Task Assignment with Federated Preference Learning in Spatial Crowdsourcing

Jiaxin Liu
University of Electronic Science and Technology of China
China
jiaxinliu1999@std.uestc.edu.cn

Liwei Deng
University of Electronic Science and Technology of China
China
deng_liwei@std.uestc.edu.cn

Hao Miao
Aalborg University
Denmark
haom@cs.aau.dk

Yan Zhao*
Aalborg University
Denmark
yanz@cs.aau.dk

Kai Zheng
University of Electronic Science and Technology of China
China
zhengkai@uestc.edu.cn

ABSTRACT

Spatial Crowdsourcing (SC) is ubiquitous in the online world today. As we have transitioned from crowdsourcing applications (e.g., Wikipedia) to SC applications (e.g., Uber), there is a substantial precedent that SC systems have a responsibility not only to effective task assignment but also to privacy protection. To address these often-conflicting responsibilities, we propose a framework, Task Assignment with Federated Preference Learning, which 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 two phases, i.e., a federated preference learning and a task assignment phase. Specifically, in the first phase, we design a local preference model for each platform center based on historical data. Meanwhile, the horizontal federated learning with a client-server structure is introduced to collaboratively train these local preference models under the orchestration of a central server. The task assignment phase aims to achieve effective and efficient task assignment by considering workers' preferences. Extensive evaluations over real data show the effectiveness and efficiency of the paper's proposals.

CCS CONCEPTS

• **Networks** → **Location based services**; • **Information systems** → *Recommender systems*; • **Security and privacy**;

KEYWORDS

preference; task assignment; federated learning; spatial crowdsourcing

*Corresponding author: Yan Zhao.

ACM Reference Format:
Jiaxin Liu, Liwei Deng, Hao Miao, Yan Zhao, and Kai Zheng. 2022. Task Assignment with Federated Preference Learning in Spatial Crowdsourcing. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511808.3557465

1 INTRODUCTION

The last decade has witnessed substantial advances of Spatial Crowdsourcing (SC), which enables people to move as multi-modal sensors that perform a variety of location-based tasks [9, 15, 33]. The social and ethical concerns raised by SC are increasingly attracting attention, including issues like privacy [10, 17, 23, 24, 26, 28, 39] and efficiency [19, 21, 36, 42, 46]. In particular, privacy is one of the most common issues 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 mainly focus on protecting the location information of workers or tasks [1, 10, 26, 39] and secure computation of distance [17, 24, 28]. However, these studies ignore worker preference, which is a critical aspect of an SC platform. Assigning workers their interested tasks is a key to ensuring continuous worker participation and satisfaction. On the contrary, when a worker is assigned an uninterested task, the worker may complete the task with low quality or may even sabotage the task, which impacts SC platforms negatively. Considering worker preference is thus a major challenge in SC. Recently, a number of studies have explored worker preference in SC [19, 21, 42, 46]. Zhao et al. [42] model different workers' preferences on different categories of tasks in different time slots with a three-dimensional tensor and supplement the missing entries of tensor with the aid of workers' task-performing history and context matrices. The study [21] uses the historical task-performing data to maximize the mutual information among workers in order to learn the informative representation vectors of groups and further learn the group

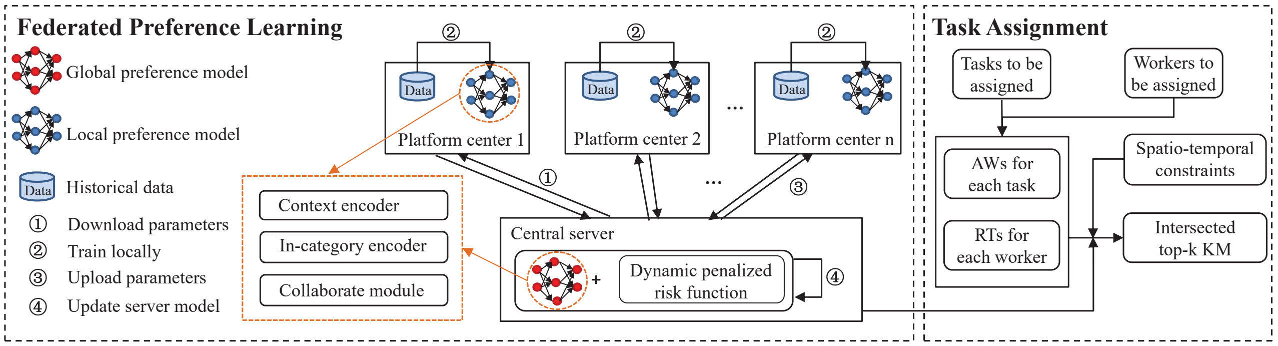


Figure 1: Framework Overview

preferences. However, they do not consider the privacy when modeling worker preferences. Instead, they transmit the raw data to the SC platform directly and train the preference model in a centralized way. In practice, data transfer migration between platform centers is cumbersome and each platform center is unwilling to share their data with others in order to prevent raw data leakage. In such a case, task assignment often fails to obtain effective task assignment solutions. To solve the above issues, we study a preference-driven task assignment problem based on federated learning, which aims to protect the raw data and take workers' preferences in task assignment. Federated Learning (FL) is a machine learning setting where many clients (e.g., mobile devices, organizations, or platforms) collaboratively 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 systemic privacy risks and costs resulting from traditional, centralized machine learning [11].

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 consists of a federated preference learning and a task assignment phase, as shown in Figure 1. In the first phase, we design a federated preference model for each local center, and all local models are combined with a central server. For each preference model, each platform center first downloads parameters from the central server and utilizes its local historical data to learn workers' preferences for all task categories by using a context encoder, an in-category encoder, and a collaborate module. Then, a central server receives the updated model parameters and aggregate them to update the global preference model. In the aggregation part, we add a dynamic penalized risk function to achieve better performance. In the task assignment phase, we first calculate Available Workers (AWs) for each task and 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 by considering workers' preference.

To summarize, the main contributions are four-fold.

1) We propose a privacy-protecting SC framework, called Task Assignment with Federated Preference Learning (TA-FPL). To the best of our knowledge, this is the first study in SC that applies federated learning, which protects the privacy of workers' historical performing raw data effectively.

2) We combine the preference model with federated learning to learn workers' preferences, where a dynamic penalized risk function is designed to aggregate the transmitted parameters of all selected platform centers.

3) We propose an intersected top-k KM algorithm based on workers' preferences to achieve effective and efficient task assignment.

4) We conduct sufficient experiments on a real-world dataset, offering evidence of the effectiveness and efficiency of the proposed framework.

The remainder of this paper is organized as follows. Preliminary concepts and notations are introduced in Section 2. We then present federated preference learning and intersected Top-k KM task allocation algorithm in Section 3. We report the experimental results in Section 4. Section 5 surveys the related work and Section 6 offers conclusions.

2 PROBLEM DEFINITION

We proceed to present necessary preliminaries and then 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 [13], 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, an 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 time instance, 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.

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.

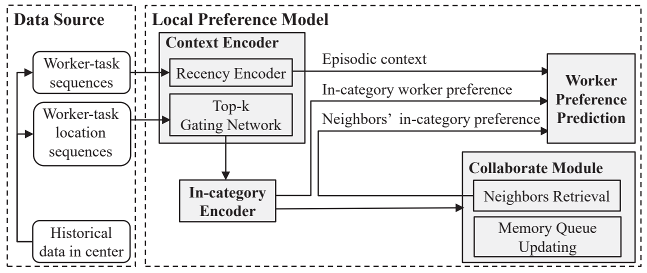


Figure 2: Local Preference Model

3.1 Federated Preference Learning

In this section, we introduce the local preference model and federated model training, respectively. Noted that 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* [16, 35], 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 [16, 35], 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_i^Q, W_i^K, W_i^V 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.

$$A_i^j = Attention(H^j W_i^Q, H^j W_i^K, H^j W_i^V),$$
$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_a/n_h}}), \quad (1)$$
$$H^{j+1} = LN(H^j + FC^j(A^j)),$$

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:

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

where $p(\hat{c}_{N+1} = j)$ represents the probability of category j being the category of the next task location and e_j^z 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:

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

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 $P^r = [P_M^r, ..., P_1^r]$ 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:

$$h^r = SelfAttention(X^r + P^r) \quad (4)$$

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 $\{h^{c_j}\}_{j=1}^k$ 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 $[l_1^c, ..., l_T^c]$ is projected through the input task location embedding layer E_{in} into a set of dense vectors $X^c = [e_{l_1}^c, ..., e_{l_T}^c]$. The relative positions $[T, ..., 1]$ of these records to the next task location are projected through the position embedding layer P into $P^c = [P_T^c, ..., P_1^c]$. 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:

$$h^c = SelfAttention(X^c + P^c) \quad (5)$$

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_w^c 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:

$$sim(h_w^c, h_i^c) \propto exp(\langle h_w^c, h_i^c \rangle) \quad (6)$$

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:

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

For the writing operation of *Mem*, *Mem* is randomly initialized. 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, which pushes the most recently served workers' representations of category c while dynamically popping out the representations of workers 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. For the preference score of task category c_j, $score(\hat{c}_j) = score(\hat{c}_{N+1} = c_j)$, we can calculate it in Eq. 2:

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

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 episodic context inferred from the most recent task location records h^r, we concatenate these three representations for each of k categories and project 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:

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

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

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

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

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

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 [2]. 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 an 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-paramter to controll the weights.

Given a central server which 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. 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 $P_t \subset \{pc_1, ..., pc_N\}$ are active, where N is the number of all platform centers. 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 5). Besides, each active platform center computes their local gradient to satisfy local optima condition (line 6), then transmitting the updated parameters to 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 [8], and model parameters θ^t (lines 11–12).

Algorithm 1: Federated Preference Model Training

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

3.2 Preference-driven Task Assignment

In this section, we 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 center and then uploaded to the central server where the task assignment will be executed.

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 $RT(w)$ ($\forall s \in RT(w), w \in W$) should satisfy the following 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).

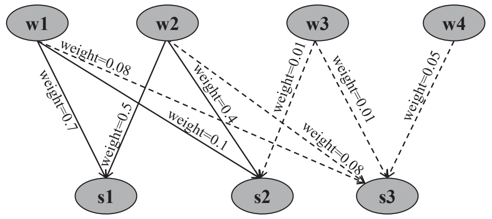


Figure 3: Worker-Task Bipartite Graph

3.2.2 Intersected Top-k KM algorithm. In this part, we transform the task assignment problem into a Bipartite Maximum Weight Matching problem [38], which is based on a graph that is represented by $G = (V, E)$ with a set of vertices V and a set of edges E. 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 number of V and the number 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 to 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 to task s_j, denoted as

$p_{w_i}(s_j)$, which can be calculated as:

$$p_{w_i}(s_j) = exp(\langle h^z_{w_i}, e^z_{c(s_j)} \rangle), \tag{17}$$

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 2: 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      $val_{worker}[w] \leftarrow max(weight(v^W_w, v^S_s))$;
4  for each worker $w \in W$ do
5      while $val_{worker}[w] > 0$ do
6          Initialize $vis_{task}$ and $vis_{worker}$ with $False$;
7          if FindTask(w,0) then break ;
8          else
9              d=$INF$;
10             for each task $s \in S$ do
11                 if ! $vis_{task}[s]$ then
12                     d=min(d,$slack[s]$);
13             for each worker $w \in W$ do
14                 Decrease $val_{worker}[w]$ by $d$ if $w$ is visited;
15             for each task $s \in S$ do
16                 Increase $val_{task}[s]$ by $d$ if $s$ is visited;
17                 Decrease $slack[s]$ by $d$ if $s$ is not visited;
18 $ReassignTask(match, \widetilde{AW})$;
19 return $match$;
```

Figure 3 depicts an example of 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, the edge (v^w_i, v^s_j) will be drawn as a solid line (e.g., edge (v^w_1, v^s_1) when $k = 2$). If these two vertexes only satisfy the spatio-temporal constraints, the edge (v^w_i, v^s_j) will be a dotted line, which will be removed from original graph in our algorithm. For example, the edge (v^w_3, v^s_2) is drawn as a dotted line, because w_3 is not in the top-2 of $\widetilde{AW}_2(s_2)$. The higher $weight(v^w_i, v^s_j)$ of the edge (v^w_i, v^s_j) 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 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 2. 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 3 (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 [38]. 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).

Algorithm 3: 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      for each task $s$ is adjacent to $w$ in $G$ do
5          if $vis_{task}[s]$ then continue ;
6          $gap \leftarrow val_{worker}[w] + val_{task}[s] - weight(v^W_w, v^S_s)$;
7          if $gap! = 0$ then
8              $slack[s] = min(slack[s], gap)$;
9          else if match[s]==-1 or FindTask(match[s],rdp+1) then
10             Assign value $w$ to $match[s]$ and return $True$;
11     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 3 is set to k, which can reduce the competition among workers (lines 2). To reduce the number of task assignments loss, a reassignment task algorithm is designed, which is shown in Algorithm 4. 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 4–8).

Algorithm 4: Reassignment Task Algorithm

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

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

We use a check-in dataset from Twitter, which is used widely in the experimental evaluation of SC platforms [4, 6, 7, 21]. It provides

Table 2: Performance 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**
FLP	0.4120	0.5832	0.6850

check-in data in the United States from September 2010 to January 2011 except Hawaii and Alaska, including 62462 venue locations and 61412 user locations. First, we use FourSquare's API[1] to generate the corresponding category information of the venue. Then, for each worker and task, we take the average value of the corresponding check-in locations as its location information. For each check-in, we simulate that the user is the worker, and 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 [38] to allocate all workers and their historical data to corresponding platform centers. A check-in record means 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 Experiment Results

4.2.1 Performance of Federated Preference Learning. We first evaluate the performance of the federated preference learning phase.

Evaluation Methods. We study the following 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 way. It means all workers belong to only one 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, where we use the first 80% of check-in records as the training set, the next 10% records as the validation set, and the remaining 10% as the testing set. We remove the 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 achieves the best followed by FLP, POISeqPop and CatePEP. The models (i.e., CTP and FTP) based on our category-aware preference model far outperform the others. That demonstrates our preference model can provide more accurate estimations for workers' preferences. Considering that the local-device level empirical loss are inconsistent with those of the global empirical loss, it is normal 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.

4.2.2 Performance of Task Assignment. We proceed to study the performance of task assignment. Table 3 shows our experimental settings, where the default values of all parameters are underlined.

Table 3: 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>

Evaluation Methods. We study the following task assignment algorithms.

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.

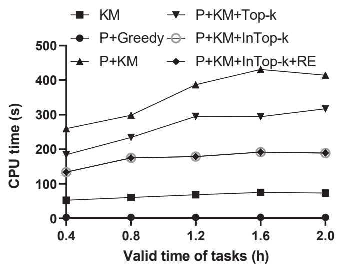
5) P+KM+InTop-k: The P+KM algorithm, using our intersected Top-k pruning strategy to prune edges.

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

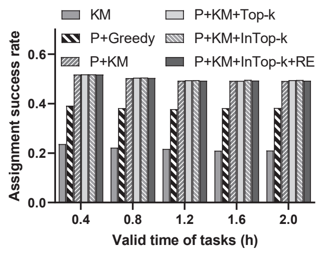
Metrics. Three metrics are compared among the methods, including CPU time, Assignment Success Rate (ASR), and the number of task assignments. The CPU time is the time cost for finding the task assignment. ASR is the ratio between the number of successful assignments of all workers and the total number 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 tasks' valid time $e - p$ on the performance of task assignments (see Figure 4). It can be seen from Figures 4(a) and 4(c) that the CPU time and the number of task assignments of all algorithms shows 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 shows the efficiency and effectiveness of our proposed algorithms. As shown in Figure 4(b), P+KM related algorithms achieve the highest Assignment Success Rate (ASR), which shows the importance of considering preferences. As shown in Figure 4(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

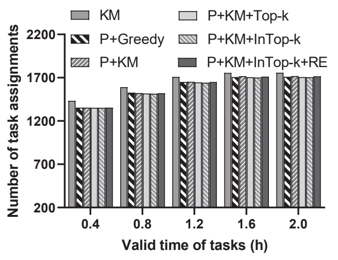
[1]https://developer.foursquare.com/



(a) CPU Time

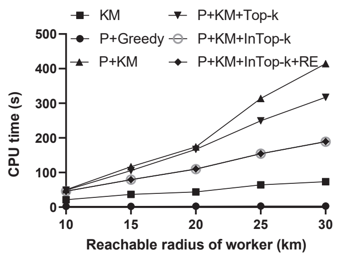


(b) Assignment Success Rate

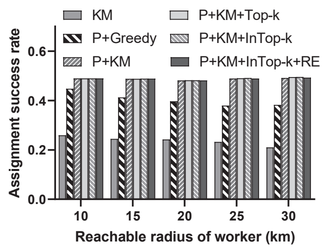


(c) Number of Task Assignments

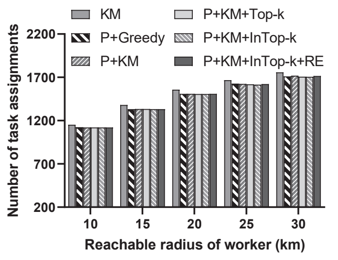
Figure 4: Performance of Task Assignment: Effect of $e - p$



(a) CPU Time

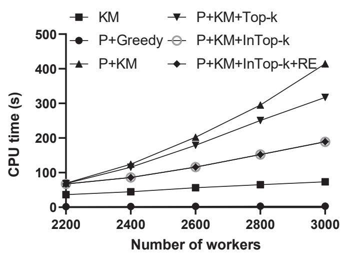


(b) Assignment Success Rate

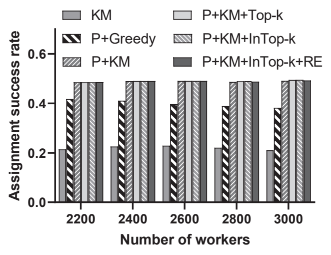


(c) Number of Task Assignments

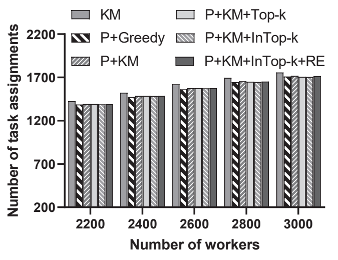
Figure 5: Performance of Task Assignment: Effect of r



(a) CPU Time



(b) Assignment Success Rate



(c) Number of Task Assignments

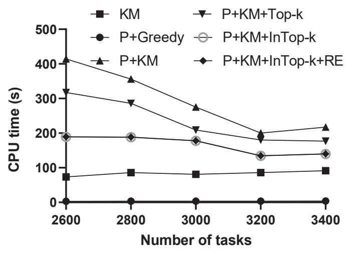
Figure 6: Performance of Task Assignment: Effect of $|W|$

greatly and many workers tend to choose their interested tasks, thus leading to a lower number of task assignments. This reflects from the side that the preference scores we learn are accurate and discriminative.

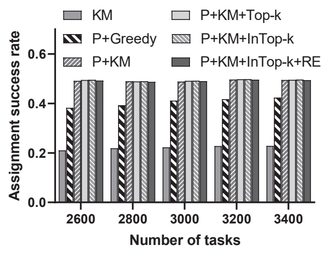
Effect of r. We further evaluate the effect of the reachable distance r of workers. It can be seen from Figure 5(a) that when r increases, the CPU time of all algorithms shows a similar growth trend. The reason behind it is that when the reachable distance of workers increases, the number of available workers and the number of reachable tasks increase, resulting in a larger search space. The P+Greedy algorithm still consumes least CPU time, but its performance in ASR is obviously not as good as other P+KM related algorithms (see Figure 5(b)). In addition, as shown in Figure 5(c), with r increasing, 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 6(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 6(b), all preference-based algorithms keep high ASR values, and the number of task assignments increases (cf. Figure 6(c)). In summary, P+KM+InTop-k+RE performs well in terms of CPU time and ASR while offering the acceptable number of task assignments.

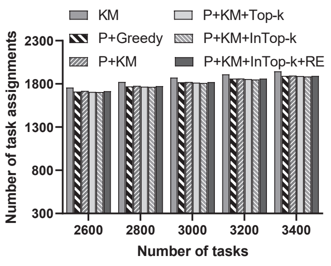
Effect of $|S|$. We study the effect of the number $|S|$ of tasks. In Figure 7(a), the CPU time of P+KM related algorithms decreases because as the number of tasks increases, the occurrence probability of tasks that workers are most interested in increases and the discriminative preference scores also disperse the competition among workers. Moreover, the CPU time of KM-related algorithms is higher 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. P+KM+InTop-k+RE performs better in terms of CPU time and ASR, showing its superiority.



(a) CPU Time

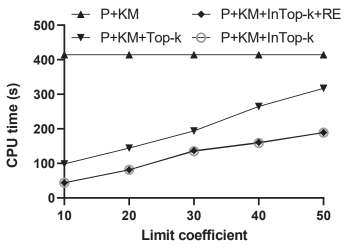


(b) Assignment Success Rate

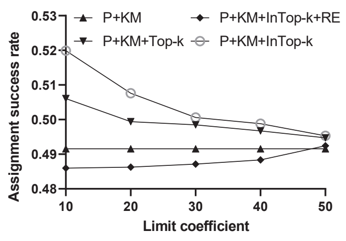


(c) Number of Task Assignments

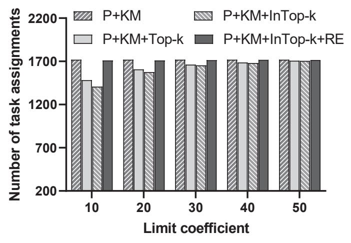
Figure 7: Performance of Task Assignment: Effect of $|S|$



(a) CPU Time



(b) Assignment Success Rate



(c) Number of Task Assignments

Figure 8: Performance of Task Assignment: Effect of k

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). We only 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 8 keep the same value. In figure 8(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.

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 [5, 20, 30, 32, 34, 36, 38, 40–45]. Most of the existing studies focus on task assignment [3, 13, 14, 27, 29, 31, 37, 47], 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 [25, 28], privacy-preserving task assignment is proposed to make users (workers/task requesters) perturb their locations with Geo-Indistinguishability [1] 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 [11]. 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 [8, 12, 18, 22]. To solve the above issues, 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

In this paper we propose a framework called Task Assignment with Federated Preference Learning (TA-FPL), which aims to find the optimal task assignment while considering workers' preferences and protect workers' raw data. TA-FPL consists of a Federated Preference Learning (FPL) phase and a Preference-driven Task Assignment (PTA) phase. Specifically, we design the local platform center's preference model combined with federated training method in the FPL phase. In the PTA phase, we propose an Intersected Top-k KM algorithm to achieve effective and efficient task assignment by considering workers' preferences obtained by the first phase. To the best of our knowledge, this is the first study in SC that applies federated learning and combine the preference modeling and protection of the raw data privacy. An empirical study based on a real dataset confirms the superiority of our proposed algorithms.

ACKNOWLEDGMENTS

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), and Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021).

REFERENCES

[1] Miguel E Andrés, Nicolás E Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: Differential privacy for location-based systems. In *SIGSAC*. 901–914.
[2] Renqin Cai, Jibang Wu, Aidan San, Chong Wang, and Hongning Wang. 2021. Category-aware collaborative sequential recommendation. In *SIGIR*. 388–397.
[3] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, and Jizhong Zhao. 2016. Task assignment on multi-skill oriented spatial crowdsourcing. *TKDE* 28, 8 (2016), 2201–2215.
[4] Peng Cheng, Xiang Lian, Lei Chen, and Cyrus Shahabi. 2017. Prediction-based task assignment in spatial crowdsourcing. In *ICDE*. 997–1008.
[5] Yue Cui, Liwei Deng, Yan Zhao, Bin Yao, Vincent W Zheng, and Kai Zheng. 2019. Hidden poi ranking with spatial crowdsourcing. In *SIGKDD*. 814–824.
[6] Hung Dang, Tuan Nguyen, and Hien To. 2013. Maximum complex task assignment: Towards tasks correlation in spatial crowdsourcing. In *IIWAS*. 77–81.
[7] Dingxiong Deng, Cyrus Shahabi, and Ugur Demiryurek. 2013. Maximizing the number of worker's self-selected tasks in spatial crowdsourcing. In *SIGSPATIAL*. 324–333.
[8] Alp Emre Durmus, Zhao Yue, Matas Ramon, Mattina Matthew, Whatmough Paul, and Saligrama Venkatesh. 2021. Federated Learning Based on Dynamic Regularization. In *ICLR*.
[9] Srinivasa Raghavendra Bhuvan Gummidi, Xike Xie, and Torben Bach Pedersen. 2019. A survey of spatial crowdsourcing. *TODS* 44, 2 (2019), 1–46.
[10] Weishan Huang, Xinyu Lei, and Hongyu Huang. 2021. PTA-SC: Privacy-Preserving Task Allocation for Spatial Crowdsourcing. In *WCNC*. 1–7.
[11] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning* 14, 1–2 (2021), 1–210.
[12] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. 2019. SCAFFOLD: Stochastic Controlled Averaging for On-Device Federated Learning. *CoRR* abs/1910.06378 (2019).
[13] Leyla Kazemi and Cyrus Shahabi. 2012. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIAL*. 189–198.
[14] Leyla Kazemi, Cyrus Shahabi, and Lei Chen. 2013. Geotrucrowd: trustworthy query answering with spatial crowdsourcing. In *SIGSPATIAL*. 314–323.
[15] Guoliang Li, Yudian Zheng, Ju Fan, Jiannan Wang, and Reynold Cheng. 2017. Crowdsourced data management: Overview and challenges. In *SIGMOD*. 1711–1716.
[16] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *WSDM*. 322–330.
[17] Maocheng Li, Jiachuan Wang, Libin Zheng, Han Wu, Peng Cheng, Lei Chen, and Xuemin Lin. 2021. Privacy-Preserving Batch-based Task Assignment in Spatial Crowdsourcing with Untrusted Server. In *CIKM*. 947–956.
[18] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *MLSys* 2 (2020), 429–450.
[19] Xiang Li, Yan Zhao, Jiannan Guo, and Kai Zheng. 2020. Group task assignment with social impact-based preference in spatial crowdsourcing. In *DASFAA*. 677–693.
[20] Xiang Li, Yan Zhao, Xiaofang Zhou, and Kai Zheng. 2020. Consensus-Based Group Task Assignment with Social Impact in Spatial Crowdsourcing. *Data Science and Engineering* 5, 4 (2020), 375–390.
[21] Yunchuan Li, Yan Zhao, and Kai Zheng. 2021. Preference-aware Group Task Assignment in Spatial Crowdsourcing: A Mutual Information-based Approach. In *ICDM*. 350–359.
[22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*. PMLR, 1273–1282.
[23] Chenxi Qiu, Anna Squicciarini, Zhuozhao Li, Ce Pang, and Li Yan. 2020. Time-efficient geo-obfuscation to protect worker location privacy over road networks in spatial Crowdsourcing. In *CIKM*. 1275–1284.
[24] Chenxi Qiu and Anna Cinzia Squicciarini. 2019. Location Privacy Protection in Vehicle-Based Spatial Crowdsourcing Via Geo-Indistinguishability. In *ICDCS*. 1061–1071.
[25] Qian Tao, Yongxin Tong, Zimu Zhou, Yexuan Shi, Lei Chen, and Ke Xu. 2020. Differentially private online task assignment in spatial crowdsourcing: A tree-based approach. In *ICDE*. 517–528.
[26] Hien To, Gabriel Ghinita, and Cyrus Shahabi. 2014. A framework for protecting worker location privacy in spatial crowdsourcing. *PVLDB* 7, 10 (2014), 919–930.
[27] Hien To, Cyrus Shahabi, and Leyla Kazemi. 2015. A server-assigned spatial crowdsourcing framework. *TSAS* 1, 1 (2015), 1–28.
[28] Hien To, Cyrus Shahabi, and Li Xiong. 2018. Privacy-preserving online task assignment in spatial crowdsourcing with untrusted server. In *ICDE*. 833–844.
[29] Yongxin Tong, Lei Chen, Zimu Zhou, Hosagrahar Visvesvaraya Jagadish, Lidan Shou, and Weifeng Lv. 2018. SLADE: A smart large-scale task decomposer in crowdsourcing. *TKDE* 30, 8 (2018), 1588–1601.
[30] Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. 2016. Online minimum matching in real-time spatial data: experiments and analysis. *PVLDB* 9, 12 (2016), 1053–1064.
[31] Yongxin Tong, Libin Wang, Zhou Zimu, Bolin Ding, Lei Chen, Jieping Ye, and Ke Xu. 2017. Flexible online task assignment in real-time spatial data. *PVLDB* 10, 11 (2017), 1334–1345.
[32] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. 2018. A unified approach to route planning for shared mobility. *PVLDB* 11, 11 (2018), 1633.
[33] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, and Cyrus Shahabi. 2020. Spatial crowdsourcing: a survey. *PVLDB* 29, 1 (2020), 217–250.
[34] Jiayang Tu, Peng Cheng, and Lei Chen. 2019. Quality-assured synchronized task assignment in crowdsourcing. *TKDE* 33, 3 (2019), 1156–1168.
[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *NIPS* 30 (2017).
[36] Ziwei Wang, Yan Zhao, Xuanhao Chen, and Kai Zheng. 2021. Task assignment with worker churn prediction in spatial crowdsourcing. In *CIKM*. 2070–2079.
[37] Jinfu Xia, Yan Zhao, Guanfeng Liu, Jiajie Xu, Min Zhang, and Kai Zheng. 2019. Profit-driven Task Assignment in Spatial Crowdsourcing.. In *IJCAI*. 1914–1920.
[38] Guanyu Ye, Yan Zhao, Xuanhao Chen, and Kai Zheng. 2021. Task allocation with geographic partition in spatial crowdsourcing. In *CIKM*. 2404–2413.
[39] Xun Yi, Fang-Yu Rao, Gabriel Ghinita, and Elisa Bertino. 2018. Privacy-preserving spatial crowdsourcing based on anonymous credentials. In *MDM*. 187–196.
[40] Yan Zhao, Xuanhao Chen, Liwei Deng, Tung Kieu, Chenjuan Guo, Bin Yang, Kai Zheng, and Christian S. Jensen. 2022. Outlier Detection for Streaming Task Assignment in Crowdsourcing. In *WWW*.
[41] Yan Zhao, Jiannan Guo, Xuanhao Chen, Jianye Hao, Xiaofang Zhou, and Kai Zheng. 2021. Coalition-based task assignment in spatial crowdsourcing. In *ICDE*. 241–252.
[42] Yan Zhao, Jinfu Xia, Guanfeng Liu, Han Su, Defu Lian, Shuo Shang, and Kai Zheng. 2019. Preference-aware task assignment in spatial crowdsourcing. In *AAAI*. 2629–2636.
[43] Yan Zhao, Kai Zheng, Yue Cui, Han Su, Feida Zhu, and Xiaofang Zhou. 2020. Predictive task assignment in spatial crowdsourcing: a data-driven approach. In *ICDE*. 13–24.
[44] Yan Zhao, Kai Zheng, Jiannan Guo, Bin Yang, Torben Bach Pedersen, and Christian S Jensen. 2021. Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches. In *ICDE*. 265–276.
[45] Yan Zhao, Kai Zheng, Yang Li, Han Su, Jiajun Liu, and Xiaofang Zhou. 2019. Destination-aware Task Assignment in Spatial Crowdsourcing: A Worker Decomposition Approach. *TKDE* (2019), 2336–2350.
[46] Yan Zhao, Kai Zheng, Hongzhi Yin, Guanfeng Liu, Junhua Fang, and Xiaofang Zhou. 2020. Preference-aware task assignment in spatial crowdsourcing: from individuals to groups. *TKDE* (2020).
[47] Libin Zheng, Lei Chen, and Jieping Ye. 2018. Order dispatch in price-aware ridesharing. *PVLDB* 11, 8 (2018), 853–865.