Task Assignment with Efficient Federated Preference Learning in Spatial Crowdsourcing

Hao Miao, Xiaolong Zhong, Jiaxin Liu, Yan Zhao, Xiangyu Zhao, Weizhu Qian, Kai Zheng, and Christian S. Jensen, *Fellow, IEEE*

Abstract—Spatial Crowdsourcing (SC) is finding widespread application in today's online world. As we have transitioned from desktop crowdsourcing applications (e.g., Wikipedia) to SC applications (e.g., Uber), there is a sense that SC systems must not only provide effective task assignment but also need to ensure privacy. To achieve these often-conflicting objectives, we propose a framework, Task Assignment with Federated Preference Learning, that performs task assignment based on worker preferences while keeping the data decentralized and private in each platform center (e.g., each delivery center of an SC company). The framework includes a federated preference learning phase and a task assignment phase. Specifically, in the first phase, we build a local preference model for each platform center based on historical data. We provide means of horizontal federated learning that makes it possible to collaboratively train these local preference models under the orchestration of a central server. Specifically, we provide a practical method that accelerates federated preference learning based on stochastic controlled averaging and achieves low communication costs while considering data heterogeneity among clients. The task assignment phase aims to achieve effective and efficient task assignment by considering workers' preferences. Extensive evaluations on real data offer insight into the effectiveness and efficiency of the paper's proposals.

Index Terms—preference; task assignment; federated learning; spatial crowdsourcing.

1 Introduction

The last decade has witnessed substantial advances in Spatial Crowdsourcing (SC), which enables people on the move to serve as multi-modal sensors that perform a variety of location-based tasks [1], [2], [3], [4], [5], [6]. The rising concerns of privacy [7], [8], [9], [10], [11], [12], [13] and efficiency [14], [15], [16], [17], [18] raised by SC are continuously attracting attention. In particular, privacy is desirable in SC. In order to achieve effective SC services, workers or platform centers (e.g., delivery centers of an SC company) are usually required to disclose their raw information (e.g., workers' locations and historical data). However, it is dangerous that real data can be used by a malicious third party. Thus, people will be unwilling to hand over their data to an SC platform, which leads to low worker participation and even worker churn.

Previous studies on privacy protection in SC focus mainly on protecting the location information of workers or tasks [7], [9], [13], [19], [20], data federation [21], [22], and on secure distance computation [8], [10], [12]. However, these studies do not take into account worker preferences, which are essential for the operation of an SC platform. These are needed in order to assign workers appropriate tasks, which is key to ensuring continuous worker participation and satisfaction. In contrast, when a worker is assigned an inappropriate task, the worker may complete the task with low quality or may even sabotage the task, which impacts the SC platforms negatively. Considering worker preferences is thus crucial in SC. Recently, a number of studies have explored worker preference in SC [14], [16], [17], [18], [23]. Zhao et al. [18] model different workers' preferences for different categories of tasks in different time slots with a three-dimensional tensor and fill-in missing entries in tensors based on workers' task-performance histories and context matrices. Another study [14] uses historical task-performance data to maximize the mutual information among workers in order to learn informative representation vectors of groups and further learn group preferences. These studies do not consider privacy when modeling worker preferences. Instead, they transmit raw data directly to the SC platform and train a preference model in a centralized manner. In practice, data transfer between platform centers is cumbersome, and platform centers may be unwilling to share their data with other centers. In this kind of setting, task assignment often fails to achieve effective task assignment. To address the above challenges, we provide a preference-driven task assignment solution that integrates federated learning to protect raw data while using workers' preferences in task assignment. Federated Learning (FL) is a machine learning approach where many clients (e.g., mobile devices, organizations, or platforms) collaboratively

- *Hao Miao, Yan Zhao, Weizhu Qian, and Christian S. Jensen are with the department of Computer Science, Aalborg University, Aalborg 9220, Denmark.*
  E-mail: {haom, yanz, wqian, csj}@cs.aau.dk
- *Xiaolong Zhong, Jiaxin Liu are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China.*
  E-mail: {xiaolongzhong2000, jiaxinliu1999}@std.uestc.edu.cn
- *Xiangyu Zhao is with the School of Data Science, City University of Hong Kong, Hong Kong, China.*
  E-mail: xianzhao@cityu.edu.hk
- *Kai Zheng is with Yangtze Delta Region Institute (Quzhou), School of Computer Science and Engineering, and Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China.*
  E-mail: zhengkai@uestc.edu.cn

Yan Zhao is the corresponding author.

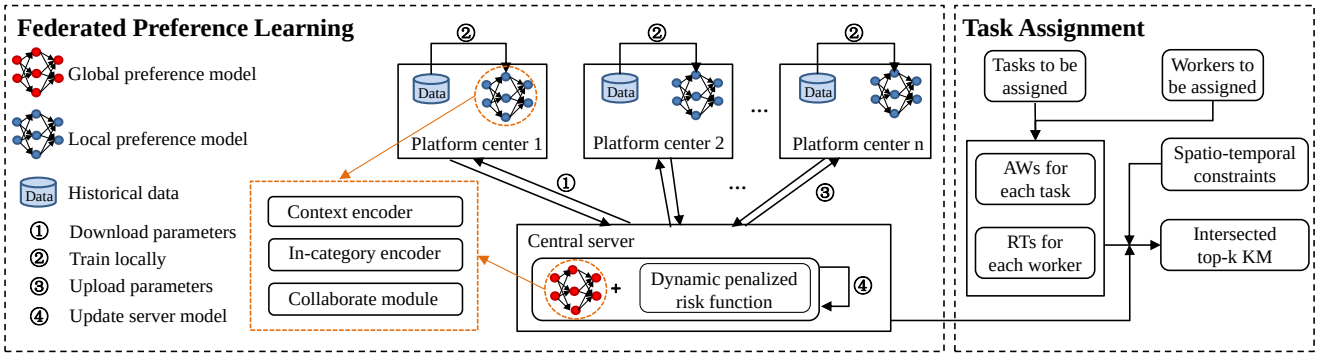


Fig. 1. Framework Overview

train a model under the orchestration of a central server while keeping the training data decentralized. It embodies the principles of focused collection and data minimization, which can mitigate the systemic privacy risks and costs associated with traditional, centralized machine learning [24]. Moreover, FL addresses privacy concerns associated with the sharing of sensitive data with a central server, reducing the risk of data leakage. MaMahan et al. [25] show that federated learning offers many practical privacy benefits.

Filling the gap between existing research focusing on privacy issues and methods ignoring worker preferences, we propose a two-phase SC framework, namely Task Assignment with Federated Preference Learning (TA-FPL), which encompasses a federated preference learning and task assignment, as shown in Figure 1. In the first phase, a federated preference model is built for each local center, and all local models are combined at a central server. For each preference model, each platform center first downloads parameters from the central server and utilizes local historical data to learn workers' preferences for all task categories by using a context encoder, an in-category encoder, and a collaboration module. Then, the central server receives the updated model parameters and aggregates them to update its global preference model. For the aggregation, we introduce a dynamic penalized risk function to achieve better performance. In the task assignment phase, we first calculate the Available Workers (AWs) for each task and the Reachable Tasks (RTs) for each worker based on spatio-temporal constraints. Then we propose an intersected top-k Kuhn Munkras (KM) algorithm, which considers the top-k AWs and RTs for each task and worker simultaneously, to achieve effective and efficient task assignment based on workers' preferences.

A preliminary version of this study [26] applies stochastic gradient descent (SGD) to optimize the federated preference learning, where clients calculate the gradients of the loss over their local data, and the central server then aggregates the gradients to update parameters. This approach requires a large number of training rounds to produce superior models, thus calling for a more communication-efficient optimization method. In real-world scenarios, the network connections between the server and clients can be unreliable due to many factors, and only a small subset of clients may be available at a given time. In addition, the distributions of the local data held by different clients are often not independent and identically distributed (Non-IID) [27], which implies large heterogeneity.

To address these issues, we first optimized the training of preference learning in TA-FPL with FederatedAveraging (FedAvg) [25]. Specifically, we select a C-fraction of the clients in each training round and update the selected clients locally multiple times by involving a server that performs model averaging. However, platform centers are generally heterogeneous (non-IID) and complex across different real-world SC settings. For instance, workers in one platform center may need to identify an attractive location for advertisements, while workers in another platform center may perform real-time traffic reporting. Such heterogeneous cases impact FedAvg negatively [27], [28], causing slow and unstable convergence due to client drift when updating clients. Client drift has the effect that the optimums of the loss functions of clients vary from client to client and from that of the server. To contend with client drift, we incorporate the idea of stochastic controlled averaging into the federated preference learning phase of TA-FPL, which ables overcoming heterogeneity and enables much faster convergence. More concretely, we consider the difference between the update direction for the server model c and each client c_i as the estimate of the client drift, which is then used to correct the local updates during local preference model training.

The major value-added extensions over our preliminary study [26] are three-fold.

1) We analyze in depth a limitation of our previous federated training method that fails to solve non-IID SC data across clients, thus improving the efficiency of federated preference learning.

2) We provide a drift-correct federated training (DCFT) method that optimizes the preference learning by stochastic controlled averaging while contending with SC data heterogeneity, thereby achieving efficient communication between clients and the server.

3) We report on extensive experiments on a real-world dataset, offering evidence of the efficiency of the paper's proposals, which reduces the convergence time of preference learning by up to 45% compared to that of a federated preference learning (FPL) model.

The remainder of the paper is organized as follows. Preliminary concepts, notation, and a problem statement are provided in Section 2. Next, federated preference learning

and intersected Top-k KM task allocation are covered in Section 3. We report on an experimental study in Section 4. Section 5 surveys related work, and Section 6 offers conclusions.

2 Preliminaries and Problem Definition

We proceed to present preliminaries and define the problem addressed. Table 1 lists the notations used throughout the paper.

TABLE 1
Summary of Notation

Symbol	Definition	Symbol	Definition
pc	Platform center	$w.l$	Current location of w
$pc.l$	Location of pc	$w.pc$	Platform center of w
$pc.W$	A worker set of pc	$w.r$	Reachable radius of w
s	Spatial task	$w.speed$	Speed of worker w
$s.l$	Location of s	$w.S$	A set of historical tasks of w
$s.p$	Publication time of s	A	A spatial task assignment
$s.e$	Expiration time of s	$A.S$	Allocated task set of S
$s.c$	Category of s	$\mathbb{A}$	Task assignment set
w	Worker		

Definition 1 (Platform Center). A platform center, denoted by $pc = (l, W)$, has a location $pc.l$ and a set of workers $pc.W$.

Definition 2 (Spatial Task). A spatial task, denoted by $s = (l, p, e, c)$, has a location $s.l$, a publication time $s.p$, an expiration time $s.e$, and a category $s.c$.

With SC, the query of a spatial task s can be answered only if a worker is physically located at that location $s.l$ and be completed only if a worker arrives at $s.l$ before its deadline $s.e$. Note that with the single task assignment mode [29], an SC server should allocate each spatial task to only one worker at a time.

Definition 3 (Worker). A worker, denoted by $w = (l, pc, r, speed, S)$, has a location $w.l$, a center $w.pc$ that the worker w works for, a reachable radius $w.r$, a movement speed $w.speed$, and a set of historical tasks $w.S$.

The reachable range of worker w is a circle with $w.l$ as the center and $w.r$ as the radius, within which w can accept assignments. In our work, a worker can handle only one task at a certain time instance and belongs to a single center, which is reasonable in practice.

Definition 4 (Spatial Task Assignment). Given a set of workers $W = \{w_1, w_2, ..., w_{|W|}\}$, a set of tasks $S = \{s_1, s_2, ..., s_{|S|}\}$, and a set of platform centers $PC = \{pc_1, pc_2, ..., pc_{|PC|}\}$, we define A as the spatial task assignment which consists of a set of tuples of form (pc, w, s), where a spatial task s is assigned to worker w who works for pc, satisfying all the workers' and tasks' spatial-temporal constraints.

Based on the above definitions, a formal problem definition is as follows.

Preference-driven Task Assignment with Privacy Protection. Given a set of centers PC with private local data (i.e., workers' historical task records and workers' locations), a set of online workers W, and a set of tasks S at the current timestamp, our problem is to find an optimal task assignment A_{opt} that maximizes the total number of assigned tasks, i.e., $\forall A_i \in \mathbb{A}\ (|A_{opt}.S| \geq |A_i.S|)$, by considering workers' preferences and protecting the privacy of each platform center, where $\mathbb{A}$ denotes all possible assignments and $A_i.S$ denotes the tasks of task assignment A_i.

Note that the proposed TA-FPL encompasses a federated preference learning and a task assignment phase. Since we aim to learn workers' preferences in the federated learning phase, which also enables privacy, we do not define preference and privacy in the problem statement.

3 Algorithm

In this section, we introduce the details of the proposed framework TA-FPL, which contains a federated preference learning phase described in Section 3.1 and a preference-driven task assignment phase described in Section 3.2. In the federated preference learning phase, a local preference model is proposed to model workers' preferences for local platform centers, which contains three modules, i.e., a context encoder, an in-category encoder, and a collaboration module. We combine the above-mentioned modules to predict the local workers' preferences. Then, we introduce a novel federated training process to update the parameters of the central server's model in order to get the global workers' preferences. In the preference-driven task assignment phase, we introduce an Intersected top-k KM algorithm to find a suitable task assignment.

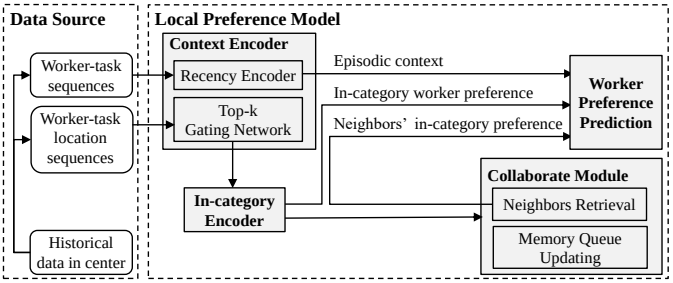


Fig. 2. Local Preference Model

3.1 Federated Preference Learning

In this section, we introduce the local preference model and federated model training, respectively. Noted that the central server's model and local platform centers' models share the same model framework, as shown in Figure 2.

3.1.1 Local Platform Center Preference Modeling

In this section, we will introduce how to use each platform center's local data to model workers' preferences.

Given a set of workers W over a set of task locations L from a set of task categories C, a task record can be denoted as a tuple $s_i = (l_i, c_i)$, where i is the index of the task in a worker's historical task record sequence, l_i is the location of task s_i that the worker interacts with, and c_i is the task's category. Different tasks may be located in the same location. A sequence of N tasks from worker w is denoted as $S_w = \{s_1, s_2, ..., s_N\}$, which is ordered according to the chronological order of tasks. $S_w^c = \{s_1^c, s_2^c, ..., s_T^c\}$ represents the subsequence of S_w under category c, where T is the number of task records in this subsequence and task records

in S_w^c are still in chronological order. Given S_w of worker w, the goal of our preference model is to predict this worker's preferences of task location and task category in the near future.

As shown in Figure 2, the local preference model is composed of three modules: a context encoder, an in-category encoder, and a collaboration module. First, to model worker preferences under a task category, a task sequence can be divided into multiple subsequences according to task categories, and each subsequence contains tasks of the same category. The in-category encoder, utilizing *SelfAttention* [30], [31], is used to model in-category transition patterns of task-to-task in the subsequences. Second, the context encoder contains episodic context and category context. Based on the categories of recent task records, the context encoder utilizes $SelfAttention$ to predict the next category, thus helping determine a worker's preference for the next task location. To get the episodic context, the context encoder models the task-to-task transition patterns among recent task records with another $SelfAttention$. Third, due to the sparsity of observations in individual worker's task records, we retrieve workers with similar in-category preferences to the target worker based on the context encoder's next category prediction. Finally, task location and task category prediction are made based on the episodic context, the in-category worker preferences, and neighboring workers' in-category preferences. We will then introduce the details of each component of this preference model.

Context Encoder. The context encoder is designed to obtain both category and episodic context for the worker preference prediction. To decide which in-category worker preferences should be used, the category of the next task is predicted. By using a top-k gating network and a recency encoder, we can obtain the category context and the episodic context in recent task records, respectively.

For the top-k gating network, we take the categories of recent tasks as input. Through the input category embedding layer, the categories of the most recent M task records can be projected into vectors $Z = [e^z_{c_{N-M}}, ..., e^z_{c_N}]$, and the relative positions of recent task records $[M, ..., 1]$ can be similarly projected into $P^z = [P^z_M, ..., P^z_1]$ with the position embedding layer. Then a *SelfAttention* network [30], [31], which is composed of n_l layers of a multi-head attention block and a Fully Connected (FC) network block, transforms the category vectors $H^0 = Z + P^z$ into hidden representations $H^{n_l} = [h^z_1, ..., h^z_T]$. For the ith attention head, the input latent states H^j will be transformed as Eq. 1, where projection matrices W^Q_i, W^K_i, W^V_i are learnable parameters and LN is a layer normalization network. We use $h^z = h^z_T$ to summarize the category information of recent task records.

$$\begin{aligned} &A^j_i = Attention(H^j W^Q_i, H^j W^K_i, H^j W^V_i), \\ &Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_a / n_h}}), \\ &H^{j+1} = LN(H^j + FC^j(A^j)), \end{aligned} \tag{1}$$

Then we feed h^z into the output category embedding layer and a softmax layer. The top-k gating network generates a probability distribution over all task categories:

$$p(\hat{c}_{N+1} = j) \propto exp(\langle h^z, e^z_j \rangle), \tag{2}$$

where $p(\hat{c}_{N+1} = j)$ represents the probability of category j being the category of the next task location and e^z_j is the category embedding of category j. Considering the uncertainty in the prediction of the next category, the gating network selects top-k most probable categories according to Eq. 3:

$$\{c_j\}^k_{j=1} = \underset{j'}{argtopk}(\{p(\hat{c}_{N+1} = j')\}^{|C|}_{j'=1}), \tag{3}$$

where $c_j \in C$.

The recency encoder is utilized to infer the episodic context from recent task records. Because of the continuity between tasks, the recent records of tasks reflect the ongoing intent of the next task location. For the recency encoder, we take the most recent M records of task locations $[l_{N-M}, ..., l_N]$ in the original sequence S as the input, which is projected through the input task location embedding layer into vectors $X^r = [e_{l_{N-M}}, ..., e_{l_N}]$. The relative positions of recent task records $[M, ..., 1]$ are projected into $P^r = [P^r_M, ..., P^r_1]$ through the position embedding layer similarly. Then, we use the $SelfAttention$ network to transform vectors $X^r + P^r$ into hidden states, and the hidden state of the last task record is used to represent the inferred episodic context of h^r:

$$h^r = SelfAttention(X^r + P^r) \tag{4}$$

The recency encoder checks the recent task locations beyond specific categories and provides a complementary view of the next task location besides the category context.

In-category Encoder. With respect to the k predicted categories calculated in Eq. 3, we can calculate the corresponding in-category worker preferences to predict the next task location, i.e., the hidden representations $\{h^{c_j}\}^k_{j=1}$ from the in-category encoder selected. Without loss of generality, we take the encoding process for a task subsequence of category c as an example. The corresponding task location subsequence $[l^c_1, ..., l^c_T]$ is projected through the input task location embedding layer E_{in} into a set of dense vectors $X^c = [e^c_{l_1}, ..., e^c_{l_T}]$. The relative positions $[T, ..., 1]$ of these records to the next task location are projected through the position embedding layer P into $P^c = [P^c_T, ..., P^c_1]$. Taking the dense vectors $X^c + P^c$ as input, the *SelfAttention* network outputs the representations of the worker preferences in this category h^c:

$$h^c = SelfAttention(X^c + P^c) \tag{5}$$

Collaboration Module. By using collaborative learning among neighboring workers with similar preferences, the sparsity issue can be mitigated. To predict the next task location, we combine the neighboring workers' information based on their similarities obtained with in-category subsequences. In collaboration module, a memory tensor Mem is used to record workers' in-category preferences, storing the latent states of the last F workers for each category c in chronological order.

Using the target worker's in-category preferences h^c_w under category c, the reading operation of Mem can be performed as follows. First, we compute the similarity of the preferences under category c between worker w and worker i as:

$$sim(h^c_w, h^c_i) \propto exp(\langle h^c_w, h^c_i \rangle) \tag{6}$$

Then we choose the top-f similar workers as the neighbors and take a weighted sum of their representations as

the neighborhood representation for the next task location prediction:

$$h^f = \sum_{i'}^{top-f} sim(h_w^c, h_{i'}^c) h_{i'}^c \tag{7}$$

We initialize Mem for the writing operation. We then update it with the latest worker's in-category preference representations h^c, generated from the in-category encoder. Thus, the memory tensor is a queue that pushes the most recently served workers' representations of category c while dynamically popping out the representations of workers who have been inactive for a long time.

Preference Prediction. Based on the three modules above, we can get the worker preference of the task location and task category. We calculate the preference score of task category c_j with Eq. 2, $score(\hat{c}_j) = score(\hat{c}_{N+1} = c_j)$:

$$score(\hat{c}_j) = exp(\langle h^z, e_j^z \rangle), \tag{8}$$

where h_z represents the category information of recent task records and e_j^z is the embedding of category j.

In order to predict the worker preference of task location l_i, $score(\hat{l}_i) = score(\hat{l}_{N+1} = l_i)$, we use the mixture of obtained representations as the worker representation. Based on in-category worker preferences of the predicted top-k categories h^{c_j}, neighboring workers' in-category preferences representation h^{f_j}, and the episodic context inferred from the most recent task location records h^r, we concatenate these three representations for each of k categories and project the concatenated representations into the output task location embedding space with a fully connected (FC) network layer. Thus, the worker representation h_j for category j is:

$$h_j = FC(h^r \oplus h^{c_j} \oplus h^{f_j}), \tag{9}$$

where $j = 1, 2, ..., k$ and $\oplus$ is the concatenation operation. Then we use inner product and softmax and get the worker preference for task location l_i under the premise of category j:

$$score_j(\hat{l}_i) = softmax(\langle h_j, E_{out} \rangle), \tag{10}$$

where E_{out} represents the task locations' embeddings. Furthermore, when we consider all top-k categories, the worker's preference for task location l_i can be calculated according to Eq. 11:

$$score(\hat{l}_i) = \sum_{j=1}^{k} score_j(\hat{l}_i) score(\hat{c}_j) \tag{11}$$

3.1.2 Federated Preference Model Training

In this section, we will introduce the global loss function and model parameters updating in federated training way.

Loss Function. The loss function of each platform center is composed of two parts: the loss of task location prediction and the loss of task category prediction.

For the loss of task location prediction, we apply the negative sampling trick, which randomly samples N_s negative task locations according to their popularity in training datasets for each positive instances [32]. The loss of task location prediction is:

$$L_{loc} = -\sum_{l=1}^{N_s+1} \delta(l_{N+1} = l) \log p(\hat{l}_{N+1} = l) \tag{12}$$

$$p(\hat{l}_{N+1} = l) = softmax(score(\hat{l}_{N+1} = l)), \tag{13}$$

where $\delta(\cdot)$ is the indicator function, l_{N+1} is the ground-truth task location, and $\hat{l}_{N+1}$ is the model's prediction. Likewise, we compute the loss of all task locations' categories:

$$L_{cate} = -\sum_{j=1}^{C} \delta(c_{N+1} = j) \log p(\hat{c}_{N+1} = j), \tag{14}$$

where $p(\hat{c}_{N+1} = j)$ is computed by Eq. 2. We compute the joint loss as follows:

$$L = \lambda \times L_{loc} + (1 - \lambda) \times L_{cate}, \tag{15}$$

where λ is a hyper-parameter to control the weights.

Given a central server that can transmit and receive messages from m sampled platform centers, platform center pc_k consists of N_k training instances. The whole federated loss function $l(\theta)$ is:

$$l(\theta) = \frac{1}{m} \sum_{k=1}^{m} L_k(\theta), \tag{16}$$

where $L_k(\theta)$ is the empirical loss (i.e. Eq. 15) of platform center pc_k, and θ are the parameters of our preference network.

Federated Training. Initially, we employ federated training, as shown in Algorithm 1, to transmit local model parameters of each platform center to the central server for the purpose of privacy protection. In communication round $t \in [1, ..., T]$, a subset of platform centers $P_t \subset \{pc_1, ..., pc_N\}$ is active. The central server transmits its current model θ^{t-1} to these platform centers (lines 2–3). Each active platform center then optimizes a local empirical risk objective, which is the sum of its local empirical loss and a dynamic penalized risk function (line 19). Besides, each active platform center computes its local gradient to satisfy local optima condition (line 21), then transmits the updated parameters to the central server (line 7). For unselected platform centers, they do not update their models (lines 8–10). Finally, the central server updates its state h^t, which implies whether they converge to a point that turns out to be a stationary point of the global risk [33], and model parameters θ^t (lines 11–12).

Algorithm 1: Federated Preference Model Training

Input: T, θ^0, α, θ_k^0
1 **for** each round t in $[1, .., T]$ **do**
2 Sample platform centers P_t;
3 Server transmits θ^{t-1} to each selected platform center;
4 **for** each platform center $pc_k \in P_t$ **do**
5 $\theta_k^t \leftarrow \arg\min_{\theta} L_k(\theta) - \langle \nabla L_k(\theta_k^{t-1}), \theta \rangle + \frac{\alpha}{2} \|\theta - \theta^{t-1}\|^2$;
6 $\nabla L_k(\theta_k^t) \leftarrow \nabla L_k(\theta_k^{t-1}) - \alpha(\theta_k^t - \theta^{t-1})$;
7 Transmit platform center model θ_k^t to central server;
8 **for** each platform center $pc_k \notin P_t$ **do**
9 $\theta_k^t \leftarrow \theta_k^{t-1}$;
10 $\nabla L_k(\theta_k^t) \leftarrow \nabla L_k(\theta_k^{t-1})$;
11 $h^t \leftarrow h^{t-1} - \frac{\alpha}{m}(\sum_{k \in P_t} \theta_k^t - \theta^{t-1})$;
12 $\theta^t \leftarrow (\frac{1}{|P_t|} \sum_{k \in P_t} \theta_k^t) - \frac{1}{\alpha} h^t$;

Algorithm 2: FedAvg based Preference Model Training

Input: T, θ^0, α, θ_k^0, E
1 **for** each round t in $[1, ..., T]$ **do**
2 Sample $m(m \geq 1)$ platform centers P_t;
3 Server transmits θ^{t-1} to each selected platform center;
4 **for** each client $k \in P_t$ in parallel **do**
5 $\theta_k^t \leftarrow ClientUpdate(k, \theta_k^{t-1})$;
6 **for** each platform center $pc_k \notin P_t$ **do**
7 $\theta_k^t \leftarrow \theta_k^{t-1}$;
8 $\nabla L_k(\theta_k^t) \leftarrow \nabla L_k(\theta_k^{t-1})$;
9 $h^t = h^{t-1} - \frac{\alpha}{m}(\sum_{k \in P_t} \theta_k^t - \theta^{t-1})$;
10 $\theta^t = (\frac{1}{|P_t|} \sum_{k \in P_t} \theta_k^t) - \frac{1}{\alpha} h^t$;
11 $ClientUpdate(k, \theta)$:
12 $\mathcal{B} \leftarrow$ split data stored in client k into batches of size B;
13 **for** each local epoch i from 1 to E **do**
14 **for** batch $b \in \mathcal{B}$ **do**
15 $\theta_{k,i}^t \leftarrow \arg\min_{\theta} L_{k,i}(\theta) - \langle \nabla L_{k,i}(\theta_{k,i}^{t-1}), \theta \rangle + \frac{\alpha}{2} \|\theta - \theta^{t-1}\|^2$;
16 $\nabla L_{k,i}(\theta_{k,i}^t) \leftarrow \nabla L_{k,i}(\theta_{k,i}^{t-1}) - \alpha(\theta_{k,i}^t - \theta^{t-1})$;
17 $\theta_k^t \leftarrow \theta_{k,E}^t$;
18 Transmit platform center model θ_k^t to central server;
19 return θ_k^t.

However, previous studies [25] show that federated training requires large numbers of rounds to train a good model which incurs high communication overheads. To tackle this problem, we first train the preference learning with FedAvg, where a selected subset of clients first update their local models multiple times and where the server aggregates the average gradients from the selected clients.

FedAvg based Training. We employ FedAvg to replace the federated training. As shown in Algorithm 2, we transmit local model parameters of selected platform centers to the central server by performing local training on these selected clients to reduce the computational load of the central server. Similarly, in a communication round $t \in [1, ..., T]$, only a subset of all N platform centers $P_t \subset \{pc_1, ..., pc_N\}$ are active, where N is the number of all platform centers. The central server allocates its current model θ^{t-1} to these platform centers (lines 2–3). But differently with federated training, each active platform center locally takes several steps of gradient descent on the current model using its local data with a local empirical risk objective, which contains a local empirical loss and a dynamic penalized risk function (line 6 and lines 11–19). Platform centers that were not selected do not update their models (lines 8–10). Finally, the central server updates its state h^t (which indicates whether they converge to a stationary point of the global risk [33]) and model parameters θ^t (lines 12–13).

Drift-Corrected Federated Training (DCFT). In some SC applications, the data held by different platform centers may vary considerably. Put differently, the data is heterogeneous and non-IID. For example, workers at one platform center may primarily deliver advertisements, while workers at another platform center may perform mostly real-time traffic reporting. Studies [27], [28] show that heterogeneity of SC

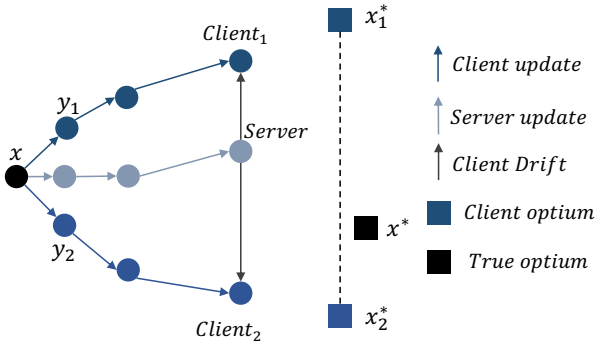


Fig. 3. Client-drift in FedAvg

data has adverse performance effects on FedAvg because it causes update drift when performing training on the clients. To illustrate, Fig. 3 shows two clients with three updating steps for example to show the client drift. For each client, the local updates move towards its individual optimum x_i^*, while the server updates move towards $\frac{1}{N}\sum_i x_i^*$ instead of the true optimum x^*, and $\frac{1}{N}\sum_i x_i^*$ may deviate substantially from x^*. Such situations may require more training rounds to achieve a good model, which degrades the performance of TA-FPL.

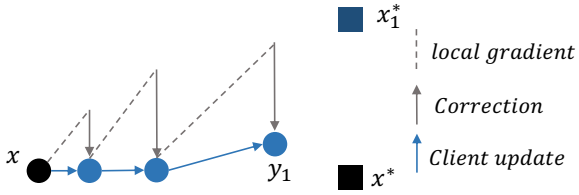


Fig. 4. Illustration of Drift-Correction

To contend with this issue, we propose so-called drift-corrected federated training that incorporates the idea of stochastically controlled averaging to optimize the preference learning. In other settings, this has proven to be able to enable stable training and faster convergence [28]. Specifically, we consider heterogeneity as 'client-variance' in the updates across different clients and aim to reduce the client-variance during the federated training. As illustrated in Fig. 4, when updating the model on a single client, the local gradient (dashed line) moves towards x_1^*, but we correct the update direction to ensure that the update points to the true optimum x^* by means of the correction term $(c - c_i)$, where c is server control variate (a state of the update direction) and c_i is the i-th client control variate. We initialize c and c_i to zero.

The training procedure of stochastic controlled averaging is similar to that of FedAvg, which has an additional drift correction (i.e., $c - c_i$). More concretely, in communication round $t \in [1, ..., T]$, we select a subset of platform centers $P_t \subset \{pc_1, ..., pc_N\}$, and the server transmits its current model θ^{t-1} to these active platform centers (lines 3 – 4), as shown in Algorithm 2. Each active platform center locally updates the current model using its local data with a local empirical risk objective with drift correction (lines 15 – 25). Meanwhile, the local control variate c_i is also updated

as follows,

$$c_i^+ = c_i - c + \frac{1}{|\mathcal{B}|\alpha}(\theta - \theta_k^t), \tag{17}$$

where the previously computed gradients are reused to update the control variate (line 22). Finally, the central server updates its state h_t, parameter θ^t, and control variate c to make the model move towards the optimum of the global model.

The difference between the original federated training and the optimized one (i.e., DCFT) is that DCFT considers data heterogeneity across platform centers (i.e., clients). This heterogeneity reduces preference learning because it causes update drift when performing training on the clients. Thus, we propose DCFT that features stochastically controled averaging to optimize the preference learning, where heterogeneity is considered as 'client-variance' in the updates. Our aim is to reduce the client-variance during the federated training, as shown in lines 22 – 23 in Algorithm 3.

Algorithm 3: Drift-Corrected Federated Preference Model Training

Input: T, θ^0, α, θ_k^0, E, c, c_i
1 % *Server execution*
2 **for** each round t in $[1, ..., T]$ **do**
3 Sample $m(m \geq 1)$ platform centers P_t from P;
4 Server transmits θ^{t-1} to each selected platform center;
5 **for** each client $k \in P_t$ in parallel **do**
6 $\theta_k^t, c_i^+ \leftarrow ClientUpdate(k, \theta_k^{t-1})$;
7 $c_i = c_i^+$
8 **for** each platform center $pc_k \notin P_t$ **do**
9 $\theta_k^t \leftarrow \theta_k^{t-1}$;
10 $\nabla L_k(\theta_k^t) \leftarrow \nabla L_k(\theta_k^{t-1})$;
11 $c \leftarrow c + \frac{1}{|P|}\sum_{i \in P_t} \delta_{c_i}$
12 $h^t \leftarrow h^{t-1} - \frac{\alpha}{m}(\sum_{k \in P_t} \theta_k^t - \theta^{t-1})$;
13 $\theta^t \leftarrow (\frac{1}{|P_t|}\sum_{k \in P_t} \theta_k^t) - \frac{1}{\alpha}h^t$;
14 % *Update on client k*
15 *ClientUpdate*(k, θ):
16 $\mathcal{B} \leftarrow$ split data stored in client k into batches of size B;
17 **for** each local epoch i from 1 to E **do**
18 **for** batch $b \in \mathcal{B}$ **do**
19 $\theta_{k,i}^t \leftarrow \arg\min_{\theta} L_{k,i}(\theta) - \langle \nabla L_{k,i}(\theta_{k,i}^{t-1}), \theta \rangle + \frac{\alpha}{2}\|\theta - \theta^{t-1}\|^2$;
20 $\nabla L_{k,i}(\theta_{k,i}^t) \leftarrow \nabla L_{k,i}(\theta_{k,i}^{t-1}) - \alpha(\theta_{k,i}^t - \theta^{t-1} + c - c_i)$;
21 $\theta_k^t \leftarrow \theta_{k,E}^t$;
22 $c_i^+ \leftarrow c_i - c + \frac{1}{|\mathcal{B}|\alpha}(\theta - \theta_k^t)$;
23 $\delta_{c_i} \leftarrow c_i^+ - c_i$;
24 Transmit platform center model θ_k^t, δ_{c_i} to central server;
25 **return** θ_k^t, c_i^+;

3.2 Preference-driven Task Assignment

We proceed to first obtain the available worker set for each task and reachable task set for each worker. Then, we use spatio-temporal and top-k constraints to build a Bipartite graph and propose an Intersected Top-k KM algorithm, which utilizes the worker preferences for different task categories calculated by the federated preference learning model. To protect each platform's local data and to get the global workers' preferences, all workers' preferences are calculated in local platform centers and are then uploaded to the central server, where the task assignment occurs.

3.2.1 Available Worker Set and Reachable Task Set

Before building the Bipartite graph, we should consider spatio-temporal constraints to filter workers and tasks. Given a set of online workers, $W = \{w_1, w_2, ..., w_{|W|}\}$ and a set of tasks, $S = \{s_1, s_2, ..., s_{|S|}\}$, the available worker set for spatial task $s \in S$ and the reachable task set for worker $w \in W$ are denoted as $AW(s)$ and $RT(w)$, respectively.

Both $AW(s)$ $(\forall w \in AW(s), s \in S)$ and $RT(w)$ $(\forall s \in RT(w), w \in W)$ must satisfy two conditions: 1) $d(w.l, s.l) \leqslant w.r$, and 2) $t_{now} + d(w.l, s.l)/w.speed \leqslant s.e$, where $d(w.l, s.l)$ represents the distance between $w.l$ and $s.l$ (e.g., Euclidean distance).

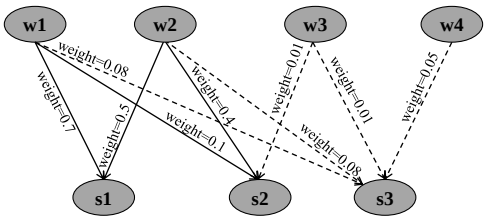


Fig. 5. Worker-Task Bipartite Graph

3.2.2 Intersected Top-k KM algorithm

Here, we transform the task assignment problem into a Bipartite Maximum Weight Matching problem [34], which is based on a graph that is represented by $G = (V, E)$ with a set V of vertices and a set E of edges. Given a set of online workers, $W = \{w_1, w_2, ..., w_{|W|}\}$, and a set of available tasks, $S = \{s_1, s_2, ..., s_{|S|}\}$, the cardinality of V and the cardinality of E are fixed to $|W| + |S|$ and $\sum_{i=1}^{|W|} n_i$, respectively, where n_i is the number of worker w_i's adaptive reachable assignments that is a subset of worker w_i's reachable assignments $RT(w_i)$.

To construct vertices, V is divided into two sets, V_W and V_S, where $V_W \cap V_S = \varnothing$. Each worker w_i maps to a vertex v_i^w, and each spatial task s_j maps to a vertex v_j^s. For the edges' construction, we add an edge from v_i^w mapped from vertex $w_i \in W$ to vertex v_j^s mapped from $s_j \in S$ if they satisfy the two following conditions:

1) spatio-temporal constraints: $w_i \in AW(s_j)$ and $s_j \in RT(w_i)$, and

2) top-k constraints: $w_i \in \widetilde{AW}_k(s_j)$ and $s_j \in \widetilde{RT}_k(w_i)$,

where $\widetilde{AW}_k(s_j)$ is a sorted set that contains the top-k workers of task s_j's available worker set when $AW(s_j)$ is sorted in descending order according to the preference of task s_j's available workers. Similarly, $\widetilde{RT}_k(w_i)$ is a set containing the top-k tasks of worker w_i's reachable task set when set $RT(w_i)$ is sorted in descending order according to the preference of worker w_i for task s, $\forall s \in RT(w_i)$. For each edge (v_i^w, v_j^s), its weight, denoted by $weight(v_i^w, v_j^s)$, can be measured as the preference of worker w_i for task s_j, denoted as $p_{w_i}(s_j)$, which can be calculated as:

$$p_{w_i}(s_j) = exp(\langle h_{w_i}^z, e_{c(s_j)}^z \rangle), \tag{18}$$

where $h^z_{w_i}$ is the representation summarizing the category information of worker w_i's recent task records, and $e^z_{c(s_j)}$ is the category embedding of category $c(s_j)$, which is the task s_j's category.

Algorithm 4: Intersected Top-k KM Algorithm

Input: graph G, k, sorted available worker set $\widetilde{AW}$
Output: $match$
1 Initialize $match$, val_{task} and $slack$;
2 **for** each worker $w \in W$ **do**
3 $\quad val_{worker}[w] \leftarrow max(weight(v_w^W, v_s^S))$;
4 **for** each worker $w \in W$ **do**
5 $\quad$ **while** $val_{worker}[w] > 0$ **do**
6 $\quad\quad$ Initialize vis_{task} and vis_{worker} to $False$;
7 $\quad\quad$ **if** FindTask(w,0) **then** break ;
8 $\quad\quad$ **else**
9 $\quad\quad\quad$ d=INF;
10 $\quad\quad\quad$ **for** each task $s \in S$ **do**
11 $\quad\quad\quad\quad$ **if** ! $vis_{task}[s]$ **then**
12 $\quad\quad\quad\quad\quad$ d $\leftarrow$ min(d,$slack[s]$);
13 $\quad\quad\quad$ **for** each worker $w \in W$ **do**
14 $\quad\quad\quad\quad$ Decrease $val_{worker}[w]$ by d if w is visited;
15 $\quad\quad\quad$ **for** each task $s \in S$ **do**
16 $\quad\quad\quad\quad$ Increase $val_{task}[s]$ by d if s is visited;
17 $\quad\quad\quad\quad$ Decrease $slack[s]$ by d if s is not visited;
18 $ReassignTask(match, \widetilde{AW})$;
19 **return** $match$;

Figure 5 depicts a graph for four workers w_i ($i = 1, 2, 3, 4$), and three tasks s_j ($j = 1, 2, 3$). If vertex w_i and vertex s_j satisfy both of the above constraints, edge (v_i^w, v_j^s) is drawn as a solid line (e.g., edge (v_1^w, v_1^s) when $k = 2$). If these two vertices only satisfy the spatio-temporal constraints, the edge (v_i^w, v_j^s) becomes a dotted line, which will be removed from the original graph in our algorithm. For example, edge (v_3^w, v_2^s) is drawn as a dotted line, because w_3 is not in the top-2 of $\widetilde{AW}_2(s_2)$. The higher $weight(v_i^w, v_j^s)$ of the edge (v_i^w, v_j^s) is, the more likely a worker is to perform this task successfully. Therefore, the mutual top-k will help filter the lower preference edge and keep the quality of assignments. However, this method will remove a number of edges and will contribute to a higher loss of the number of task assignments (e.g., w_4 and s_3). Therefore, we redistribute the task s_j to worker w_i if $w_i \in \widetilde{AW}(s_j)$ and w_i is not assigned to any task, where $\widetilde{AW}(s_j)$ is a sorted set which contains all available workers of task s_j.

The Intersected Top-k KM Algorithm is shown in Algorithm 4. Suppose that we have a bipartite graph G, which is composed of two vertices sets V_S and V_W. First, in graph G, the expectation of each vertex in V_W is equal to the largest weight among the edges associated with it (lines 2–3). Second, matching tasks for worker w are recursively found through the Find Task Algorithm 5 (line 7). Third, if w fails to match a task, to make more workers assigned, the expectations of workers and tasks involved in the last matching are adjusted, thus changing the competitive relationship among workers (lines 8–17). Some refinements should be made to the original KM algorithm, which is used to find the perfect matching of a weighted bipartite graph, to take into account cases where perfect matches do not exist [34]. Since the original KM algorithm may cause an endless loop in our problem, we stop matching tasks for the worker if the expectation of w is less than 0 (line 5). The time complexity of the proposed Intersected Top-k KM Algorithm is $O(nlogN)$.

Algorithm 5: FindTask Algorithm

Input: $worker\ w$, $recursion\ depth\ rdp$
Output: $Bool$
1 $vis_{worker}[w] = True$;
2 **if** $rdp > \lambda$ **then return** $False$;
3 **else**
4 $\quad$ **for** each task s is adjacent to w in G **do**
5 $\quad\quad$ **if** $vis_{task}[s]$ **then** continue ;
6 $\quad\quad$ $gap \leftarrow$
$\quad\quad\quad val_{worker}[w] + val_{task}[s] - weight(v_w^W, v_s^S)$;
7 $\quad\quad$ **if** $gap \neq 0$ **then**
8 $\quad\quad\quad$ $slack[s] \leftarrow min(slack[s], gap)$;
9 $\quad\quad$ **else if** match[s] ==-1 *or*
$\quad\quad\quad$ FindTask(match[s],rdp+1) **then**
10 $\quad\quad\quad$ Assign value w to $match[s]$ and **return** $True$;
11 $\quad$ **return** $False$;

To improve efficiency, we limit the number of edges in the graph. Since we use the intersected Top-k method to filter edges, the number of edges associated with a worker node or a task node cannot exceed k. Thus, the recursion depth of Algorithm 5 is set to k, which can reduce the competition among workers (lines 2). The time complexity of the FindTask algorithm is $O(n)$. To reduce the number of task assignments loss, a reassignment task algorithm is designed, which is shown in Algorithm 6. First, the assignment result will be checked (lines 1–3) in order to keep assignments satisfying the constraints proposed before. Then, we assign the remaining available tasks to the worker with the highest preference score among the corresponding available workers (lines 4–9). The time complexity of the proposed reassignment task algorithm is $O(n)$.

Algorithm 6: Reassignment Task Algorithm

Input: KM output assignments $match$, $\widetilde{AW}$
Output: modified assignments $match$
1 **for** each task s in $match$ **do**
2 $\quad$ **if** match[s]≠-1 and match[s] not in $\widetilde{AW}(s)$ **then**
3 $\quad\quad$ match[s] $\leftarrow$ -1;
4 **for** each task s in $match$ **do**
5 $\quad$ **if** match[s] == -1 and $|\widetilde{AW}(s)| > 0$ **then**
6 $\quad\quad$ **for** each worker w in $\widetilde{AW}(s)$ **do**
7 $\quad\quad\quad$ **if** w not in match **then**
8 $\quad\quad\quad\quad$ Assign value w to $match[s]$;
9 $\quad\quad\quad\quad$ **return** $True$;
10 **return** $match$;

4 Experimental Evaluation

4.1 Experimental Setup

We use a check-in dataset from Twitter, which is used widely for experimental evaluations of SC platforms [14],

[35], [36], [37]. It provides check-in data in the United States from September 2010 to January 2011 and includes 62,462 venue locations and 61,412 user locations. First, we use FourSquare's API[1] to generate the corresponding category information of venues. Then, for each worker and task, we take the average of the corresponding check-in locations as a venue's location information. For each check-in, we simulate that the user is a worker and that each venue accessed by the user is a task performed by the worker. The publication time of the task is set to the earliest check-in time of the task in a day. We use the category information of the venues in 18 kinds of check-ins to simulate the category information of tasks. Moreover, we randomly and uniformly generate 32 platform centers and use a Voronoi diagram-based algorithm [34] to allocate all workers and their historical data to corresponding platform centers. It is common practice in experimental studies of spatial crowdsourcing platforms to use uniformly distributed attribute values [38], [39], [40], the argument being that this captures the effects of the attributes on a more fair basis. A check-in record denotes that the worker has accepted and completed the task. All the experiments are implemented on an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz and NVidia TITAN Xp GPU.

4.2 Performance of Federated Preference Learning

4.2.1 Overall Accuracy

We first evaluate the overall accuracy of the federated preference learning.

Evaluation Methods. We study four models.

1) CatePEP: The Category-based Personal Equal Popularity (CatePEP) model, where the popularity of task category c is set to 1 if the target worker did tasks in category c; otherwise, it is set to 0. The popularity of task category c is considered as the worker's preference for c.

2) POISeqPop: The POI-based Sequence Popularity (POISeqPop) model, which ranks tasks according to their popularity in the target worker's sequence in a descending order. The popularity of task category c is calculated as $1/rank(c)$, which is regarded as the target worker's preference for c.

3) CTP: The Centralized Training based Preference (CTP) model, which trains our local preference model in a centralized manner. It means all workers belong to a single center.

4) FLP: Our Federated Learning based Preference (FLP) model.

Metrics. To evaluate the accuracy of worker preference learning, we use $Recall@K$ as the evaluation metric. This metric counts the proportion of times when the categories of ground-truth tasks are ranked among the top-K predictions. We use workers' historical check-in sequences stored in local platform centers for the experiments and we use the first 80% of the check-in records for training, the next 10% for validation, and the remaining 10% for testing. We remove POIs associated with fewer than 5 records and workers with fewer than 10 or more than 300 check-in records.

Results. Table 2 shows the evaluation results, in which CTP performs best followed by FLP, POISeqPop, and CatePEP. The models (CTP and FTP) based on our category-aware preference model outperform the others by far. This shows that our preference model can provide more accurate estimates of worker preferences. Considering that the local-device level empirical losses are inconsistent with the global empirical loss, it is expected that FLP performs worse than CTP. Moreover, with the aid of linear and quadratic penalty terms, the $Recall@K$ ($k = 1, 2, 3$) of FLP is 97.6%–98.0% of that of CTP.

1. https://developer.foursquare.com/

TABLE 2
Overall Accuracy of Different Preference Models

Methods	Recall@1	Recall@2	Recall@3
CatePEP	0.1190	0.236	0.3630
POISeqPop	0.3876	0.5210	0.6065
CTP	**0.4204**	**0.5978**	**0.6994**
FPL	0.4120	0.5832	0.6850

TABLE 3
Accuracy and Convergence Time of Different Training Manners

Methods	Recall@1	Recall@2	Recall@3	Convergence Time
FLP	0.4120	0.5832	0.6850	17.97 min
FLP-FedAvg	**0.4278**	0.5927	0.6933	11.72 min
FLP-DCFT	0.4267	**0.5946**	**0,6946**	**9.77** min

4.2.2 Effect of Different Federated Training Approaches

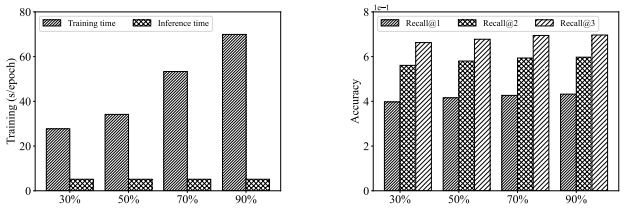
We proceed to evaluate the performance of preference learning across different federated training approaches.

Evaluation Methods. We train the preference model according to three different approaches and report the performance.

1) FLP: Federated training (see Algorithm 1) is used for preference learning.

2) FLP-FedAvg: The preference learning is optimized using FedAvg (see Algorithm 2).

3) FLP-DCFT: The preference learning is optimized with drift-corrected federated training to overcome data heterogeneity (see Algorithm 3).



(a) Training Time (b) Accuracy

Fig. 6. Parameter Sensitivity Analysis

Metrics. We quantify the performance of preference learning with the different optimization approaches by measuring the accuracy (i.e., $Recall@k$) of worker preference learning and the convergence time. We set the ratio of sampled clients to 0.7 during the training of FLP-FedAvg and FLP-DCFT.

Results. In Table 3, the overall best results are highlighted in bold. Fed-DCFT performs the best among all methods w.r.t. $Recall@2$, and $Recall@3$. This is because DCT has the ability to contend with data heterogeneity by correcting client drift, which improves the performance of preference learning. As popular optimization methods, FLP-FedAvg and FLP achieve comparable results as both suffer from the issue of client drift, which affects the performance negatively.

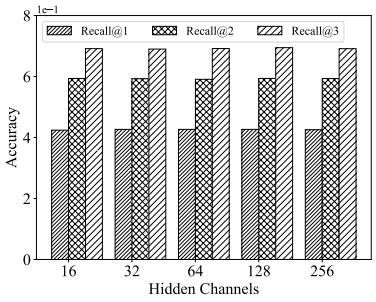


Fig. 7. Effect of Hidden Features

To observe whether DCFT is able to accelerate the convergence of preference learning, we report the convergence time across different training approaches. We see that FLP-DCFT converges the fastest, followed by FLP-FedAvg and FLP. More specifically, FLP-DCFT converges faster than FedAvg by 16.64% and faster than FLP by 45.63%. This is because that FLP-DCFT only involves a subset of clients (i.e., 70%) for training and accelerates the training by performing drift correction, which makes it easier for the preference model to reach the optimum.

4.2.3 Parameter Sensitivity of Preference Learning

Next, we assess the sensitivity of FLP-DCFT in terms of the number of clients involved in the training and the number of hidden features in the local preference learning models.

Effect of the Number of Clients. We conduct experiments in which we randomly sample differently-sized subsets of clients (i.e., 30%, 50%, 70%, and 90% of clients) when training FLP-DCFT in each training round. Fig. 6(a) shows the training time across different ratios of clients. The training time increases from around 27s to 70s per epoch as the number of clients increases. This shows that involving more clients in the training can increase the training time, which is due to more training data being introduced. Next, we report the accuracy of FLP-DCFT in Fig. 6(b). We observe that $Recall@k$ shows an increasing trend, meaning that involving more clients with more training data leads to better accuracy.

Effect of the Number of Hidden Features. We also study how sensitive the model is to the deep neural network structure with different parameters. As indicated in Fig. 7, we vary the number of channels (features) of the hidden layers in the local preference models from 16 to 256. We see that the model accuracy first increases and then decreases slightly with the increase of hidden features, indicating that the proposed preference model is robust to different parameters (i.e., hidden features) settings. We also observe that our local preference model achieves the best performance when setting the number of hidden features to 128. Therefore, we use 128 hidden features as the default in experiments.

4.3 Performance of Task Assignment

We proceed to study the performance of task assignment. Table 4 shows our experimental settings, where the default values of all parameters are underlined.

Evaluation Methods. We study the following task assignment algorithms.

TABLE 4
Experiment Parameters

Parameter	Values
Valid time of tasks (h) $e-p$	0.4, 0.8, 1.2, 1.6, <u>2.0</u>
Reachable distance of workers (km) r	10, 15, 20, 25, <u>30</u>
Number of workers $\|W\|$	2200, 2400, 2600, 2800, <u>3000</u>
Number of tasks $\|S\|$	<u>2600</u>, 2800, 3000, 3200, 3400
Limit coefficient k	10, 20, 30, 40, <u>50</u>

1) KM: The original KM algorithm that does not consider workers' preferences.

2) P+Greedy: The Greedy algorithm with workers' Preferences calculated by our FLP model.

3) P+KM: The original KM algorithm with workers' Preferences calculated by our FLP model.

4) P+KM+Top-k: The P+KM algorithm, using the worker-sided Top-k pruning strategy to prune edges.

5) P+KM+InTop-k: The P+KM algorithm, using our intersected Top-k pruning strategy to prune edges.

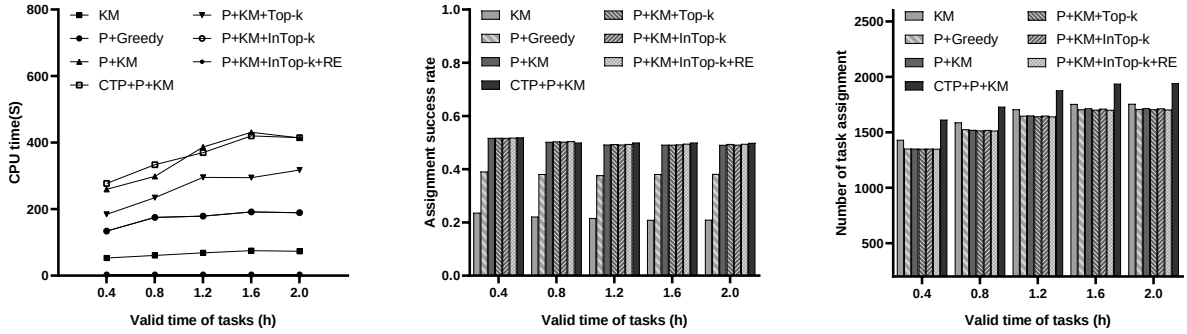
6) P+KM+InTop-k+RE: The P+KM algorithm, using our Intersected Top-k pruning strategy and the Reassignment optimization strategy.

7) CTP+P+KM: The original KM algorithm with workers' preferneces calculated by CTP model.

Metrics. Three metrics are compared among the methods, including CPU time, Assignment Success Rate (ASR), and the number of task assignments. The CPU time is the time cost for finding the task assignment. ASR is the ratio between the number of successful assignments of all workers and the total number of assignments in a time instance. In our experiments, if a worker performs (checks in) tasks (locations) with the same category in the next two check-ins, the assignment of this task can be considered successful.

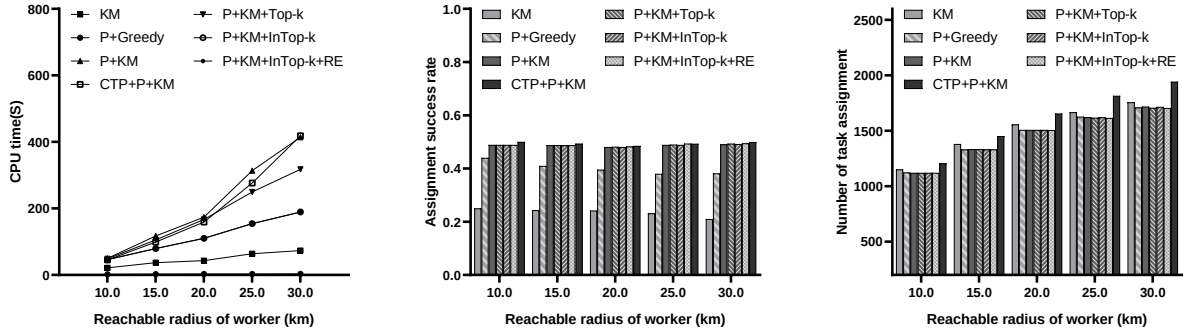
Effect of $e-p$. First, we evaluate the effect of the valid time $e-p$ of tasks on the performance of task assignments (see Figure 8). It can be seen from Figures 8(a) and 8(c) that the CPU time and the number of task assignments of all algorithms exhibit an increasing trend as the valid time of tasks increases. This is because as the valid time of tasks increases, there will be more available workers and tasks, which leads to a larger search space and higher probability of being assigned to a task for each worker. The CPU time of P+KM+InTop-k is less than those of P+KM+Top-k and P+KM while keeping almost the same ASR and number of task assignments, which is evidence of the efficiency and effectiveness of our proposed algorithms. As shown in Figure 8(b), P+KM related algorithms achieve the highest Assignment Success Rate (ASR), which shows the importance of considering preferences. As shown in Figure 8(c), the KM algorithm has the most task assignments, while other preference-related algorithms achieve fewer task assignments. This is due to the fact that the preferences of some workers among tasks vary greatly and many workers tend to choose the tasks they are interested in, thus leading to a lower number of task assignments. This reflects from the side that the preference scores we learn are accurate and discriminative.

Effect of r. We further evaluate the effect of the reachable distance r of workers. It can be seen from Figure 9(a) that when r increases, the CPU time of all algorithms shows a similar growth trend. The reason is that when the reachable



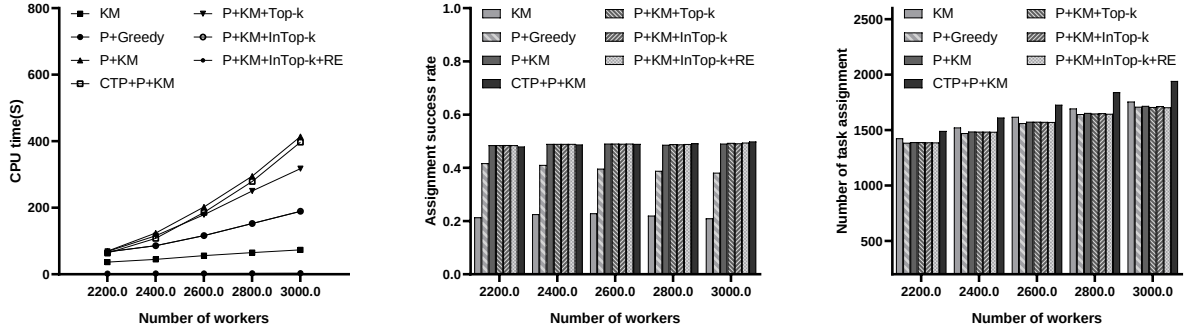
(a) CPU Time (b) Assignment Success Rate (c) Number of Task Assignments

Fig. 8. Performance of Task Assignment: Effect of $e - p$



(a) CPU Time (b) Assignment Success Rate (c) Number of Task Assignments

Fig. 9. Performance of Task Assignment: Effect of r



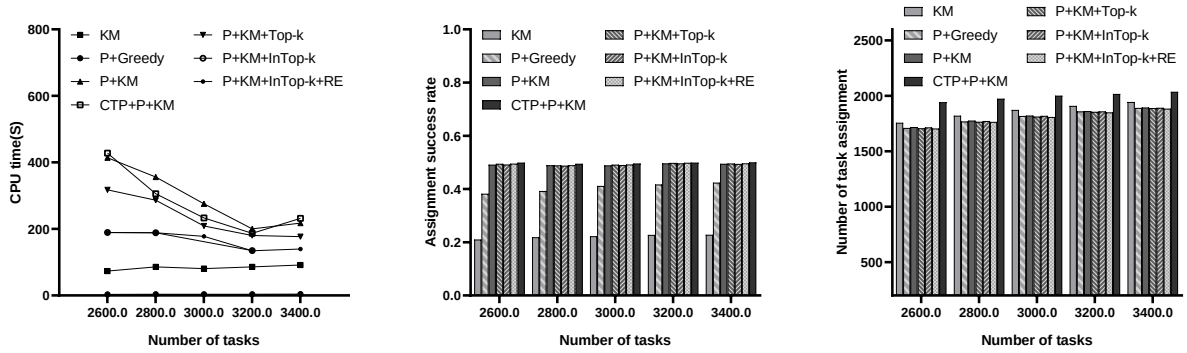
(a) CPU Time (b) Assignment Success Rate (c) Number of Task Assignments

Fig. 10. Performance of Task Assignment: Effect of $|W|$

distance of workers increases, the number of available workers and the number of reachable tasks increase, yielding a larger search space. The P+Greedy algorithm still consumes the least CPU time, but its performance in ASR is obviously not as good as those of the other P+KM related algorithms (see Figure 9(b)). In addition, as shown in Figure 9(c), when r increases, the number of task assignments also increases since workers are more likely to be assigned their available tasks with greater r.

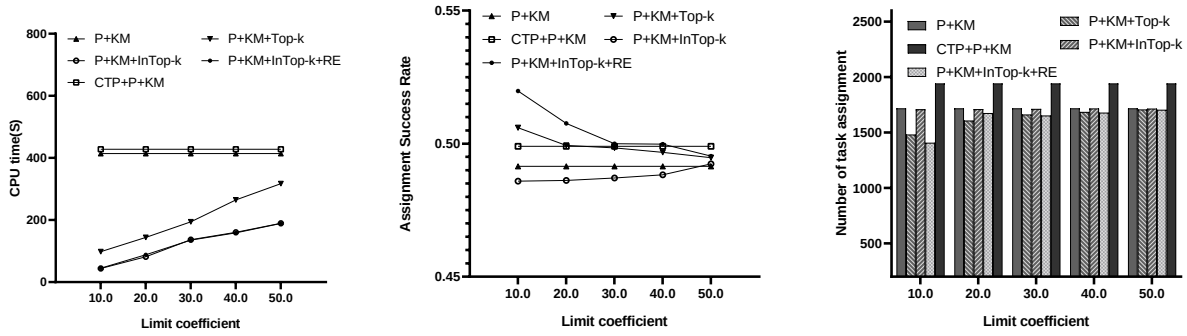
Effect of $|W|$. Next, we evaluate the effect of $|W|$. As shown in Figure 10(a), the larger $|W|$ is, the longer the CPU time is. This is because more available workers need to be assigned, which leads to more competition for limited tasks and generates more time overhead. When it comes to ASR in Figure 10(b), all preference-based algorithms keep high ASR values, and the number of task assignments increases (cf. Figure 10(c)). In summary, P+KM+InTop-k+RE, which considers privacy protection via federated learning, performs well in terms of CPU time and ASR while offering the acceptable number of task assignments. In addition, CTP+P+KM performs well than P+KM+InTop-k+RE. This is because CTP learns preference by centralized learning, however, which has high risks of privacy leakage.

Effect of $|S|$. We study the effect of the number $|S|$ of tasks. In Figure 11(a), the CPU time of P+KM related algorithms decreases because as the number of tasks increases, the occurrence probability of tasks that workers are most interested in increases and the discriminative preference scores also disperse the competition among workers. Moreover, the CPU time of KM-related algorithms is higher



(a) CPU Time (b) Assignment Success Rate (c) Number of Task Assignments

Fig. 11. Performance of Task Assignment: Effect of $|S|$



(a) CPU Time (b) Assignment Success Rate (c) Number of Task Assignments

Fig. 12. Performance of Task Assignment: Effect of k

than that of Greedy algorithm. Because it is time-consuming for KM-related algorithms to find a perfect matching with multiple iterations. The number of tasks assigned by KM is more than those of preference-based algorithms (i.e., all algorithms except KM), which sacrifice the number of task assignments in order to improve the total preferences of workers. In addition, with the increase of $|S|$, a worker can access more available and interested tasks with less competition, so both the number of task assignments and the ASR value increase. In terms of decentralized training, P+KM+InTop-k+RE performs better in terms of CPU time and ASR, showing its superiority.

Effect of k. Finally, we study the effect of k, which limits the number of edges associated with vertices and recursion depth in pruning-based algorithms (i.e., P+KM+Top-k, P+KM+InTop-k and P+KM+InTop-k+RE) and CTP+P+KM. We report the results of pruning-based algorithms, where P+KM is used as a reference. As k is a parameter only for the pruning-based algorithms, the results of P+KM in Figure 12 keep the same value. In figure 12(a), InTop-k based algorithms (i.e., P+KM+InTop-k and P+KM+InTop-k+RE) cost the least cpu time because they use the intersection operation to filter more edges with low weights. In the graphs of P+KM+Top-k and P+KM+InTop-k, each vertex is associated with more edges, leading to more opportunities to be assigned to available tasks as k increases. However, k has little impact on the high weight edge. Thus, the number of task assignments of these two algorithms increases while ASR decreases as k grows. Compared with P+KM+InTop-k, P+KM+InTop-k+RE takes less time but is more effective to keep the number of task assignments, which shows the necessity of the reassignment strategy. In addition, CTP+P+KM performs the best in task assignment due to the centralized training with more data.

5 Related Work

Spatial crowdsourcing (SC) is a new framework that has emerged recently, requiring workers with GPS devices to reach a specific location physically under certain restrictions to perform spatial tasks [15], [18], [34], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53]. Most of the existing studies focus on task assignment [29], [54], [55], [56], [57], [58], [59], [60], [61], however, putting their focus on effectiveness without considering the privacy of users' raw data and the tediousness of data migration in reality, which leads to the risk of privacy leaks. To make the SC server assign tasks properly, workers need to upload their highly sensitive data (e.g., locations and historical task records), which disclose their private attributes. Thus, in recent studies [8], [62], privacy-preserving task assignment is proposed to make users (workers/task requesters) perturb their locations with Geo-Indistinguishability [19] and upload only the perturbed locations. However, these studies mainly focus on the privacy of locations without considering the preferences of workers, failing to achieve a satisfying task assignment. With federated learning, multiple entities (clients) collaborate to solve a problem, under the coordination of a central server or service provider [24]. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are

used to achieve the learning objective [25], [27], [33], [63]. For example, a novel quantized federated averaging algorithm [63] is designed to apply a stochastic quantization scheme to the local and global model parameters, which is the first work to provide theoretical analysis for quantized federated learning algorithms with convex functions. In this paper, we propose a federated preference learning model to protect the privacy of workers' raw data and learn worker preferences without data migration, based on which we assign tasks to suitable workers.

6 Conclusion

We propose a framework called Task Assignment with Federated Preference Learning (TA-FPL), which aims to find optimal task assignments while considering workers' preferences and protecting workers' raw data. TA-FPL consists of an efficient Federated Preference Learning (FPL) phase and a Preference-driven Task Assignment (PTA) phase. For the FPL phase, we present local means of learning platform center models and combine these with a drift-corrected federated training method. For the PTA phase, we propose an Intersected Top-k KM algorithm to achieve effective and efficient task assignments based on worker preferences obtained in the first phase. To the best of our knowledge, this is the first study in SC that applies federated learning and enables preference modeling while offering raw data privacy. An empirical study based on a real dataset offers evidence of the superiority of our proposed algorithms. In future research, it is of interest to achieve privacy-preserving clustering so that similar clients can be clustered, thereby further accelerating preference learning.

Acknowledgments

This work is partially supported by NSFC (No. 61972069, 61836007, 61832017, 62272086), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), Municipal Government of Quzhou under Grant No. 2022D037, and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

References

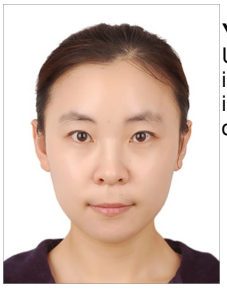
[1] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: a survey," *PVLDB*, vol. 29, no. 1, pp. 217–250, 2020.
[2] S. R. B. Gummidi, X. Xie, and T. B. Pedersen, "A survey of spatial crowdsourcing," *TODS*, vol. 44, no. 2, pp. 1–46, 2019.
[3] G. Li, Y. Zheng, J. Fan, J. Wang, and R. Cheng, "Crowdsourced data management: Overview and challenges," in *SIGMOD*, 2017, pp. 1711–1716.
[4] R. Wang, S. Wang, H. Yan, and X. Wang, "Wsip: Wave superposition inspired pooling for dynamic interactions-aware trajectory prediction," in *AAAI*, vol. 37, no. 4, 2023, pp. 4685–4692.
[5] X. Chen, Y. Zhao, and K. Zheng, "Task publication time recommendation in spatial crowdsourcing," in *CIKM*, 2022, pp. 232–241.
[6] Y. Zhao, T. Lai, Z. Wang, K. Chen, H. Li, and K. Zheng, "Worker-churn-based task assignment with context-lstm in spatial crowdsourcing," *TKDE*, 2023.
[7] H. To, G. Ghinita, and C. Shahabi, "A framework for protecting worker location privacy in spatial crowdsourcing," *PVLDB*, vol. 7, no. 10, pp. 919–930, 2014.
[8] H. To, C. Shahabi, and L. Xiong, "Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server," in *ICDE*, 2018, pp. 833–844.
[9] X. Yi, F.-Y. Rao, G. Ghinita, and E. Bertino, "Privacy-preserving spatial crowdsourcing based on anonymous credentials," in *MDM*, 2018, pp. 187–196.
[10] C. Qiu and A. C. Squicciarini, "Location privacy protection in vehicle-based spatial crowdsourcing via geo-indistinguishability," in *ICDCS*, 2019, pp. 1061–1071.
[11] C. Qiu, A. Squicciarini, Z. Li, C. Pang, and L. Yan, "Time-efficient geo-obfuscation to protect worker location privacy over road networks in spatial crowdsourcing," in *CIKM*, 2020, pp. 1275–1284.
[12] M. Li, J. Wang, L. Zheng, H. Wu, P. Cheng, L. Chen, and X. Lin, "Privacy-preserving batch-based task assignment in spatial crowdsourcing with untrusted server," in *CIKM*, 2021, pp. 947–956.
[13] W. Huang, X. Lei, and H. Huang, "Pta-sc: Privacy-preserving task allocation for spatial crowdsourcing," in *WCNC*, 2021, pp. 1–7.
[14] Y. Li, Y. Zhao, and K. Zheng, "Preference-aware group task assignment in spatial crowdsourcing: A mutual information-based approach," in *ICDM*, 2021, pp. 350–359.
[15] Z. Wang, Y. Zhao, X. Chen, and K. Zheng, "Task assignment with worker churn prediction in spatial crowdsourcing," in *CIKM*, 2021, pp. 2070–2079.
[16] X. Li, Y. Zhao, J. Guo, and K. Zheng, "Group task assignment with social impact-based preference in spatial crowdsourcing," in *DASFAA*, 2020, pp. 677–693.
[17] Y. Zhao, K. Zheng, H. Yin, G. Liu, J. Fang, and X. Zhou, "Preference-aware task assignment in spatial crowdsourcing: from individuals to groups," *TKDE*, 2020.
[18] Y. Zhao, J. Xia, G. Liu, H. Su, D. Lian, S. Shang, and K. Zheng, "Preference-aware task assignment in spatial crowdsourcing," in *AAAI*, 2019, pp. 2629–2636.
[19] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi, "Geo-indistinguishability: Differential privacy for location-based systems," in *SIGSAC*, 2013, pp. 901–914.
[20] Q. Tao, Y. Tong, S. Li, Y. Zeng, Z. Zhou, and K. Xu, "A differentially private task planning framework for spatial crowdsourcing," in *MDM*, 2021, pp. 9–18.
[21] Y. Wang, Y. Tong, Z. Zhou, Z. Ren, Y. Xu, G. Wu, and W. Lv, "Fed-ltd: Towards cross-platform ride hailing via federated learning to dispatch," in *SIGKDD*, 2022, pp. 4079–4089.
[22] Y. Tong, X. Pan, Y. Zeng, Y. Shi, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu *et al.*, "Hu-fu: Efficient and secure spatial queries over data federation," *PVLDB*, vol. 15, no. 6, p. 1159, 2022.
[23] Y. Zhao, K. Zheng, Z. Wang, L. Deng, B. Yang, T. B. Pedersen, C. S. Jensen, and X. Zhou, "Coalition-based task assignment with priority-aware fairness in spatial crowdsourcing," *VLDBJ*, 2023.
[24] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*. PMLR, 2017, pp. 1273–1282.
[26] J. Liu, L. Deng, H. Miao, Y. Zhao, and K. Zheng, "Task assignment with federated preference learning in spatial crowdsourcing," in *CIKM*, 2022, pp. 1279–1288.
[27] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *MLSys*, vol. 2, pp. 429–450, 2020.
[28] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "SCAFFOLD: stochastic controlled averaging for on-device federated learning," *CoRR*, vol. abs/1910.06378, 2019.
[29] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *SIGSPATIAL*, 2012, pp. 189–198.
[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NIPS*, vol. 30, 2017.
[31] J. Li, Y. Wang, and J. McAuley, "Time interval aware self-attention for sequential recommendation," in *WSDM*, 2020, pp. 322–330.
[32] R. Cai, J. Wu, A. San, C. Wang, and H. Wang, "Category-aware collaborative sequential recommendation," in *SIGIR*, 2021, pp. 388–397.
[33] A. E. Durmus, Z. Yue, M. Ramon, M. Matthew, W. Paul, and S. Venkatesh, "Federated learning based on dynamic regularization," in *ICLR*, 2021.
[34] G. Ye, Y. Zhao, X. Chen, and K. Zheng, "Task allocation with geographic partition in spatial crowdsourcing," in *CIKM*, 2021, pp. 2404–2413.

[35] P. Cheng, X. Lian, L. Chen, and C. Shahabi, "Prediction-based task assignment in spatial crowdsourcing," in *ICDE*, 2017, pp. 997–1008.
[36] H. Dang, T. Nguyen, and H. To, "Maximum complex task assignment: Towards tasks correlation in spatial crowdsourcing," in *IIWAS*, 2013, pp. 77–81.
[37] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *SIGSPATIAL*, 2013, pp. 324–333.
[38] P. Cheng, X. Jian, and L. Chen, "An experimental evaluation of task assignment in spatial crowdsourcing," *PVLDB*, vol. 11, no. 11, pp. 1428–1440, 2018.
[39] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*, 2016, pp. 49–60.
[40] S. R. B. Gummidi, T. B. Pedersen, and X. Xie, "Transit-based task assignment in spatial crowdsourcing," in *SSDBM*, 2020, pp. 1–12.
[41] Y. Cui, L. Deng, Y. Zhao, B. Yao, V. W. Zheng, and K. Zheng, "Hidden poi ranking with spatial crowdsourcing," in *SIGKDD*, 2019, pp. 814–824.
[42] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: experiments and analysis," *PVLDB*, vol. 9, no. 12, pp. 1053–1064, 2016.
[43] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *PVLDB*, vol. 11, no. 11, p. 1633, 2018.
[44] J. Tu, P. Cheng, and L. Chen, "Quality-assured synchronized task assignment in crowdsourcing," *TKDE*, vol. 33, no. 3, pp. 1156–1168, 2019.
[45] Y. Zhao, K. Zheng, Y. Cui, H. Su, F. Zhu, and X. Zhou, "Predictive task assignment in spatial crowdsourcing: a data-driven approach," in *ICDE*, 2020, pp. 13–24.
[46] Y. Zhao, K. Zheng, J. Guo, B. Yang, T. B. Pedersen, and C. S. Jensen, "Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches," in *ICDE*, 2021, pp. 265–276.
[47] Y. Zhao, X. Chen, L. Deng, T. Kieu, C. Guo, B. Yang, K. Zheng, and C. S. Jensen, "Outlier detection for streaming task assignment in crowdsourcing," in *WWW*, 2022.
[48] Y. Zhao, K. Zheng, Y. Li, H. Su, J. Liu, and X. Zhou, "Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach," *TKDE*, pp. 2336–2350, 2019.
[49] X. Li, Y. Zhao, X. Zhou, and K. Zheng, "Consensus-based group task assignment with social impact in spatial crowdsourcing," *Data Science and Engineering*, vol. 5, no. 4, pp. 375–390, 2020.
[50] Y. Zhao, J. Guo, X. Chen, J. Hao, X. Zhou, and K. Zheng, "Coalition-based task assignment in spatial crowdsourcing," in *ICDE*, 2021, pp. 241–252.
[51] S. Wang, J. Cao, and S. Y. Philip, "Deep learning for spatio-temporal data mining: A survey," *TKDE*, vol. 34, no. 8, pp. 3681–3700, 2020.
[52] Y. Zhao, K. Zheng, Y. Li, J. Xia, B. Yang, T. B. Pedersen, R. Mao, C. S. Jensen, and X. Zhou, "Profit optimization in spatial crowdsourcing: Effectiveness and efficiency," *TKDE*, 2022.
[53] Y. Zhao, J. Liu, Y. Li, D. Zhang, C. S. Jensen, and K. Zheng, "Preference-aware group task assignment in spatial crowdsourcing: Effectiveness and efficiency," *TKDE*, 2023.
[54] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: trustworthy query answering with spatial crowdsourcing," in *SIGSPATIAL*, 2013, pp. 314–323.
[55] H. To, C. Shahabi, and L. Kazemi, "A server-assigned spatial crowdsourcing framework," *TSAS*, vol. 1, no. 1, pp. 1–28, 2015.
[56] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "Slade: A smart large-scale task decomposer in crowdsourcing," *TKDE*, vol. 30, no. 8, pp. 1588–1601, 2018.
[57] Y. Tong, L. Wang, Z. Zimu, B. Ding, L. Chen, J. Ye, and K. Xu, "Flexible online task assignment in real-time spatial data," *PVLDB*, vol. 10, no. 11, pp. 1334–1345, 2017.
[58] P. Cheng, X. Lian, L. Chen, J. Han, and J. Zhao, "Task assignment on multi-skill oriented spatial crowdsourcing," *TKDE*, vol. 28, no. 8, pp. 2201–2215, 2016.
[59] J. Xia, Y. Zhao, G. Liu, J. Xu, M. Zhang, and K. Zheng, "Profit-driven task assignment in spatial crowdsourcing." in *IJCAI*, 2019, pp. 1914–1920.
[60] L. Zheng, L. Chen, and J. Ye, "Order dispatch in price-aware ridesharing," *PVLDB*, vol. 11, no. 8, pp. 853–865, 2018.
[61] Y. Zhao, L. Deng, and K. Zheng, "Adataskrec: An adaptive task recommendation framework in spatial crowdsourcing," *TOIS*, 2023.
[62] Q. Tao, Y. Tong, Z. Zhou, Y. Shi, L. Chen, and K. Xu, "Differentially private online task assignment in spatial crowdsourcing: A tree-based approach," in *ICDE*, 2020, pp. 517–528.
[63] Y. Li, W. Li, and Z. Xue, "Federated learning with stochastic quantization," *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 11 600–11 621, 2022.

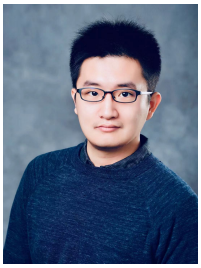
Hao Miao received the M.E. in Computer science and technology from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2021. He is currently a PhD Fellow in Computer Science at Aalborg University. From September 2019 to November 2019, he was a Visiting Student at PolyU, Hong Kong, China. His research interests include spatio-temporal data analytics, incremental learning, and federated learning.

Xiaolong Zhong received a bachelor's degree from Chongqing University of Technology, in 2018. He is currently studying for a master's degree in Computer Science and Technology at the University of Electronic Science and Technology of China. His research interests include federated learning and spatio-temporal data mining.

Jiaxin Liu received the bachelor's degree in Computer Science and Technology from Southwest Jiaotong University, in 2021. She is currently studying for a master's degree in Computer Science and Technology at the University of Electronic Science and Technology of China. Her research interests include spatial crowdsourcing, trajectory data mining and spatio-temporal databases.



Yan Zhao is an Assistant Professor with Aalborg University. She received the Doctoral Degree in Computer Science from Soochow University in 2020. Her research interests include spatial database and trajectory computing.



Prof. Xiangyu Zhao is an assistant professor of the school of data science at City University of Hong Kong (CityU). Prior to CityU, he completed his PhD (2021) at MSU. His current research interests include data mining and machine learning, especially their applications in Urban Computing and Recommender Systems. He has published more than 80 papers in top conferences and journals. His research has been awarded ICDM'22 and ICDM'21 Best-ranked Papers, Global Top 100 Chinese New Stars in AI, CCF-Ant Research Fund, CCF-Tencent Open Fund, Criteo Faculty Research Award, and Bytedance Research Collaboration Award. He serves as top data science conference (senior) program committee members and session chairs (e.g., KDD, WWW, SIGIR, IJCAI, AAAI, ICML, ICLR), and journal guest editors and reviewers (e.g., TKDE, TKDD, TOIS, TIST, CSUR, Frontiers in Big Data). Please find more information at https://zhaoxyai.github.io/.

Weizhu Qian is a postdoctoral researcher at Aalborg University. He received the PhD degree in Computer Science from University Bourgogne Franche-Comté in 2021. His research interests mainly focus on deep learning and data science.

Kai Zheng is a Professor of Computer Science with University of Electronic Science and Technology of China. He received his PhD degree in Computer Science from The University of Queensland in 2012. He has been working in the area of spatial-temporal databases, uncertain databases, social-media analysis, in-memory computing and blockchain technologies. He has published over 100 papers in prestigious journals and conferences in data management field such as SIGMOD, ICDE, VLDB Journal, ACM Transactions and IEEE Transactions. He is a senior member of IEEE.

Christian S. Jensen is a professor of computer science at Aalborg University, Denmark. He was a professor at Aarhus University from 2010 to 2013, and he was previously with Aalborg University for two decades. His research concerns data management and analytics, and its focus is on temporal and spatio-temporal data. He is a member of the Academia Europaea, the Royal Danish Academy of Sciences and Letters, and the Danish Academy of Technical Sciences. He has received several national and international awards for his research, most recently the IEEE TCDE impact award. He is a fellow of the IEEE and ACM.