FedFPM: A Unified Federated Analytics Framework for Collaborative Frequent Pattern Mining

Zibo Wang*, Yifei Zhu*†, Dan Wang‡, and Zhu Han§
*UM-SJTU Joint Institute, Shanghai Jiao Tong University
†Department of Electronic Engineering, Shanghai Jiao Tong University
‡Department of Computing, The Hong Kong Polytechnic University
§Department of Electrical and Computer Engineering, University of Houston
Email: wangzibo@sjtu.edu.cn, yifei.zhu@sjtu.edu.cn, csdwang@comp.polyu.edu.hk, zhan2@uh.edu

***Abstract*—Frequent pattern mining is an important class of knowledge discovery problems. It aims at finding out high-frequency items or structures (*e.g.*, itemset, sequence) in a database, and plays an essential role in deriving other interesting patterns, like association rules. The traditional approach of gathering data to a central server and analyze is no longer viable due to the increasing awareness of user privacy and newly established laws on data protection. Previous privacy-preserving frequent pattern mining approaches only target a particular problem with great utility loss when handling complex structures. In this paper, we take the first initiative to propose a unified federated analytics framework (FedFPM) for a variety of frequent pattern mining problems, including item, itemset, and sequence mining. FedFPM achieves high data utility and guarantees local differential privacy without uploading raw data. Specifically, FedFPM adopts an interactive query-response approach between clients and a server. The server meticulously employs the Apriori property and the Hoeffding's inequality to generates informed queries. The clients randomize their responses in the reduced space to realize local differential privacy. Experiments on three different frequent pattern mining tasks demonstrate that FedFPM achieves better performances than the state-of-the-art specialized benchmarks, with a much smaller computation overhead.**

***Index Terms*—federated analytics, frequent pattern mining, differential privacy, collaborative computing**

I. Introduction

A tremendous amount of data are being generated by widely deployed edge devices. It is estimated that there will be 27.1 billion networked devices in 2021 with 278.1 Exabytes of data being generated every month [1]. The increasing amount of accessible data provide great opportunities for useful knowledge derivation. Frequent pattern mining (FPM) is an important class of knowledge discovery problems that aim at finding out high-frequency items or structures (*e.g.*, itemset, sequence) in a database [2]. It also plays an essential role in many other pattern mining tasks, such as association rules, correlations, and classifiers. FPM has been widely studied and applied in a wide range of real-world applications. For instance, personalized recommendation systems rely on mining frequent web usage data (itemsets) to characterize users [3]. Ride-sharing service providers like Uber greatly benefit from mining users' trajectories (sequences of locations) [4].

Traditional FPM solutions adopt the gather-and-analyze paradigm, where the data born in the edge is gathered by a centralized server before being analyzed [2]. However, with the increasing awareness of privacy preservation, laws and regulations, such as the GDPR in European Union [5] and the CPPA in California [6], are established worldwide to restrict the access of raw edge data, making the previous gather-and-analyze paradigm no longer viable.

Recognizing the importance of privacy preservation, various privacy-preserving FPM solutions have also been proposed in recent years. Particularly, researchers from academia and industry develop algorithms to satisfy local differential privacy (LDP), a de facto criteria of local privacy [7]. Google presents RAPPOR which uses bloom filters to encode the browser homepage choices [8]. Apple presents SFP to discover more complex frequent sequences [9]. However, these privacy-preserving solutions have two major drawbacks. First, their realization of LDP is based on randomizing the raw data. The data utility significantly deteriorates with the sophistication of the data structures and the increasing system scales. Second, previous solutions only handle a particular form of patterns with limited power of generalization.

The federated paradigm emerges as a promising direction to perform AI or data science tasks on edge devices with the privacy preserved [10]–[13]. A federated system consists of a centralized server and many clients (edge devices) holding private raw data. Server and clients collaboratively work on a data-centric computing task without sharing the raw data. As its earliest application, federated learning (FL) focuses on collaboratively training a neural network, in which a neural network model is transmitted between clients and a server; local computation calculates gradient descent; the central server aggregates them by averaging-like operations [10], [11]. It has been widely used to support training various types of neural networks, including deep recurrent models for language modelling, convolutional models for computer vision, graph convolutional models for graph-structured data [10], [11], [14].

On the contrary, as a newly emerged sibling to FL, federated analytics (FA) tackles the data science tasks that are

This work is supported by SJTU Explore-X grant and SJTU Global Strategic Partnership Fund (2021 SJTU-HKUST). Dan Wang's work is supported in part by GRF 15210119, 15209220, 15200321, ITF-ITSP ITS/070/19FP, CRF C5026-18G, C5018-20G, PolyU 1-ZVPZ, and a Huawei Collaborative Project. Zhu Han's work is partially supported by NSF CNS-2128368, CNS-2107216, Toyota and Amazon. Corresponding author: Yifei Zhu

not suitable for neural networks, like model evaluation [13], out-of-dictionary words discovery [15], data heterogeneity measurement [16]. In particular, a specific frequent item mining problem has also been investigated. In federated out-of-dictionary words discovery [15], a tree-based structure is used to discover the most frequent word in users' typing records. However, this endeavor only addresses a particular FPM problem, and sacrifices LDP for better data utility. A wide variety of FPM problems in the federated settings remain to be handled. Considering the close relationship between these problems and more stringent requirements on both utility and privacy protection, *the focus of this paper is to have a unified FA framework that can handle different FPM problems.*

Designing a unified FA-based FPM framework is non-trivial. First, the mining procedure is highly dependent on the pattern structure. Identifying the right decomposition unit and the central aggregation approach for a general federated framework is difficult. Second, naïvely adding noises to the raw data greatly jeopardizes the data utility, especially for complex patterns. Therefore, the interaction between the server and the clients should be ingeniously designed to reduce the utility loss in the process of satisfying LDP. Third, considering the reliability in real-world systems, the framework should theoretically bound its mining error, even based on the noisy responses from the sampled clients.

In this paper, we propose the first unified FA-based framework for FPM problems (FedFPM). FedFPM's modular design is capable of handling classical frequent item, itemset, sequence mining situations. FedFPM adopts an interactive query-response approach between the clients and the server to accomplish the mining tasks. Unlike non-FA FPM solutions that require clients to directly process their local data [8], [9], FedFPM only requires sampled clients to respond to queries on their partial dataset with simple binary answers. These responses are aggregated further at the server side to generate informed queries for the next round. The whole process runs iteratively until all frequent patterns are identified. In summary, our contributions are

- To the best of our knowledge, FedFPM is the first unified FA framework for a variety of FPM problems, including frequent item, itemset, and sequence mining.
- Our framework achieves high data utility, satisfies LDP, and incurs minor communication and computation costs for each individual client via efficient candidate generation and verification design.
- The performance of FedFPM is theoretically guaranteed. FedFPM adopts the Hoeffding's inequality to adaptively thresholding each candidate with no assumptions on data statistics. The local differential privacy and bounded mining error are theoretically proved to ensure the privacy and data utility of our framework.
- Extensive experiments on three large-scale datasets show that FedFPM achieves comparative or even better performances (97%∼411% of data utility) than the state-of-the-art specialized benchmarks in different FPM scenarios, with much smaller computation overhead (86%∼98% less).

TABLE I
COMPARISON BETWEEN THE ALGORITHMS. WE PRESENT THE OUTPUT QUALITY OF FPM AS BAD (+), MEDIUM (++), AND GOOD (+++), AND THEIR ABILITY IN GENERALIZATION OR PRIVACY PRESERVATION AS NO ABILITY (BLANK), LIMITED ABILITY (△), AND FULL ABILITY (✓).

	Pattern type			Performance	
	Item	Itemset	Sequence	Utility	Privacy
RAPPOR [8]	✓	△		+++	✓
SFP [9]			✓	+	✓
TrieHH [15]			✓	++	△
FedFPM	✓	✓	✓	+++	✓

The rest of the paper is organized as follows: related work is surveyed in Section II; the common variations of FPM are reviewed in Section III; in Section IV, we present the system model and problem formulation of federated FPM problems; an overview of FedFPM is presented in Section V, with its detailed design introduced in Section VI; the evaluation part is presented in Section VII; Section VIII concludes this paper.

II. RELATED WORK

A. Federated analytics

Compared to its FL sibling, which aims to collaboratively train neural networks, FA has a wide spectrum of applications in the field of data science and has been relatively unexplored. In [13], the FA-based algorithms evaluate the FL model and identify the songs played in surrounding smartphones. In [15], the TrieHH algorithm finds frequently typed words in the edge devices, which is a constrained variety of FSM, and tradeoff privacy for data utility. In [16], data heterogeneity of the clients is measured and quantified to perform intelligent client selection for other federated tasks. However, existing FA solutions are all task-specific, and are hard to be transferred to other scenarios. In the paper, we take the initiative to design a FA mechanism framework for a class of FPM problems.

Distributed data mining has also been extensively studied [17]. Although general distributed data mining and FA both aim at performing data mining tasks in a distributed way. FA focuses more on the large-scale, heterogeneous environments, and especially emphasizes privacy preservation without uploading raw data. On the contrary, in general distributed data mining, various heterogeneity are usually mitigated after the prepossessing operations, and the privacy is of little concern.

B. Privacy-preserving FPM

Given the importance of FPM and the increasing awareness of privacy protection, privacy-preserving FPM has gained traction in recent years. These works usually satisfy *user level* privacy schemes, represented by LDP, to guarantee strong privacy preservation. Various sketching methods are adopted to encode the raw data, such as bloom filters used by RAPPOR [8] and count-mean-sketch proposed by Apple [9]. Extra efforts have also been made based on these works to optimize complexity [18] and data utility [19]. However, the usage of sketching disables the direct decoding of raw data. Therefore,

they are effective only when the possible patterns are within a small domain. Some researchers design application-specific algorithms to enable the privacy-preserving FPM for other types of patterns, but these algorithms only work in constrained scenarios [20]–[23]. In [9], the SFP algorithm handles the complex sequence pattern, but the usage of two count-mean-sketches significantly harms the data utility.

Compared to the existing non-FA FPM solutions, FedFPM has two major advantages. First, while the usages of existing applications are limited to one or several pattern types, FedFPM supports wide variations of FPM scenarios. Second, FedFPM is superior in utility and privacy comprehensively. It improves the data utility with the power of the FA paradigm, and satisfies the strong privacy guarantee of LDP. A comparison between FedFPM and existing privacy-preserving FPM algorithms is presented in Table I.

III. Review of the FIM, FIsM and FSM

In this section, we first review the representative FPM problems: frequent item mining (FIM), frequent itemset mining (FIsM), and frequent sequence mining (FSM).

A. Frequent item mining

Denote $\mathcal{I} = \{i_1, i_2, \cdots, i_n\}$ as a set of n attributes called items. In the FIM scenario, each client data d is a subset of $\mathcal{I}$, and each pattern p is an element of $\mathcal{I}$, *i.e.*,

$$d \subseteq \mathcal{I} \text{ and } p \in \mathcal{I}. \tag{1}$$

We claim that p appears in d when $p \in d$. p is a frequent item when it appears in a sufficient proportion of client data d that exceeds a given threshold f. Given this threshold f, FIM aims at identifying all frequent items existing in the system.

B. Frequent itemset mining

We reuse the notion of $\mathcal{I}$ as a set of items. In the FIsM scenario, each client data d is a subset of $\mathcal{I}$, and each pattern p is also a subset of $\mathcal{I}$, *i.e.*,

$$d \subseteq \mathcal{I} \text{ and } p \subseteq \mathcal{I}. \tag{2}$$

We claim that p appears in d when $p \subseteq d$. p is a frequent itemset when it appears in a sufficient proportion of client data d that exceeds a given threshold f. Similarly, given this threshold f, FIsM aims at identifying all frequent subsets existing in the system.

C. Frequent sequence mining

We reuse the notion of $\mathcal{I}$ as a set of items. In the FSM scenario, each client data d is a sequence of items in $\mathcal{I}$. Denote $|d|$ as the length of d, we can express d as

$$d := a_1^{<d>} \rightarrow a_2^{<d>} \rightarrow \cdots \rightarrow a_{|d|}^{<d>}, \tag{3}$$

where $a_k^{<d>}$ refers to the k-th item in sequence d, *i.e.*, $a_k^{<d>} \in \mathcal{I}, \forall k \in [1, |d|]$. In this paper, we focus on *continuous subsequences* of client data in FSM. The restriction on continuity enables us to preserve the microscopic structure of raw data. Here we use $Sub(d, i, j)$ to define a continuous subsequence of d between indexes i and j, *i.e.*,

$$Sub(d, i, j) := a_i^{<d>} \rightarrow a_{i+1}^{<d>} \rightarrow \cdots \rightarrow a_j^{<d>}, \tag{4}$$

where $1 \leq i \leq j \leq |d|$. p is considered to appear in d when p is a continuous subsequence of d, *i.e.*, there exist any i, j that p is equal to $Sub(d, i, j)$. p is considered as a frequent sequence when it appears in a sufficient proportion of client data d that exceeds a given threshold f. Given this threshold f, FSM aims at identifying all frequent subsequences existing in the system.

IV. System Model and Problem Formulation

In this section, we provide the system model and mathematically formulate the problem we solve. We consider the system of N clients, with indexes $1, 2, ..., N$. Client i holds data $d_i \in \mathcal{D}$. We aim at finding pattern p that appears frequently in $\mathcal{D}$. In different scenarios, the definitions of the pattern are different (*e.g.*, item, subset, subsequence). For the sake of simplicity, we borrow the mathematical symbol to represent the relationship that "p appears in d_i" as

$$p \subseteq d_i. \tag{5}$$

For any pattern p, we define its *support*:

$$\mathcal{S}(p) = \sum_{i=1}^{N} \mathbb{I}(p \subseteq d_i), \tag{6}$$

where $\mathbb{I}$ is the indicator function, returning 1 when the statement $p \subseteq d_i$ is true, and returning 0 otherwise. We can represent the *frequency* of a pattern p as

$$\mathcal{F}(p) = \frac{\mathcal{S}(p)}{N}. \tag{7}$$

We say p is a frequent pattern if its frequency exceeds a threshold f:

$$p \text{ is a frequent pattern } \iff \mathcal{F}(p) \geq f. \tag{8}$$

In our federated FPM problem, no raw data is allowed to be uploaded to the server; any uploaded information from the client is required to satisfy a widely used local privacy criterion, local differential privacy (LDP). LDP is firstly proposed in [7]. It aims to regulate the output of each individual client to be closed with each other, so that it cannot be identified by the server. The formal definition of LDP is given as follows:[1]

Definition 1 (local differential privacy)**.** *A randomized algorithm $\mathcal{M}$ satisfies ϵ-LDP when for any two theoretically possible inputs d_i and d_j,*

$$\mathbb{P}(\mathcal{M}(d_i) = y) \leq e^{\epsilon}\mathbb{P}(\mathcal{M}(d_j) = y), \ \forall y \in \text{Range}(\mathcal{M}). \tag{9}$$

We aim to design an upload algorithm $\mathcal{M}$ that performs abstraction of the raw data to prevent uploading raw data. In addition, LDP needs to be enforced on the abstracted uploads. Based on $\mathcal{M}$, we identify a set $\mathcal{P}$ that contains all frequent

[1] ϵ controls the strength of privacy preservation, where lower ϵ indicates stronger privacy protection.

patterns. This federated privacy-preserving FPM problem can then be formally formulated as follows:

$$\max |\mathcal{P}|, \tag{10}$$

$$s.t.\ \mathcal{F}(p) > f,\ \forall p \in \mathcal{P}, \tag{11}$$

$$\mathbb{P}(\mathcal{M}(d_i) = y) \leq e^{\epsilon}\mathbb{P}(\mathcal{M}(d_j) = y),\ \forall y \in \mathrm{Range}(\mathcal{M}). \tag{12}$$

V. Design Overview

In this section, we provide an overview of FedFPM as also being demonstrated in Fig. 1. In general, the execution of FedFPM is an iterative procedure of candidate generation and validation. The server holds a candidate pool, which is made up of candidates that are likely to be frequent patterns, together with their historical response profile. In each round, the server samples a portion of available clients as participants, queries the participants to gain knowledge about the frequency of the candidates, and tries to filter the pool. The procedure can be split into five phases:

1) *Candidate generation.* Once the server is initialized or any candidate is accepted, the server tries to generate new candidates, and put them into the candidate pool with the profile initialized to 0.
2) *Candidate distribution.* The server randomly chooses a subset of clients to be the participants of this round. For each participant, the server sends a candidate randomly chosen from the candidate pool.
3) *Client response.* After receiving the candidate, each participant checks whether the candidate is a pattern of its local data and derives a binary response to it. Then, the clients perform the randomized response technique on the response to realize LDP.
4) *Candidate profile update.* After receiving the noisy response from the candidates, the server directly updates the profile, representing the numbers of yes/no responses it has received.
5) *Candidate filtering.* The server estimates each candidate whether there is sufficient confidence to claim it is (or it is not) a frequent pattern based on its profile. Each candidate can be moved to the accepted pool, moved to the rejected pool, or reserved in the candidate pool based on the decision.

For the satisfaction of LDP, each client can only participate in one round. The aforementioned procedure repeats until the candidate pool is cleaned. In the end, all the frequent patterns are outputted from the accepted pool.

The design of FedFPM has three major advantages: 1) the design is highly modular. Users can adapt FedFPM into different FPM tasks by solely changing the candidate generation scheme; 2) the design minimizes the privacy leakage. Client sampling, queries on partial information, and the randomized response technique all prevent the upload of raw data and decrease the total information leakage; 3) the design is interactive. Instructions from the server are based on aggregations from multiple clients, which intelligently guide knowledge discovery. The iterative query-response approach allows the system to achieve high data utility with minimal client usage.

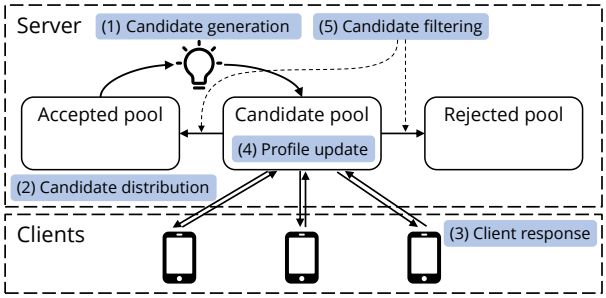


Fig. 1. An overview of FedFPM.

VI. Unified FA Framework Design for FPM

A. Candidate generation

The phase of candidate generation is with two variations. The first is in initialization, where the server needs to generate a few candidates without prior knowledge. The second is in the later rounds, where the server needs to generate new candidates based on currently accepted ones. The idea of candidate generation is to exploit the underlying data structure to make interactive query-response possible.

When the patterns are in simple structure, like FIM, since the pattern is exactly discrete items, we can either enumerate all possible discrete items or exploit context-based structural relationships to generate candidates. When the patterns are in complex structures, like FIsM and FSM, we can initialize the candidate pool with candidates in the simplest structure, and generate complex candidates based on the accepted simple ones. Since the candidate generation takes place at the server side, they can borrow the wisdom of existing candidate-based centralized FPM algorithms. As an example, we adopt Apriori-based algorithms to generate candidates in all pattern scenarios. We present our adaptations of FedFPM into the scenarios of FIM, FIsM, and FSM as follows.

Candidate generation for FIM We have two solutions to adapt FedFPM to the simple FIM problem. First, we can just perform initialization one time for FIM, which enumerates all possible items in the pool. Second, we can use the context of the items, *e.g.*, in the location-based services, geometric locations can be represented as items. We can then grid the area and start generating candidates with lower granularity and fine-tune them into higher granularity candidates based on the results in each iteration. To guarantee generalization, we use the first strategy in the evaluation part.

Candidate generation for FIsM The Apriori property shows that the subpattern of any frequent pattern must be frequent. As a result, it provides a tool to generate candidates of potential frequent patterns: a pattern is considered as a candidate only when all of its subpatterns have been proven to be frequent. FIsM is exactly the first application of the Apriori property [24]. The procedure of candidate generation is mathematically defined as follows:

$$\mathcal{F}(q) \geq f,\ \forall q \subseteq p \iff p \text{ is a candidate.} \tag{13}$$

In practice, for a pattern p with n items, instead of checking all subsets of p, whose number is 2^n, we only check the subsets with $n-1$ items, whose number is n. It has implicitly checked all subsets of p by recursively applying the Apriori property to the checked subsets. In addition, in the FIsM scenario, we take all itemsets with one item for initialization. This approach meets the requirements of the Apriori-based algorithms and minimizes the number of candidates in initialization.

Candidate generation for FSM The Apriori property can also be adapted into FSM, because the subsequences of a frequent sequential pattern must be also frequent. We abuse the notion n to be the length of any pattern p, the procedure of candidate generation for FSM can be defined as follows:

$$\big\{\mathcal{F}\big(Sub(p,1,n-1)\big) \geq f\big\} \ \wedge \ \big\{\mathcal{F}\big(Sub(p,2,n)\big) \geq f\big\} \iff p \text{ is a candidate.} \tag{14}$$

Therefore, we only check two subsequences because it has implicitly checked all subsequences. In the initialization phase, we place all sequences with one item into the candidate pool which meets the requirements of the Apriori property and minimizes the number of candidates.

B. Candidate distribution

In this phase, the server first samples M available clients, and distributes a random candidate in the candidate pool to each client. In practice, there may be less than M available clients at one moment. FedFPM is robust to these cases because it does not require the participating clients to conduct the task synchronously. Instead, FedFPM waits until M clients participate and their responses are received, before continuing to the later phases. As a result, M only controls the frequency of starting a new round and does not propose any requirement on the number of available clients.

C. Client response

After receiving the candidate, each participant first generates a yes/no response to whether the candidate pattern appears in the local data. We use a bit b to encode the response, where $b = 1$ for "yes" response, and $b = 0$ for "no" response. Randomized response, as the necessity of the realization of LDP, is then employed to randomize the yes/no response (*i.e.*, it flips the input bit with possibility η). We formally define the binary randomized response as follows:

Definition 2 (Binary randomized response)**.** *Given an input bit b, and a noise factor $\eta \in [0, 0.5)$, randomized response outputs b', a noisy version of b, following the below rules.*

$$\mathbb{P}(b' = 0) = \begin{cases} 1-\eta, & \text{if } b = 0, \\ \eta, & \text{if } b = 1. \end{cases} \tag{15}$$

Algorithm 1 shows how a client generates a response. After receiving the candidate from the server, the client first verifies whether the candidate is a pattern of its local data. Then, the client performs a binary randomized response to realize LDP, and uploads the noisy response to the server. In FedFPM, the clients are only required to check whether a candidate appears in the local data, and then randomize this single bit information. Therefore, the computation on the client side is significantly reduced compared to the existing methods. In Section VII-B, we also validate the claim by recording the client runtime of FedFPM and existing methods.

Algorithm 1 Client Response (RESPOND)

Input: Local data: d; noise factor: η; candidate: c.
Output: Response: r'

```
r ← 𝕀(c ⊆ d)                          ▷ True response
r' ← randomize r with (15)            ▷ Uploaded response
return r'
```

Randomized response helps the client scheme of FedFPM to satisfy LDP, which can be proved by the following theorem.

Theorem 1. *The client response scheme of FedFPM satisfies ϵ-LDP when*

$$\eta = \frac{1}{1+e^{\epsilon}}. \tag{16}$$

Proof. See Appendix A. □

In FedFPM, the interaction scheme is elaborately designed so that the client response only takes one bit, much simpler than the existing methods. As a result, FedFPM requires much smaller noise on the response to realize the same LDP level, which helps to gain high data utility.

D. Candidate profile update

Each candidate c in the candidate pool has a profile with two attributes, c_y and c_n. c_y records the times c is responded "yes" from the clients, which means that a client reports that c is a pattern of its local data. c_n records the times c is responded "no" from the clients, which means that a client reports that c is not a pattern of its local data. Once a client responds on c, the profile of c is updated by adding 1 to c_y or c_n.

Algorithm 2 Candidate Filtering (FILTER)

Input: Candidate: c.
Output: Operation on c.

```
if c satisfies (17) then
    return move c to 𝒫                 ▷ 𝒫 is the accepted pool
end if
if c satisfies (18) then
    return remove c from 𝒞             ▷ 𝒞 is the candidate pool
end if
if c_y + c_n ≥ κ then
    if c satisfies (19) then
        return move c to 𝒫
    else
        return remove c from 𝒞
    end if
end if
return reserve c in 𝒞
```

E. Candidate filtering

After the records of the candidates are updated, the server filters the candidates, whose procedure is demonstrated in Algorithm 2. Essentially, we make three decisions for each candidate in each round: 1) whether we can claim that c is a frequent pattern; 2) whether we can claim that c is not a frequent pattern; 3) the status of c needs further responses to verify. Due to the client sampling and randomized response, we cannot make the above claims with 100% confidence. Therefore, we introduce a parameter ξ as the allowed error rate of our claim, which controls the minimum confidence required to make a decision. Denote the frequency threshold of frequent patterns as f, and the noise factor as η, borrowing the power of the Hoeffding's inequality, we conclude the rationale of candidate filtering into the following theorems:

Theorem 2. *The frequency of any candidate c is higher than f with $1-\xi$ confidence when*

$$\frac{c_y}{c_y+c_n} \geq f+\eta-2f\eta+\sqrt{\frac{-\ln\xi}{2(c_y+c_n)}}. \tag{17}$$

Proof. See Appendix B. □

Theorem 3. *The frequency of any candidate c is lower than f with $1-\xi$ confidence when*

$$\frac{c_y}{c_y+c_n} \leq f+\eta-2f\eta-\sqrt{\frac{-\ln\xi}{2(c_y+c_n)}}. \tag{18}$$

Proof. See Appendix C. □

For the candidates satisfying Theorem 2, their observed frequency is high enough for us to claim that it is a frequent pattern with a given confidence. Therefore, they will be accepted as a frequent pattern, and vise versa for those satisfying Theorem 3. The parameter ξ exposes a knob to control the strictness of candidate generation, which allows users to freely balance the trade-off between data utility and the client usage to match different application scenarios: a higher ξ accept or reject a candidate with lower confidence, which decrease the data utility, but use a lower number of clients to verify each candidate. Efficacy of ξ is validated in Section VII-B.

The employment of the Hoeffding's inequality is a key design of FedFPM, which adapts the number of responses for each candidate in a rigorous way. For a candidate with a true frequency far from f, it only uses a small number of clients to verify the candidate. For a candidate with true frequency closed to f, it uses more clients to get sufficient confidence. Such an approach makes the client usage to be "just sufficient" by adjusting the total rounds of FedFPM. On the other hand, methods in the literature require the users to predefine the total client usage, which may lead to resource wastage or impetuous decisions. The effects of adaptive client usage in practice are further demonstrated in Section VII-B.

In addition to filtering the candidate with sufficient confidence, we set a threshold κ as the maximal response each client can receive. When the candidate receives too many responses from the clients, even if it cannot leave the candidate pool via Theorems 2 or 3, the server is forced to filter out the candidate. The observed frequency of the candidate is compared with the expectation of a pattern with frequency exactly f, *i.e.*, the candidate will be accepted if

$$\frac{c_y}{c_y+c_n} \geq f+\eta-2f\eta, \tag{19}$$

and rejected otherwise. This scheme aims at limiting client usage by preventing a candidate to be in the pool for a long time. After that, the remaining candidates are reserved in the candidate pool to gain more responses from the clients.

Overall, we summarize the whole structure of FedFPM in Algorithm 3. For the sake of clarity, hyperparameters including f, ϵ, ξ, and κ are not present in the pseudocode inputs.

Algorithm 3 FedFPM: overall structure

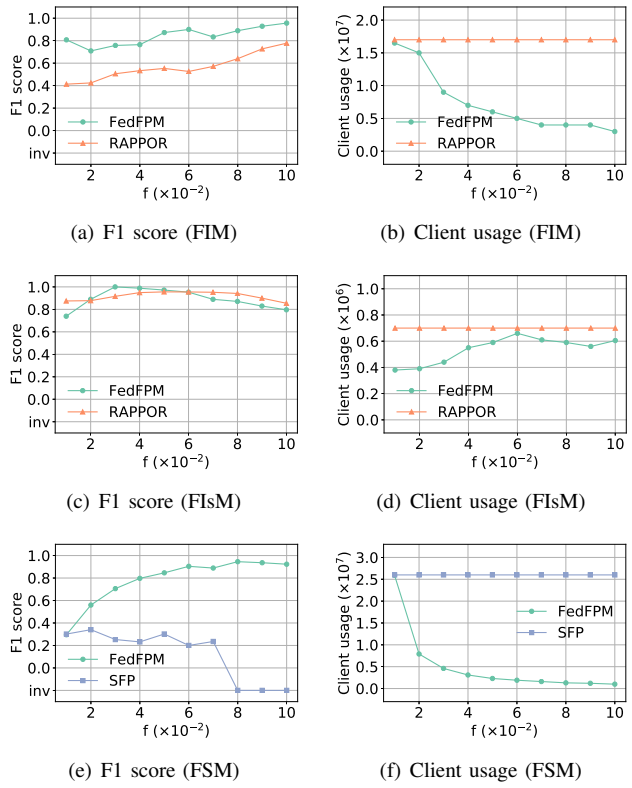
Input: Data in the clients: $\mathcal{D}$; set of clients: $\mathcal{N}$; number of participating clients in each round: M.
Output: Derived frequent patterns: $\mathcal{P}$

$\mathcal{C} \leftarrow$ initialize the candidate pool
$\mathcal{P} \leftarrow$ an empty set
$\eta \leftarrow$ derive noise factor from ϵ with (16)
while $\mathcal{C}$ is not empty **do**
  $A \leftarrow$ draw M clients from $\mathcal{N}$
  for $i \in A$ **do**
    $c \leftarrow$ draw a candidates form $\mathcal{C}$
    $r' \leftarrow$RESPOND(d_i, η, c)
    if $r' = 1$ **then**
      Add 1 to c_y
    else
      Add 1 to c_n
    end if
  end for
  for $c \in \mathcal{C}$ **do**
    FILTER(c)
  end for
  Generate new candidates $\mathcal{C}'$ with updated $\mathcal{P}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
end while
return $\mathcal{P}$

VII. Evaluation

A. Experimental setup

Datasets We use three real-world datasets, Kosarak [25], MovieLens [26], and MSNBC [27], to evaluate the performance of FedFPM in the scenarios of FIM, FIsM, and FSM, respectively. Kosarak contains 2.5×10^5 users' browsing records in the Hungarian news portal. Our goal is to identify the frequent websites (items) in the population. MovieLens contains 1.5×10^5 users' preferences on movie genres. Our goal is to identify the frequent sets of genres (frequent itemsets). MSNBC contains the sequential browsing history of 4.7×10^5 users in msnbc.com in categories of news. Our goal is to identify the common browsing trajectories



(a) F1 score (FIM) (b) Client usage (FIM)

(c) F1 score (FIsM) (d) Client usage (FIsM)

(e) F1 score (FSM) (f) Client usage (FSM)

Fig. 2. Performance of FedFPM and the benchmarks under different scenarios.

(sequences). Considering the significant running time of the specialized approaches, we only select the top 500 frequent websites in Kosarak and the top 10 genres in MovieLens. The performance of the benchmarks and our framework under different candidate scales will be discussed in the details later.

Benchmarks We compare FedFPM with two benchmarks: RAPPOR [8] and SFP [9]. RAPPOR is an industrial privacy-preserving FIM solution. It encodes the raw data into binary in fixed bits, and perform randomize response on the binary. We also modify RAPPOR to handle the FIsM scenario using one-hot encoding based on their suggestion [8]. SFP is a privacy-preserving FSM solution. It uses two count-mean-sketches to form a frequency oracle, which enables the decoding of the complex pattern structure. Note that the benchmarks are both specialized for particular forms of the pattern, while FedFPM is a unified framework for all kinds of patterns.

Implementation We implement FedFPM, the benchmarks, and the corresponding simulation environment on Python 3.8. All experiments are run on a desktop equipped with Intel(R) Core(TM) i7-10700 CPU @2.9GHz and 32GB RAM. The client schemes of FedFPM and the benchmarks all run in parallel with 14 worker processes, so that FedFPM and the benchmarks are fairly allocated with computation resources.

Settings We conduct FPM tasks with target frequencies f ranging from 0.01 to 0.10. The LDP parameter ϵ is set to 2.0, which is a moderate value (values of ϵ from the literature range between 0.01 to 10, according to the authors of [8]). The default allowed filtering error threshold ξ is set to 0.01, and the maximal response for each candidate κ is set to 10^5. We assume there are sufficient available clients, and each client randomly possesses one record in the dataset. In FedFPM, the numbers of participating clients in each round M are set to 10^6, 10^4, and 10^5, for the FIM, FIsM, and FSM scenarios, respectively. Since other benchmarks are non-interactive, we set the total participating clients of other benchmarks to be slightly higher than FedFPM (1.7×10^7, 7×10^5, and 2.6×10^7 participating clients for FIM, FIsM and FSM scenarios, respectively). It guarantees that FedFPM cannot unfairly gain an advantage by having more participating clients to improve data utility.

Metrics To quantify the quality of the derived patterns, we calculate precision, recall, and F1 score.[2] We use *F1 score* to measure data utility, where a higher F1 score indicates a better performance. In addition, we record the total number of participating clients, referred to as *client usage*, and the average *client runtime* for each client. The first one measures the system cost, the latter one measures the client overhead.

B. Results and analysis

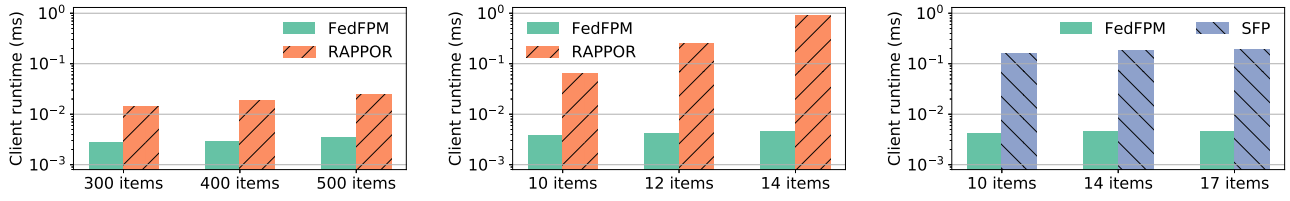
Data utility Fig. 2 presents the F1 score and client usage of FedFPM and other compared approaches in all subproblems[3]. In the three scenarios of FIM, FIsM, and FSM, FedFPM gets average F1 scores of 0.84, 0.89, and 0.78, respectively, while the benchmarks get 0.57, 0.92, and 0.19. Using less participating clients, FedFPM has relative performances of 147%, 97%, and 411% compared to the benchmarks regarding data utility. FedFPM outperforms the specialized benchmarks in the FIM and FSM scenarios. Although RAPPOR has a comparable performance with FedFPM in the FIsM scenarios, its applicability significantly degrades as the problem size increases, which is discussed in the following runtime part.

Client runtime analysis Fig. 3 presents the average runtime of the participating clients in FedFPM and other benchmarks. FedFPM only takes 2%∼14% of runtime for each participating client compared to its benchmarks, because FedFPM only requires the clients to check about one candidate, while the benchmarks require the clients to encode the whole raw data. FedFPM thus gains extra advantages in real-world applications, where clients have limited computing capacity. In addition, for each scenario, we generate two extra datasets with different numbers of items. As is shown in Fig. 3(b), while the runtime of FedFPM and SFP is almost independent of the settings, the runtime of RAPPOR grows exponentially with respect to the number of items in the FIsM scenario. The result is consistent with the theoretical expectation, because the computation of encoding is linear to the number of possible patterns, and exponential to the number of items.

Efficacy of filtering error threshold As is discussed in Section VI-E, the parameter of allowed filtering error threshold

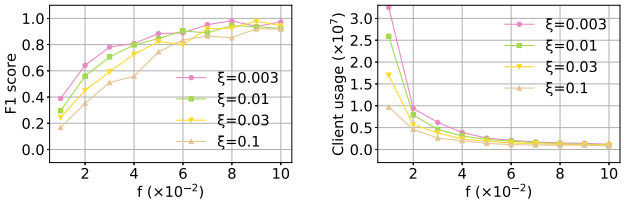
[2]F1 score equals $\frac{2pr}{p+r}$, where p is precision and r is recall.

[3]SFP does not output any pattern in some cases, leading to invalid F1 score (marked as "inv" in the figure.)



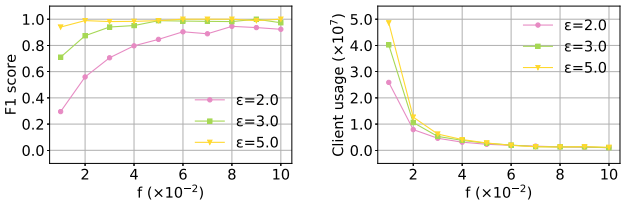
(a) FIM (b) FIsM (c) FSM

Fig. 3. Client runtimes in three scenarios. For each scenario, we generate three datasets by modifying the number of total items.



(a) F1 score (b) Client usage

Fig. 4. Performance of FedFPM with different ξ in FSM scanrio.

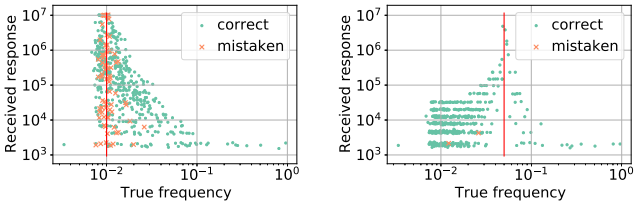


(a) F1 score (b) Client usage

Fig. 5. Performance of FedFPM with different ϵ in FSM scanrio.

ξ quantifies the trade-off between data utility and client usage. To explore its efficacy, we use different ξ values and conducts experiments on the FSM scenario, where FedFPM gets the worst performance and the change is the most observable. The results are summarized in Fig. 4. When ξ gets larger, the server will make a decision to accept/reject a candidate more "recklessly". It harms the data utility but decreases the client usage, and vise versa when ξ gets smaller. By adjusting ξ, the users can achieve a trade-off between resource usage and quality of output depending on their own system requirements.

Performance under different privacy level ϵ controls the level of privacy preservation, where a lower ϵ provides better privacy preservation, but requires a higher noise on the uploads. To investigate the functionality of FedFPM under different practical scenarios, we record its performance in different settings of ϵ. We choose the FSM scenario to conduct the experiments, because it is the most complex situation in our studied FPM problems. The results are presented in Fig. 5. When we enforce weaker privacy preservation, the performance of FedFPM increases significantly regarding data utility. Especially, when ϵ is 5.0, FedFPM performs almost perfectly. On the other hand, FedFPM utilizes more participating clients to validate each candidate when ϵ gets higher, which is resulted from the nature of the Hoeffding's inequality.



(a) $f = 0.01$ (b) $f = 0.05$

Fig. 6. Response received by each filtered candidate of FedFPM in FIM scanrio ($\kappa = 10^7, \xi = 10^{-5}$).

Details of candidate filtering FedFPM's candidate filtering verifies the frequency of each candidate based on their difficulties, avoids wasting client resources on easy judging tasks. To investigate whether it works as expected, we take the FIM scenario as an example, and record each candidate with whether it is correctly classified and the number of client responses it receives before leaving the candidate pool. We choose two values of f: 0.01 and 0.05, representing the case that FedFPM performs comparatively poorly and the case that FedFPM performs almost perfectly. As is shown in Fig. 6(a), the difficult candidates fill up the majority of mistakenly classified patterns. Overall in Fig. 6, difficult candidates whose true frequency are closed to f indeed tend to receive more client response, while the candidates that are easy to validate receive fewer responses before being filtered, which also empirically proves the correctness of our design.

Discussion Any FPM algorithm with looser privacy requirements and more client usages has better data utility. FedFPM is designed to require each client to participate only once to preserve each individual's privacy to the best. As a result, the efficacy of FedFPM relies on sufficient client usages, and its advantage over benchmarks shrinks when the participating clients are not sufficient. Two further approaches can be applied to FedFPM under limited participating clients. First, ξ can be increased as is shown in Fig. 4(b). Second, based on the composition theorem of LDP, we can allow each client to participate in multiple rounds (n), but enforce stricter privacy levels (ϵ/n) in each round. This approach enables each participating client to be counted for multiple client usages, and also decreases total client usage as is shown in Fig. 5(b).

VIII. Conclusion

We propose FedFPM as the first unified FA framework for FPM problems. FedFPM takes a query-response ap-

proach between clients and servers to collaboratively mine out frequent patterns. Queries are meticulously generated at the server side based on the aggregation of responses from sampled clients. Responses based on partial local data are randomized to realize data privacy. FedFPM achieves high data utility with provable loss and guarantees local differential privacy with minimum participating clients. Experiment results show that FedFPM achieves superior data utility compared to state-of-the-art specialized industrial solutions with minor computational overheads in all three FedFPM problems. The authors have provided public access to their code and data at https://github.com/inslab-ji/FFPA.

Appendix A
Proof of Theorem 1

Based on the definition of LDP in Definition 1, we have

$$\frac{\mathbb{P}(\mathcal{M}(d_i)=y)}{\mathbb{P}(\mathcal{M}(d_j)=y)} \leq e^{\epsilon}, \tag{20}$$

where d_i and d_j are any possible input. In the design of FedFPM, according to the randomized repsonse we used in (15), the possibility of any output is only with two options: η and $1-\eta$. Therefore, we have

$$\frac{\mathbb{P}(\mathcal{M}(d_i)=y)}{\mathbb{P}(\mathcal{M}(d_j)=y)} \leq \frac{\max\{\mathbb{P}(\mathcal{M}(d)=y)\}}{\min\{\mathbb{P}(\mathcal{M}(d)=y)\}} = \frac{1-\eta}{\eta}. \tag{21}$$

Combining with (20), the satisfaction of LDP needs

$$\frac{1-\eta}{\eta} \leq e^{\epsilon}, \tag{22}$$

which yields Theorem 1 via simple arithmetic transformation.

Appendix B
Proof of Theorem 2

We use a binary variable x to represent the response of a client, where $x=1$ when the client responds "yes", and $x=0$ when the client responds "no". For any candidate c, assume its true frequency is f_0, and consider the binary response scheme we used in (15), we can derive the expectation of x as

$$\begin{aligned}\mathbb{E}(x) &= f_0(1-\eta) + (1-f_0)\eta \\ &= f_0 + \eta - 2\eta f_0.\end{aligned} \tag{23}$$

Recall that $\eta < 0.5$, $\mathbb{E}(x)$ is a strictly increasing function with respect to f_0. Therefore, when $f_0 \geq f$, the corresponding $\mathbb{E}(x)$ is also larger, *i.e.*,

$$f_0 \geq f \iff \mathbb{E}(x) \geq f + \eta - 2\eta f. \tag{24}$$

For simplicity, we denote the bound of $\mathbb{E}(x)$ in (24) as $\hat{x}$:

$$\hat{x} = f + \eta - 2\eta f. \tag{25}$$

Since $f_0 \geq f$ suffices the requirement of a frequent pattern, this requirement is equivalent to $\mathbb{E}(x) \geq \hat{x}$ according to (24),(25).

On the other hand, recall that c_y and c_n are the numbers of yes/no responses candidate c received. We can write the total number of responses c received as

$$m = c_y + c_n, \tag{26}$$

and the average value of x for c as

$$\overline{x} = \frac{c_y}{c_y + c_n} \tag{27}$$

The Hoeffding's inequality provides probabilistic bound between average value of random variables and its expectation [28]. By applying it on the values of x, we have

$$\mathbb{P}\big(\overline{x} - \mathbb{E}(x) \geq \delta\big) \leq \exp(-2\delta^2 m). \tag{28}$$

Since δ can be arbitrary value, we use

$$\delta = \sqrt{\frac{-\ln\xi}{2m}} = \sqrt{\frac{-\ln\xi}{2(c_y+c_n)}}, \tag{29}$$

and we can rewrite (28) as

$$\mathbb{P}\left(\mathbb{E}(x) \leq \overline{x} - \sqrt{\frac{-\ln\xi}{2(c_y+c_n)}}\right) \leq \xi. \tag{30}$$

We want to prove that candidate c is a frequent pattern with confidence $1-\xi$, *i.e.*,

$$\mathbb{P}\big(\mathbb{E}(x) \geq \hat{x}\big) \geq 1-\xi, \tag{31}$$

or

$$\mathbb{P}\big(\mathbb{E}(x) \leq \hat{x}\big) \leq \xi. \tag{32}$$

Consider (30) and (32). Since (30) has been proven by the Hoeffding's inequality, we find that (32) must be satisfied when it provides a tighter bound than (30), *i.e.*,

$$\hat{x} \leq \overline{x} - \sqrt{\frac{-\ln\xi}{2(c_y+c_n)}} \implies \mathbb{P}\big(\mathbb{E}(x) \leq \hat{x}\big) \leq \xi, \tag{33}$$

which directly yields Theorem 2.

Appendix C
Proof of Theorem 3

Proof of Theorem 3 is similar to that for Theorem 2 by reversing the sign, so we reuse the definitions of $x, f_0, \hat{x}, m$, and $\overline{x}$, and some inference procedure. We starts with an alternative expression of (24):

$$f_0 \leq f \iff \mathbb{E}(x) \leq f + \eta - 2\eta f, \tag{34}$$

and an alternative version of the Hoeffding's inequality:

$$\mathbb{P}\big(\overline{x} - \mathbb{E}(x) \leq \delta\big) \leq \exp(-2\delta^2 m), \tag{35}$$

which can be rewritten as

$$\mathbb{P}\left(\mathbb{E}(x) \geq \overline{x} + \sqrt{\frac{-\ln\xi}{2(c_y+c_n)}}\right) \leq \xi. \tag{36}$$

We want to prove that candidate c is not a frequent pattern with confidence $1-\xi$, *i.e.*,

$$\mathbb{P}\big(\mathbb{E}(x) \geq \hat{x}\big) \leq \xi. \tag{37}$$

By considering (36) and (37), we find that (37) must be satisfied when it provides a tighter bound than (36), *i.e.*,

$$\hat{x} \geq \overline{x} + \sqrt{\frac{-\ln\xi}{2(c_y+c_n)}} \implies \mathbb{P}\big(\mathbb{E}(x) \geq \hat{x}\big) \leq \xi, \tag{38}$$

which directly yields Theorem 3.

REFERENCES

[1] Cisco Annual Internet Report (2018–2023). Cisco. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf

[2] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: current status and future directions," *Data Mining Knowl. Discovery*, vol. 15, no. 1, pp. 55–86, Jan. 2007.

[3] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Effective personalization based on association rule discovery from web usage data," in *Proc. Int. Workshop Web Inform. Data Manage.*, Atlanta, GA, Nov. 2001, pp. 9–15.

[4] L. Tang, Z. Duan, Y. Zhu, J. Ma, and Z. Liu, "Recommendation for ridesharing groups through destination prediction on trajectory data," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 2, pp. 1320–1333, Feb. 2021.

[5] 2018 reform of EU data protection rules. European Commission. [Online]. Available: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf

[6] California consumer privacy act. California Government. [Online]. Available: https://www.oag.ca.gov/privacy/ccpa

[7] A. Evfimievski, J. Gehrke, and R. Srikant, "Limiting privacy breaches in privacy preserving data mining," in *Proc. ACM-SIGACT-SIGART Symp. Princ. Database Syst.*, San Diego, CA, Jun. 2003, pp. 211–222.

[8] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proc. ACM Conf. Comput. Commun. Secur.*, Scottsdale, AZ, Nov. 2014, pp. 1054–1067.

[9] Learning with privacy at scale. Apple. [Online]. Available: https://machinelearning.apple.com/research/learning-with-privacy-at-scale

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat.*, Fort Lauderdale, FL, Apr. 2017, pp. 1273–1282.

[11] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., Standford, CA, Mar. 2019.

[12] P. Kairouz, B. McMahan, and V. Smith. Federated learning and analytics: Industry meets academia. NeurIPS 2020 tutorial. [Online]. Available: https://sites.google.com/view/fl-tutorial/home

[13] Federated analytics: Collaborative data science without data collection. Google AI. [Online]. Available: https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html

[14] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "Fedgnn: Federated graph neural network for privacy-preserving recommendation," *arXiv preprint arXiv:2102.04925*, 2021.

[15] W. Zhu, P. Kairouz, B. McMahan, H. Sun, and W. Li, "Federated heavy hitters discovery with differential privacy," in *Proc. Int. Conf. Artif. Intell. Statist.*, Palermo, Italy, Jun. 2020, pp. 3837–3847.

[16] Z. Wang, Y. Zhu, D. Wang, and Z. Han, "Fedacs: Federated skewness analytics in heterogeneous decentralized data environments," in *Proc. IEEE/ACM Int. Symp. Qual. Service*, virtual event, Jun. 2021, pp. 1–10.

[17] W. Gan, J. C.-W. Lin, H.-C. Chao, and J. Zhan, "Data mining in distributed environment: a survey," *Wiley Interdisciplinary Rev. Data Mining Knowl. Discovery*, vol. 7, no. 6, p. e1216, Jul. 2017.

[18] R. Bassily, K. Nissim, U. Stemmer, and A. Thakurta, "Practical locally private heavy hitters." *J. Mach. Learn. Res.*, vol. 21, pp. 16–1, Dec. 2020.

[19] J. Jia and N. Z. Gong, "Calibrate: Frequency estimation and heavy hitter identification with local differential privacy via incorporating prior knowledge," in *Proc. IEEE Conf. Comput. Commun.*, Paris, France, Apr. 2019, pp. 2008–2016.

[20] B. Ding, J. Kulkarni, and S. Yekhanin, "Collecting telemetry data privately," in *Proc. Adv. Neural Inf. Process. Syst.*, Long Beach, CA, Dec. 2017.

[21] T. Wang, N. Li, and S. Jha, "Locally differentially private frequent itemset mining," in *Proc. IEEE Symp. Secur. Privacy*, San Francisco, CA, May 2018, pp. 127–143.

[22] S. Wang, L. Huang, Y. Nie, P. Wang, H. Xu, and W. Yang, "Privset: Set-valued data analyses with locale differential privacy," in *Proc. IEEE Conf. Comput. Commun.*, Honolulu, HI, Apr. 2018, pp. 1088–1096.

[23] S. Wang, Y. Nie, P. Wang, H. Xu, W. Yang, and L. Huang, "Local private ordinal data distribution estimation," in *Proc. IEEE Conf. Comput. Commun.*, Atlanta, GA, May 2017, pp. 1–9.

[24] R. Agarwal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. VLDB Conf.*, vol. 487, Santiago de Chile, Chile, Sep. 1994, p. 499.

[25] A. R. Benson, R. Kumar, and A. Tomkins, "A discrete choice model for subset selection," in *Proc. ACM Int. Conf. Web Search Data Mining*, Marina Del Rey, CA, Feb. 2018.

[26] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Jan. 2015.

[27] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White, "Visualization of navigation patterns on a web site using model-based clustering," in *Proc. ACM Int. Conf. Knowl. Discovery Data Mining*, Boston, MA, Aug. 2000, pp. 280–284.

[28] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.