Federated and Balanced Clustering for High-dimensional Data

Yushuai Ji[1†], Shengkun Zhu[1†], Shixun Huang[2], Zepeng Liu[1], Sheng Wang[1*], Zhiyong Peng[1,3]
[1]School of Computer Science, Wuhan University, [2]School of Computing and Information Technology, The University of Wollongong, [3]Big Data Institute, Wuhan University
[yushuai,whuzsk66,liuzp_063,swangcs,peng]@whu.edu.cn,shixunh@uow.edu.au

ABSTRACT

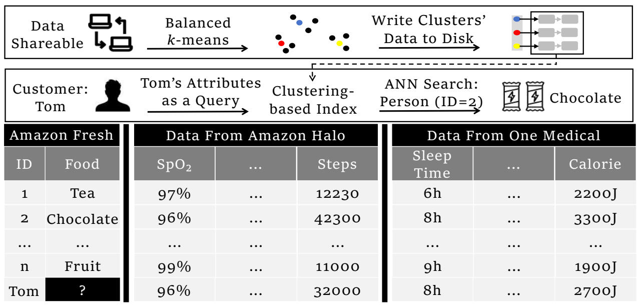
Balanced k-means ensures representative centroids by forming equal-sized clusters, but struggles with slow clustering of massive distributed attributes and data-sharing restrictions. A common approach is adapting it to a vertical federated learning (VFL) framework, preventing raw data exposure by only intermediate result exchange and accelerating clustering via parallelism, yet it remains unexplored. In this paper, we propose a time-efficient, federated, and balanced k-means algorithm, called `Teb-means`, to bridge the gap. We first formulate the balanced k-means problem as a trace maximization problem (TMP) and propose an efficient coordinate-wise optimization (CO) scheme to solve it. We then integrate TMP and CO into the VFL framework by demonstrating that TMP can be decomposed into multiple subproblems based on each party's data, which can be solved using CO while exchanging only intermediate results. Notably, we build a trade-off between utility and communication efficiency by designing a greedy block-based strategy for CO (GBCO). Our theoretical analysis shows that `Teb-means` achieves linear time complexity on each client, and our communication round is constant in the mild condition. Experiments show that `Teb-means` is on average 12.18× faster than other balanced clustering algorithms that can be federated, while achieving better balance without disrupting the cluster structure.

PVLDB Reference Format:
Yushuai Ji, Shengkun Zhu, Shixun Huang, Zepeng Liu, Sheng Wang, Zhiyong Peng. Federated and Balanced Clustering for High-dimensional Data. PVLDB, 18(11): 4032 - 4044, 2025.
doi:10.14778/3749646.3749673

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/whu-totemdb/Teb-means.

1 INTRODUCTION

The k-means algorithm [34] can partition a dataset through clustering to support efficient analytics and learning [12, 20, 21, 29, 36, 44]. As one of the most well-known variants, balanced k-means [14, 31, 32] aims to partition data points into k clusters such that the points within a cluster are as close as possible, while also ensuring equal cluster sizes. This algorithm improves the interpretability or efficiency of analysis methods [10, 13, 23, 41, 48, 50], such as feature quality in embedding [48], boosting retrieval precision in vector search [13], and balancing workloads in wireless sensor networks [49], by preventing the dominance of any single cluster.


Data Shareable
Balanced k-means
Write Clusters' Data to Disk
Customer: Tom
Tom's Attributes as a Query
Clustering-based Index
ANN Search: Person (ID=2)
Chocolate

Amazon Fresh		Data From Amazon Halo			Data From One Medical		
ID	Food	SpO_2	...	Steps	Sleep Time	...	Calorie
1	Tea	97%	...	12230	6h	...	2200J
2	Chocolate	96%	...	42300	8h	...	3300J
...	...	...	...	...	...	...	...
n	Fruit	99%	...	11000	9h	...	1900J
Tom	?	96%	...	32000	8h	...	2700J

Figure 1: Balanced k-means is used in clustering-based index construction for targeted advertising but loses effectiveness under data-sharing restrictions.

Demand for Federated and Balanced k-means. However, the rapid growth of massive attributes slow balanced clustering due to the failure of pruning techniques [43], caused by the curse of dimensionality [17, 25]. Meanwhile, stricter legal mandates [24, 28, 30, 51] have outlawed data sharing, rendering balanced k-means inapplicable and stalling analysis [13, 42, 48, 49]. For example, as shown in Figure 1, balanced k-means serves as a cornerstone for clustering-based indices, such as those used in SPANN [13] and SPfresh [45], facilitating approximate nearest neighbor (ANN) search. The core idea is that each centroid's corresponding data is stored on disk, and when a query arrives, it first identifies the closest centroids and then retrieves the nearest point from the corresponding disk, where balance ensures that no single disk is overloaded, preventing increased latency [13]. Considering the following scenario:

Amazon Fresh, a grocery subsidiary of Amazon, purchases data from sources such as Amazon Halo and One Medical, which collect health and fitness attributes. It uses ANN to find users similar to new customers and recommend their most frequently purchased foods. However, the large-scale integrated user data often contains dozens or even hundreds of attributes, significantly slowing balanced k-means and causing the training server to frequently run out of memory or time out due to high computational cost. Meanwhile, privacy laws restrict data sharing, making recommendations impossible. For example, with data sharing, the most similar user to Tom is ID = 2, but without such sharing, no recommendations can be made, resulting in missed opportunities for relevant products like chocolate for Tom.

Challenges in Adapting Existing Work to VFL. In this context, efficient balance clustering with high dimensions, without exposing raw data, has become a necessary step forward. One of the

* Sheng Wang is the corresponding author.
† Yushuai Ji and Shengkun Zhu are co-first authors.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749673

Table 1: Performance of two balanced clustering algorithms. The deviation between the data points and their centers is referred to as clustering loss, while the balance loss reflects the deviation between each cluster size and the ideal size (total number of data points divided by k).

Dataset	Clustering Loss			Balance Loss		
	Lloyd	FCFC	BCLS	Lloyd	FCFC	BCLS
NYC	2.20E+06	2.43E+06	6.57E+06	1.30E+10	1.23E+10	4.09E+07
Amazon	8.18E+07	7.00E+07	1.75E+08	7.39E+10	7.01E+10	4.69E+07

most popular approaches is adapting the algorithm to VFL frameworks [22, 26, 27, 47, 52], which effectively handles non-shareable data containing unique attributes while sharing the same user set. In the VFL, accelerating model training can be achieved by parallelizing the process across multiple clients while exchanging only intermediate results. However, no federated and balanced k-means algorithm exists. While several algorithms [11, 31, 32, 35] can be adapted to VFL, two key challenges arise in our scenario: 1) disruption of cluster structure in high dimensions and 2) communication bottlenecks in VFL, which we will elaborate as below:

Adopting soft-balanced k-means in VFL is a suitable choice, as it achieves balance without strict enforcement and has lower time complexity compared to the hard-balanced approach [11, 14, 35], which exhibits superlinear time complexity. For instance, Liu et al. [31] proposed BCLS, which applies square regression with a balance constraint to achieve approximate balance. In a separate study, Liu et al. [32] introduced FCFC, which modifies the objective function in Lloyd's algorithm [34] to enforce balance. However, it is sensitive to the trade-off between clustering loss and balance loss in high dimensions. As shown in Table 1, we compare the clustering performance of BCLS and FCFC on two datasets, NYC (100d) and Amazon (2400d), where d represents the number of dimensions. BCLS achieves better balance than Lloyd's algorithm but significantly compromises clustering loss. In contrast, FCFC maintains a clustering loss comparable to Lloyd's algorithm but fails to preserve cluster balance.

Existing VFL frameworks for clustering algorithms face communication bottlenecks [16, 24, 30, 51] due to the strong influence of dataset size on the communication round. To mitigate this, Ding et al. [16] introduced a constant approximation scheme for k-means clustering, resulting in a linear dependence between communication rounds and dataset size. Furthermore, Huang et al. [24] proposed a method that reduces communication rounds by using a coreset, achieving a sublinear dependence. However, even such improvements remain inadequate in the era of big data [51].

Our Methodology and Contributions. In this paper, we propose Teb-means to fill the gap while addressing the limitations of existing methods in the VFL framework. We first define the federated and balanced k-means problem, incorporating both clustering loss and balance loss into the loss function while specifying the loss computations assigned to the clients and the central server. To optimize the loss function efficiently, we reformulate it as a TMP and propose a CO scheme for its solution. We integrate TMP and CO into the VFL framework by demonstrating that TMP can be decomposed into multiple subproblems, which are solved using CO while transmitting only intermediate results to the central server. Moreover, we introduce GBCO, which batch-optimizes the data points, achieving a trade-off between utility and communication efficiency. Overall, our main contributions are:

- To the best of our knowledge, Teb-means is the first federated and balanced k-means algorithm that can efficiently divide high-dimensional data into approximately equal clusters, with attributes distributed across different parties (see Section 4).
- Our theoretical analysis shows that Teb-means has linear time complexity on each client and a constant communication round in the mild condition (see Section 5).
- Experiments on 8 real-world datasets show that Teb-means is on average 12.18× faster than other balanced clustering algorithms adaptable to VFL, achieves better cluster balance, and maintains clustering loss comparable to Lloyd's algorithm (see Section 6).

2 RELATED WORK

In this section, we review existing work on balanced clustering algorithms, including hard-balanced and soft-balanced clustering. Specifically, we discuss their methodologies and the bottlenecks encountered by both types of balancing. We then review federated clustering algorithms and discuss the limitations they face.

2.1 Balanced k-means Methodology

Hard-balanced Clustering. Hard-balanced k-means algorithms [11, 35] were developed to enforce strict cluster size constraints. Bradley et al. [11] introduced k constraints into Lloyd's algorithm to enforce balance, while Malinen et al. [35] formulated the assignment process as a pairing problem and solved it using the Hungarian algorithm to ensure strict balance. However, the time complexities of their approaches are $O(n^{3.5})$ and $O(n^3)$, respectively, where n represents the data size. A bunch of acceleration techniques improve the speed of this clustering by using the triangle inequality [18] and bounding strategies [19]. However, these methods become ineffective in high-dimensional spaces (e.g., above 10d) due to the curse of dimensionality [25].

Soft-balanced Clustering. Hard-balanced k-means is too restrictive for real-world applications, which has led to the adoption of soft-balanced k-means, which aims for balance without strict enforcement. Specifically, these algorithms add different types of penalty constraints to Lloyd's algorithm [34] to bring the clusters closer to balance. For example, Althoff et al. [10] modified the k-means assignment by introducing a penalty term based on cluster size. Liu et al. [31] proposed using square regression with a balance constraint for clustering. Liu et al. [32] proposed adding cluster size variances as a penalty for balancing the clusters. However, our experiments (see Section 6) show that both approaches disrupt the structure of the clusters formed by Lloyd's algorithm or have limited effectiveness in maintaining balance.

2.2 VFL for k-means

Most existing studies [16, 24, 30, 51] on federated k-means have been conducted within the framework of VFL. Ding et al. [16] proposed an approximation algorithm for distributed dimensional scenarios in k-means. This algorithm operates by the central server, which is derived from the product of the clients, and the communication rounds depend on the number of clients and the dataset size.

Table 2: Summary of notations.

Symbol	Description
$d \in \mathbb{Z}^+$	The dataset dimension
$n \in \mathbb{Z}^+$	The size of dataset
$k \in \mathbb{Z}^+$	The number of clusters
$M \in \mathbb{Z}^+$	The number of data owner
$c \in \mathbb{Z}^+$	The number of dataset partitions
$n_c \in \mathbb{Z}^+$	The partitioned data scale
$\mathbf{X} \in \mathbb{R}^{n\times d}$	The dataset
$\mathbf{x}_i \in \mathbb{R}^d$	The i-th row of $\mathbf{X}$
$\mathbf{o} \in \mathbb{R}^d, \mathbf{O} \in \mathbb{R}^{k\times d}$	The cluster centroid, a set of centroids
$\mathbf{G} \in \mathbb{R}^{k\times n}$	The indicator matrix with n one-hot vectors
$\mathbf{g}_j \in \mathbb{R}^k$	The j-th column of $\mathbf{G}$
$g_{ij} \in \mathbb{R}^k$	The (i,j)-entry of $\mathbf{G}$

Huang et al. [24] researched communication-efficient approaches for VFL, focusing on scalability. They developed a comprehensive paradigm based on the coreset. This coreset construction method with sublinear communication under mild conditions. However, both approaches still depend on the dataset size, and an increase in data volume raises the number of communication rounds.

Li et al. [30] proposed a method to ensure differential privacy in federated k-means with multiple clients and an untrusted server, where each party generates a differentially private data synopsis. This algorithm achieves a communication round that is independent of n and has the lowest observed communication round among existing approaches. However, this method does not fully exploit the potential for balancing. Moreover, Zhu et al. [51] proposed F^{3}KM, which we extend to support balanced k-means in VFL. After our modification, its communication rounds can be constant. However, F^{3}KM exhibits poor performance in balancing (see Section 6).

3 PRELIMINARIES

In this section, several key concepts are introduced for understanding our methodology. Specifically, we first present the notations used in this paper and the formulation of the k-means problem. Then, we provide an overview of VFL, followed by the formal definition of the federated and balanced k-means problem.

3.1 Notations

We use different text formatting styles to represent mathematical concepts: plain letters for scalars, bold lowercase letters for data points, and bold uppercase letters for a set containing data points. For example, k stands for a scalar, $\mathbf{x}$ represents a data point, and $\mathbf{X}$ represents a dataset. Without loss of generality, we denote the d-dimensional Euclidean space as $\mathbb{R}^d$, and the set of positive integers as $\mathbb{Z}^+$. Moreover, we use $[k]$ to represent the set $\{1, 2, \ldots, k\}$.

We consider a matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, where the entry in the (i,j)-th position is denoted as $\mathbf{x}_{ij}$, and the i-th column of $\mathbf{X}$ is represented by $\mathbf{x}_i$. The transpose of $\mathbf{x}_i$ is written as $\mathbf{x}_i^\top$, and the transpose of the matrix $\mathbf{X}$ is denoted by $\mathbf{X}^\top$. The Frobenius norm of $\mathbf{X}$, expressed as $\|\mathbf{X}\|_F$, is the square root of the sum of the squared elements of $\mathbf{X}$:

$$\|\mathbf{X}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{d} \mathbf{x}_{ij}^2} = \sqrt{\mathrm{Tr}(\mathbf{X}^\top \mathbf{X})}, \tag{1}$$

where $\mathrm{Tr}(\cdot)$ refers to the trace of a matrix, which is the sum of its diagonal entries. Additionally, we define $\mathbf{G} \in \mathbb{R}^{k\times n}$ as an indicator matrix, where each column contains exactly one "1" and the rest are "0". A full list of the notation is provided in Table 2.

3.2 k-means Problem

We first introduce the classical k-means loss function, followed by a reformulated version that depends solely on the indicator matrix $\mathbf{G}$. Several studies [32, 37, 38] have demonstrated that this reformulation can improve clustering performance. Notably, the reformulated one serves as the foundation for understanding `Teb-means`.

Classical k-means Formulation. Let $\mathbf{X} \in \mathbb{R}^{n\times d}$ represent a matrix of n data points, where the i-th row, $\mathbf{x}_i \in \mathbb{R}^d$, denotes a single data point. The k-means problem aims to partition the dataset into k disjoint subsets, denoted as $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k\}$, that minimize the sum of squared errors, expressed as:

$$\min_{\mathcal{P}} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathbf{P}_j} \|\mathbf{x}_i - \mathbf{o}_j\|_2^2, \tag{2}$$

where the centroid $\mathbf{o}_j = \frac{1}{|\mathbf{P}_j|}\sum_{\mathbf{x}_i\in\mathbf{P}_j} \mathbf{x}_i$ is the mean of data points in the cluster $\mathbf{P}_j$.

Reformulation. The k-means problem can be formally defined using the cluster indicator matrix. Recall that the Frobenius norm of a matrix ($\|\cdot\|_F$) is the square root of the sum of the squared entries. Hence, Problem (2) is equivalent to the following formulation:

$$\min_{\mathbf{G},\mathbf{O}} \sum_{i=1}^{n}\sum_{j=1}^{k} \|\mathbf{x}_i - \mathbf{o}_j\|_2^2 g_{ij} = \min_{\mathbf{G},\mathbf{O}} \|\mathbf{X} - \mathbf{G}^\top \mathbf{O}\|_F^2, \tag{3}$$

where $\mathbf{o}_j$ is the j-th row of $\mathbf{O}$, and $\mathbf{O} \in \mathbb{R}^{k\times d}$ represents the centers of the clusters. Additionally, $\mathbf{G} \in \mathbb{R}^{k\times n}$ is an indicator matrix consisting of n one-hot data points, where $g_{ij} \in \{0,1\}$ is the (i,j)-th entry of $\mathbf{G}$. For example, if $\mathbf{x}_i$ belongs to cluster $\mathbf{P}_j$, then $g_{ij} = 1$, while all other entries in the j-th column of $\mathbf{G}$ are set to 0.

As shown in Problem (3), both the indicator matrix $\mathbf{G}$ and the center matrix $\mathbf{O}$ are involved. Following [32, 38], we reformulate the center matrix in terms of the indicator matrix, allowing the optimization to focus on a single variable—the indicator matrix.

When $\mathbf{O}$ is fixed, the solution to $\mathbf{G}$ is

$$g_{ij} = \begin{cases} 1, & \text{if } j = \arg\min_l \|\mathbf{x}_i - \mathbf{o}_l\|_2^2, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathbf{o}_l$ is the l-th row of $\mathbf{O}$.

When $\mathbf{G}$ is fixed, Problem (3) can be rewritten as:

$$\min_{\mathbf{G},\mathbf{O}} = \mathrm{Tr}((\mathbf{X} - \mathbf{G}^\top\mathbf{O})(\mathbf{X} - \mathbf{G}^\top\mathbf{O})^\top), \tag{5}$$

where $\mathrm{Tr}(\cdot)$ denotes the trace of a matrix (see Section 3.1). This is a minimization problem with respect to $\mathbf{O}$. A common approach to finding the minimum is to compute the derivative and set it to zero. Hence, we have:

$$\mathbf{O} = (\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}\mathbf{X}. \tag{6}$$

By replacing $\mathbf{O}$ with $(\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}\mathbf{X}$, the optimization problem can be reformulated as follows:

$$\min_{\mathbf{G}} \Theta(\mathbf{G}) = \min_{\mathbf{G}} -\mathrm{Tr}\left((\mathbf{G}\mathbf{G}^\top)^{-1}\mathbf{G}\mathbf{X}\mathbf{X}^\top\mathbf{G}^\top\right), \tag{7}$$

where $G^\top G$ is a diagonal matrix with the (i,i) element equal to $g_i^\top g_i$; g_i denotes the i-th row of G. By removing the negative sign, Problem (7) is reformulated as the maximization problem:

$$\max_{G} \Theta(G) = \max_{G} \sum_{i=1}^{k} \frac{g_i XX^\top g_i^\top}{g_i g_i^\top}. \tag{8}$$

3.3 Vertical Federated Learning (VFL)

Since k-means is an unsupervised learning method, we explain the concept of VFL in the unsupervised setting to help readers understand the federated nature of our problem. Specifically, for training an unsupervised machine learning (ML) model on a single server, the loss of the ML model on a dataset X can be defined as:

$$\min_{\Theta} f(\Theta; X), \tag{9}$$

where $f(\cdot)$ is the loss function, parameterized by Θ. While training a joint model across multiple clients, ensuring that data processing remains local makes the VFL framework the optimal choice [33].

Following [33, 46], a VFL assumes that data is partitioned by feature space and aims to collaboratively train a joint model. Let X be distributed across $M \in \mathbb{Z}^+$ parties, such that $X = [X_1, X_2, \dots, X_M] \in \mathbb{R}^{n\times d}$, where $X_m \in \mathbb{R}^{n\times d_m}$ and $\sum_{m=1}^{M} d_m = d$. Each X_m represents the data held by the m-th data owner. Without loss of generality, the global model can be decomposed into M local models, each operating exclusively on local data. Specifically, $\mathcal{G}_m$ denotes the loss function of the m-th local model, parameterized by θ_m. The global model relies solely on intermediate results from the local models to refine itself, with its loss function represented as $\mathcal{F}$ and parameterized by β. Hence, we rewrite $f(\Theta; X)$ as:

$$f(\Theta; X) = \mathcal{F}\left(\beta; \{\mathcal{G}_m(\theta_m; X_m)\}_{m=1}^{M}\right). \tag{10}$$

3.4 Problem Definition

We introduce the federated setting in our problem, and then present the formal problem definition. Notably, instead of directly adding a penalty term to the loss function that acts only as a constraint [31, 32] during clustering, we introduce a weighting coefficient to quantify the relative importance of the clustering and balance losses, e.g., $\eta \times$ clustering loss + $(1-\eta) \times$ balance loss.

In the client, we define G and O as distributed across $M \in \mathbb{Z}^+$ parties, such that $G = [G_1, G_2, \dots, G_M] \in \mathbb{R}^{k\times d}$ and $O = [O_1, O_2, \dots, O_M] \in \mathbb{R}^{k\times d_m}$, where $G_m \in \mathbb{R}^{d_m\times n}$, $O_m \in \mathbb{R}^{d_m\times n}$, and $\sum_{m=1}^{M} d_m = d$. Hence, we define the loss of the m-th client as:

$$\mathcal{G}_m(G_m, O_m; X_m) = \|X_m - G_m^\top O_m\|_F^2, \tag{11}$$

where X_m is only accessible to the m-th local model.

In the central server, the global model is refined by aggregating intermediate results in two ways: 1) acquiring the value of $\mathcal{G}_m(G_m, O_m; X_m)$ from the local models, as shown in Equation (11), and 2) calculating the balance loss as expressed by:

$$\mathcal{L}(G, n, k) = \sum_{j=1}^{k} \left(\sum_{i=1}^{n} g_{ij} - \frac{n}{k} \right)^2, \tag{12}$$

where $\mathcal{L}$ measures the deviation of each cluster size from the ideal balanced size, defined as the total number of data points divided by k; $g_{ij} \in \{0,1\}$ denotes the (i,j)-entry of G; and G is globally shared, containing only the cluster to which each point is assigned, without exposing any point-specific details.

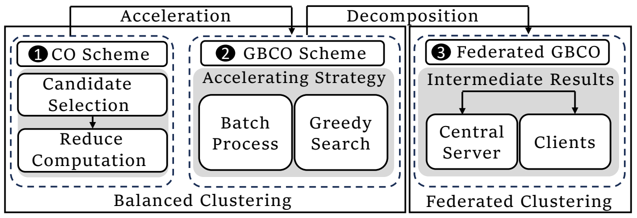


Figure 2: An overview of Teb-means.

Definition 1. *(**Federated and Balanced k-means**) Federated balanced k-means enables data owners to collaboratively partition a dataset into k disjoint subsets, minimizing the total squared distance between distributed data and their cluster centers while balancing each cluster, as formulated below:*

$$\min_{G,O} \{\mathcal{F}(G, O, n, k; X) = \eta \sum_{m=1}^{M} \mathcal{G}(G_m, O_m; X_m) + (1-\eta)\mathcal{L}(G, n, k)\}, \tag{13}$$

where $\mathcal{F}$ is the loss function of the global model, and η serves as a weighting coefficient, indicating the importance of the deviation between data points and their corresponding cluster centers, referred to as the clustering loss, and the deviation between the size of each cluster and the ideal size, known as the balance loss.

4 TEB-MEANS

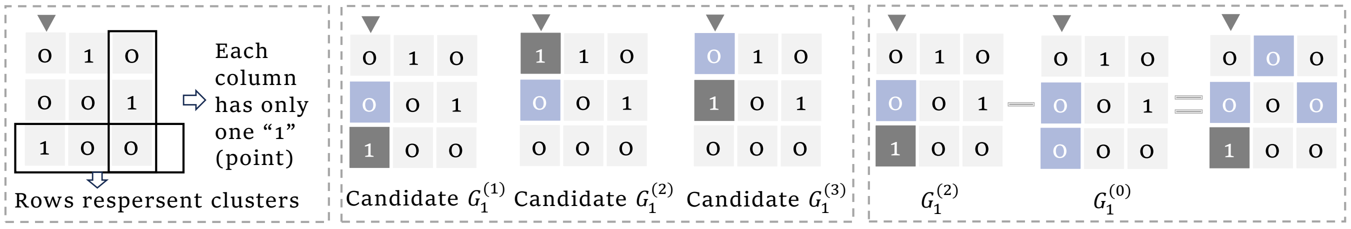
When addressing our problem (see Definition 1), we convert it into the following three subproblems:

(1) How can the loss function (13) be efficiently minimized?
(2) How do the proposed methods perform in a VFL framework if the data can only remain on clients and cannot be transmitted?
(3) How can we improve communication efficiency, given each outer iteration requires n inner iterations?

We propose Teb-means to address the subproblems. We first introduce an effective stepwise optimization strategy that transforms the loss function (13) into a TMP, and solves it using the CO scheme, as shown in Figure 2(a) (see Section 4.1). Before addressing (2), we refine the CO scheme to improve iteration efficiency and reduce communication rounds in the VFL setting. We propose GBCO, a CO variant that improves iteration efficiency by eliminating inefficient one-by-one processing and mitigating slow loss reduction from sequential updates, as illustrated in Figure 2(b) (see Section 4.2.1). For (2), we show that TMP can be decomposed into multiple subproblems, each solved on a client using GBCO. Clients execute computations as directed by the central server and transmit only intermediate results as shown in Figure 2(c) (see Section 4.2.2).

4.1 CO for Stepwise Optimization

We first convert the loss function (13) into a TMP. We then apply the CO to iteratively assign data points to clusters by minimizing the loss function along coordinate directions, avoiding full reliance on previous iteration information, as in Lloyd-based balanced clustering algorithms. Notably, we introduce a variable to eliminate parsing and redundancy in G, reducing memory cost and precomputing intermediate results to avoid unnecessary computations.



(a) A $\mathbf{3}\times\mathbf{3}$ Indicator Matrix G (b) Candidates of G_1 (c) Example of $G_1^{(0)}$

Figure 3: Examples of different forms of indicator matrices.

Loss Function Reformulation. We reformulate the loss function (13) into a form executable on a single server, expressed as follows:

$$\min_{\mathbf{G},\mathbf{O}} \eta\|\mathbf{X}-\mathbf{G}^\top\mathbf{O}\|_F^2 + (1-\eta)\sum_{i=1}^{k}\Big(\sum_{j=1}^{n} g_{ij} - \frac{n}{k}\Big)^2. \tag{14}$$

Recall Equations (6) and (8), we transform Equation (14) into a univariate optimization problem, which can be expressed as follows:

$$\max_{\mathbf{G}}\Big\{\varphi(\mathbf{G}) := \eta\sum_{i=1}^{k}\frac{\mathbf{g}_i\mathbf{X}\mathbf{X}^\top\mathbf{g}_i^\top}{\mathbf{g}_i\mathbf{g}_i^\top} + (1-\eta)\sum_{i=1}^{k}\|\mathbf{g}_i\mathbf{g}_i^\top - \frac{n}{k}\|_2^2\Big\}. \tag{15}$$

Recall that $\mathbf{G}$ has n columns, each representing a data point, where the position of 1 in a column indicates its assigned cluster. For example, as shown in Figure 3(a), the 3×3 indicator matrix shows that the first data point (1st column) belongs to the third cluster (3rd row), the second to the first cluster, and the third to the second cluster. Optimizing objectives (15) determines how to assign data points to clusters. Here, we optimize the objective function by completing the assignment in n iterations, with each iteration updating one column (i.e., assigning one data point to a cluster), greedily ensuring that each assignment maximizes the loss function.

Candidate Selection. Updating one column of $\mathbf{G}$ means adjusting the position of 1 in that column (i.e., determining which row it belongs to). This can be achieved by selecting one from the k candidates in $\{\mathbf{G}^{(1)},\mathbf{G}^{(2)},\ldots,\mathbf{G}^{(k)}\}$. Specifically, when updating the j-th column of $\mathbf{G}$, the only difference between $\mathbf{G}_j^{(h)}$ and $\mathbf{G}_j^{(l)}$ lies in the (h,j)-th and (l,j)-th entries. The (h,j)-th entry of $\mathbf{G}_j^{(h)}$ is 1, and all other entries are 0, with a similar structure for $\mathbf{G}_j^{(l)}$. For instance, as shown in Figure 3(b), the only difference between $\mathbf{G}_1^{(2)}$ and $\mathbf{G}_1^{(3)}$ is the position of the "1" in the first column, while the entries in the other columns remain the same.Hence, the j-th subproblem can be defined as follows:

$$\max_{h\in[k]}\Big\{\varphi(\mathbf{G}_j^{(h)}) = \eta\sum_{i=1}^{k}\frac{\mathbf{g}_i^{(h)}\mathbf{X}\mathbf{X}^\top(\mathbf{g}_i^{(h)})^\top}{\mathbf{g}_i^{(h)}(\mathbf{g}_i^{(h)})^\top} + (1-\eta)\sum_{i=1}^{k}\|\mathbf{g}_i^{(h)}(\mathbf{g}_i^{(h)})^\top - \frac{n}{k}\|_2^2\Big\}, \tag{16}$$

where $h\in[k]$, with $[k]$ denoting the set $\{1,2,\ldots,k\}$, and the i-th row of $\mathbf{G}_j^{(h)}$ is denoted as $\mathbf{g}_i^{(h)}$, with the subscript j omitted.

Loss Function Simplication. Equation (16) contains redundant computations that can be removed to improve efficiency. This is because $\mathbf{G}^{(h)}$ and $\mathbf{G}^{(l)}$ $(h,l=1,2,\ldots,k, h\neq l)$ share many common entries and differ only in the h-th and l-th columns. We apply the method from [38] to simplify Equation (16). A variable $\mathbf{G}_j^{(0)}\in\mathbb{R}^{k\times n}$ is introduced to simplify the loss function and reduce computational complexity. The entries in the j-th column of $\mathbf{G}_j^{(0)}$ are 0, while the entries in the other columns remain the same as those in $\mathbf{G}_j^{(h)}$. Hence, we provide an equivalent formulation for Problem (16):

$$\begin{aligned}\max_{h\in[k]}\ \psi(h,j) &= \max_{h\in[k]}\ \varphi(\mathbf{G}_j^{(h)}) - \varphi(\mathbf{G}_j^{(0)})\\
&= \max_{h\in[k]}\eta\Big(\sum_{i=1}^{k}\frac{\mathbf{g}_i^{(h)}\mathbf{X}\mathbf{X}^\top(\mathbf{g}_i^{(h)})^\top}{\mathbf{g}_i^{(h)}(\mathbf{g}_i^{(h)})^\top} - \sum_{j=1}^{k}\frac{\mathbf{g}_i^{(0)}\mathbf{X}\mathbf{X}^\top(\mathbf{g}_i^{(0)})^\top}{\mathbf{g}_i^{(0)}(\mathbf{g}_i^{(0)})^\top}\Big)\\
&\quad+(1-\eta)\Big(\sum_{i=1}^{k}\|\mathbf{g}_i^{(h)}(\mathbf{g}_i^{(h)})^\top - \frac{n}{k}\|_2^2 - \sum_{i=1}^{k}\|\mathbf{g}_i^{(0)}(\mathbf{g}_i^{(0)})^\top - \frac{n}{k}\|_2^2\Big), \qquad (17)\\
&= \max_{h\in[k]}\eta\Big(\frac{\mathbf{g}_h^{(h)}\mathbf{X}\mathbf{X}^\top(\mathbf{g}_h^{(h)})^\top}{\mathbf{g}_h^{(h)}(\mathbf{g}_h^{(h)})^\top} - \frac{\mathbf{g}_h^{(0)}\mathbf{X}\mathbf{X}^\top(\mathbf{g}_h^{(0)})^\top}{\mathbf{g}_h^{(0)}(\mathbf{g}_h^{(0)})^\top}\Big)\\
&\quad+(1-\eta)\Big(\|\mathbf{g}_h^{(h)}(\mathbf{g}_h^{(h)})^\top - \frac{n}{k}\|_2^2 - \|\mathbf{g}_h^{(0)}(\mathbf{g}_h^{(0)})^\top - \frac{n}{k}\|_2^2\Big)\end{aligned}$$

where the equation only contains only two terms, whereas Problem (16) includes k terms. Despite this difference, the two optimization problems are equivalent, which significantly decreases the computational complexity. An example is shown in Figure 3(c), where the result of $\mathbf{G}_1^{(2)} - \mathbf{G}_1^{(0)}$ leaves only the first column containing 1, while all other columns contain only 0 and are removed.

Space Storage Reduction. When updating the j-th column of $\mathbf{G}$, $k+1$ variables, denoted as $\mathbf{G}_j^{(0)},\mathbf{G}_j^{(1)}\cdots,\mathbf{G}_j^{(k)}$, need to be stored in memory, which is memory-intensive. To reduce the memory cost, we use the current variable $\mathbf{G}$ to replace the $k+1$ variable, including $\mathbf{G}^h$ $(h=1,2,\ldots,k)$ and $\mathbf{G}^0$. Specifically, we record the position of element 1 in the i-th row as $w(w=1,2,\ldots,k)$. $\mathbf{g}_h^{(h)}$ denotes the h-th row of $\mathbf{G}^{(k)}$, $\mathbf{g}_h^{(0)}$ is the h-th row of $\mathbf{G}^{(0)}$, and $\mathbf{g}_h$ represents the h-th row of the current $\mathbf{G}$. We derive $\psi(h,j)$ in Problem (17) under two cases:

Case 1. When $h=w$ and $\mathbf{g}_h^{(h)}=\mathbf{g}_h$, the j-th entry of both $\mathbf{g}_h^{(h)}$ and $\mathbf{g}_h$ is 1. We define $\kappa_h=\mathbf{g}_h-\mathbf{g}_h^{(0)}$, meaning that the i-th entry of κ_h is 1, while all other entries are 0. The equation $\mathbf{g}_h^{(h)}(\mathbf{g}_h^{(h)})^\top - z_h = \mathbf{g}_h\mathbf{g}_h^\top - z_h$ holds. Moreover, since $\mathbf{g}_h^{(0)}=\mathbf{g}_h-\kappa_h$, we have $(\mathbf{g}_h-\kappa_h)(\mathbf{g}_h-\kappa_h)^\top=\mathbf{g}_h\mathbf{g}_h^\top-1$. To simplify notation, we define $\mathbf{a}_h=\mathbf{g}_h\mathbf{X}$, $\mathbf{a}_h\mathbf{a}_h^\top=\mathbf{g}_h\mathbf{X}\mathbf{X}^\top\mathbf{g}_h^\top$, and $\mathbf{b}_h=\mathbf{g}_h\mathbf{g}_h^\top$. The resulting simplified equation is as follows:

$$\psi(h,j)=\eta\Big(\frac{\mathbf{a}_h\mathbf{a}_h^\top}{\mathbf{b}_h} - \frac{\mathbf{a}_h\mathbf{a}_h^\top - 2\mathbf{x}_j\mathbf{a}_h^\top + \mathbf{x}_j\mathbf{x}_j^\top}{\mathbf{b}_h-1}\Big) + 2(1-\eta)\Big(\frac{k\mathbf{b}_h-n-k}{k}\Big). \tag{18}$$

Case 2. When $h\neq w$, we have $\mathbf{g}_h^{(0)}=\mathbf{g}_h$, and the j-th entry of both $\mathbf{g}_h^{(h)}$ and $\mathbf{g}_h$ is 1. We define $\kappa_h=\mathbf{g}_h^{(h)}-\mathbf{g}_h$, meaning that the j-th entry of κ_h is 1, while all other entries remain 0. Thus, $\mathbf{g}_h^{(h)}=$

Algorithm 1: CO_for_Balanced_k_Means($\mathbf{X}, k, n$)

```
Input: X: the dataset, k: number of clusters, n: number of points.
Output: G: the indicator matrix.
1 Initialize G by k-means++;
2 Compute and store $g_h X$, $g_h g_h^\top$, $g_h XX^\top g_h^\top$, and $x_j x_j^\top$,
    $\forall h \in [k], j \in [n]$;
3 while not converge do
4     for $j \leftarrow 1$ to $n$ do
5         Compute $\varphi(h, j), h \in [k]$ by Equation (20);
6         Update the $j$-th column of G by Equation (21);
7         Update $g_h X$, $g_h g_h^\top$, and $g_h XX^\top g_h^\top$ by Equations (22)
            & (23);
8 return G;
```

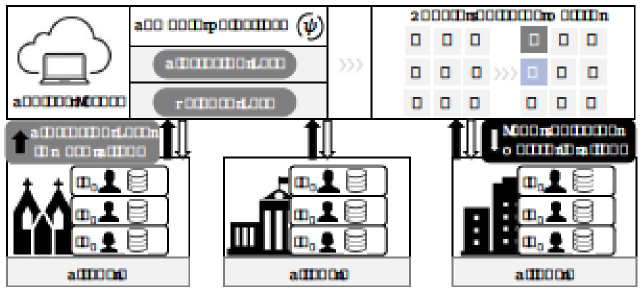


Figure 4: An example of our VFL framework for GBCO.

$\kappa_h + g_h$, and the expansion gives $(\kappa_h + g_h)(\kappa_h + g_h)^\top = g_h g_h^\top + 1$. Additionally, since $g_h^{(0)} = g_h$, it follows that $g_h(g_h^{(0)})^\top = g_h g_h^\top$. The corresponding simplified equation is as follows:

$$\psi(h, j) = \eta\left(\frac{a_h a_h^\top + 2x_j a_h^\top + x_j x_j^\top}{b_h + 1} - \frac{a_h a_h^\top}{b_h}\right) + 2(1-\eta)\left(\frac{kb_h - n + k}{k}\right). \quad (19)$$

Therefore, based on the above two cases, $\psi(h, j)$ is given by:

$$\psi(h, j) = \begin{cases} \eta\left(\frac{a_h a_h^\top}{b_h} - \frac{a_h a_h^\top - 2x_j a_h^\top + x_j x_j^\top}{b_h - 1}\right) + 2(1-\eta)\left(\frac{kb_h - n - k}{k}\right), \text{ if } h = w, \\ \eta\left(\frac{a_h a_h^\top + 2x_j a_h^\top + x_j x_j^\top}{b_h + 1} - \frac{a_h a_h^\top}{b_h}\right) + 2(1-\eta)\left(\frac{kb_h - n + k}{k}\right), \text{ if } h \neq w. \end{cases} \quad (20)$$

Recall Equation (4) and Equation (20), the update rule for the j-th column of G is given by:

$$g_{ir} = \begin{cases} 1, & \text{if } r = \arg\max_h \psi(h, j), \\ 0, & \text{otherwise.} \end{cases} \quad (21)$$

Time Complexity Reduction. To improve the efficiency of the algorithm, we also follow [15] that proposes an incremental updating approach. This approach reduces computational costs by updating only the modified terms, thereby generating results more quickly and requiring less storage compared to recomputing all terms.

Consider the loss function defined in Equation (20), which has four terms: $g_h X$, $g_h XX^\top g_h^\top$, $g_h g_h^\top$, and $x_j^\top x_j$. The term $x_j^\top x_j$ does not need to be updated, as it can be precomputed and stored. For updating a_h and b_h, two cases should be considered: 1) When $w = r$, the indicator matrix G remains unchanged, so the other terms do not require updating; 2) When $w \neq r$, the (w, j)-th and (r, j)-th elements of G should be swapped, meaning that g_{wj} and g_{rj} are exchanged. The update process is then described as follows:

$$\begin{aligned} gX_w &\leftarrow gX_w - x_j, \quad gX_r \leftarrow gX_r + x_j, \\ g_w g_w^\top &\leftarrow g_w g_w^\top - 1, \quad g_r g_r^\top \leftarrow g_r g_r^\top + 1. \end{aligned} \quad (22)$$

Further, the update of $a_h a_h^\top = g_h XX^\top g_h^\top$ depends on the update of Xg_h . Specifically, when $w = r$, Xg_h remains unchanged for the next iteration, and consequently, $g_h XX^\top g_h^\top$ also remains unchanged. On the other hand, when $q \neq p$, only gX_h (for $h = w, r$) needs to be updated according to Equation (22). Therefore, we only need to update $g_h XX^\top g_h^\top$ using the following equation:

$$\begin{aligned} g_r XX^\top g_r^\top &\leftarrow g_r XX^\top g_r^\top - 2x_j X_w^\top g_w^\top + x_j x_j^\top, \\ g_w XX^\top g_w^\top &\leftarrow g_w XX^\top g_w^\top + 2x_j X_r^\top g_r^\top + x_j x_j^\top. \end{aligned} \quad (23)$$

Algorithm 2: GBCO_for_Balanced_k_Means($\mathbf{X}, k, n, c$)

```
Input: X: the dataset, k: # of clusters, n: # of points, f: # of blocks.
Output: G: the indicator matrix.
1  Initialize G by k-means++;
2  Divide $\{1, 2, \ldots, n\}$ into c blocks: $\{F_1, \ldots, F_c\}$;
3  Compute and store $g_h X$, $g_h g_h^\top$, $g_h XX^\top g_h^\top$, and $x_j x_j^\top$,
     $h \in [k], j \in [n]$;
4  $\mathcal{G} \leftarrow$ An empty set used to store the indicator matrix;
5  while not converge do
6      for idx $\leftarrow 1$ to $f$ do
7          Compute $\phi(h, j), j \in F_{idx}, h \in [k]$ using Equation (20);
8          $G_0 \leftarrow$ Temporary variable equal to G;
9          Update the columns of $F_{idx}$ of $G_0$ using Equation (21);
10         Add $G_0$ to $\mathcal{G}$;
11     Find the optimal G using Equation (24);
12     Update $Xg_h$, $g_h^\top g_h$, and $g_h^\top X^\top X g_h$ using Equations (22) & (23) ;
13 return G;
```

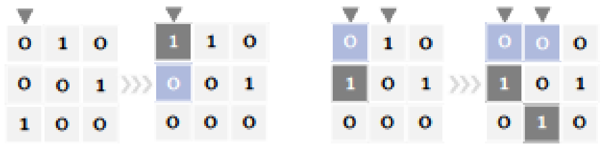

(a) Updating One at a Time (b) Updating in Batches

Figure 5: Examples of two updating methods.

Algorithm Design. Algorithm 1 shows that we first initialize G using k-means++ (Lines 1), which helps avoid local optima and accelerates convergence. Then, we compute and store $g_h X$, $g_h g_h^\top$, $g_h XX^\top g_h^\top$, and $x_j x_j^\top$, for all $h \in [k]$ and $j \in [n]$ (Lines 2). Next, the algorithm iteratively updates G by calculating the loss and updating the parameters (Lines 5-7) until convergence.

4.2 VFL for GBCO

Before introducing our federated algorithm, we propose GBCO, which improves CO's iteration efficiency, inherently improving communication efficiency in the federated setting. We then show that TMP can be regarded as multiple subproblems, where clients perform computations under the central server's guidance to greedily update the indicator matrix in batches. Before delving into the details, we provide an example to illustrate the workflow below.

A Toy Example. Recall that the goal of the problem is calculating G, we use a simple example in Figure 4 to illustrate the workflow of our

VFL framework. It shows the process of updating a 3×3 indicator matrix through a central server and three clients. In each iteration, each client computes the clustering loss based on its local data and uploads it to the central server. The central server then aggregates the clustering loss, computes the balance loss, and updates $\mathbf{G}$, which is sent back to the clients. This process repeats until convergence.

4.2.1 Improving Iteration Efficiency. We improve iteration efficiency by mitigating the slow increase in Equation (17) caused by one-by-one processing. As shown in Figure 5, we illustrate this with a 3×3 indicator matrix: Figure 5(a) updates one entry at a time, while Figure 5(b) updates in batches (batch size = 2). Moreover, we update one block while keeping the others fixed and compute the indicator matrix. The indicator matrix with the greatest increase in Equation (17) is selected for the update.

Block-based Optimization. Unlike Algorithm 1 (Line 5), which optimizes variables sequentially, we define $n_c = \lceil n/c \rceil$ as the number of blocks, where $c \le n$ represents the number of columns (data points) in each block. In each iteration of GBCO, the computation step solves a block of columns rather than a single column of $\mathbf{G}$ and updates the relevant terms. This approach lowers Algorithm 1's time complexity and improves its fit for distributed use.

Block Selection. Instead of cyclically updating (Algorithm 1, Line 6), we select the block that maximizes the increase in Equation (17) and update them. For each block in $\{\mathbf{F}_1, \mathbf{F}_2, \dots, \mathbf{F}_c\}$, we compute the corresponding $\{\mathcal{G} = \mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_c\}$, and determine the optimal $\mathbf{F}^*$ using the following formula:

$$\mathbf{G}^* = \arg\max_{\mathbf{G}\in\mathcal{G}} \varphi(\mathbf{G}), \tag{24}$$

where the corresponding $\mathbf{F}^*$ of $\mathbf{G}^*$ is the block of columns to be updated in the current iteration.

Algorithm Design. As shown in Algorithm 2, after initializing $\mathbf{G}$ using `k-means++`, we divide $\{1, 2, \cdots, n\}$ into c blocks (Line 2). In each iteration, we first compute $\phi(h, j)$ for $j \in \mathbf{F}_{\texttt{idx}}$ (Line 7), which identifies the row (cluster) to which the "1" in the j-th column (data point) belongs. Then, we set a temporary variable $\mathbf{G_0}$ equal to $\mathbf{G}$, update the columns of $\mathbf{F}_{\texttt{idx}}$ in $\mathbf{G_0}$, and add $\mathbf{G_0}$ to $\mathcal{G}$ (Line 8-10). Next, we evaluate Equation (13) over $\mathbf{G_0} \in \mathcal{G}$, and select one for this iteration using Equation (21) (Line 11). This approach reduces time complexity while keeping the increase in loss small during batch updates. We iteratively repeat this process until convergence.

4.2.2 VFL Framework for GBCO. To better understand the notation in this section, recall that $\{\mathbf{X}_m\}_{h=1}^{M}$ represents the partition of $\mathbf{X}$ along the feature dimension across M local parties, where d_m is the number of features assigned to each client. Additionally, for the i-th data point $\mathbf{x}_i \in \mathbb{R}^d$, we express it as $\mathbf{x}_i = (\mathbf{x}_i^1, \mathbf{x}_i^2, \dots, \mathbf{x}_i^M)$, where $\mathbf{x}_i^m \in \mathbb{R}^{d_m}$ for $m \in [M]$.

Client. For each client with data $\mathbf{X}_m$, its role is to assist the central server by computing Equation (17), denoted as `ComputeObj`, and updating $\mathbf{G}$, denoted as `Update_G`. In `ComputeObj`, the client focuses solely on clustering loss, as balance loss is handled by the central server. Thus, Equation (17) on the client can be rewritten as follows:

$$\overline{\gamma}(h, j, m) = \frac{\mathbf{g}_h^{(h)} \mathbf{X}_m \mathbf{X}_m^\top (\mathbf{g}_h^{(h)})^\top}{\mathbf{g}_h^{(h)} (\mathbf{g}_h^{(h)})^\top} - \frac{\mathbf{g}_h^{(0)} \mathbf{X}_m \mathbf{X}_m^\top (\mathbf{g}_h^{(0)})^\top}{\mathbf{g}_h^{(0)} (\mathbf{g}_h^{(0)})^\top}. \tag{25}$$

Algorithm 3: `GBCO_Center`(n, k, c)

```
Input: $n$: # of points, $k$: # of clusters, $c$: # of blocks.
Output: $\mathbf{G}$: the indicator matrix.
1  Initialize $\mathbf{G}$ by Federated $k$-means++ ;
2  Divide $\{1, 2, \dots, n\}$ into $c$ blocks: $\{\mathbf{F}_1, \dots, \mathbf{F}_c\}$;
3  for $m \in [M]$ in parallel over local parties do
4      Call INITIALIZATION in Algorithm 4;
5  $\mathcal{G} \leftarrow$ An empty set used to store the indicator matrix;
6  while not converge do
7      for idx $\leftarrow 1$ to $c$ do
8          Compute $\mathcal{L}(\mathbf{G})$ by calling Equation (12);
9          $\mathbf{G}_0 \leftarrow$ Temporary variable equal to $\mathbf{G}$;
10         Compute $\overline{\mathcal{F}}(h, j), j \in \mathbf{F}_{\texttt{idx}}, h \in [k]$ by calling
             COMPUTEOBJ in Algorithm 4;
11         Compute $\mathbf{G}$ by calling UPDATE_G in Algorithm 4;
12         Adding $\mathbf{G}$ to $\mathcal{G}$;
13     Find the optimal $\mathbf{G}$ using Equation (24);
14 return F;
```

Algorithm 4: `GBCO_Client`$(\mathbf{X}_m, \mathbf{G})$

```
Input: $\mathbf{X}_m$: the dataset of $m$-th client, $\mathbf{G}$: the indicator matrix.
1  if the server calls INITIALIZATION then
2      Compute and store $\mathbf{g}_h\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m$, and
         $\mathbf{x}_j^\top\mathbf{x}_j$, $m \in [M], j \in [n]$ locally;
3  if the center calls COMPUTEOBJ then
4      Compute $\overline{\gamma}(h, j, m), h \in [k], j \in \mathbf{F}_{\text{idx}}, m \in [M]$ by (27);
5      Upload $\overline{\gamma}(h, j, m), h \in [k], j \in \mathbf{F}_{\text{idx}}, m \in [M]$ to the center;
6  if the center calls UPDATE_G then
7      Update $\mathbf{g}_h\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m$, and
         $\mathbf{x}_j\mathbf{x}_j^\top$, $m \in [M], j \in [n]$ by (29);
```

Here, we omit the cases $h = w$ and $h \neq w$ as they have already been discussed in Section 4.1. Thus, we directly present the formula:

$$\overline{\gamma}(h, j, m) = \begin{cases} \frac{\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top}{\mathbf{g}_h\mathbf{g}_h^\top} - \frac{\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top - 2\mathbf{x}_j\mathbf{X}_m^\top\mathbf{g}_h^\top + \mathbf{x}_j\mathbf{x}_j^\top}{\mathbf{g}_h\mathbf{g}_h^\top - 1}, & \text{if } h = w, \\ \frac{\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top + 2\mathbf{x}_j\mathbf{X}_m^\top\mathbf{g}_h^\top + \mathbf{x}_j\mathbf{x}_j^\top}{\mathbf{g}_h\mathbf{g}_h^\top + 1} - \frac{\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top}{\mathbf{g}_h\mathbf{g}_h^\top}, & \text{if } h \neq w. \end{cases} \tag{26}$$

After computations, $\overline{\gamma}(h, j, m)$ is sent to the central server as an intermediate result for computing Equation (17).

Central Server. The central server clusters the distributed data by: 1) requesting each client to compute `ComputeObj` for clustering loss and 2) calculating balance loss, as shown in Equation (12). Specifically, after gathering intermediate results from clients, including $\overline{\gamma}(h, j, m)$ for $m \in [M]$, and computing the balance loss, Equation (20) is expressed as follows:

$$\psi(h, j) = \begin{cases} \eta \sum_{m\in M} \overline{\gamma}(h, j, m) + 2(1-\eta)(\mathbf{g}_h\mathbf{g}_h^\top - \frac{n}{k}) - (1-\eta), & \text{if } h = w, \\ \eta \sum_{m\in M} \overline{\gamma}(h, j, m) + 2(1-\eta)(\mathbf{g}_h\mathbf{g}_h^\top - \frac{n}{k}) + (1-\eta), & \text{if } h \neq w. \end{cases} \tag{27}$$

We use the following equation to update the j-th column of $\mathbf{G}$:

$$\mathbf{g}_{jr} = \begin{cases} 1, & \text{if } r = \arg\max_h \overline{\mathcal{F}}(h, j), \\ 0, & \text{otherwise.} \end{cases} \tag{28}$$

Table 3: Complexity analysis of different methods.

Method Name	Cluster Balance	Time Complexity	Communication Round
`Lloyd` [34]	×	$O(nd_mkT)$	T
`CDKM` [38]	×	$O(nd_mkT)$	nT
`BCLS` [31]	✓	$O(n^2kT + d_m^2T)$	nT
`FCFC` [32]	✓	$O(nd_mkT)$	T
F^3`KM` [52]	✓	$O(n(d_m + l_m)kT)$	bT
`Teb-means`	✓	$O(nd_mkT)$	c^2T

Reconsider the four terms in the loss function (27): $\mathbf{g}_h\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top$, $\mathbf{g}_h\mathbf{X}_m$, and $\mathbf{x}_j^m(\mathbf{x}_j^m)^\top$. Note that $\mathbf{x}_j^m(\mathbf{x}_j^m)^\top$ is a constant on the client and does not require updating. Let r represent the current index of 1 in the j-th column of the indicator matrix $\mathbf{G}$. If $r = s$, $\mathbf{G}$ remains unchanged. Otherwise, if $r \neq s$, the following update formulas can be directly applied, using computations similar to those in Equation (22) and Equation (23):

$$\begin{aligned} &\mathbf{gX}_w \leftarrow \mathbf{Xg}_w - \mathbf{x}_j^m, \quad \mathbf{gX}_r \leftarrow \mathbf{gX}_r + \mathbf{x}_j^m, \\ &\mathbf{g}_w\mathbf{g}_w^\top \leftarrow \mathbf{g}_w\mathbf{g}_w^\top - 1, \quad \mathbf{g}_r\mathbf{g}_r^\top \leftarrow \mathbf{g}_r\mathbf{g}_r^\top + 1, \\ &\mathbf{g}_r\mathbf{XX}^\top\mathbf{g}_r^\top \leftarrow \mathbf{g}_r\mathbf{XX}^\top\mathbf{g}_r^\top - 2\mathbf{x}_j^m\mathbf{X}_w^\top\mathbf{g}_w^\top + \mathbf{x}_j^m(\mathbf{x}_j^m)^\top, \\ &\mathbf{g}_w\mathbf{XX}^\top\mathbf{g}_w^\top \leftarrow \mathbf{g}_w\mathbf{XX}^\top\mathbf{g}_w^\top + 2\mathbf{x}_j^m\mathbf{X}_r^\top\mathbf{g}_r^\top + \mathbf{x}_j^m(\mathbf{x}_j^m)^\top. \end{aligned} \tag{29}$$

Algorithm Design. Details of federated GBCO on the central server and client sides are detailed in Algorithms 3 and 4. The central server first initializes the indicator matrix $\mathbf{G}$ using federated k-means++ (available in our code repository [7]). It then computes the four terms in (29) by calling INITIALIZATION (LINES 3–4 IN ALGORITHM 3; LINE 2 IN ALGORITHM 4). It then instructs clients to compute clustering loss using local data and return the results (Line 10 of Algorithm 3; Lines 4–5 of Algorithm 4). The central server aggregates the results and updates $\mathbf{G}$ by calling UPDATE_G (Line 11 of Algorithm 3; Line 7 of Algorithm 4). This iterative process continues until convergence, yielding $\mathbf{G}$.

5 COMPLEXITY ANALYSIS OF `TEB-MEANS`

This section demonstrates that `Teb-means` is efficient in terms of both communication rounds and time complexity. Specifically, under the mild condition that the number of iterations T is fixed and the partition of $\mathbf{G}$ is divided into a fixed number of blocks c, the communication rounds remain constant. Meanwhile, the time complexity of each client is linear, making the method efficient.

THEOREM 1. *Let c be the number of blocks and T the number of iterations. The communication round is $O(c^2T)$.*

PROOF. As shown in Algorithm 4, each iteration of `Teb-means` requires c^2 communication rounds between the central server and clients to solve $\mathbf{G}$. This is because traditional block coordinate descent divides $\mathbf{G}$ into c blocks, with each block containing $\frac{n}{c}$ columns, and each communication round solves one block of columns. Notably, our method requires selecting the best block, which, in the worst case, involves c communication rounds to identify the optimal block. Hence, the total communication rounds per iteration amount to c^2. If T denotes the total number of iterations of `Teb-means`, the overall communication round is $O(c^2T)$. □

Table 4: An overview of datasets (M for millions).

Dataset	`MTG`	`Census`	`Game`	`NYC`	`Crime`	`Retail`	`Amazon`	`MovieLens`
Ref.	[5]	[9]	[2]	[6]	[8]	[3]	[1]	[4]
d	53	69	79	100	127	600	2400	5000
Scale	5.98M	2.46M	3M	0.1M	0.01M	1.64M	21M	5.3M

THEOREM 2. *The time complexity on each client is $O(nd_mkT)$.*

PROOF. For stored variables, under the condition $h, j = 1, 2, \ldots, k$, computing $\mathbf{g}_h\mathbf{X}_m$ and $\mathbf{g}_h\mathbf{g}_h^\top$ requires nd_m and n additions, respectively, while computing $x_jx_j^\top$ requires nd_m multiplications, and computing $\mathbf{g}_h\mathbf{X}_m\mathbf{X}_m^\top\mathbf{g}_h^\top$ requires d_mk multiplications. For computing $\psi(h, j)$ $(h = 1, 2, \ldots, k)$ using Equation (20), i.e., calculating $\mathbf{x}_j^m\mathbf{X}_m^\top\mathbf{g}_h^\top$ requires d_mk multiplications, and calculating $k\mathbf{b}_h$ requires 1 multiplication. The time complexity of updating the j-th column of $\mathbf{G}$ using Equation (21) can be ignored, as it involves only constant multiplications. Updating $\mathbf{g}_h\mathbf{X}_m$ and $\mathbf{g}_h\mathbf{g}_h^\top$ based on (22) requires $2d_m + 2$ additions. Specifically, since $x_j^m \in \mathbb{R}^{1\times d_m}$, the first formula requires d_m additions, as does the second. The third formula involves a scalar and needs 1 addition, and the fourth formula also requires 1 addition. For (23), since the required values have already been computed in advance, there is no need to compute them again. Therefore, the overall computational complexity is $nd_mkT + nd_m + d_mk$ and time complexity is $O(nd_mkT)$. □

Remark. GBCO maintains communication efficiency under mild conditions. Specifically, instead of updating one row at a time, our algorithm updates $\frac{n}{c}$ rows of the indicator matrix simultaneously. As a result, `Teb-means` requires fewer communication rounds than other balanced k-means algorithms [31, 32] adapted for VFL. Moreover, `Teb-means` achieves lower time complexity compared to other balanced k-means algorithms, including [11, 31, 35]. While F^3`KM` [51] and `FCFC` [32] offer constant communication rounds in each iteration (notably, F^3`KM` [51] depends only on constants $b \in \mathbb{Z}^+$) and linear per-client time complexity, `Teb-means` achieves better balance performance compared with them (see Section 6.3). A summary is provided in Table 3.

6 EXPERIMENTS

In this section, we conduct experiments on 8 real-world datasets to validate the superiority of `Teb-means`. We begin by describing the experimental setup (see Section 6.1), followed by an analysis of key parameters (see Section 6.2). Next, we evaluate the clustering performance of `Teb-means`, showing that it is more balanced, preserves cluster quality better, and faster compared to other balanced clustering algorithms adapted for VFL (see Section 6.3).

6.1 Experimental Setup

Dataset. We use eight real-world datasets: `MTG`, `Census`, `Game`, `NYC`, `Crime`, `Retail`, `Amazon`, and `MovieLens`. `MTG`, `Amazon`, and `MovieLens` each contain over 5 million data points. Additionally, `Amazon` and `MovieLens` have more than a thousand dimensions. Notably, each dataset can simulate scenarios suitable for VFL, and our method can extract useful information as described in Section

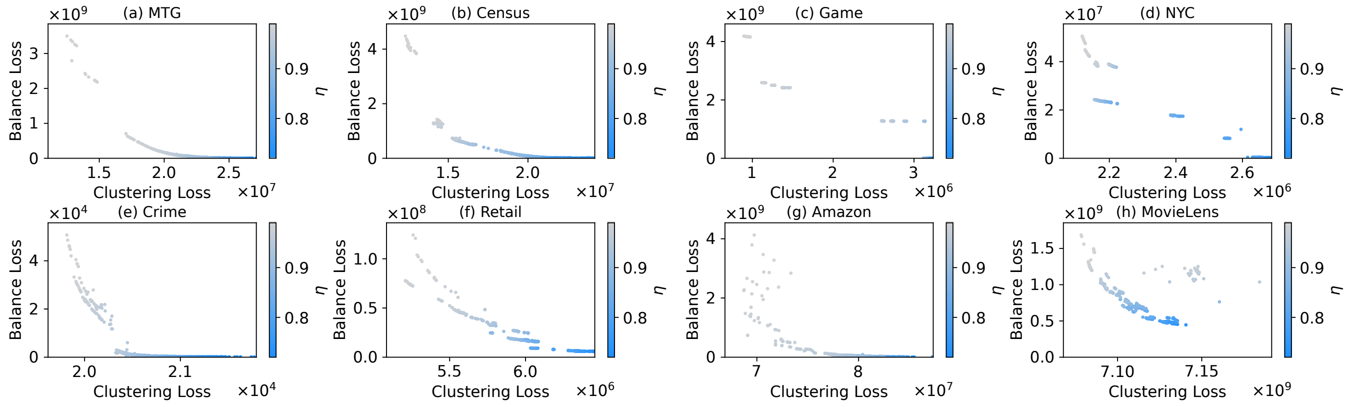


Figure 6: Clustering loss vs. balance loss for different values of η.

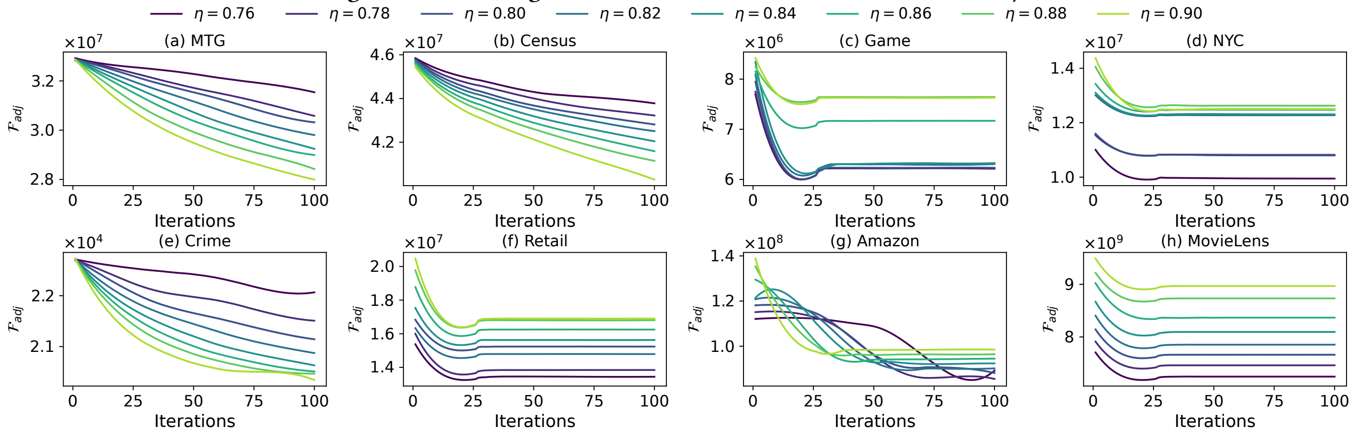


Figure 7: Impact of η on iterations.

1. Due to page limitations, detailed discussions of datasets are provided in our code repository [7]. The dimensions and scales of the datasets are provided in Table 4.

Implementations. We implemented Teb-means and conducted comparisons using MATLAB R2022a. All performance tests were performed on a server with an i9-14900KF CPU and 128GB of RAM. Unless otherwise noted, the setup consists of one central server and two clients, with feature dimensions evenly split between the clients. For example, in the case of Amazon, which has 2,400 dimensions ($d = 2400$), each client holds 1,200 dimensions ($d = 1200$). The code for the federated simulation and the proposed methodology is available in [7]. All algorithms run for a maximum of 100 iterations, with $k = 10$, $\eta = 0.8$, and $n_c = 8$ (see Section 6.2)

Comparisons. Teb-means againsts five methods that can be adapted for VFL: Lloyd's algorithm [34] (Lloyd), BCLS [31], FCFC [32], CDKM [38], and F^{3}KM [51], all discussed in Section 2. Lloyd and CDKM are clustering methods that do not consider balance, providing a baseline for clustering loss (as balancing inherently sacrifices clustering loss). BCLS, FCFC, and F^{3}KM represent state-of-the-art (SOTA) balanced k-means algorithms that can be adapted for VFL.

6.2 Parameter Studies

Impact of η on Clustering Performance. Theoretically, clustering loss and balance loss exhibit a roughly inverse relationship. When η is large, clustering loss is given more emphasis, while smaller values of η prioritize balance loss. We conduct 100 random k-means tasks, where η is randomly selected from the range $(0.7, 1)$. Notably, the range of η is determined based on the observation that when $\eta < 0.7$, the balance loss approaches zero. This occurs because, with a small η, optimizing balance loss is more effective than clustering loss, causing it to diminish first.

Observations: As shown in Figure 6, there exists a trade-off between clustering loss and balance loss as η varies, which generally exhibits an inverse relationship. We find that when η is around 0.8, both losses are relatively small, avoiding cases where either becomes large. Therefore, $\eta = 0.8$ is adopted as a suitable choice.

Impact of η on Iterations. We investigate the Impact of η on Iterations. To account for variations in η, we adjust the loss and define $\mathcal{F}_{adj}$ as the sum of clustering loss and balance loss. This removes the direct influence of η on loss function (13), ensuring a fair comparison across different η values. We set $\eta \in [0.76, 0.90]$, as values of η that are too small (e.g., $\eta < 0.76$) cause the optimization process to overemphasize balance loss, leading to high clustering loss and a poor cluster structure. Conversely, when η is too large, clustering loss dominates, which compromises balance.

Observations: As shown in Figure 7, $\mathcal{F}_{adj}$ decreases as iterations increase, regardless of η. In most cases, increasing η reduces $\mathcal{F}_{adj}$. This is because the loss function is defined as $\eta \times$ clustering loss + $(1 - \eta) \times$ balance loss. A larger η makes optimizing clustering loss more effective. Additionally, in real-world datasets, clustering loss

Table 5: Clustering loss and balance loss of Teb-means with different block sizes.

Dataset	Clustering Loss					Balance Loss				
	$n_c = 2$	$n_c = 8$	$n_c = 32$	$n_c = 128$	$n_c = 512$	$n_c = 2$	$n_c = 8$	$n_c = 32$	$n_c = 128$	$n_c = 512$
MTG	2.43E+07	2.52E+07	8.08E+07	8.04E+07	8.03E+07	6.46E+06	5.08E+06	1.69E+08	4.77E+09	3.05E+11
Census	2.02E+07	2.14E+07	5.40E+07	5.35E+07	5.42E+07	2.08E+07	2.14E+07	1.12E+09	4.44E+10	3.13E+12
Game	1.89E+07	3.14E+07	3.24E+07	3.20E+07	3.24E+07	1.18E+05	3.10E+06	4.88E+07	1.39E+09	1.03E+11
NYC	2.54E+06	2.55E+06	2.62E+06	4.34E+06	4.89E+06	6.74E+06	8.27E+06	7.66E+06	3.21E+08	3.01E+10
Crime	2.09E+04	2.09E+04	3.87E+04	3.83E+04	3.88E+04	1.71E+02	1.98E+02	1.00E+04	2.22E+05	1.82E+07
Retail	6.00E+06	6.08E+06	6.08E+06	6.82E+06	6.84E+06	9.16E+06	9.16E+06	9.25E+06	1.04E+07	9.94E+07
Amazon	7.45E+07	8.27E+07	8.88E+07	1.12E+08	1.01E+08	6.02E+06	6.39E+06	1.88E+07	1.62E+08	2.47E+10
MovieLens	7.12E+09	7.12E+09	7.12E+09	7.12E+09	7.14E+09	5.37E+08	5.41E+08	5.52E+08	5.48E+08	7.52E+08

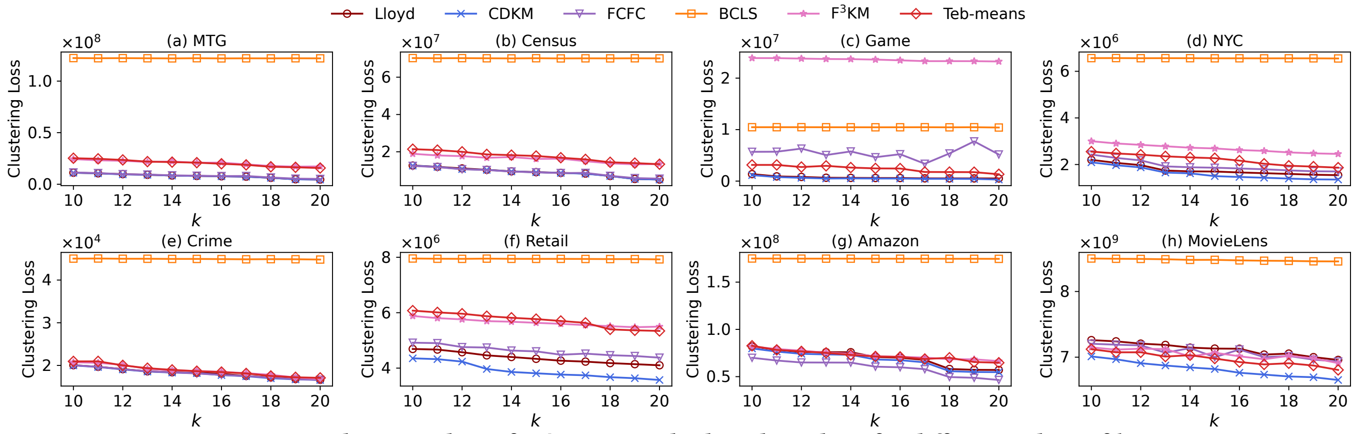


Figure 8: Clustering loss of Teb-means and other algorithms for different values of k.

Table 6: Communication rounds for different block sizes.

Dataset	$n_c = 2$	$n_c = 8$	$n_c = 32$	$n_c = 128$	$n_c = 512$
MTG	4.47E+08	2.79E+07	1.75E+06	1.09E+05	6.82E+03
Census	8.71E+07	5.45E+06	3.40E+05	2.13E+04	1.33E+03
Game	1.13E+08	7.03E+06	4.39E+06	2.75E+04	1.72E+03
NYC	1.25E+05	7.81E+04	4.88E+03	3.05E+02	1.91E+01
Crime	2.22E+00	3.56E+01	5.69E+05	9.10E+06	1.46E+08
Retail	3.36E+07	2.10E+06	1.31E+05	8.21E+03	5.13E+02
Amazon	5.51E+09	3.44E+08	2.15E+07	1.35E+06	8.41E+04
MovieLens	3.51E+08	2.19E+07	1.37E+06	8.57E+04	5.36E+03

is typically the dominant term. These two factors lead to a more significant reduction in clustering loss compared to balance loss.

Impact of Block Size. The block size affects both clustering loss, balance loss, and communication round. In general, as the block size increases, both clustering and balance losses tend to rise, while communication rounds decrease, and vice versa. We consider the following block sizes: $n_c \in \{2, 8, 32, 128, 512\}$.
Observations: As shown in Table 5, the block size has limited impact on the clustering loss, which remains within the same order of magnitude. In contrast, the balance loss becomes increasingly sensitive as the block size grows. For instance, when $n_c = 32$, the balance loss is more than ten times higher than that at $n_c = 8$. Moreover, as shown in Table 6, although $n_c = 32$ or larger values would result in fewer communication rounds, the significant increase in balance loss leads to poorer balance performance. Hence, $n_c = 8$ is the optimal choice, as it achieves low clustering and balance loss, while also improving the efficiency of sequentially scanning.

6.3 Superiority of Teb-means

Quality of Cluster Structure. We evaluate clustering loss to assess the quality of the resulting cluster structure compared to other methods. We also examine its variation across different k, where $k \in \{10, 11, \ldots, 20\}$, to test the sensitivity of Teb-means to k.
Observations: As shown in Figure 8, Teb-means achieves lower clustering loss compared to the balance algorithms, including F^{3}KM and BCLS. While Teb-means has a higher clustering loss than FCFC, the balance loss of FCFC is significantly higher than that of Teb-means, indicating its inferior ability to ensure balance. Most of the methods demonstrate a decrease in clustering loss as k increases. This is because data points are more likely to be assigned to closer cluster centers, reducing intra-cluster errors. Notably, BCLS does not exhibit a decrease in clustering loss as k increases, because it transforms the discrete indicator matrices into continuous matrix variables and optimizes a relaxed objective function that is relatively insensitive to k. Moreover, Teb-means has a slightly higher clustering loss than CDKM and Lloyd as it optimizes both clustering and balance loss, reflecting the inherent trade-off between them.

Balance Performance. We use several metrics to evaluate the balance performance of Teb-means. In addition to balance loss, we also include the coefficient of variation (CV) and N_{entro} [38]. The

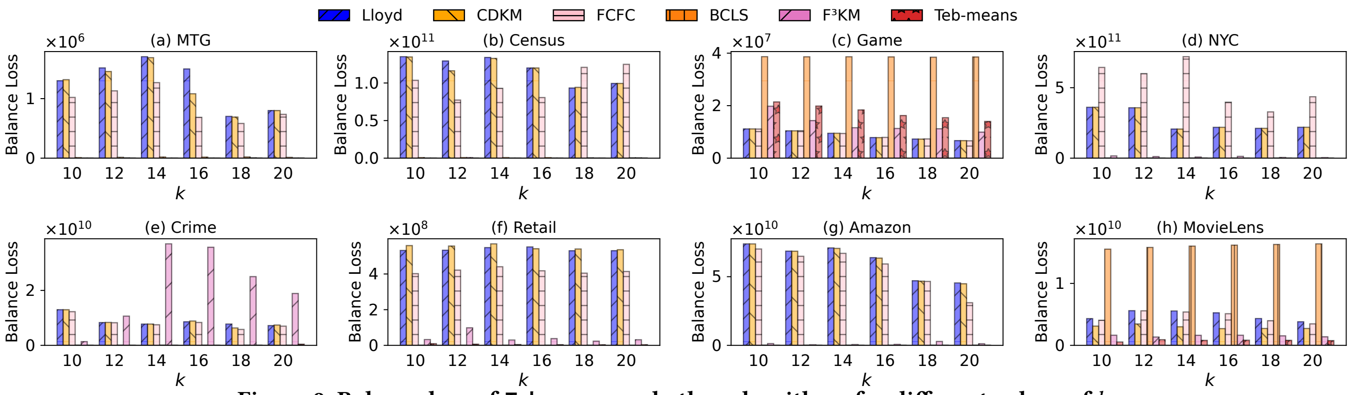


Figure 9: Balance loss of Teb-means and other algorithms for different values of k.

Table 7: Balance performance and runtime efficiency.

Dataset	Method	CV	N_{entro}	Runtime (s)
MTG	FCFC	13.19	0.90	2.22E+02
	BCLS	1.28	0.99	2.59E+03
	F^{3}KM	1.87	0.99	3.22E+02
	Teb-means	0.49	1.72	2.60E+02
Census	FCFC	15.40	0.81	2.79E+02
	BCLS	1.28	0.99	2.86E+03
	F^{3}KM	1.14	1.00	3.51E+02
	Teb-means	0.80	1.00	2.97E+02
Game	FCFC	26.64	0.56	4.52E+02
	BCLS	1.28	0.99	4.17E+03
	F^{3}KM	6.00	0.95	5.22E+02
	Teb-means	0.24	1.00	1.02E+02
NYC	FCFC	13.69	0.88	2.58E+01
	BCLS	1.28	0.99	4.87E+03
	F^{3}KM	13.62	0.81	6.11E+02
	Teb-means	8.86	0.93	2.04E+03
Crime	FCFC	7.96	0.96	6.05E-02
	BCLS	1.60	0.99	9.04E-01
	F^{3}KM	0.25	1.00	5.75E-01
	Teb-means	0.21	1.00	1.95E-01
Retail	FCFC	30.24	0.36	8.02E+00
	BCLS	1.09	1.00	2.51E+01
	F^{3}KM	15.12	0.84	1.59E+01
	Teb-means	7.52	0.94	2.73E+01
Amazon	FCFC	96.99	0.09	7.99E+03
	BCLS	94.44	0.11	3.76E+03
	F^{3}KM	94.44	0.11	2.12E+03
	Teb-means	94.44	0.11	6.49E+02
MovieLens	FCFC	23.72	0.62	3.23E+02
	BCLS	1.67	1.00	5.86E+02
	F^{3}KM	18.06	0.78	7.60E+02
	Teb-means	18.62	0.81	1.33E+02

formulas for these two metrics are as follows:

$$\text{CV} = \frac{k}{n}\sqrt{\sum_{i=1}^{k}\left(\mathbf{z}_i - \frac{n}{k}\right)^2},\ N_{\text{entro}} = -\frac{1}{\log k}\sum_{i=1}^{k}\frac{\mathbf{z}_i}{n}\log\left(\frac{\mathbf{z}_i}{n}\right), \qquad (30)$$

where $\mathbf{z} \in \mathbb{Z}^+$ is a vector containing each cluster size, with $\mathbf{z}_i$ representing the size of the i-th cluster. In this context, a smaller coefficient of variation (CV) indicates better balance performance, while N_{entro} closer to 1 indicates better balance performance.

Observations: As shown in Figure 9, Teb-means achieves the lowest balance loss in most cases. Moreover, as shown in Table 7, when using CV as the metric, our method achieves the lowest CV, demonstrating the best balance performance. When N_{entro} is used, both our method and BCLS achieve $N_{\text{entro}} = 1$ in most cases. However, BCLS requires significantly more time than Teb-means and results in much higher clustering loss (see Figure 8).

Runtime Performance. We show that Teb-means is a time-efficient algorithm compared with SOTAs. Notably, our method is slower than Lloyd and CDKM, as it explicitly accounts for balance loss during execution. This introduces additional computations not reflected in the asymptotic complexity (since it remains proportional to nd_mkT). A theoretical summary is provided in Table 3, and the detailed discussion is presented below.

Observations: As shown in Table 7 shows that Teb-means achieves a shorter runtime than BCLS and F^{3}KM. It runs longer than FCFC. Nevertheless, FCFC exhibits the poorest balance, as indicated by its highest CV and lowest N_{entro}, making it the least balanced method.

Robustness. Teb-means and other balanced algorithms are heuristic methods, which are non-deterministic concerning a specific performance criterion and can be highly sensitive to initialization. Here, we calculate the mean, denoted as Obj_Mean, and the standard deviation, denoted as Obj_Std, of the cluster loss and balance loss based on 100 random seeds (different initializations) to assess the robustness of the algorithm. To reduce time cost, we limit clustering to a sample of 10,000 points from each dataset.

Observations: In Table 8, Teb-means achieves the lowest Obj_Mean and Obj_Std for clustering loss in most cases. However, in a few instances, it performs slightly worse than FCFC. Nevertheless, FCFC has poor balance performance, which is comparable to that of Lloyd. For balance loss, Teb-means achieves the lowest Obj_Mean and Obj_Std. The low variance and mean in both clustering and balance loss indicate that Teb-means is robust to initialization.

Verification for ANN. We demonstrate that Teb-means improves the ANN efficiency of clustering-based indexes. Following the clustering approach in Section 6.1, data points in the same cluster are stored as posting lists on disk. Upon receiving a query, the central

Table 8: Comparison of clustering loss and balance loss under different initializations.

Dataset	Method	Clustering Loss		Balance Loss	
		Obj_Mean	Obj_Std	Obj_Mean	Obj_Std
MTG	FCFC	1.91E+04	6.00E+02	6.00E+06	1.37E+06
	BCLS	2.05E+05	2.09E+02	6.72E+04	2.11E+04
	F^3KM	4.07E+04	1.35E+03	3.10E+05	3.10E+05
	Teb-means	4.22E+04	1.20E+03	8.50E+03	2.37E+03
Census	FCFC	5.06E+04	2.34E+03	2.26E+07	1.13E+06
	BCLS	2.86E+05	4.11E+01	8.62E+04	2.71E+04
	F^3KM	7.70E+04	2.83E+03	2.18E+05	3.45E+04
	Teb-means	8.70E+04	9.60E+02	8.70E+04	6.91E+02
Game	FCFC	1.90E+04	4.79E+03	1.35E+08	5.03E+07
	BCLS	3.49E+04	6.23E+01	2.09E+05	6.56E+04
	F^3KM	7.97E+04	5.40E+02	5.33E+06	2.07E+06
	Teb-means	1.05E+05	2.11E+02	1.03E+04	2.77E+03
NYC	FCFC	2.43E+06	6.26E+04	2.11E+07	7.13E+06
	BCLS	6.57E+06	4.97E+03	2.11E+05	6.64E+04
	F^3KM	3.00E+06	6.93E+05	5.15E+07	7.40E+07
	Teb-means	2.55E+06	1.27E+05	8.27E+06	2.59E+06
Crime	FCFC	2.01E+04	1.14E+02	2.29E+05	1.00E+05
	BCLS	4.50E+04	2.94E+01	1.11E+04	5.75E+03
	F^3KM	2.07E+04	5.64E+01	2.14E+02	1.05E+01
	Teb-means	2.10E+04	4.45E+01	1.98E+02	5.75E+01
Retail	FCFC	3.00E+04	7.59E+02	2.49E+06	2.44E+04
	BCLS	4.85E+04	3.47E+01	1.13E+03	1.58E+02
	F^3KM	3.59E+04	2.50E+03	2.00E+05	3.88E+04
	Teb-means	3.70E+04	3.06E+02	5.59E+04	1.81E+04
Amazon	FCFC	3.33E+04	1.32E+03	4.07E+06	4.88E+05
	BCLS	8.33E+04	3.50E+01	5.17E+03	1.12E+03
	F^3KM	3.90E+04	2.44E+03	2.29E+05	1.14E+05
	Teb-means	3.94E+04	3.95E+02	3.04E+03	1.16E+03
MovieLens	FCFC	1.36E+07	5.42E+04	3.58E+06	2.63E+05
	BCLS	1.60E+07	4.34E+03	1.14E+04	6.20E+03
	F^3KM	1.35E+07	8.54E+04	1.44E+06	3.54E+05
	Teb-means	1.34E+07	3.58E+04	1.02E+06	6.41E+04

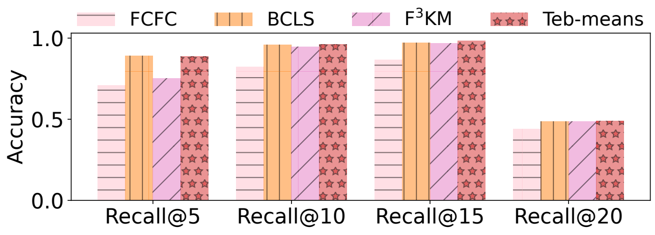


Figure 10: ANN search accuracy of federated clustering-based indexes: average over 8 datasets.

server identifies the nearest centroid(s). Each party then retrieves as much relevant data as possible from the corresponding posting list of the identified centroid(s) and transmits it back to the server with distance information. Finally, the central server computes the distances and identifies the nearest point. Notably, we generate 1,000 random ANN queries and compute recall@5, @10, and @15 under a 1 ms query latency constraint.

Observations: As shown in Figure 10, the clustering-based index built by Teb-means achieves higher recall than other balanced clustering algorithms in most cases, indicating superior query performance. Among them, FCFC shows the poorest performance, primarily due to its poor clustering balance, which results in significant variation in cluster sizes. Small clusters may be fully searched within the given time, limiting further opportunities to retrieve additional data points, while large clusters cannot be fully searched in time.

Table 9: Clustering accuracy assessment.

Dataset	Metrics	Lloyd	CDKM	BCLS	FCFC	F^3KM	Teb-means
Game	ARI	0.08	0.09	0.00	0.08	0.04	**0.21**
	AMI	0.06	0.07	0.00	0.06	0.04	**0.21**

Accuracy Measurement. We also use Adjusted Rand Index (ARI) [40] and Adjusted Mutual Information (AMI) [39] to measure clustering accuracy; both favor higher values for better performance. Among our selected datasets, Game includes class labels. To align with the balanced design of our method, we modify Game to ensure an equal number of samples for each class label.

Observations: As shown in Table 9, Teb-means achieves highly ARI and AMI scores, showing superior accuracy. BCLS can result in a value of zero. This occurs because k-means is an unsupervised algorithm that does not utilize label information. Additionally, using Euclidean distance for clustering may not always effectively differentiate between labels, as distance alone is not a sufficient criterion. As a result, even high-quality balanced clustering does not necessarily yield higher ARI or AMI scores.

7 CONCLUSIONS & FUTURE WORK

In this paper, we proposed Teb-means to efficiently perform balanced clustering in a VFL setting. We first introduced a new loss function that incorporated balance loss into the k-means objective and used CO to iteratively assign data points to clusters. We then proposed GBCO to improve the efficiency of CO by dividing the indicator matrix into multiple blocks and selecting the block that reduced the loss function the most for updating. Next, we designed a VFL framework for GBCO, where only intermediate results were exchanged. Teb-means demonstrated that its communication round was a constant in the mild condition, and the time complexity on clients was linear lower than most of existing balanced k-means algorithms. Our experiments show the superiority of Teb-means in runtime, cluster balance, and robustness to initialization.

In future work, we will explore extending Teb-means to support clustering of multimodal data while maintaining soft balance and preserving the cluster structure. We also plan to investigate adapting Teb-means for efficient execution on emerging hardware, such as TPUs, to further accelerate clustering.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2023YFB4503600), National Natural Science Foundation of China (62202338 and 62372337), and the Key R&D Program of Hubei Province (2023BAB081).

REFERENCES

[1] 2025. Amazon Product Reviews. https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews.

[2] 2025. Gaming Profiles 2025 (Steam, PlayStation, Xbox). https://www.kaggle.com/datasets/artyomkruglov/gaming-profiles-2025-steam-playstation-xbox.

[3] 2025. Global Fashion Retail Sales. https://www.kaggle.com/datasets/ricgomes/global-fashion-retail-stores-dataset/data?select=customers.csv.

[4] 2025. GroupLens Research collects ratings data from the MovieLens website. https://grouplens.org/datasets/movielens/.

[5] 2025. HMDA Data Browser allows you to filter, aggregate, download, and visualize HMDA datasets. https://ffiec.cfpb.gov/data-browser/.

[6] 2025. NYC Open Data. https://opendata.cityofnewyork.us/.

[7] 2025. Repository of Teb-means. https://github.com/whu-totemdb/Teb-means.

[8] 2025. UCI Credit Card. https://archive.ics.uci.edu/dataset/183/communities+and+crime.

[9] 2025. US Census Data. https://archive.ics.uci.edu/dataset/116/us+census+data+1990.

[10] Tim Althoff, Adrian Ulges, and Andreas Dengel. 2011. Balanced Clustering for Content-based Image Browsing. In *Informatiktage*, Vol. S-10. 27–30.

[11] Paul S Bradley, Kristin P Bennett, and Ayhan Demiriz. 2000. Constrained k-means clustering. *Microsoft Research, Redmond* 20 (2000).

[12] Fuyuan Cao, Xuechun Jing, Kui Yu, and Jiye Liang. 2025. FWCEC: An Enhanced Feature Weighting Method via Causal Effect for Clustering. *IEEE Trans. Knowl. Data Eng.* 37, 2 (2025), 685–697.

[13] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighborhood Search. In *NeurIPS*. 5199–5212.

[14] Yixin Chen, Ya Zhang, and Xiang Ji. 2005. Size Regularized Cut for Data Clustering. In *NIPS*. 211–218.

[15] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75.

[16] Hu Ding, Yu Liu, Lingxiao Huang, and Jian Li. 2016. K-Means Clustering with Distributed Dimensions. In *ICML*, Vol. 48. 1339–1348.

[17] Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, and Dongmei Zhang. 2015. YADING: fast clustering of large-scale time series data. *Proc. VLDB Endow.* 8, 5 (2015), 473–484.

[18] Jonathan Drake. 2013. Faster k-means Clustering. In *MS Thesis*.

[19] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *ICML*. 147–153.

[20] Dan Feldman, Melanie Schmidt, and Christian Sohler. 2020. Turning Big Data Into Tiny Data: Constant-Size Coresets for k-Means, PCA, and Projective Clustering. *SIAM J. Comput.* 49, 3 (2020), 601–657.

[21] Manuel Fritz, Michael Behringer, and Holger Schwarz. 2020. LOG-Means: Efficiently Estimating the Number of Clusters in Large Datasets. *Proc. VLDB Endow.* 13, 11 (2020), 2118–2131.

[22] Fangcheng Fu, Huanran Xue, Yong Cheng, Yangyu Tao, and Bin Cui. 2022. BlindFL: Vertical Federated Machine Learning without Peeking into Your Data. In *SIGMOD*. 1316–1330.

[23] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In *WWW*. 3406–3416.

[24] Lingxiao Huang, Zhize Li, Jialin Sun, and Haoyu Zhao. 2022. Coresets for Vertical Federated Learning: Regularized Linear Regression and K-Means Clustering. In *NeurIPS*.

[25] Yushuai Ji, Zepeng Liu, Sheng Wang, Yuan Sun, and Zhiyong Peng. 2025. On Simplifying Large-Scale Spatial Vectors: Fast, Memory-Efficient, and Cost-Predictable k-means. In *ICDE*. 863–876.

[26] Linshan Jiang, Moming Duan, Bingsheng He, Yulin Sun, Peishen Yan, Yang Hua, and Tao Song. 2024. OFL-W3: A One-shot Federated Learning System on Web 3.0. *Proc. VLDB Endow.* 17, 12 (2024), 4461–4464.

[27] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurelien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adria Gascon, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konevcny, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrede Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Ozgur, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramer, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* 14, 1-2 (2021), 1–210.

[28] He Li, Lu Yu, and Wu He. 2019. The impact of GDPR on global technology development. , 6 pages.

[29] Kun Li, Liang Yuan, Yunquan Zhang, and Gongwei Chen. 2022. An Accurate and Efficient Large-Scale Regression Method Through Best Friend Clustering. *IEEE Trans. Parallel Distributed Syst.* 33, 11 (2022), 3129–3140.

[30] Zitao Li, Tianhao Wang, and Ninghui Li. 2023. Differentially Private Vertical Federated Clustering. *Proc. VLDB Endow.* 16, 6 (2023), 1277–1290.

[31] Hanyang Liu, Junwei Han, Feiping Nie, and Xuelong Li. 2017. Balanced Clustering with Least Square Regression. In *AAAI*. 2231–2237.

[32] Hongfu Liu, Ziming Huang, Qi Chen, Mingqin Li, Yun Fu, and Lintao Zhang. 2018. Fast Clustering with Flexible Balance Constraints. In *IEEE BigData 2018*. 743–750.

[33] Yang Liu, Xinwei Zhang, Yan Kang, Liping Li, Tianjian Chen, Mingyi Hong, and Qiang Yang. 2022. FedBCD: A Communication-Efficient Collaborative Learning Framework for Distributed Features. *IEEE Trans. Signal Process.* 70 (2022), 4277–4290.

[34] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

[35] Mikko I. Malinen and Pasi Franti. 2014. Balanced K-Means for Clustering. In *S+SSPR*, Vol. 8621. Springer, 32–41.

[36] Kasper Overgaard Mortensen, Fatemeh Zardbani, Mohammad Ahsanul Haque, Steinn Ymir Agustsson, Davide Mottin, Philip Hofmann, and Panagiotis Karras. 2023. Marigold: Efficient k-means Clustering in High Dimensions. *Proc. VLDB Endow.* 16, 7 (2023), 1740–1748.

[37] Feiping Nie, Ziheng Li, Rong Wang, and Xuelong Li. 2023. An Effective and Efficient Algorithm for K-Means Clustering With New Formulation. *IEEE Trans. Knowl. Data Eng.* 35, 4 (2023), 3433–3443.

[38] Feiping Nie, Jingjing Xue, Danyang Wu, Rong Wang, Hui Li, and Xuelong Li. 2022. Coordinate Descent Method for k-means. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 5 (2022), 2371–2385.

[39] Simone Romano, James Bailey, Xuan Vinh Nguyen, and Karin Verspoor. 2014. Standardized Mutual Information for Clustering Comparisons: One Step Further in Adjustment for Chance. In *ICML*, Vol. 32. 1143–1151.

[40] Jorge M. Santos and Mark J. Embrechts. 2009. On the Use of the Adjusted Rand Index as a Metric for Evaluating Supervised Classification. In *ICANN*, Vol. 5769. 175–184.

[41] Yitong Song, Bin Yao, Zhida Chen, Xin Yang, Jiong Xie, Feifei Li, and Mengshi Chen. 2025. Efficient top-k spatial-range-constrained approximate nearest neighbor search on geo-tagged high-dimensional vectors. *VLDB J.* 34, 1 (2025), 14.

[42] Yongye Su, Yinqi Sun, Minjia Zhang, and Jianguo Wang. 2024. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *Proc. ACM Manag. Data* 2, 3 (2024), 187.

[43] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. 2022. A Survey on Trajectory Data Management, Analytics, and Learning. *ACM Comput. Surv.* 54, 2 (2022), 39:1–39:36.

[44] Sheng Wang, Yuan Sun, and Zhifeng Bao. 2020. On the Efficiency of K-Means Clustering: Evaluation, Optimization, and Algorithm Selection. *Proc. VLDB Endow.* 14, 2 (2020), 163–175.

[45] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, Peng Cheng, and Mao Yang. 2023. SPFresh: Incremental In-Place Update for Billion-Scale Vector Search. In *SOSP*. 545–561.

[46] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10, 2 (2019), 12:1–12:19.

[47] Shuyuan Zheng, Yang Cao, and Masatoshi Yoshikawa. 2023. Secure Shapley Value for Cross-Silo Federated Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1657–1670.

[48] Peng Zhou, Jiangyong Chen, Liang Du, and Xuejun Li. 2023. Balanced Spectral Feature Selection. *IEEE Trans. Cybern.* 53, 7 (2023), 4232–4244.

[49] Weiwei Zhou and Bin Yu. 2017. An Efficient Energy-Hole Alleviating Algorithm for Wireless Sensor Network Based on Energy-Balanced Clustering Protocol. In *CWSN*, Vol. 812. 103–116.

[50] Pengfei Zhu, Wangmeng Zuo, Lei Zhang, Qinghua Hu, and Simon C. K. Shiu. 2015. Unsupervised feature selection by regularized self-representation. *Pattern Recognit.* 48, 2 (2015), 438–446.

[51] Shengkun Zhu, Quanqing Xu, Jinshan Zeng, Sheng Wang, Yuan Sun, Zhifeng Yang, Chuanhui Yang, and Zhiyong Peng. 2023. F3KM: Federated, Fair, and Fast k-means. *Proc. ACM Manag. Data* 1, 4 (2023), 241:1–241:25.

[52] Zeqi Zhu, Zeheng Fan, Yuxiang Zeng, Yexuan Shi, Yi Xu, Mengmeng Zhou, and Jin Dong. 2024. FedSQ: A Secure System for Federated Vector Similarity Queries. *Proc. VLDB Endow.* 17, 12 (2024), 4441–4444.