted to heterogeneous setups by adjusting the view size of each node proportionally to its upload capability. Combining this approach with theoretical results, which show that an average logarithmic view size preserves the dissemination of a gossip protocol, [16] enhances considerably the robustness of gossip protocols to churn in heterogeneous contexts. More recently, preliminary solutions have been considered for gossip learning (e.g., [19, 24]). For instance, in [19], a simple yet effective mechanism that skips the aggregation and update phases when the view size of the sending node is small, ensures that models coming from low-degree nodes propagate faster. As opposed to this approach, Han et al. [24] proposed a data-driven one where the ratio and the nature of training data taken in consideration by a node is adjusted accordingly to its computing and networking abilities. A combination of such solutions with PEPPER shall be investigated in future work.

7 Related Work

In the past several solutions have been proposed to build distributed recommender systems. The first ones focused mainly on memory-based methods of recommendations, which usually use similarity metrics to build communities of users and/or items. In addition to being highly time consuming [6], these methods require sharing recommendation information (e.g., ratings, user-items interactions, etc.) between users which was done either through distributed hash tables [23, 34] or directly via gossip protocols [49, 50]. This information sharing is sometimes transgressing user privacy so it motivated more careful designs such as [12, 26], which imply a notion of trust between subsets of users. However, these setups were neither practical nor generalizable so various privacy-preserving recommender systems started to emerge.

The privacy-preserving recommender systems fall into three categories. First, there are *homomorphic-encryption-based* solutions [21, 35, 63], where user profiles, ratings, and any sensitive information is encrypted before being processed. This approach is computationally expensive and introduces a significant overhead. A state-of-the-art implementation of fully homomorphic encryption [52] requires 28 hours to make a recommendation on MovieLens [25]. The second category is composed of *solutions based on differential privacy (DP)* [4, 17, 18, 42] where user profiles are obfuscated with noise. However, due to noise interference, DP often introduces a compromise between the privacy and the accuracy of recommendations. The third category of privacy-preserving recommender systems is represented by the *Federated Learning-based solutions*. Depending on the presence of parameter servers or not, Federated Recommender Systems can be divided in two categories:

Centralized FL recommender systems. These systems are characterized by the presence of a centralized server which coordinates the learning process. In the works of Qin and Liu [51] this entity allows the users to collaboratively train a recommender model based on sensitive information such as location and age which are kept private on client devices. In addition, user-items interactions are considered to be publicly available so they are centralized and leveraged to train another model on the server-side. However, it can be argued that such interactions are also private by nature. Therefore, other works [45, 54, 64] are more strict in this regard as they do not assume any publicly available data. For personalization and user satisfaction purposes, two distinct approaches can be identified: Meta-Learning-based solutions and similarity-based ones. The former [14, 54, 64] are based on the idea of training a global model following

a traditional method (e.g., Federated Averaging) then fine-tuning it on local data by making meta-steps, which turn the global model into a more personalized one. While such techniques have been found to be effective in many use cases, they incur a risk of overfitting due to the purely local personalization. In contrast, in PEPPER, a model is personalized by taking advantage of similar models, which reduces the risk of overfitting. On the other hand, these Meta-Learning techniques depend considerably on having a good global model as a starting point. For that reason, they have not, to the best of our knowledge, been considered in decentralized contexts, where obtaining such a global model is often fairly difficult. In the similarity-based class of solutions, FedFast [45] can be considered as one of the most influential works. It incorporates a similarity-based client sampling technique employed to cluster users so that an active aggregation function can be applied within these clusters, which improves the performance and convergence speed. However, the client sampling technique is based on a per-round clustering, which, as stated by the authors, is quite costly (e.g., a time complexity $\mathcal{O}(n^{81})$ per round on MovieLens-100k). In contrast, PEPPER emulates a cheaper and gradual clustering along the learning process, by keeping similar nodes close to each other. Another difference is that PEPPER considers all models and only ponders their magnitude with respect to their similarity, while in contrast, FedFast completely excludes parts of the models that are outside of the cluster of a user. Finally, FedFast also inherits the main drawback of centralized FL solutions which is the presence of a central server that can face different kinds of challenges (See Section 2.1).

Decentralized FL recommender systems. In decentralized FL recommender systems, there is no central server and clients instead exchange models with each other. To the best of our knowledge, the work of Hegedűs et al. [28] is the only decentralized Federated Learning recommender system solution in the literature. In this work, the authors show that decentralization’s impact on performance can be mitigated. For that purpose, different model compression methods as well as algorithmic enhancements in the form of flow control mechanisms were presented. While they do not specifically target aggregation functions, they still implement an aggregation function that takes into consideration the age of models, which allows models that have seen more data to have a larger weight. We consider this work as a baseline in our current evaluation settings to quantify the added value of our performance-based aggregation function.

8 Conclusion

In this work, we presented PEPPER, a decentralized and privacy-preserving recommender system. PEPPER relies on Gossip Learning principles for enabling nodes to asynchronously train a model that better responds to their needs. At the heart of PEPPER resides two key components: i) a personalized peer-sampling protocol, which allows each node to keep in his neighborhood a proportion of similar nodes (taste wise), leading to a gradual clustering of users and ii) a simple yet effective model aggregation function that builds a model that is better suited to each user. We implemented PEPPER on a networking simulator and evaluated its performance using three real datasets involving up to 1000 nodes and implementing two use cases: a location check-in recommendation and a movie recommendation. Our results show that, on average, nodes in PEPPER converge with up to 42% less communications rounds than with other decentralized approaches while providing up to 8% improvement on average performance and up to 30% improvement on tail performance compared to decentralized competitors. As part of our future work we plan to investigate the impact of fully decentralizing recommender systems as in PEPPER on their resilience to adversaries.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. <https://www.tensorflow.org/> Software available from tensorflow.org.
- [2] Naveen Farag Awad and Mayuram S Krishnan. 2006. The personalization privacy paradox: an empirical evaluation of information transparency and the willingness to be profiled online for personalization. *MIS quarterly* (2006), 13–28.
- [3] Ranieri Baraglia, Patrizio Dazzi, Matteo Mordacchini, and Laura Ricci. 2013. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *J. Comput. Syst. Sci.* 79 (2013), 291–308.
- [4] Alon Ben Horin and Tamir Tassa. 2021. *Privacy Preserving Collaborative Filtering by Distributed Mediation*. Association for Computing Machinery, New York, NY, USA, 332–341. <https://doi.org/10.1145/3460231.3474251>

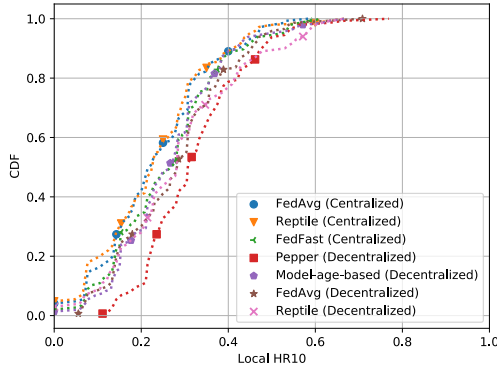
- [5] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in Neural Information Processing Systems* 30 (2017).
- [6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. 2013. Recommender Systems Survey. *Know.-Based Syst.* 46 (jul 2013), 109–132. <https://doi.org/10.1016/j.knosys.2013.03.012>
- [7] B. Bollobás and O. Riordan. 2004. The Diameter of a Scale-Free Random Graph. *Combinatorica* 24 (2004), 5–34.
- [8] Mathias Brandt. 2015. Location Based Services Still In Its Infancy. <https://www.statista.com/chart/3713/smartphone-use-of-location-based-services/>.
- [9] CiscoMag. [n.d.]. *Nearly 140 million user data leaked in Canva hack.* <https://www.forbes.com/sites/daveywinder/2020/08/19/massive-data-leak235-million-instagram-tiktok-and-youtube-user-profiles-exposed/>
- [10] Ugo Maria Colesanti, Carlo Crociani, and Andrea Vitaletti. 2007. On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed. In *Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*. 25–31.
- [11] Qiang Cui, Yuyuan Tang, Shu Wu, and Liang Wang. 2019. Distance2Pre: Personalized Spatial Preference for Next Point-of-Interest Prediction. In *Advances in Knowledge Discovery and Data Mining*, Qiang Yang, Zhi-Hua Zhou, Zhiguo Gong, Min-Ling Zhang, and Sheng-Jun Huang (Eds.). Springer International Publishing, Cham, 289–301.
- [12] Tobias Eichinger, Felix Beierle, Robin Papke, Lucas Rebscher, Hong Chinh Tran, and Magdalena Trzeciak. 2019. On Gossip-based Information Dissemination in Pervasive Recommender Systems. *CoRR* abs/1908.05544 (2019). arXiv:1908.05544 <http://arxiv.org/abs/1908.05544>
- [13] David Enthoven and Zaid Al-Ars. 2021. An overview of federated deep learning privacy attacks and defensive strategies. *Federated Learning Systems* (2021), 173–196.
- [14] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. 2020. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948* (2020).
- [15] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized ranking metric embedding for next new poi recommendation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- [16] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quéma. 2009. Heterogeneous gossip. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 42–61.
- [17] Chen Gao, Chao Huang, Dongsheng Lin, Depeng Jin, and Yong Li. 2020. *DPLCF: Differentially Private Local Collaborative Filtering*. Association for Computing Machinery, New York, NY, USA, 961–970. <https://doi.org/10.1145/3397271.3401053>
- [18] Chen Gao, Chao Huang, Yue Yu, Huandong Wang, Yong Li, and Depeng Jin. 2019. Privacy-preserving cross-domain location recommendation. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 1 (2019), 1–21.
- [19] Lodovico Giarretta and Šarūnas Girdzijauskas. 2019. Gossip learning: Off the beaten path. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 1117–1124.
- [20] Neil Zhenqiang Gong and Bin Liu. 2016. You are who you know and how you behave: Attribute inference attacks via users’ social friends and behaviors. In *25th USENIX Security Symposium (USENIX Security 16)*. 979–995.
- [21] Rachid Guerraoui, Anne-Marie Kermarrec, Richeek Patra, Mahammad Valiyev, and Jingjing Wang. 2017. I Know Nothing about You But Here is What You Might Like. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 439–450. <https://doi.org/10.1109/DSN.2017.22>
- [22] Yeting Guo, Fang Liu, Zhiping Cai, Hui Zeng, Li Chen, Tongqing Zhou, and Nong Xiao. 2021. PREFER: Point-of-Interest REcommendation with Efficiency and Privacy-Preservation via Federated Edge Learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 1, Article 13 (mar 2021), 25 pages. <https://doi.org/10.1145/3448099>
- [23] Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. 2004. A scalable P2P recommender system based on distributed collaborative filtering. *Expert Syst. Appl.* 27 (2004), 203–210.
- [24] Rui Han, Shilin Li, Xiangwei Wang, Chi Harold Liu, Gaofeng Xin, and Lydia Y Chen. 2020. Accelerating gossip-based deep learning in heterogeneous edge computing platforms. *IEEE Transactions on Parallel and Distributed Systems* 32, 7 (2020), 1591–1602.

- [25] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. <https://doi.org/10.1145/2827872>
- [26] Tanzima Hashem, Rubaba Hasan, Flora Salim, and Mehnaz Tabassum Mahin. 2018. Crowd-enabled processing of trustworthy, privacy-enhanced and personalised location based services with quality guarantee. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–25.
- [27] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *CoRR* abs/1708.05031 (2017). arXiv:1708.05031 <http://arxiv.org/abs/1708.05031>
- [28] István Hegedűs, Gábor Danner, and Márk Jelasity. 2020. Decentralized Recommendation Based on Matrix Factorization: A Comparison of Gossip and Federated Learning. In *Machine Learning and Knowledge Discovery in Databases*, Peggy Cellier and Kurt Driessens (Eds.). Springer International Publishing, Cham, 317–332.
- [29] István Hegedűs and Márk Jelasity. 2017. Differentially private linear models for gossip learning through data perturbation. *OPEN JOURNAL OF INTERNET OF THINGS* 3, 1 (2017), 62–74.
- [30] István Hegedűs, Gábor Danner, and Márk Jelasity. 2021. Decentralized learning works: An empirical comparison of gossip learning and federated learning. *J. Parallel and Distrib. Comput.* 148 (02 2021), 109–124. <https://doi.org/10.1016/j.jpdc.2020.10.006>
- [31] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. 2021. Data poisoning attacks to deep learning based recommender systems. *arXiv preprint arXiv:2101.02644* (2021).
- [32] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. 2004. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In *Middleware 2004*, Hans-Arno Jacobsen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 79–98.
- [33] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badi Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. *CoRR* abs/1912.04977 (2019). arXiv:1912.04977 <http://arxiv.org/abs/1912.04977>
- [34] Jae Kyeong Kim, Hyea Kim, and Yoon Cho. 2004. A user-oriented contents recommendation system in peer-to-peer architecture. *Expert Systems with Applications* 34 (01 2004), 300–312. <https://doi.org/10.1016/j.eswa.2006.09.034>
- [35] Sungwook Kim, Jinsu Kim, Dongyoung Koo, Yuna Kim, Hyunsoo Yoon, and Junbum Shin. 2016. Efficient privacy-preserving matrix factorization via fully homomorphic encryption. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. 617–628.
- [36] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 426–434.
- [37] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>.
- [38] Chengxi Li, Gang Li, and Pramod K. Varshney. 2020. Decentralized Federated Learning via Mutual Knowledge Transfer. *CoRR* abs/2012.13063 (2020). arXiv:2012.13063 <https://arxiv.org/abs/2012.13063>
- [39] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. *arXiv e-prints*, Article arXiv:1705.09056 (May 2017), arXiv:1705.09056 pages. arXiv:1705.09056 [math.OC]
- [40] David Manheim and Scott Garrabrant. 2018. Categorizing variants of Goodhart’s Law. *arXiv preprint arXiv:1803.04585* (2018).
- [41] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. *CoRR* abs/1602.05629 (2016). arXiv:1602.05629 <http://arxiv.org/abs/1602.05629>
- [42] Frank McSherry and Ilya Mironov. 2009. Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge*

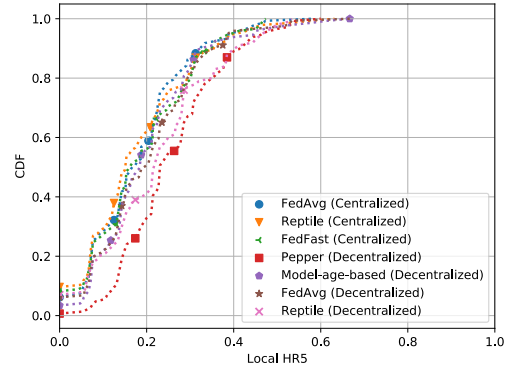
- Discovery and Data Mining* (Paris, France) (*KDD '09*). Association for Computing Machinery, New York, NY, USA, 627–636. <https://doi.org/10.1145/1557019.1557090>
- [43] Marcos Modenesi. 2020. OMNetPy: OMNeT++ meets python. <https://github.com/mmodenesi/omnetpy>
 - [44] Itishree Mohallick, Katrien De Moor, Özlem Özgöbek, and Jon Atle Gulla. 2018. Towards new privacy regulations in europe: Users’ privacy perception in recommender systems. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 319–330.
 - [45] Khalil Muhammad, Qinqin Wang, Diarmuid O’Reilly-Morgan, Elias Tragos, Barry Smyth, Neil Hurley, James Geraci, and Aonghus Lawlor. 2020. Fedfast: Going beyond average for faster training of federated recommender systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1234–1242.
 - [46] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. 2022. Local and central differential privacy for robustness and privacy in federated learning. *Proceedings of the 29th Network and Distributed System Security Symposium (NDSS 2022)* (2022).
 - [47] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.
 - [48] Róbert Ormándi, István Hegedüs, and Márk Jelasity. 2011. Efficient P2P Ensemble Learning with Linear Models on Fully Distributed Data. *CoRR* abs/1109.1396 (2011). arXiv:1109.1396 <http://arxiv.org/abs/1109.1396>
 - [49] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. 2008. TRIBLER: A Social-Based Peer-to-Peer System: Research Articles. *Concurr. Comput.: Pract. Exper.* 20, 2 (feb 2008), 127–138.
 - [50] Konstantin Pussep, Sebastian Kaune, Jonas Flick, and Ralf Steinmetz. 2009. A Peer-to-Peer Recommender System with Privacy Constraints. In *2009 International Conference on Complex, Intelligent and Software Intensive Systems*. 409–414. <https://doi.org/10.1109/CISIS.2009.32>
 - [51] Jiangcheng Qin and Baisong Liu. 2020. A Novel Privacy-Preserved Recommender System Framework based on Federated Learning. *CoRR* abs/2011.05614 (2020). arXiv:2011.05614 <https://arxiv.org/abs/2011.05614>
 - [52] Kurt Rohloff and David Cousins. 2014. A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU. In *Financial Cryptography Workshops*.
 - [53] Jeff Sandvig, Bamshad Mobasher, and Robin Burke. 2008. A Survey of Collaborative Recommendation and the Robustness of Model-Based Algorithms. *IEEE Data Eng. Bull.* 31 (01 2008), 3–13.
 - [54] Marco Scavuzzo, Amir Jalalirad, Michael Sprague, and Catalin Capota. 2019. A Simple and Efficient Federated Recommender System. <https://doi.org/10.1145/3365109.3368788>
 - [55] Johannes Schneider and Michail Vlachos. 2019. Mass Personalization of Deep Learning. *CoRR* abs/1909.02803 (2019). arXiv:1909.02803 <http://arxiv.org/abs/1909.02803>
 - [56] Hyejin Shin, Sungwook Kim, Junbum Shin, and Xiaokui Xiao. 2018. Privacy Enhanced Matrix Factorization for Recommendation with Local Differential Privacy. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1770–1782. <https://doi.org/10.1109/TKDE.2018.2805356>
 - [57] Jason Silverstein. [n.d.]. *Hundreds of millions of Facebook user records were exposed on Amazon cloud server*. <https://www.forbes.com/sites/daveywinder/2020/08/19/massive-data-leak235-million-instagram-tiktok-and-youtube-user-profiles-exposed/>
 - [58] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. 2019. Can You Really Backdoor Federated Learning? *CoRR* abs/1911.07963 (2019). arXiv:1911.07963 <http://arxiv.org/abs/1911.07963>
 - [59] WANG TianXiang, ZhongLong ZHENG, TANG ChangBing, and PENG Hao. 2019. Aggregation rules based on stochastic gradient descent in byzantine consensus. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*. IEEE, 317–324.
 - [60] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *European Symposium on Research in Computer Security*. Springer, 480–501.
 - [61] Andras Varga and Rudolf Hornig. 2020. OMNeT++ Discrete Event Simulator. <https://github.com/omnetpp/omnetpp>
 - [62] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris S. Papailiopoulos. 2020. Attack of the Tails: Yes, You Really Can Backdoor Federated Learning. *CoRR* abs/2007.05084 (2020). arXiv:2007.05084 <https://arxiv.org/abs/2007.05084>

- [63] Jun Wang, Qiang Tang, Afonso Arriaga, and Peter Y. A. Ryan. 2019. Novel Collaborative Filtering Recommender Friendly to Privacy Protection. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (Macao, China) (IJCAI'19)*. AAAI Press, 4809–4815.
- [64] Qinyong Wang, Hongzhi Yin, Tong Chen, Junliang Yu, Alexander Zhou, and Xiangliang Zhang. 2021. Fast-adapting and Privacy-preserving Federated Recommender System. *CoRR* abs/2104.00919 (2021). arXiv:2104.00919 <https://arxiv.org/abs/2104.00919>
- [65] Davey Winder. [n.d.]. 235 Million Instagram, TikTok And YouTube User Profiles Exposed In Massive Data Leak. <https://www.forbes.com/sites/daveywinder/2020/08/19/massive-data-leak235-million-instagram-tiktok-and-youtube-user-profiles-exposed/>
- [66] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *CoRR* abs/1902.04885 (2019). arXiv:1902.04885 <http://arxiv.org/abs/1902.04885>
- [67] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*. PMLR, 5650–5659.

A Individual Performance Comparison

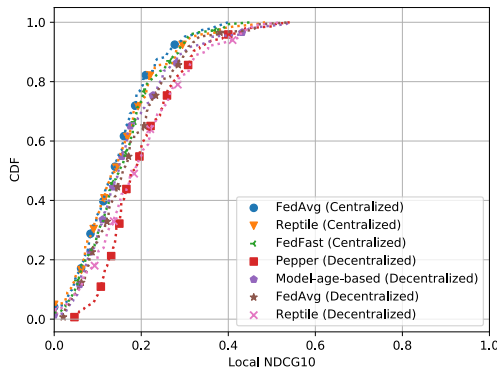


(a) Local Hit Ratio@10 cumulative distribution function.

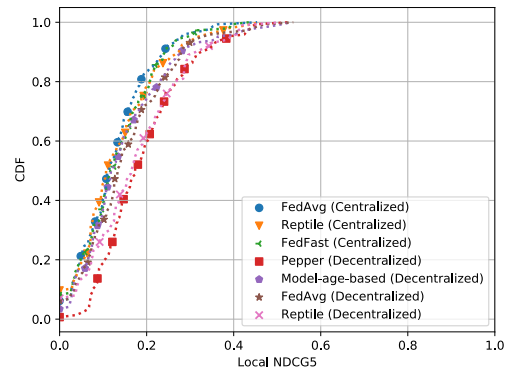


(b) Local Hit Ratio@5 cumulative distribution function.

Figure 10: Top-K recommendation quality distribution comparison on Foursquare-NYC (GMF, K = 10 and K = 5).

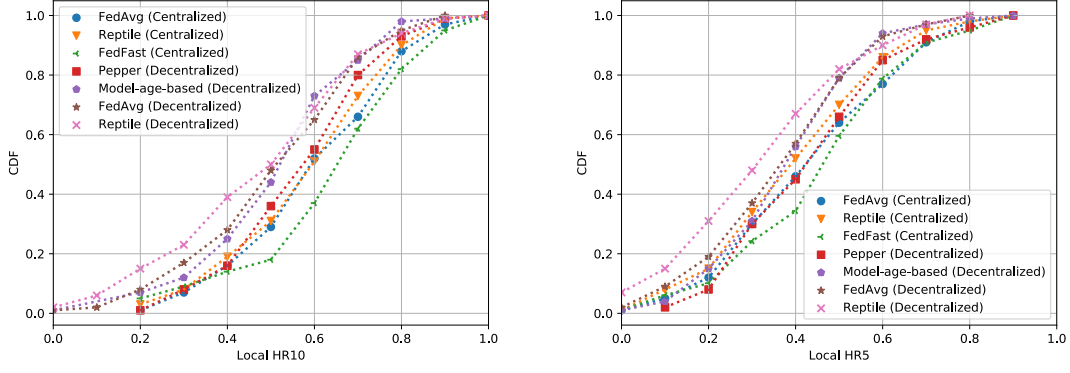


(a) Local NDCG@10 cumulative distribution function.



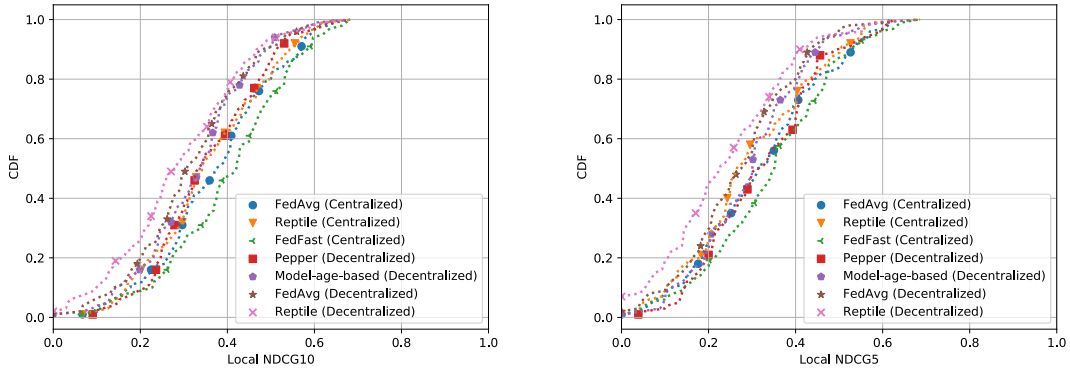
(b) Local NDCG@5 cumulative distribution function.

Figure 11: Top-K recommendation quality distribution comparison on Foursquare-NYC (GMF, K = 10 and K = 5).



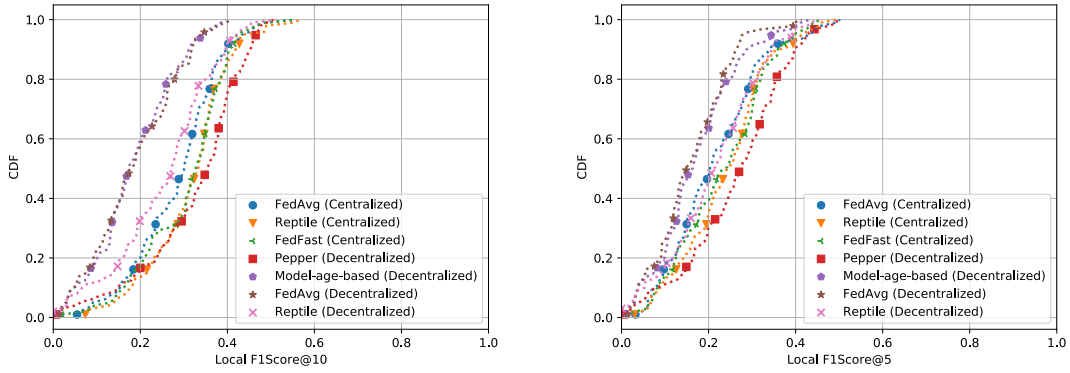
(a) Local Hit Ratio@10 cumulative distribution function. (b) Local Hit Ratio@5 cumulative distribution function.

Figure 12: Top-K recommendation quality distribution comparison on MovieLens-100K (GMF, $K = 10$ and $K = 5$).



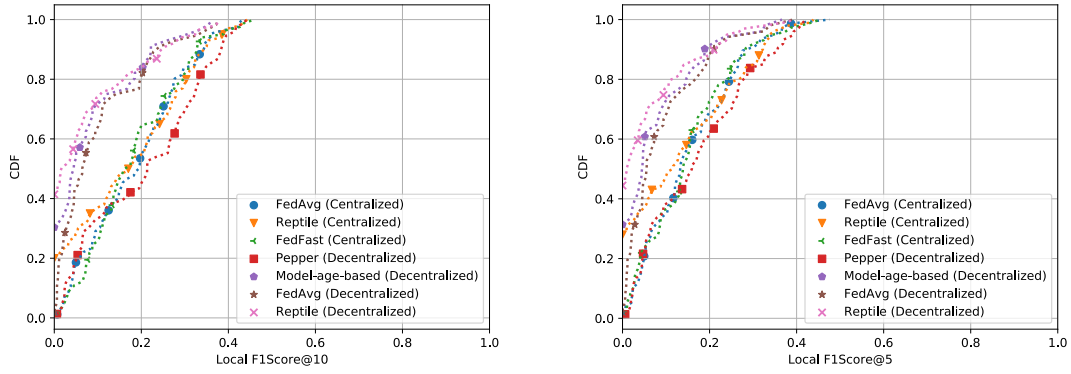
(a) Local NDCG@10 cumulative distribution function. (b) Local NDCG@5 cumulative distribution function.

Figure 13: Top-K recommendation quality distribution comparison on MovieLens-100K (GMF, $K = 10$ and $K = 5$).



(a) Local F1 score@10 cumulative distribution function. (b) Local F1 score@5 cumulative distribution function.

Figure 14: Top-K F1-Score cumulative distribution function on Foursquare-NYC (PRME-G, $K = 10$ and $K = 5$).



(a) Local F1 score@10 cumulative distribution function. (b) Local F1 score@5 cumulative distribution function.

Figure 15: Top-K F1-Score cumulative distribution function on Gowalla-NYC (PRME-G, K = 10 and K = 5).