Differentially-Private Multiparty Clustering

by

Abdelrahman Ahmed

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In an era marked by the widespread application of Machine Learning (ML) across diverse domains, the necessity of privacy-preserving techniques has become paramount. The Euclidean k-Means problem, a fundamental component of unsupervised learning, brings to light this privacy challenge, especially in federated contexts. Existing Federated approaches utilizing Secure Multiparty Computation (SMPC) or Homomorphic Encryption (HE) techniques, although promising, suffer from substantial overheads and do not offer output privacy. At the same time, differentially private k-Means algorithms fall short in federated settings. Recognizing the critical need for innovative solutions safeguarding privacy, this work pioneers integrating Differential Privacy (DP) into federated k-Means. The key contributions of this dissertation include the novel integration of DP in Horizontally-Federated k-Means, a lightweight aggregation protocol offering three orders of magnitude speedup over other multiparty approaches, the application of cluster-size constraints in DP k-Means to enhance state-of-the-art utility, and a meticulous examination of various aggregation methods in the protocol. Unlike traditional privacy-preserving approaches, our innovative design results in a faster, more private, and more accurate solution, significantly advancing the state-of-the-art in privacy-preserving machine learning.

Acknowledgements

I would love to thank Florian Kerschbaum not just for being a resourceful, supportive, and insightful supervisor but also for setting an example of how a leader should be. He taught me how to be diligent and patient, compassionate but also encouraging towards progress. His management style and personality are elements that all of his students admire, and we are all very grateful for.

This personality transferred exceptionally well to his students; seniors help their juniors in their struggles in research and non-research settings. I would love to specifically thank Thomas Humphries for being a mentor in times of stress and offering unconditional help and insight.

This environment also expanded to all CrySP lab members, where I made close friendships that transcend the workspace with the most wonderful of people. Thank you for making this experience worthwhile and enjoyable: Norhan, Lucas, John, Shreya, Rasoul, Jason, Simon, Yunji, and everyone else.

Dedication

To my parents, whose love transcends all science. To my sisters, to whom I wish the highest levels of success. To my friends, who never left my side. To you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the prevalent era of Machine Learning (ML) with diverse applications such as recommendation systems, fraud detection, and healthcare analytics, the center-based clustering problem, particularly the Euclidean k-Means problem, forms an essential part of unsupervised learning. The usage of this approach with various sensitive data necessitates the importance of privacy-preserving techniques, which goes beyond mere protection of sensitive information. The handling of data from multiple entities, especially in federated scenarios, brings forth profound ethical, legal, and technological challenges, thus demanding innovative solutions to ensure individual privacy without compromising the clustering quality.

Exact privacy-preserving approaches that employ Secure Multiparty Computation or Homomorphic Encryption, whether serverless [1, 2, 3, 4, 5, 6, 7, 8] or outsourced [9, 10, 11, 12, 13, 14], encounter significant problems: (1) Even though they mitigate leakage during computation, information can still be exposed through the output which has no protection against attacks [15]; (2) Substantial computational and communication overheads arise, making them unsuitable for large-scale deployment. Additionally, having a high overhead makes naive application of differential privacy to existing methods, still unsuitable.

Differential Privacy (DP) has emerged as an accepted approach to offer robust privacy guarantees. Considerable research has been conducted on developing differentially private k-Means algorithms [16, 17, 18, 19, 20, 21, 22, 23, 24]. However, existing methods do not address federated situations where data ownership is distributed among various entities without a trusted central aggregator.

In response, we present the first examination of incorporating DP into federated k-Means. Instead of naively combining the state-of-the-art in DP [18] and Federated k-

Means [8, 14], our work provides enhancements to both components, resulting in a design that is faster, more private, and more accurate than previous work.

Firstly, since we provide DP assurances, we can relax the need to conceal intermediate centroids from participating clients. We thus devise an aggregation protocol that is more efficient than previous works: masked secure aggregation, a variant of secure aggregation [25] that employs a semi-honest server to aggregate values across all clients without observing the outcome. Secondly, we significantly improve the utility of Su et al.'s solution [18] by considering variations of the constrained k-Means problem [26], where cluster sizes are bounded, which also of independent interest in the centralized differential privacy model. Thirdly, we analyze different aggregation methods, including (1) employing distinct variables for cluster sums and counts and (2) a stricter option where only partial centroids are aggregated among clients in scenarios where even sharing a differentially-private count for clusters is not applicable. In summary, our contributions encompass:

1. The initiation of DP in Horizontally-Federated k-Means.
2. The introduction of a lightweight aggregation protocol, offering three orders of magnitude speedup over other federated k-Means solutions.
3. The application of cluster-size constraints in DP k-Means, enhancing utility relative to the previous state-of-the-art in the central model.
4. A thorough analysis and evaluation of various aggregation methods for the protocol, along with an investigation of their respective instantiations and hyperparameters.

Chapter 2

Preliminaries

2.1 k-Means

The k-Means problem is a discrete optimization problem that seeks to partition a set of N d-dimensional observations into k clusters, each represented by its mean or centroid. The objective function, the within-cluster sum of squares (WCSS), is to be minimized. Mathematically, given a set of observations $(x_1, x_2, ..., x_N)$, where each observation is a d-dimensional real vector, the k-Means problem is to find an assignment of data points to clusters, and a set of cluster centers, that minimizes the WCSS objective:

$$\underset{O,\mu}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x_j \in O_i} ||x_j - \mu_i||^2 \tag{2.1}$$

where $O = \{O_1, O_2, ..., O_k\}$ are the clusters, O_i contains the points in the i-th cluster, $\mu = \{\mu_1, \mu_2, ..., \mu_k\}$ are the centroids of the clusters, and μ_i is the (arithmetic) mean of points in O_i.

This problem is NP-hard in general Euclidean space and spaces of high dimensionality [27]. However, a simple heuristic known as Lloyd's algorithm, first introduced in [28], is commonly applied for practical applications to find a local minimum of the objective function, even though it does not guarantee to reach a global minimum.

2.1.1 Lloyd's Algorithm

Lloyd's algorithm, commonly called the k-Means algorithm, is a heuristic iterative method used to solve the k-Means problem. Despite not guaranteeing a global optimum, it often provides satisfactory results in practical applications. The most basic implementation of it operates as follows:

1. Initialization step: Randomly sample k centroids $\{\mu_1, \mu_2, ..., \mu_k\}$ from the datapoints.
2. Repeat until convergence or a predetermined number of iterations has been reached:
   (a) Assignment step: Assign each observation x_j to the nearest centroid, using the Euclidean distance. This creates clusters O_i for $i = 1, 2, ..., k$. Formally, the assignment is:
   $$O_i^{(t)} = \left\{x_j : ||x_j - \mu_i^{(t)}||^2 \leq ||x_j - \mu_c^{(t)}||^2 \ \forall c, 1 \leq c \leq k\right\}$$
   (b) Update step: Calculate the new centroids to be the mean of the observations in the cluster:
   $$\mu_i^{(t+1)} = \frac{\sum_{x_j \in O_i^{(t)}} x_j}{|O_i^{(t)}|}$$

The algorithm converges when the assignments no longer change. However, Lloyd's algorithm might converge to a local minimum of the WCSS because of its dependence on the initial centroids. Hence, it is often run multiple times with different initializations, and the solution with the smallest WCSS is selected.

While the k-Means algorithm is widely used for its simplicity and efficiency, it possesses several limitations. First, it is sensitive to outliers. Since the algorithm minimizes the squared distance of data points in a cluster, a single outlier can significantly skew the centroid and hence the cluster distribution. Second, k-Means may generate clusters of widely varying sizes, including very small or even singleton clusters. This can be particularly problematic when there is an underlying expectation or requirement for clusters of comparable sizes. An approach to mitigate these issues is to impose a minimum cluster size constraint during the assignment step, which we explore in this dissertation.

2.1.2 Convergence Criteria

The convergence of the k-Means algorithm is usually determined when the cluster assignments do not change. However, there are other stopping criteria, each with advantages and limitations. These criteria are typically used in combination with one another to guide the stopping of the algorithm.

- **Change in Cluster Assignments**: This is the most common convergence criterion used in practice, as described in subsection 2.1.1. Mathematically, this criterion is expressed as:

$$O^{(t)} = O^{(t-1)} \tag{2.2}$$

  Where $O^{(t)}$ and $O^{(t-1)}$ represent the set of clusters at the current and previous iteration, respectively. The algorithm stops when there are no changes in the assignment of points to clusters. While this criterion is straightforward and intuitive, it does not guarantee that the algorithm has found a minimum (even a local one) of the WCSS objective. It just guarantees that the, under the current initial conditions, Lloyd's algorithm will not improve further.

- **Maximal Iterations**: As a safeguard against excessive computational time or any other costs incurred due to running a high number of iterations, a pre-specified number of maximal iterations, denoted by t_{max}, can be set. This is particularly useful when the dataset is large, or the dimensionality is high, as the algorithm could continue for a substantial amount of time without satisfying the above criterion. The algorithm stops if $t > t_{max}$.

- **Centroid Shift**: This criterion assesses the amount of movement of the centroids. The algorithm stops when the centroid's sum (or maximum) shifts below a specified threshold. Let δ represent this threshold; then the criterion can be formulated as:

$$\max_{1 \leq i \leq k} ||\mu_i^{(t)} - \mu_i^{(t-1)}||^2 < \delta \tag{2.3}$$

  This criterion essentially measures the stability of the centroids.

- **Change in WCSS**: Another common stopping criterion is monitoring the WCSS change. This criterion checks if the decrease in the objective function from one iteration to the next is less than a predetermined small value, δ. If it is, the algorithm

stops, indicating that further iterations will not significantly improve the partitioning. Formally, this criterion can be stated as follows:

$$\left| \sum_{i=1}^{k} \sum_{x_j \in O_i^{(t)}} ||x_j - \mu_i^{(t)}||^2 - \sum_{i=1}^{k} \sum_{x_j \in O_i^{(t-1)}} ||x_j - \mu_i^{(t-1)}||^2 \right| < \delta \tag{2.4}$$

This condition provides more direct control over the optimization process by closely monitoring the objective function.

The choice of convergence criteria depends largely on the specific problem at hand, the size and complexity of the dataset, and the computational resources available. The above convergence criteria collectively offer a comprehensive way to control the termination of the k-Means algorithm. In this thesis, we are interested in multiparty privacy-preserving settings, which limit our choices to the maximal iterations or centroid shifts criteria most of the time because other approaches are evaluated on private data. However, we explore different evaluation metrics to assess the quality of the produced clustering.

2.1.3 Evaluation Metrics

The evaluation of the k-Means algorithm's output is crucial to understand the quality and usefulness of the formed clusters. Several metrics exist to evaluate the results of the k-Means algorithm, which can be broadly divided into unsupervised and supervised metrics.

Unsupervised Metrics: Unsupervised metrics evaluate the cluster quality without reference to ground truth. These are particularly useful when labeled data is not present. Key unsupervised metrics include:

- **Within-Cluster Sum of Squares (WCSS):** As mentioned before, it is the sum of squared distances of all points in a cluster to their respective centroid. It is mathematically expressed as:

$$WCSS = \sum_{i=1}^{k} \sum_{x_j \in O_i} ||x_j - \mu_i||^2 \tag{2.5}$$

- **Between-Cluster Sum of Squares (BCSS):** This metric measures the variability between clusters. It is the sum of the squared distances of each cluster centroid to the overall data mean. It is represented as:

$$BCSS = \sum_{i=1}^{k} |O_i| ||\mu_i - \mu_G||^2 \tag{2.6}$$

  where μ_G is the global mean of all data points, and $|O_i|$ denotes the number of data points in the i-th cluster.

- **Silhouette Score [29]:** This score measures how similar a point is to its cluster compared to others. The silhouette score for the j-th point, $\mathbf{Sil}(x_j)$, is calculated as:

$$\mathbf{Sil}(x_j) = \frac{b(x_j) - a(x_j)}{\max\{a(x_j), b(x_j)\}} \tag{2.7}$$

  Where $a(x_j)$ is the average distance from the j-th point to the other points in the same cluster, and $b(x_j)$ is the minimum average distance from the j-th point to points in a different cluster, minimized over clusters. The silhouette score for the clustering solution is the average silhouette score overall points.

Other notable unsupervised metrics include the Dunn index [30] and the Davies-Bouldin index [31].

Supervised Metrics: Supervised metrics require a ground truth - a known set of true cluster assignments. These are particularly useful when data is labeled and one wants to compare the output of our algorithm with the true labels. Key supervised metrics include:

- **Adjusted Rand Index (ARI) [32]:** This index measures the similarity between the true and predicted assignments, correcting for chance. An ARI score of 1 indicates that the predicted assignments perfectly match the true assignments.

- **Normalized Mutual Information (NMI) [33]:** NMI is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation) and is suitable for comparing clusters with ground truth.

- **Fowlkes-Mallows Index (FMI) [34]:** The FMI is the geometric mean of the precision and recall and is interpreted similarly to the F1-score commonly used in binary classification settings. An FMI of 1 indicates perfect precision and recall.

These unsupervised and supervised evaluation metrics provide a comprehensive understanding of the performance of the k-Means algorithm. However, it is essential to keep in mind that the best evaluation metric depends on the specific application and requirements of the clustering task.

2.1.4 Algorithmic Complexity

The complexity of the k-Means algorithm is categorized into time and space complexity, both vital for evaluating its adaptability to voluminous datasets or high-dimensional scenarios.

Time Complexity: The time complexity of the k-Means algorithm is governed by the number of data points (N), clusters (k), dimensions (d), and iterations (t). Each iteration necessitates the computation of Euclidean distances from every data point to every centroid, an $\mathcal{O}(Nkd)$ operation. Consequently, the worst-case time complexity is $\mathcal{O}(tNkd)$. Although k-Means typically converges quickly in practice, worst-case scenarios may lead to exponential convergence time [35] due to adverse initialization or data configurations.

Space Complexity: The space complexity primarily depends on the data points (N) and dimensionality (d), resulting in $\mathcal{O}(Nd)$ complexity. Including the cluster assignments would not alter the order, retaining $\mathcal{O}(Nd)$ complexity.

These complexities highlight the limitations of the k-Means algorithm with large-scale, high-dimensional data. Adaptations like dimensionality reduction, KD-Trees for efficient distance computations [36], or scalable variants like mini-batch k-Means [37] can augment the algorithm's practical applicability.

2.2 Masked Secure Aggregation

Building on the concept of Secure Aggregation [25], we introduce Masked Secure Aggregation (MSA). In MSA, like all secure aggregation protocols, a service provider P_S securely aggregates numbers from M clients, all while oblivious to every client's contributions. However, in MSA, the server is also oblivious to the result of the aggregation operation; the server only acts as an aggregator and is oblivious to both the input and output.

We assume a multiparty setup consisting of M clients in this privacy-preserving mechanism. Each client P_l possesses a private value $v_l \in \mathbf{R}$ where $\mathbf{R}$ is $\mathbb{Z}_{32}$ or $\mathbb{Z}_{64}$. These clients aim to collectively compute the sum of their private values: $v = \sum_{l=1}^{M} v_l$ while preventing any individual client from gaining knowledge about the private values of others. We assume that the number of clients involved (M) is more than two because, knowing the sum and their value, a client can trivially compute the other's value.

2.2.1 Method

To facilitate the computation, the clients share a common seed and a Pseudo-Random Number Generator (PRNG), critical in generating common pseudo-random masks that assist in the protocol. The PRNG takes a uniformly random seed of some fixed length and has a uniform output distribution over $\mathbf{R}$.

It is pivotal that the server, assumed to be semi-honest, can aid in computation without gaining any insight into the private values. Similarly, it is critical to design the mechanism so that no client can infer the private values (v_l) of others. Our proposed solution is outlined as follows:

1. **Random Mask Generation:** Using the shared seed and the same PRNG, all clients (except the server) generate the same set of M random numbers: $\{r_1, r_2, \ldots r_M\}$. Every client, then, computes the sum $r \equiv \sum_{l=1}^{M} r_l$, which would be used as the decryption (unmasking) key.

2. **Data Masking:** Each client P_l sends $Enc_l(v_l) \equiv r_l + v_l$ to the service provider P_S. This step masks the actual value v_l from the service provider because r_l acts as a one-time pad. Note that this masked value is never sent to other clients, who would be able to unmask it very easily.

3. **Sum Calculation:** The service provider sums the received values to reveal the masked sum: $sum(v) \equiv \sum_{l=1}^{M} Enc_l(v_l) \equiv r + \sum_{l=1}^{M} v_l$ and is ready to send it back to all clients. The server cannot interpret this value as r acts as a one-time pad.

4. **(Optional) Plain Addition:** The service provider adds a value to the sum before returning it to the clients: $Enc(v) \equiv r + b + \sum_{l=1}^{M} v_l$.

5. **Data Unmasking:** Each client unmasks the received sum using r to retrieve the sum of the actual values: $Dec(Enc(v)) \equiv b + \sum_{l=1}^{M} v_l$.

This solution enhances privacy by ensuring that the P_S only interacts with masked values, and individual clients cannot deduce other clients' values from the calculated sum. The idea is illustrated in Table 2.1.

Step	**Clients** $l \in \{1, \ldots, M\}$	**Server** P_S
1	- Sample $\{r_1, r_2, \ldots r_M\}$ - $r \equiv \sum_{l=1}^{M} r_l$ - Send $Enc_l(v_l) \equiv r_l + v_l$ to P_S	
2		- $Sum(v) \equiv \sum_{l=1}^{M} Enc_l(v_l) \equiv v + r$ - Add own value: $Enc(v) \equiv (v + b) + r$ - Send $Enc(v)$ to all clients
3	- Unmask $v + b = Enc(v) - r$	

Table 2.1: Masked Secure Aggregation Protocol with M Clients.

2.2.2 Numerical Considerations

We need to encode the private values in the ring $\mathbf{R}$ to enable working with real-valued data. This can be achieved using fixed-point number representation, where a fixed number of digits after the decimal point represent fractional parts. Given a real-valued private value v_l, we convert it to a fixed-point representation by scaling it up by a power of two. This operation facilitates arithmetic operations in the integer domain while preserving the fractional information. The following steps outline the modifications required in the secret-sharing protocol:

1. **Data Conversion:** Each private value v_l is converted to a fixed-point format, $\tilde{v}_l$, by scaling it up with a power-of-2 scale factor, $\mathbf{\Delta}$. Formally, $\tilde{v}_l = \lceil v_l \times \mathbf{\Delta} \rfloor$. This ensures that the value is now an integer, which fits into the 32-bit (or 64-bit) integer ring used for computations.

2. **Data Unmasking:** Each client unmasks the received sum using r and then scales it down to retrieve the sum of the actual values. This is done by dividing the unmasked sum by the scale factor, $\mathbf{\Delta}$, to reverse the initial scaling operation. Formally, $\sum_{l=1}^{M} v_l \approx \frac{\sum_{l=1}^{M} \tilde{v}_l - r}{\mathbf{\Delta}}$. Where $\approx$ is used to indicate the loss in precision due to quantization.

All fractional values are rounded to the nearest representable value in this fixed-point representation. The choice of the scale factor, $\mathbf{\Delta}$, determines the precision of the representation. A larger $\mathbf{\Delta}$ allows for greater precision but also reduces the number of bits available for the integer part of the number, which might cause overflows. A workaround is to increase the bitwidth of the operations, which increases the computational load and the communication cost.

2.3 Differential Privacy

Differential Privacy (DP) offers a rigorous framework to protect individual records in statistical databases. It focuses on publishing aggregate information from a private dataset while maintaining individual privacy. This privacy guarantee is achieved by bounding the impact of modifying a single record on any query's outcome. An algorithm (a mechanism) adheres to differential privacy if it obfuscates every individual detail in its output to some bounded degree. The essence of DP is captured by the bounded probability, which limits the capability of an adversary to infer specific details about any individual record.

Definition 2.3.1 (Statistical Database)**.** A statistical database is a structured collection of data to compute aggregate statistics. It ensures that derived statistics do not compromise individual privacy.

Definition 2.3.2 (Dataset space)**.** The dataset space $\mathcal{X}$ is a set of all possible datasets. A single dataset $X \in \mathcal{X}$ is a list of data entries.

Definition 2.3.3 (Output space)**.** The output space $\mathcal{R}$ is the set of all possible outputs a mechanism can produce.

Definition 2.3.4 (Neighboring Datasets)**.** Two datasets $X, Y \in \mathcal{X}$ are called neighboring datasets (adjacent datasets), denoted as $X \sim Y$ if they differ by exactly one data entry.

In the context of this paper, we employ a specific variant of differential privacy, referred to as *unbounded differential privacy*. Particularly, our model of neighboring datasets corresponds to the scenario in which one dataset can be derived from the other by adding or removing exactly one datapoint as opposed to *bounded differential privacy*, where modifying/swapping a datapoint is allowed.

Given two datasets $X, Y \in \mathcal{X}$, they are considered neighboring datasets (in an unbounded sense), denoted as $X \sim Y$, if X can be obtained from Y by the removal of exactly

one datapoint, or equivalently, Y can be obtained from X by adding exactly one data point. To put it formally, we say that $X \sim Y$ if a datapoint d exists, such as $X = Y \cup \{d\}$ or $Y = X \cup \{d\}$. This implies that $Y = X \setminus \{d\}$ or $X = Y \setminus \{d\}$, respectively.

Unbounded differential privacy focuses on the privacy guarantees when data is removed from the dataset. It allows us to focus on the implications of the inclusion or exclusion of a specific data point, exploring the notion of "plausible deniability".

Definition 2.3.5 ((Randomized) Mechanism f)**.** A (randomized) mechanism f is a function from dataset space $\mathcal{X}$ to output space $\mathcal{R}$, considering that the mechanism is allowed to use randomness.

Sensitivity is a critical parameter in differential privacy. It quantifies the maximum change in the output of a function due to the alteration of a single data entry in its input. It is vital in noise addition mechanisms, where the variance of the added noise is proportional to the function's sensitivity. Two types are often considered: L_1 sensitivity and L_2 sensitivity. These measure the maximum change in the function's output according to the L_1 and L_2 norms, respectively.

Definition 2.3.6 (L1 Sensitivity)**.** The L1 sensitivity of a function $f : \mathcal{X} \to \mathbb{R}^d$ under the L1 norm is given by:

$$\Delta_1 f = \max_{X \sim Y} ||f(X) - f(Y)||_1,$$

where $X \sim Y$ denotes that X and Y are neighboring datasets, and $||\cdot||_1$ represents the L1 norm.

The L_1 sensitivity measures the maximum change in each dimension of the function's output and sums these values.

Definition 2.3.7 (L2 Sensitivity)**.** The L2 sensitivity of a function $f : \mathcal{X} \to \mathbb{R}^d$ under the L2 norm is given by:

$$\Delta_2 f = \max_{X \sim Y} ||f(X) - f(Y)||_2,$$

where $X \sim Y$ denotes that X and Y are neighboring datasets, and $||\cdot||_2$ represents the L2 norm.

The L_2 sensitivity measures the Euclidean distance between the function's output on two neighboring datasets. It is often used when we are concerned with the geometric distance between the multidimensional outputs.

In differentially private mechanisms such as the Laplace or Gaussian mechanisms, noise is added to the function's output to obfuscate the presence of individual data entries. The noise scale is generally proportional to the function's sensitivity, so a function with higher sensitivity will add more noise. As a result, the sensitivity directly impacts the level of privacy and utility of the mechanism: a high sensitivity requires more noise, which preserves privacy but decreases utility.

Differential Privacy (ϵ-DP) is a seminal notion that guarantees that the presence or absence of any individual in the database does not significantly change the likelihood of any specific outcome. The parameter ϵ controls the maximum allowable increase in the likelihood of any outcome due to the inclusion of an individual in the database.

Definition (ϵ-DP [38])**.** A (randomized) mechanism f is ϵ-differentially private (ϵ-DP) if for all adjacent datasets $X, Y \in \mathcal{X}$ and for all measurable $S \subseteq \mathcal{R}$, we have:

$$\Pr[f(X) \in S] \leq e^{\epsilon} \Pr[f(Y) \in S]$$

Understanding ϵ The parameter ϵ in ϵ-DP quantifies the degree of privacy protection, and it essentially bounds the amount by which the probability of a given output changes when a single record in the database is modified. Lower values of ϵ imply stronger privacy guarantees. However, this also increases the amount of noise added to the query result, thus reducing its accuracy or utility. This depicts the classic trade-off scenario in differential privacy between the privacy level and data utility.

The Laplace mechanism introduces noise to query outputs to ensure ϵ-DP.

Definition 2.3.8 (Laplace Mechanism [38])**.** The Laplace Mechanism is a method for introducing controlled statistical noise into data, thereby ensuring ϵ-DP. Given a function $f : \mathcal{X} \to \mathbb{R}^d$ with sensitivity Δf, the Laplace Mechanism is defined as $M(X) = f(X) + (Z_1, \ldots, Z_d)$ where Z_i are i.i.d. random variables drawn from the Laplace distribution. The Laplace distribution, denoted as $Lap(\lambda)$, is defined for $\lambda > 0$ and $x \in \mathbb{R}$ as:

$$Lap(x|\lambda, \mu) = \frac{1}{2\lambda} e^{-\frac{|x-\mu|}{\lambda}}$$

In the context of differential privacy, the scale parameter λ is set as $\Delta_1 f/\epsilon$, making the mechanism ϵ-differentially private. Thus, $Z_i \sim Lap(\Delta_1 f/\epsilon)$.

The composition theorems are fundamental properties of differential privacy that guide the application of differentially private mechanisms to more complex queries and algorithms.

Theorem 2.3.1 (Sequential Composition [39])**.** *If a sequence of k mechanisms $f_1, f_2, \ldots, f_k$ each provides ϵ-differential privacy, the sequence of these mechanisms provides $k\epsilon$-differential privacy.*

Sequential composition is essential in understanding the privacy implications of running multiple differentially private queries or mechanisms on the same dataset. Specifically, the theorem states that the privacy parameters of individual mechanisms in a sequence add up. If a mechanism with ϵ-DP is applied k times, the resulting mechanism is $k\epsilon$-DP. This property illustrates a "privacy budget" concept, where each application of a differentially private mechanism consumes a portion of the privacy budget, and the total budget used is the sum of the consumed portions.

Theorem 2.3.2 (Parallel Composition [39])**.** *If a mechanism f provides ϵ-differential privacy and is run independently on disjoint subsets of the database, then the overall mechanism also provides ϵ-differential privacy.*

Parallel composition is useful when analyzing different subsets of the data independently (like computing the centroids for a mutually-exclusive set of clusters). The key aspect here is that the subsets of data must be disjoint. According to this theorem, no matter how many disjoint subsets of the database the mechanism is applied to, the overall privacy guarantee remains the same. This is because the presence or absence of a single record affects only one of these disjoint subsets.

These two theorems, while simple, form the backbone for analyzing the privacy guarantees of more complex differentially private algorithms. They allow us to understand and measure how privacy guarantees evolve as we apply sequences of mechanisms or partition our dataset into disjoint subsets. The theorems also demonstrate the robustness and versatility of differential privacy, showing that differential privacy guarantees can be maintained under various data analysis scenarios.

The post-processing lemma is another vital property of differential privacy. The privacy guarantees of a differentially private mechanism are preserved under any post-processing, meaning that the results can be further manipulated, analyzed, or combined with other data without weakening the privacy protections.

Theorem 2.3.3 (Post-processing [38])**.** *Let f be an ϵ-differentially private mechanism, and let g be any function. Then the composition $g \circ f$ is also ϵ-differentially private.*

Differential privacy can be implemented using two main models: Central Differential Privacy (CDP) and Local Differential Privacy (LDP).

Definition 2.3.9 (Central Differential Privacy (CDP) [38])**.** Central Differential Privacy focuses on privacy protection on the data collector's side. A mechanism adheres to ϵ-central differential privacy if it satisfies the above definition of differential privacy.

Definition 2.3.10 (Local Differential Privacy (LDP) [40, 41])**.** A (randomized) mechanism f satisfies ϵ-local differential privacy (ϵ-LDP) if, for all possible data entries d and d' in the dataset, and for all measurable $S \subseteq \mathcal{R}$, we have:

$$\Pr[f(d) \in S] \leq e^{\epsilon} \Pr[f(d') \in S]$$

Local Differential Privacy (LDP) is a variant of differential privacy that shifts the responsibility for adding noise from a centralized curator to the individual data contributors. In this model, each data contributor perturbs their data locally before sending it to the aggregator, providing an additional layer of privacy. LDP is advantageous in scenarios where the trust in the central aggregator is limited, and it ensures that even the curator does not have access to exact individual data entries. However, this increased privacy often comes at the cost of higher noise and, consequently, reduced utility.

Chapter 3

Related Work

We outline the literature landscape in terms of DP k-Means and multiparty protocols for k-Means clustering, both in the presence of an assistant server or without a server.

3.1 Multiparty k-Means (Serverless)

In multiparty k-Means clustering, data are partitioned across several clients. There are two modes of partitioning: horizontal, where each client possesses a subset of records, and vertical, where each client has a set of features of the same records. Previous research on privacy-preserving clustering solutions in this setting falls broadly into two categories: methods revealing intermediate centroids, potentially breaching privacy, and those exposing only final centroids, incurring increased computational complexity.

Notable works in the first category include **Vaidya and Clifton** [1], who proposed privacy-preserving k-Means clustering for vertically partitioned databases. **Jha, Kruger, and McDaniel** [2] later shifted the focus to privacy-preserving k-Means clustering for horizontally partitioned databases. Yet, similar to the work of Vaidya and Clifton, their approach revealed intermediate candidate cluster centers.

Subsequent works by **Jagannathan and Wright** [3, 4] devised privacy-preserving k-Means clustering methods for databases partitioned horizontally and arbitrarily. Their latter method, named ReCluster, circumvents the disclosure of intermediate candidate cluster centers but reveals the pattern of cluster merging, providing a potential adversary with insights into which local clusters are likely to merge next.

Another contribution is by **Gheid et al.** [5], who proposed a scheme for horizontally partitioned databases that also exposes intermediate cluster centers but enhances the process's speed. The authors achieved this speedup by relaxing the secure centroid computation requirement into securely computing the sums and the counts separately, thus reducing the computation to a multiparty sum protocol, obviating the need for more computationally expensive division protocols.

In contrast, a different line of research has sought to reveal only final centroids. Pioneering work in this direction is by **Bunn and Ostrovsky** [6]. Their 2-party solution guarantees complete privacy in a semi-honest security model by revealing only the final cluster centers. However, the extensive use of homomorphic encryption, although preserving privacy, significantly inflates the computational costs. Similarly, **Jäschke et al.** [7] proposed a protocol heavily reliant on homomorphic encryption. While theoretically sound, the scheme suffers from scalability issues, limiting its applicability to large datasets.

Most recently, **Mohassel et al.** [8] introduced an efficient, secure squared Euclidean distance protocol and a custom garbled circuit for computing the minimum value. Thanks to these innovations, their protocol can handle significantly larger datasets than previous works. However, it still requires substantial communication overhead (in gigabytes) and has a slower runtime (in the order of minutes) than plaintext k-Means clustering algorithms.

In summary, the multiparty k-Means clustering field exhibits a dichotomy in approach, grappling with the trade-offs between privacy and computational efficiency. While early works reveal intermediate centroids and provide a relatively faster computation, this often compromises privacy. Conversely, modern protocols safeguarding privacy by revealing only the final centroids often involve considerable computational overhead. Recent advancements, such as those by **Mohassel et al.** [8], are pushing the boundaries in handling larger datasets. Yet, they still struggle with substantial communication overhead and slower runtimes. This landscape of existing works underscores the ongoing challenge of achieving an optimal balance between privacy preservation and computational efficiency in multiparty k-Means clustering, setting a compelling stage for further research and innovation.

3.2 Multiparty k-Means (Outsourced)

Privacy-preserving k-Means clustering has been extensively studied in various settings, among which is the multiparty scenario where the computations are outsourced to an assisting server or aggregator. Various protocols have been proposed to facilitate this scenario.

Patel et al. [9] proposed a distributed k-Means clustering scheme based on Shamir's secret sharing [42]. However, the scheme requires more than two non-colluding servers to ensure privacy. Moreover, the computation mechanism for the distance metric remains unclear, creating ambiguity in the practical application of this method.

Jiawei Yuan and Yifan Tian [10] offered a practical privacy-preserving k-Means clustering scheme tailored for outsourcing to cloud servers. The scheme leverages the integration of MapReduce, making it adaptable to a cloud computing environment. However, the intermediate closest clustering centers are revealed to the server, thus compromising the level of privacy to some extent.

Rao et al. [11] developed a unique protocol that mitigates privacy issues by employing two semi-honest servers and utilizing homomorphic encryption. Their protocol notably avoids division by scaling up the centroids – specifically by the product of all cluster counts, with division by omission of the corresponding cluster count in the scale – and simultaneously adjusts the distance computation to incorporate this scaling. While this method successfully conceals everything except the final clusters, it does suffer from long runtimes and high communication costs, making it less feasible for larger-scale applications.

Similarly, **Liu et al.** [12] introduced a two-party protocol wherein most computations are outsourced to the cloud. Each party encrypts their data using two different homomorphic encryption schemes [43, 44], and the encrypted data is then processed by the cloud to determine cluster assignments and centroids. However, the practicality of this approach remains uncertain due to the lack of implementation or evaluation. Also, extending this approach to handle more than three clients is unclear from their work since the authors only provided an example of the three-client extension. Additionally, multiple rounds of interactions are required between the cloud and the users every iteration, and the intermediate centroids are revealed in the clear to all clients. **Jiang et al.** [14] optimize the protocol in the two-party case by utilizing only one homomorphic encryption scheme and replacing the other with SMPC protocols. However, the approach still suffers the same limitations and has runtimes in the order of minutes.

Silva et al. [13] conducted a comprehensive study focusing on privacy-preserving multiparty clustering techniques. Unlike the methods that are tailored specifically for k-Means clustering, their work addresses the broader domain of clustering with applications in various fields like customer segmentation and information retrieval, especially for Software-as-a-Service (SaaS) providers.

In the context of outsourced multiparty k-Means clustering, the literature presents a wide array of techniques catering to privacy concerns. Utilizing diverse cryptographic primitives, such as homomorphic encryption and secure multiparty computation protocols,

emphasizes the complexity of achieving privacy in outsourced computations. Although strides have been made to address privacy in different server-assisted scenarios, these solutions commonly grapple with trade-offs between privacy, efficiency, and scalability. The revelation of intermediate clustering centers and high runtimes are recurring issues that pervade existing protocols. Moreover, practical extensions to more than two clients and the necessity of non-colluding servers further impede their real-world applicability. These challenges delineate a fertile ground for further exploration, emphasizing striking a delicate balance between preserving privacy and enhancing computational efficacy in outsourced multiparty k-Means clustering.

3.3 Centralized DP k-Means

Although valuable, pure secure multiparty computation (SMC) based methods discussed thus far fall short as their final outputs lack provable resilience against membership or reconstruction attacks. Additionally, the computation and communication overheads are generally high, often making these methods unsuitable for practical applications.

This section surveys a collection of centralized differential privacy k-Means algorithms. The goal is to understand the current state of the art, its benefits, drawbacks, and their applicability in a multiparty setting. We specifically focus on centralized DP instead of local DP because the local DP k-Means treats each user as a separate data point. Our objective is to handle the scenario where every user holds a dataset partition, constituting a federated setup.

Blum et al. [16] adopted a simple mechanism where Laplace noise was directly injected into the iterations of Lloyd's algorithm to ensure differential privacy. The privacy budget was allocated uniformly across all iterations. This method, however, required significant computational resources as the number of iterations was determined empirically. Building on this work, **Dwork** [17] employed a similar noise-injection approach but allocated the privacy budget using a decreasing exponential distribution. This increased noise injection as the privacy budget decreased, leading to poor clustering quality.

Su et al. [18] presented a refined privacy budget allocation and fixed the number of iterations using a theoretically guaranteed optimal allocation method. This improved upon the methods of both Blum et al. and Dwork by enhancing the clustering quality. However, the scheme assumed a uniform cluster size, limiting its universal applicability. In practice, the scheme performs best across all approaches that use the standard computations of the centroids, as evidenced in the analysis by **Lu et al.** [22].

Mohan et al. [19] introduced GUPT, a differentially private framework based on sample and aggregate techniques applied to Lloyd's algorithm. However, due to its uniform sampling mechanism, GUPT might over-sample from one cluster leading to excessive noise during aggregation, thus yielding unsatisfactory clustering quality. **Zhang et al.** [20] proposed a completely different approach in PrivGene, a DP k-Means clustering algorithm based on a genetic algorithm, utilizing the exponential mechanism [45] for sampling surviving candidates. While PrivGene achieved satisfactory clustering quality for relatively small datasets, it faced efficiency issues due to a required predefined iteration number.

Alternative high-dimensional solutions were proposed by **Balcan et al.** [21], where they used the Johnson-Lindenstrauss to project data into a lower-dimensional space and a private recursive subdivision technique to obtain small regions for candidate centroids. **Lu et al.** [22] made further improvements by incorporating knowledge from previous and potential future iterations to control the movement orientation of centroids, thereby guaranteeing convergence. While these works show promise, they are constrained by complex non-standard computations, limiting their adaptability to a multiparty setting.

On the other hand, **Park et al.** [23] ensured (ϵ, δ)-DP by making assumptions on the input dataset distribution. **Ni et al.** [24] suggested a different approach of dividing the k-Means problem into more clusters (nk)-Means with the intent of canceling out Laplace noises. However, this reduces the size of the clusters, making them susceptible to outliers, especially in the multiparty case.

In summary, while the state-of-the-art offers several insightful and effective methodologies for centralized DP k-Means clustering, none are readily adaptable to a multiparty setup due to a variety of reasons ranging from high computational cost to non-standard computations, uniform cluster size assumptions, and distribution assumptions of the input dataset. It is, therefore, crucial to investigate efficient and practical ways of extending these centralized DP methods to a federated or multiparty setup, which is the main focus of this work.

3.4 Federated DP k-Means

In the existing body of knowledge, only two published works attempt to address the problem of differentially private (DP) k-Means clustering in a multiparty federated setup.

Li et al. [46] proposed a practical solution tailored to the vertical federated setting, which relies on an untrusted central server to aggregate differentially private local centers and membership encodings from the clients. The central server then generates a weighted

grid as a synopsis of the global dataset. The weights are estimated using a novel differentially private set intersection cardinality estimation algorithm based on the Flajolet-Martin sketch, and the grid is formed as the Cartesian product of all the partial centroids shared by the clients. However, this approach is specifically designed for vertical federated setups and cannot be readily applied to other configurations.

Zhang et al. [47] proposed a method for the horizontal federated setting. Their approach requires the involvement of two non-colluding servers responsible for secure division and independent Laplace noise injection at the centroid level. While this work marks an essential step towards combining Secure Multiparty Computation (SMPC) and DP, several critical issues must be addressed. Firstly, the privacy proof is informal and does not explicitly state the sensitivity or account for the data-dependent initialization. Secondly, the evaluation obscures the effect of the expensive multiparty division and the quality degradation due to centroid-level noise. Because of issues with the privacy proof, we consider our work to be the first in this space as this instantiation is not really private.

In conclusion, while both studies present important contributions to differentially private k-Means clustering in a federated setting, significant limitations and concerns hinder their practical applicability. Therefore, it is imperative to further explore solutions that are both privacy-preserving and practically implementable in a multiparty setup, which is the focus of this research.

Chapter 4

Customizing k-Means

4.1 Assignment Methods

In the k-Means clustering algorithm, various assignment strategies can impose constraints on the cluster sizes. The assignment step is a critical component of the algorithm, determining how data points are associated with clusters before the centroids are updated. We discuss four main strategies: (1) Unconstrained Exclusive, (2) Constrained Non-Exclusive, (3) Constrained Exclusive, and (4) Constrained E-Exclusive.

4.1.1 Unconstrained Exclusive

In the original k-Means algorithm [28], every data point is associated with its nearest (in a squared-euclidean sense) centroid from the last iteration. This does not restrict the cluster size and ensures that every point is exclusively associated with one and only one cluster. Formally, the assignment for the i-th cluster in the t-th iteration is:

$$O_i^{(t)} = \left\{ x_p : ||x_p - \mu_i^{(t)}||^2 \leq ||x_p - \mu_c^{(t)}||^2 \ \forall c, 1 \leq c \leq k \right\}$$

4.1.2 Constrained Non-exclusive

The Constrained Non-exclusive assignment method introduces a minimum cluster size constraint c_{min} in the k-Means algorithm. For each cluster O_i, the nearest c_{min} data points are chosen and associated with it. This means the assignment is not exclusive; a data point

can belong to multiple (or even all) clusters. Formally, the assignment for the i-th cluster in the t-th iteration is:

$$O_i^{(t)} = \left\{ x_p : x_p \text{ is among the closest } c_{min} \text{ points to } \mu_i^{(t)} \right\}$$

This approach ensures a minimum cluster size and introduces overlap among clusters as data points can be assigned to multiple clusters. While this might be undesirable in some contexts where clear separation of clusters is sought, in other scenarios it may provide a more flexible representation of the data, reflecting the potential for data points to exhibit characteristics of multiple clusters simultaneously.

4.1.3 Constrained Exclusive

The Constrained Exclusive assignment method modifies the k-Means algorithm to enforce a constraint on the cluster size while ensuring that each data point is exclusively assigned to one cluster. We expand on the work of [26] which enforces a minimum cluster, to further impose a maximum cluster size requirement. Specifically, the size of each cluster O_i is constrained to be between a minimum and maximum value, $c_{min} \leq |O_i| \leq c_{max}$.

The objective in this case is to minimize the cost function:

$$J(O, \mu) = \sum_{i=1}^{k} \sum_{x_j \in O_i} ||x_j - \mu_i||^2$$

subject to the constraints:

$$c_{min} \leq |O_i| \leq c_{max} \quad \forall i \quad \text{and} \quad \sum_{i=1}^{k} I(x_j \in O_i) = 1 \quad \forall j$$

where $I(x_j \in O_i)$ is an indicator function that equals 1 if $x_j \in O_i$ and 0 otherwise.

This method seeks to minimize the total squared distance of points to their assigned centroids while ensuring that each point is associated with exactly one cluster and each cluster size falls within the specified range. However, this approach leads to a more complex optimization problem that may require advanced techniques or heuristics to solve effectively.

One viable approach to solving the Constrained Exclusive method is formulating it as a minimum-cost flow problem. We extend the analysis in [26] to include the maximum constraint. In this model, data points x_j are represented as nodes and connected to centroid nodes μ_i through directed edges. The cost of each edge (x_j, μ_i) is defined as the Euclidean distance between x_j and the centroid μ_i. The capacity of each edge is set to 1, ensuring that a data point cannot contribute to the same cluster more than once.

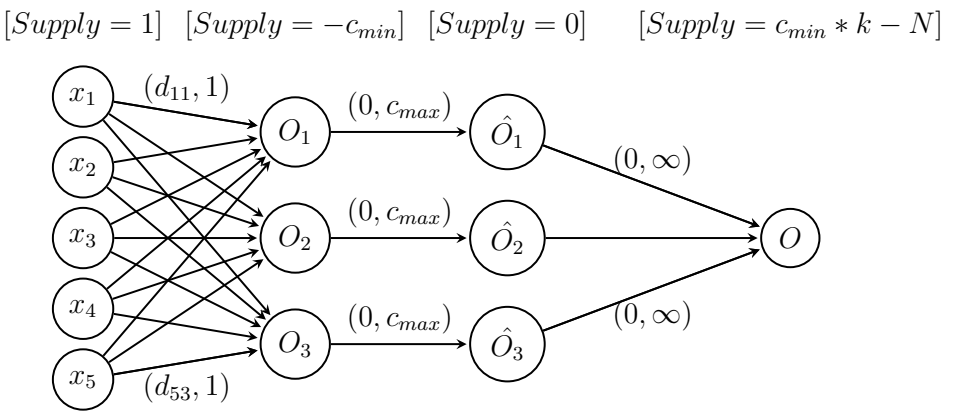
Each data point node has a supply of 1, ensuring that each data point is associated with exactly one cluster. On the other hand, each centroid node has a supply of $-c_{min}$ (equivalently, a demand of c_{min}). As every valid flow solution must satisfy this demand, a minimum of c_{min} data points will flow into each cluster, thereby meeting the minimum size constraint.

To impose the maximum size constraint, we create clone nodes for each centroid, and connect each centroid with its clone through an edge with zero cost and a capacity of c_{max}. This constrains the maximum flow into each cluster to be c_{max}.

Lastly, a sink node is introduced to consume the remaining flow. Since there are N data points each supplying 1 unit of flow, and each cluster node demands c_{min} units, the remaining flow is $N - c_{min} \cdot k$. Hence, the sink node demands this remaining flow, completing the model.

Thus, this network flow model provides a practical means of solving the Constrained Exclusive k-Means assignment, ensuring that the constraints on cluster sizes are respected and the total distance of points to their assigned centroids is minimized.

In the figure below, we show the reduction of an assignment instance ($N = 5, k = 3$) to a network flow graph that will be solved by any off-the-shelf min-cost flow solver.



4.1.4 Constrained E-Exclusive

The Constrained E-Exclusive assignment method introduces an additional parameter E to the k-Means algorithm, enabling each data point to contribute to multiple, yet limited, clusters. Specifically, each data point can be associated with up to E clusters where $E \leq k$.

The objective remains the minimization of the cost function:

$$J(O, \mu) = \sum_{i=1}^{k} \sum_{x_j \in O_i} ||x_j - \mu_i||^2$$

subject to the constraints:

$$c_{min} \leq |O_i| \leq c_{max} \quad \forall i \quad \text{and} \quad \sum_{i=1}^{k} I(x_j \in O_i) \leq E \quad \forall j$$

This method thus provides more flexibility in the assignment of data points, allowing them to reflect the characteristics of multiple clusters. It retains the cluster size constraints of the Constrained Exclusive method while relaxing the exclusivity of data point assignment. This also allows for instances where $N < k \cdot c_{min}$ which were impossible to be solved in the Constrained Exclusive assignment. However, similar to the Constrained Exclusive method, this results in a more complex optimization problem that may require more sophisticated methods or heuristics to solve efficiently.

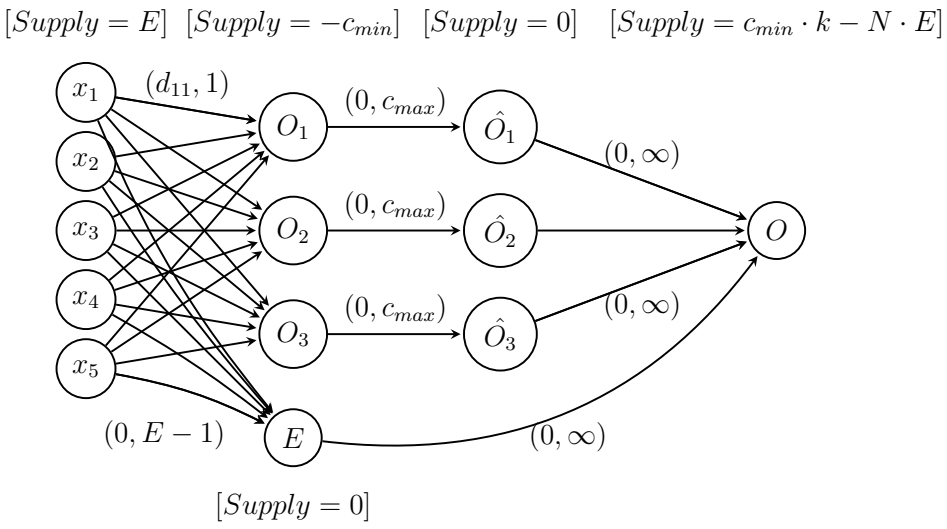
The minimum-cost flow formulation can also be adapted to solve the Constrained E-Exclusive method. In this scenario, each data point node is given a supply of E, representing the maximum number of clusters a data point can contribute to. The capacity between data points x_j and centroid nodes μ_i remains 1 to prevent a data point from being assigned to the same cluster multiple times.

However, this modification enforces each data point to contribute to exactly E clusters, which might not always be desirable. We introduce an "escape" node with neither supply nor demand to address this. Each data point node x_j is connected to the escape node through an edge with zero cost and a capacity of $E - 1$. The zero cost allows the flow to divert to the escape node unless the minimum cluster demand c_{min} is not met. In this case, the solver will opt for the least costly data points to satisfy the constraint. The capacity of $E - 1$ ensures that each data point contributes at least once, but no more unless necessary.

Finally, the escape node is connected to the sink node with an edge of zero cost and infinite capacity, which allows any remaining flow to be absorbed. The sink node now has a demand of $E \cdot N - c_{min} \cdot k$ to account for the increased total supply of $E \cdot N$.

This adjusted network flow model provides a practical method to solve the Constrained E-Exclusive k-Means problem while allowing for flexible data point contributions and satisfying the cluster size constraints.

In the figure below, we show the reduction of the same assignment instance ($N = 5, k = 3$) to a network flow graph that is to be solved by any off-the-shelf min-cost flow solver.



4.2 Initialization Methods

The k-Means algorithm is significantly influenced by the initial choice of centroids. Consequently, careful selection of these initial centroids is imperative, as an improper choice might lead to convergence to a local optimum far from the global optimum. A variety of methods have been proposed in the literature to tackle this issue, some of which are discussed in this section.

4.2.1 Data-driven Initialization

One intuitive approach is to derive initial centroids directly from the data points themselves. This is advantageous because these data points inherently reflect the underlying

distribution and structure of the data, and starting from them could potentially lead to more meaningful and representative clusters.

Random Initialization In the simplest case, one could randomly select k data points from the dataset as the initial centroids. This method is straightforward and does not require any additional computations. However, its effectiveness largely depends on chance, and it could lead to poor convergence or local optima.

k-means++ Initialization [48] A more sophisticated approach is the k-means++ initialization algorithm. This method aims to spread out the initial centroids in the data space such that they are not too close to each other, reducing the probability of converging to a sub-optimal solution. The algorithm operates as follows:

1. Select the first centroid μ_1 uniformly at random from the data points.
2. For $i = 2, ..., k$, compute the squared Euclidean distance $d(x_j, \mu_c)^2$ from each data point x_j to the nearest centroid that has already been chosen, and select the next centroid μ_i randomly from the data points with probability proportional to this squared distance.

Formally, the probability $P(x_j)$ of choosing data point x_j as the next centroid is given by:

$$P(x_j) = \frac{d(x_j, \mu_c)^2}{\sum_{i=1}^{N} d(x_i, \mu_c)^2}$$

where $d(x_j, \mu_c)^2$ is the squared Euclidean distance from x_j to the nearest chosen centroid μ_c, and the denominator is the sum of these distances for all data points, effectively normalizing the probabilities.

The k-Means++ method offers a significant improvement over random initialization by providing a more systematic approach to distribute the initial centroids across the data space. However, it requires computation of distances between data points and centroids, which might be computationally expensive for large datasets.

4.2.2 Domain-driven Initialization

There are situations where direct access to the dataset is not possible due to privacy concerns, latency issues, or when dealing with streaming data where the entire dataset is not available upfront. In such scenarios, it is still possible to initialize the centroids using the known domain of the data. Although it may not yield optimal results, normalizing the data within the domain $[-B, B]^d$ can provide a plausible approach. Here, we discuss a few of these methods.

Random Initialization A straightforward way is to randomly generate k points within the domain $[-B, B]^d$ as initial centroids. This method is computationally simple, but just like the data-driven random initialization, it heavily relies on chance, and the effectiveness of the result is largely unpredictable.

Sphere-packing Initialization [18] A more nuanced approach is the Sphere Packing or repulsive initialization method, which seeks to emulate the spread enhancement of k-means++ but without direct access to the data points. This method generates the initial centroids in a way that they are far apart from each other.

The process of space-filling initialization can be outlined as follows:

1. Initialize a radius parameter a.
2. For $i = 1, 2, ..., k$, generate a point μ_i such that it is at least of distance a away from the domain boundaries and at least of distance $2a$ away from any previously chosen centroid. If a randomly generated point does not meet this condition, generate another one.
3. If after repeated attempts, it is not possible to find such a point, decrease the radius a and repeat the process.

The radius a can be determined via a binary search, iteratively adjusted until a suitable value is found that allows successful generation of k centroids. This would be the radius that maximally separates the initial centroids which provides for better convergence.

Formally, a new centroid μ_i is chosen by satisfying:

$$||\mu_i - \mu_c||^2 \geq (2a)^2 \ \forall c, 1 \leq c < i \text{ and } ||\mu_i - f||^2 \geq a^2 \ \forall f \in \partial B(B)$$

Where $\partial B(B)$ is the boundary specified as:

$$\partial B(B) = \left\{ b \in \mathbb{R}^d : \underline{b} \leq b \leq \overline{b} \text{ and } (\exists i, 1 \leq i \leq d : b_i \in \{-B, B\}) \right\}$$

and

$$\underline{b} = [-B, -B, ..., -B]^T \in \mathbb{R}^d, \quad \overline{b} = [B, B, ..., B]^T \in \mathbb{R}^d$$

This method requires no knowledge of the data points, maintaining the privacy of the data, and can give a good spread of initial centroids across the domain, thereby providing a good starting point for the k-Means algorithm.

Chapter 5

Multiparty k-Means

This section introduces the methodology of the multiparty k-Means algorithm. We consider M clients denoted by $P = \{P_1, P_2, ..., P_M\}$, with each party P_l owning a private dataset $x_{.l} = \{x_{l1}, x_{l2}, ..., x_{lN_l}\}$, where $x_{lj} \in \mathbb{R}^d$, d denotes the dimensionality of the dataset and N_l is the size of the dataset of party l. The objective is to compute a collaborative k-Means clustering across the union of these datasets, ensuring that no individual data points are exposed to the other clients. A service provider P_S assumed to be semi-honest, can assist with multiparty computations.

5.1 Multiparty Initialization

We explore two methods for multiparty initialization that are based on the domain-centered approach. This approach ensures the data privacy by avoiding data-driven approaches for initialization that may risk revealing sensitive information about the individual clients' datasets. Additionally, it ensures that all clients have the same centroid as a starting point.

5.1.1 Synchronized Random Initialization

In the first method, all clients share the same seed and use the same Pseudo Random Number Generator (PRNG) to generate the same centroids as a starting point. For example, each party P_l could generate k points within the domain $[-B, B]^d$ to form the initial centroids μ_i, where $1 \leq i \leq k$. Formally, we can write this as:

$$\mu_{il} = \text{PRNG}(\text{seed}, i),\ \forall i, 1 \leq i \leq k$$

This procedure ensures a common starting point for all clients without revealing any information about the individual datasets. Alternatively, clients could also use the Sphere-packing (repulsive) Initialization [18] using the same PRNG to arrive at the same centroids.

However, it's worth noting that if clients use different seeds, every centroid from the first iteration could contain points from all across the domain since they won't agree on the initialization. This results in effectively concentrating the initialization at the center of the domain due to averaging, which is undesirable as it restricts the algorithm's ability to explore different regions of the domain.

5.1.2 Server-generated Initialization

The second method involves the service provider P_S generating the initial centroids and distributing them to all the clients. Since the initialization methods used are all domain-driven and the domain is available as part of the parameters shared with the server, the server can generate the centroids using any method and publish them to the clients. However, this gives an important control point to the server, which is acceptable as long as we assume the server is semi-honest and will not maliciously generate centroids in such a way to reveal information.

In general, we prefer using client-side initialization, since the clients are already assumed to share a common seed and because it would deduct an unnecessary round of communication at no cost to the accuracy or privacy of the protocol. In fact, it further hides the initial centroids from the server.

5.2 Multiparty Lloyd's Algorithm (Naive)

5.2.1 Protocol

The multiparty setting presents a simplistic application of Lloyd's algorithm that follows these steps:

1. **Initialization:** Clients initialize their centroids as in subsection 5.1.

2. The following steps are repeated until either convergence is achieved or a specified number of iterations are completed:

   (a) **Assignment:** Every party P_l calculates their local assignments based on a predefined assignment method. In the case of the Unconstrained Exclusive assignment, for instance, each party computes clusters O_{li} for $i \in \{1,2,...,k\}$, such that

   $$O_{li}^{(t)} = \left\{ x_{lj} : ||x_{lj} - \mu_i^{(t)}||^2 \leq ||x_{lj} - \mu_c^{(t)}||^2 \ \forall c, 1 \leq c \leq k \right\}$$

   (b) **Local Update:** Each party P_l computes the sum and the count of their local clusters and transmits them to P_S. Consequently, each party sends $(S_{li}^{(t)}, C_{li}^{(t)})$ for $i \in \{1,2,...,k\}$, where

   $$S_{li}^{(t)} = \sum_{x_j \in O_{li}^{(t)}} x_j$$

   and

   $$C_{li}^{(t)} = |O_{li}^{(t)}|$$

   (c) **Global Update:** P_S computes the new centroids by aggregating the sums and the counts and then dividing:

   $$\mu_i^{(t+1)} = \frac{\sum_{l=1}^{M} S_{li}^{(t)}}{\sum_{l=1}^{M} C_{li}^{(t)}} = \frac{S_i^{(t)}}{C_i^{(t)}}$$

5.2.2 Information Leakage

In this algorithm, the service provider P_S receives the local cluster sizes C_{li} and the local cluster sums S_{li} from every party P_l. This enables P_S to compute the local and global centroids, which are subsequently shared with the other clients. Such a mechanism reveals some information about the individual data distributions across different clients, potentially constituting an unacceptable privacy infringement. The shared variables are illustrated in table 5.1.

5.3 Masked Multiparty Lloyd's Algorithm (Sum/Count)

Taking advantage of the Masked Secure Aggregation technique discussed in subsection 2.2, we modify the multiparty k-Means algorithm to make the data completely invisible

Entity	Local Variables		Global Variables		
	C_{li}	S_{li}	C_i	S_i	μ_i
P_S	✓	✓	✓	✓	✓
$P_{l'}$	$l = l'$	$l = l'$	×	×	✓

Table 5.1: Leakage analysis of the Naive Multiparty k-Means protocol.

to the server while preserving the accuracy of computations. This proposed modification primarily changes the local update phase of the algorithm, concealing the true sums and counts of local clusters from the server. This allows the clients to see the sums and counts instead of the final centroids; an approach introduced by [5] to avoid secure division.

5.3.1 Protocol

Under the masked multiparty Lloyd's algorithm, the following steps are executed:

1. **Initialization:** This step remains unchanged from the basic approach.
2. The following steps are iterated until a specified number of iterations are completed:
   (a) **Assignment:** The assignment step also remains unchanged. Each party P_l computes local assignments based on the predefined assignment method.
   (b) **Masked Local Update:** In this modified step, each party P_l calculates the sum and count of their local clusters, just as in the basic approach. However, instead of directly transmitting these values to P_S, they first mask the sums and counts using the additive masking technique described earlier. The party then sends the masked values $(Enc_l(S_{li}), Enc_l(C_{li}))$ for $i \in \{1,2,...,k\}$ to P_S.
   (c) **Global Update:** P_S aggregates the masked sums and counts separately and sends these aggregated masked values back to the clients $(Enc(S_i), Enc(C_i))$. Each party then unmasks the sums and counts and performs the division themselves to compute the new centroids.

5.3.2 Information Leakage

The novel aspect of this approach is the masking of sums and counts, which ensures that the server cannot access any meaningful data from the clients. Consequently, privacy is

enhanced without affecting the core computations or accuracy of the results. However, the clients can now obtain the total sums and counts across all clients. The information shared among the server and the clients in this masked algorithm is illustrated in Table 5.2.

	Local Variables		**Global Variables**		
Entity	C_{li}	S_{li}	C_i	S_i	μ_i
P_S	×	×	×	×	×
$P_{l'}$	$l = l'$	$l = l'$	✓	✓	✓

Table 5.2: Leakage analysis of the Sum/Count Masked Multiparty k-Means protocol.

5.4 Masked Multiparty Lloyd's Algorithm (Centroid)

Aiming to limit the information leakage even further, we propose an alternative approach that conceals the total sums and counts from the clients. This, however, comes with a trade-off in the accuracy of the algorithm.

The central concept here is to refrain from transmitting the local sums and counts to the server, instead sending a "scaled down" version of the centroids. For each centroid i and for each party P_l, we send a masked version of

$$\tilde{\mu}_{li} = \frac{S_{li}}{M \cdot C_{li}}$$

to the server. In doing so, when the server aggregates these values, the result is not the true centroid but rather an "average of local centroids".

The deviation in accuracy arises from the fact that the average of averages does not necessarily equate to the true average, unless each average is computed over an equally-sized set. To illustrate, consider two clients, one with a three-element set (1,2,3) and the other with a two-element set (2,3). The average of averages is $\frac{\frac{1+2+3}{3}+\frac{2+3}{2}}{2} = 2.25$, while the true average of all five elements is $\frac{1+2+3+2+3}{5} = 2.2$. This discrepancy becomes more pronounced with larger differences in set sizes across clients.

Despite this decrease in accuracy, opting for the "average of local centroids" as an estimate may be beneficial in terms of privacy preservation. Specifically, counts could represent sensitive information because they reveal the volume of data each party holds. For instance, in a healthcare context, the count could indirectly indicate the number of patients a hospital treated for a particular condition, information that might be considered confidential.

5.4.1 Protocol

Under this proposed Masked Centroid-Level Multiparty Lloyd's Algorithm, the modified steps become:

1. **Masked Local Update:** Instead of calculating and transmitting masked sums and counts, each party P_l computes the masked scaled down version of their local centroids: $Enc_l(\tilde{\mu}_{li})$, and sends these to the server.

2. **Global Update:** The server aggregates the scaled down centroids and returns the sum ($Enc(\tilde{\mu}_i)$) directly to the clients to be unmasked and used as the new centroids for the next iteration.

5.4.2 Information Leakage

By employing this strategy, privacy is further enhanced as it limits the data each party can get from the process. However, it is essential to weigh this benefit against the potential loss in accuracy, depending on the context and the sensitivity of the data involved.

	Local Variables		**Global Variables**		
Entity	C_{li}	S_{li}	C_i	S_i	μ_i
P_S	$\times$	$\times$	$\times$	$\times$	$\times$
$P_{l'}$	$l = l'$	$l = l'$	$\times$	$\times$	✓

Table 5.3: Leakage analysis of the Centroid Masked Multiparty k-Means protocol.

Chapter 6

Differentially-Private k-Means

Until this point in our exploration, we have primarily focused on ensuring that the central server, tasked with aggregating the data, remains oblivious to the data points of each client. This approach successfully hides the dataset from potential misuse or abuse by a potentially untrustworthy server.

However, this approach may prove insufficient in scenarios where the clients themselves are not trustworthy or, more importantly, do not trust each other. For example, in a collaborative data analysis scenario, participating entities may be competing businesses willing to share data for a collective benefit but wary of revealing their individual customer-level data. It could also be a case where the data points belong to individual users who do not want their personal information exposed to others.

This raises the bar for privacy considerations, urging us to contemplate the notion of privacy from a more granular perspective. We need to think beyond protecting data at an aggregate level and ensure that each data point's contribution is not discernible. This would imply that the information of any data point of a single client cannot be inferred even by another participating client, thereby maintaining the confidentiality of each data contributor.

Differential privacy comes handy in these scenarios. Differential privacy is a robust privacy framework that guarantees that adding or removing a single data point does not significantly affect the outcome of any analysis. This provides a provable level of protection to each data point against other parties, ensuring that the output of a differentially private mechanism does not reveal the presence or absence of any single data point in the dataset.

In this work, we extend the concepts of our masking-based multiparty algorithm to incorporate differential privacy, aiming to deliver a more encompassing privacy-preserving

k-Means algorithm. In this section, we explore the effects of differential privacy on the k-Means algorithm itself (locally). We will explore two possible instantiations based on the aggregation method used to compute the centroids. In the following sections, we will explore how to adapt these ideas to the multiparty case.

6.1 Customizing k-Means for Differential Privacy

To instantiate multiparty differentially-private k-Means, a couple of core components need careful customization: the initialization, assignment and update procedures. We aim to adapt these procedures in the light of differential privacy.

6.1.1 Initialization Methods

For initialization of the centroids, we utilize *Domain-driven* approaches. The advantage of this method lies in its operational flexibility: it can be performed either by the clients or the server as discussed earlier, which is a valuable asset in the context of a decentralized setting. Moreover, domain-driven initialization has the merit of being *privacy-preserving*, as it does not demand any additional privacy budget (ϵ). To evaluate the impact of the initialization scheme on the overall protocol, we consider two types of centroid initialization: *Random* and *Sphere-packing*. While *Sphere-packing* approach was shown to be superior in [18], we noticed that the *Random* initialization performed similarly in our preliminary evaluations while not requiring as much computations as the space-filling one, so we evaluate both approaches.

6.1.2 Assignment Method

For the assignment method, we choose the *Constrained E-Exclusive* method. This choice is motivated by its universality and flexibility: most other assignment methods can be reduced to it by a change in parameters, and it provides a parametrized fine-grained control over the differentially-private noise needed in multiple scenarios as will be shown later in this section.

6.1.3 Postprocessing

When incorporating differential privacy into the k-Means algorithm, we add noise to the sum and count of every cluster (or to the centroid). These noise additions can push the calculated centroids beyond the specified domain $[-B, B]$. To prevent this from occurring, we can employ two post-processing strategies in the update phase: truncation and folding.

Truncation is the simpler of the two strategies. If a calculated centroid lies outside of the specified domain, it is truncated to the nearest boundary. More formally, if x is the value of a centroid, we enforce $x = \max(\min(x, B), -B)$. However, truncation has a drawback: it can create an undue concentration of values at the boundaries of the domain, distorting the true distribution of the data and leading to potential biases in subsequent analyses.

Folding, on the other hand, reflects out-of-bound values into the domain. More formally, if $x < -B$, we adjust it to $x' = -2B - x$, and if $x > B$, we adjust it to $x' = 2B - x$. This folding process effectively "folds" (mirrors) the value over the nearest boundary, preserving its distance from the boundary. This method allows for a more even distribution of noise within the domain. Figure 6.1 illustrates this idea on a number line with 3 different values to fold.

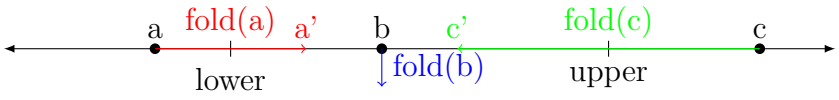


Figure 6.1: Illustration of folding as a postprocessing step

We can apply these methods at three levels: the sum, the count, and the centroid itself. The sum is constrained to lie within $[-Bc_{max}, Bc_{max}]$, since in our constrained k-means variant, the number of points in every cluster is bounded by c_{max}, each are in the original domain, $[-B, B]$. The count lies within $[c_{min}, c_{max}]$ as per the constraints. Finally, the centroid is kept within the original domain, $[-B, B]$.

For the sum and count, we may use either truncation or folding, but we recommend folding as it maintains a more natural distribution of the noise within the specified range. Truncation could be advantageous in specific scenarios where the introduction of additional noise may not be desired.

For the centroid, we typically employ folding, as truncation can lead to a biased estimate of the true cluster centroid, pushing them towards the edges of the domain. Additionally, folding the centroids back into the domain ensures that they get exposed to more points in the next iteration and hence get a better estimate.

It's worth noting that the optimal method might vary depending on the particular properties of the dataset and the requirements of the analysis. Therefore, in our experiments, we evaluate the performance of all the discussed strategies. This empirical evaluation complements our theoretical analysis and provides a comprehensive understanding of how best to integrate differential privacy into KMeans clustering while preserving the utility of the output.

6.2 Sum/Count Method (DPLloyd)

The first differentially-private version of Lloyd's algorithm (DPLloyd) was proposed in [16]. It offers a straightforward and intuitive application of differential privacy for the k-Means algorithm. The essential modifications to the algorithm in this approach are twofold: firstly, Laplace noise is added to both the sum and count of each cluster during the update phase; secondly, to control the amount of noise added, the number of iterations for the algorithm is decided beforehand. This approach, because the noise is added to the sum and count separately, works well in the multiparty protocols we have discussed where sum and count components are treated separately. In this section, we will be presenting this approach with the modificiations and improvements from [18], but with our parameterization of the Constrained-E-Exclusive assignment method.

6.2.1 Sensitivity Analysis

To formally illustrate DPLloyd we first inspect the sensitivity of the used mechanisms to compute the centroids. To start with, in every iteration, every data point (a d-dimensional tuple) contributes to d sum queries and 1 count query if the default (Unconstrained) association method is used. However, in our analysis, we are using the Constrained-E-exclusive assignment method. Therefore, every data point may contribute up to $E \cdot d$ sum queries and E count queries every iteration.

Assuming the data is normalized between $[-B, B]$, then sensitivity of every sum query is B (in the unbounded DP model). The sensitivity of every count query is, trivially, 1. Therefore, the sum sensitivity for every iteration (and every dimension) is:

$$\Delta^{\Sigma} = BE$$

Similarly, the sensitivity of the count mechanisms is:

$$\Delta^{C} = E$$

We split the provided privacy budget (ϵ) equally over every iteration and analyze, later in the section, the optimal number of iterations. Additionally, we split the budget such that there is an ϵ^{Σ} for every sum dimension and ϵ^{C} for the count. That is:

$$\epsilon = t_{max} \cdot (d \cdot \epsilon^{\Sigma} + \epsilon^{C})$$

Therefore, the modified centroid update step for dimension h is as follows:

$$\widehat{\mu}_i^{(t+1)}[h] = \frac{S_i^{(t)}[h] + \eta^{\Sigma}}{C_i^{(t)} + \eta^{C}}$$

where $\eta^{\Sigma} = Lap(0, \Delta^{\Sigma}/\epsilon^{\Sigma})$, $\eta^{C} = Lap(0, \Delta^{C}/\epsilon^{C})$ and $Lap(a, b)$ is a Laplace random variable centered around a with scale b. In the next subsections, we will discuss how to optimally split the privacy budget (ϵ) in such a way that minimizes the Mean Squared Error from the plain k-Means.

6.2.2 Error Analysis

The noise added in DPLloyd is introduced every iteration when updating the centroids. To study how this error affects the centroids, we analyse the mean squared error (MSE) between the true centroids and the centroids in DPLloyd every iteration. We adapt the analysis from [18] to our parameters and focus on one centroid update in one iteration and analyze it:

The true centroid's h'th dimension should be $\mu_i[h] = \frac{S_i[h]}{C_i}$, where C_i is the number of data points in the cluster and $S_i[h]$ is the sum of h'th dimension coordinates of data points in the cluster i. Consider the noisy centroid $\widehat{\mu}_i$; its h'th dimension is $\widehat{\mu}_i[h] = \frac{S_i[h]+\eta^{\Sigma}}{C_i+\eta^{C}}$, where η^{C} is the noise added to the count and η^{Σ} is the noise added to the $S_i[h]$. The MSE is thus:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) = \mathsf{E}\left[\sum_{h=1}^{d}\left(\frac{S_i[h] + \eta^{\Sigma}}{C_i + \eta^{C}} - \frac{S_i[h]}{C_i}\right)^2\right]$$

Focusing only on the h'th dimension, and expanding:

$$\begin{aligned}
&\mathsf{MSE}\left(\widehat{\mu}_i[h]\right) = \mathsf{E}\left[\left(\frac{S_i[h]+\eta^{\Sigma}}{C_i+\eta^{C}} - \frac{S_i[h]}{C_i}\right)^2\right] \\
\approx\ & \mathsf{E}\left[\left(\frac{C_i\eta^{\Sigma}-S_i[h]\eta^{C}}{C_i^2}\right)^2\right] \\
=\ & \frac{\mathsf{E}[(\eta^{\Sigma})^2]}{C_i^2} + \frac{\mathsf{E}[S_i^2[h](\eta^{C})^2]}{C_i^4} + \frac{2C_iS_i[h]\mathsf{E}[\eta^{\Sigma}\eta^{C}]}{C_i^4} \\
=\ & \frac{\mathsf{Var}\left(\eta^{\Sigma}\right)}{C_i^2} + \frac{S_i^2[h]\mathsf{Var}\left(\eta^{C}\right)}{C_i^4}
\end{aligned}$$

The last step holds, because η^{Σ} and η^{C} are independent Laplacian variables with zero mean and so the following formulas hold:

$$\begin{cases}
\mathsf{E}[\eta^{\Sigma}\eta^{C}] = 0 \\
\mathsf{E}[(\eta^{\Sigma})^2] = \mathsf{Var}\left(\eta^{\Sigma}\right) - (\mathsf{E}[\eta^{\Sigma}])^2 = \mathsf{Var}\left(\eta^{\Sigma}\right) \\
\mathsf{E}[(\eta^{C})^2] = \mathsf{Var}\left(\eta^{C}\right) - (\mathsf{E}[\eta^{C}])^2 = \mathsf{Var}\left(\eta^{C}\right),
\end{cases}$$

where $\mathsf{Var}\left(\eta^{\Sigma}\right)$ and $\mathsf{Var}\left(\eta^{C}\right)$ are the variances of η^{Σ} and η^{C}, respectively.

To simplify the expression, we follow the same assumptions from [18]; we assume an average normalized coordinate of every centroid $\rho = \frac{|S_i[h]|}{2B\cdot C_i}$, we assume equal-sized clusters, such that $C_i \approx \frac{N}{k}$. Therefore, the error can be approximated as:

$$\mathsf{MSE}\left(\widehat{\mu}_i[h]\right) \approx \frac{k^2}{N^2}\left(\mathsf{Var}\left(\eta^{\Sigma}\right) + (2\rho B)^2 \cdot \mathsf{Var}\left(\eta^{C}\right)\right)$$

Plugging $\mathsf{Var}\left(\eta^{\Sigma}\right)$ and $\mathsf{Var}\left(\eta^{C}\right)$, we obtain

$$\begin{aligned}
\mathsf{MSE}\left(\widehat{\mu}_i[h]\right) &\approx \frac{k^2}{N^2}\left(\mathsf{Var}\left(\eta^{\Sigma}\right) + (2\rho B)^2 \cdot \mathsf{Var}\left(\eta^{C}\right)\right) \\
&= \frac{k^2}{N^2}\left(2\cdot\left(\frac{B\cdot E}{\epsilon^{\Sigma}}\right)^2 + (2\rho B)^2\cdot 2\cdot\left(\frac{E}{\epsilon^{C}}\right)^2\right) \\
&= \frac{2(EkB)^2}{N^2}\left(\frac{1}{(\epsilon^{\Sigma})^2} + \frac{4\rho^2}{(\epsilon^{C})^2}\right)
\end{aligned}$$

As the noise added to each dimension is independent, we know

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) = \sum_{h=1}^{d}\mathsf{MSE}\left(\widehat{\mu}_i[h]\right) \approx \frac{2d(EkB)^2}{N^2}\left(\frac{1}{(\epsilon^{\Sigma})^2} + \frac{4\rho^2}{(\epsilon^{C})^2}\right) \tag{6.1}$$

To inspect the size of the error, we assume that we split the privacy budget equally, that is $\epsilon^{\Sigma} = \epsilon^{C} = \frac{\epsilon}{t_{max} \cdot (d+1)}$. We also assume the normalizing bound $B = 1$. Plugging this into Equation 6.1, we get:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) \approx \frac{2d(d+1)^2(t_{max}Ek)^2}{(\epsilon N)^2}\left(1+4\rho^2\right) \approx \Theta\left(\frac{d^3(t_{max}Ek)^2}{(\epsilon N)^2}\right) \tag{6.2}$$

6.2.3 Splitting the Privacy Budget

The analysis for splitting the privacy budget stays the same as in [18], because including the contribution parameter (E) does not affect the optimal proportion for splitting the budget. To illustrate in our notation, we expand on the analysis here. If we want to alleviate the assumption of equal-splitting of the budget, we need to revisit Equation 6.1. The objective is to minimize:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) \approx \frac{2d(EkB)^2}{N^2}\left(\frac{1}{(\epsilon^{\Sigma})^2} + \frac{4\rho^2}{(\epsilon^{C})^2}\right)$$

Subject to the constraint that $d * \epsilon^{\Sigma} + \epsilon^{C} = \epsilon^{(t)}$ (where $\epsilon^{(t)} = \frac{\epsilon}{t_{max}}$). This can be solved using Lagrange Multipliers and results in the following optimal proportions:

$$\epsilon^{\Sigma} : \epsilon^{C} = 1 : \sqrt[3]{4d\rho^2}$$

Plugging this back into our constraint, we obtain:

$$\epsilon^{\Sigma} = \frac{\epsilon^{(t)}}{d + \sqrt[3]{4d\rho^2}}$$

Similary,

$$\epsilon^{C} = \frac{\sqrt[3]{4d\rho^2} \cdot \epsilon^{(t)}}{d + \sqrt[3]{4d\rho^2}}$$

Plugging these to Equation 6.1 to get the new error estimate, we obtain:

$$\begin{aligned}
\mathsf{MSE}\left(\widehat{\mu}_i\right) &\approx \frac{2d(EkB)^2}{N^2}\left(\frac{(d+\sqrt[3]{4d\rho^2})^2}{(\frac{\epsilon}{t_{max}})^2} + (\frac{(d+\sqrt[3]{4d\rho^2})^2) \cdot 4\rho^2}{(\frac{\epsilon}{t_{max}} \cdot \sqrt[3]{4d\rho^2})^2}\right) \\
&= \frac{2(Et_{max}kB)^2(d+\sqrt[3]{4d\rho^2})^2}{(\epsilon N)^2}\left(d + \frac{4d\rho^2}{(\sqrt[3]{4d\rho^2})^2}\right) \\
&= \frac{2(Et_{max}kB)^2}{(\epsilon N)^2}\left(d+\sqrt[3]{4d\rho^2}\right)^3
\end{aligned}$$

So, if we assume a $B = 1$ as we did before, we have the following updated error estimate:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) \approx \frac{2(Et_{max}k)^2}{(\epsilon N)^2}\left(d + \sqrt[3]{4d\rho^2}\right)^3 \approx \Theta\left(\frac{d^3(t_{max}Ek)^2}{(\epsilon N)^2}\right) \tag{6.3}$$

In [18], the authors experimented with 81 synthetic datasets and conjectured that a good estimate of $\rho \approx 0.225$. In this work, we continue, following the same assumption.

6.2.4 Iterations

Since the error in Equation 6.3 is directly proportional to t_{max}^2, it would make sense to not use a large t_{max}. This is because we split the budget over the allocated number of iterations, so more iterations could cause very small budget for every iteration and hence a huge amount of noise. We follow the same approach in [18] to determine the number of iterations.

Firstly, we calculate a minimum privacy budget per iteration ($\epsilon^{(m)}$) so that the mean-squared error is within a certain bound. Secondly, we divide the available budget (ϵ) by $\epsilon^{(m)}$ in order to determine the number of iterations t_{max}. However, we note that, in practice, there is no need to go beyond 7 iterations, and that you need at least 2 iterations to gain useful results. Therefore, after dividing, we truncate the number of iterations to be in the range $[2, 7]$.

To calculate the minimum privacy budget per iteration, we apply the heuristic described in [18], which states that the total mean squared error (over all centroids, per iteration) should not exceed 0.004 every centroid update. That is:

$$\mathsf{MSE}\left(\widehat{\mu}\right) \approx \frac{2E^2k^3}{(\epsilon^{(m)}N)^2}\left(d + \sqrt[3]{4d\rho^2}\right)^3 \leq 0.004$$

Which gives us

$$\sqrt{\frac{500E^2k^3}{N^2}\left(d + \sqrt[3]{4d\rho^2}\right)^3} \leq \epsilon^{(m)}$$

6.3 Centroid Method (CDPLloyd)

Another approach which might appeal to centroid-level aggregation protocol discussed earlier, is the Centroid Method. In this method, instead of adding the noise to the sum

and count separately and then dividing, we divide first, to obtain the centroid, and then we add the Laplace noise to the final value of the centroid. This introduces more error, as the sensitivity of the mechanism is very high (consider the case when a cluster is a single element), but would be very useful in cases where publishing a differentially-private count is still not acceptable. We call this method Centroid-DPLloyd or CDPLloyd.

6.3.1 Sensitivity Analysis

To formally illustrate CDPLloyd we first inspect the sensitivity of the used mechanisms to compute the centroids. To start with, every iteration, every data point (a d-dimensional tuple) contributes to d average queries if the default (Unconstrained) association method is used. For our analysis (using the Constrained E-Exclusive assignment method), every data point may contribute up to $E \cdot d$ average queries and every iteration.

Assuming the data is normalized between $[-B, B]$, then sensitivity of every average query is B (in the unbounded DP model). This is because, in the worst case, an empty dataset and a dataset with one element are also neighbors in this setting. Therefore, the sensitivity of the average would be exactly the sensitivity of the sum (since the count is 1 in that case). However, this is precisely the reason why Constrained assignment methods are very useful in this use-case. By constraining every cluster to have at least c_{min} datapoints, we ensure that worst-case difference in every centroid between neighboring datasets is $\frac{B}{c_{min}}$. This way, we can add noise in a meaningful way without losing all the information to differential privacy.

Therefore, the average sensitivity for every iteration (and every dimension) is:

$$\Delta^{\mu} = \frac{BE}{c_{min}}$$

We split the provided privacy budget (ϵ) equally over every iteration and analyze, later in the section, the optimal number of iterations. Additionally, we split the budget such that there is an ϵ^{μ} for every dimension. That is:

$$\epsilon = t_{max} \cdot d \cdot \epsilon^{\mu}$$

Therefore, the modified centroid update step for dimension h is as follows:

$$\widehat{\mu}_i^{(t+1)}[h] = \frac{S_i^{(t)}[h]}{C_i^{(t)}} + \eta^{\mu}$$

where $\eta^\mu = L(0, \Delta^\mu/\epsilon^\mu)$and $L(a, b)$ is a Laplace random variable centered around a with scale b.

6.3.2 Error Analysis

The error of CDPLloyd is much more straightforward to compute, as it is directly computed from the Laplace variable introduced and nothing more. To study how this error affects the centroids, we analyse the mean squared error (MSE) between the true centroids and the centroids in CDPLloyd every iteration. We focus on one centroid update in one iteration and analyze it:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) = \mathsf{E}\left[\sum_{h=1}^{d}(\eta^\mu)^2\right] = \sum_{h=1}^{d}\mathsf{Var}\left(\eta^\mu\right) = d \cdot \frac{2(BE)^2}{(\epsilon^\mu c_{min})^2}$$

Since, there's no epsilon-splitting in this use-case, we can directly plug-in $\epsilon = t_{max} \cdot d \cdot \epsilon^\mu$ into the above. We get:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) = \frac{2d(t_{max}dBE)^2}{(\epsilon c_{min})^2} = \frac{2d^3(t_{max}BE)^2}{(\epsilon c_{min})^2} \tag{6.4}$$

To inspect the size of the error, we assume the normalizing bound $B = 1$. Plugging this into Equation 6.4, we get:

$$\mathsf{MSE}\left(\widehat{\mu}_i\right) = \frac{2d^3(t_{max}E)^2}{(\epsilon c_{min})^2} \approx \Theta\left(\frac{d^3(t_{max}E)^2}{(\epsilon c_{min})^2}\right) \tag{6.5}$$

Remark. This error is very similar to what we observed in the DPLloyd with the following differences: 1. c_{min} replaces the assumed expected cluster size of $\frac{N}{k}$. 2. The error is independent of the data or the actual clustering result; it depends completely on the parameters set before running the algorithm.

Remark. The higher we allow c_{min} to be, the closer this error gets to the "Noise then Average" protocol. However, this would usually lower the cluster quality in cases where the sizes of the clusters are not close. Therefore, careful choice of c_{min} is crucial for the success of CDPLloyd.

6.3.3 Iterations

We follow the same approach for calculating the number of iterations as in DPLloyd. To calculate the minimum privacy budget per iteration , we apply the heuristic described in [18], which states that the total mean squared error (over all centroids, per iteration) should not exceed 0.004 every centroid update. That is:

$$\mathsf{MSE}\left(\widehat{\mu}\right) = \frac{2kd^3E^2}{(\epsilon c_{min})^2} \leq 0.004$$

Which gives us

$$\sqrt{\frac{500kd^3E^2}{c_{min}^2}} \leq \epsilon^{(m)}$$

Chapter 7

Multiparty Instantiations of DP k-Means

In this section, we delve into the multiple instantiations of differentially-private k-Means in a multiparty context, connecting various concepts from all the preceding sections. The primary objectives are threefold: First, we examine the parameters from each component of the system; Second, we study the scenario that includes a semi-honest server (P_S), providing the specific instantiation and operation; Third, we discuss potential modifications for the protocol to achieve server-independent operation.

7.1 Parameters

In this section, we examine the parameters for multiparty differentially private (DP) k-Means from four main perspectives: high-level parameters, parameters related to the arithmetic of the protocol, parameters of the k-Means algorithm itself, and parameters of the underlying DP mechanisms.

7.1.1 High-level Multiparty Parameters

High-level parameters are parameters that affect the entire protocol and must be known to the server, as they determine the behavior of the protocol.

Aggregation Method: The aggregation method specifies which process should be used to aggregate the centroids across multiple clients. This choice ultimately specifies the DP method to be used and its implications on the privacy-utility tradeoff. There are two main types of aggregation methods to consider: Centroid and Sum/Count.

- **Centroid** aggregation combines "Masked Multiparty Lloyd's Algorithm (Centroid)" with CDPLloyd. This method can protect against the release of individual sums and counts. However, the quality of the result may be compromised due to the larger noise magnitude and the error introduced during the averaging process before noise addition. The protocol assigned to this method is called "Average, Mask then Noise (AMN)".
- **Sum/Count** combines "Masked Multiparty Lloyd's Algorithm (Sum/Count)" with the DPLloyd. Although this approach generally provides better quality results due to smaller error terms, it reveals the sums and counts separately, which may lead to a higher degree of information disclosure. The protocol assigned to this method is called "Noise, Mask then Average (NMA)".

Number of clients (M): The number of participating clients, denoted as M, refers to the total number of clients involved in the computation. This parameter should be consistently announced across all clients to ensure accurate and consistent computation, particularly in centroid-level operations.

Dataset Size (N): The size of the multiparty dataset (across all clients), denoted as N, is assumed to be known to the server beforehand. This parameter can be securely obtained by performing a secure aggregation protocol with the server. It provides crucial information for tuning some hyperparameters of the protocol without breaching privacy, as the server is oblivious to any other details and the clients only receive some adjusted parameters from the server.

7.1.2 Arithmetic Parameters

Arithmetic parameters are parameters related to the arithmetic of the protocol, specifically for the Masked Secure Aggregation (MSA) operation.

Normalization Range: We need to ensure all clients have the same data bounds, and the differentially-private mechanisms can operate correctly. Hence, we normalize the data to a particular range. Given that our data domain lies in the multidimensional space $[-B, B]^d$, normalization scales the data points to lie within this range. This normalization ensures that all clients work on the same scale and maintains the relative distance between data points, an essential feature for distance-based clustering algorithms such as k-Means.

Heuristic: we assume data was normalized beforehand into the range $[-B, B]^d$ where $B = 1$.

Ring ($\mathcal{R}$): The ring size for computations in the MSA protocol lies between 32 and 64 bits, with the rings chosen from $\mathbb{Z}_{2^{32}}$ and $\mathbb{Z}_{2^{64}}$. The choice depends on the precision and overflow risk considerations. A larger ring size allows for more space for the integer part of the number, reducing the risk of overflow, but it increases the computational load and the communication cost.

Heuristic: we use $\mathbb{Z}_{2^{32}}$ as our ring of operations as there was no specific need for higher width that justifies the computation or communication cost.

Precision factor ($\boldsymbol{\Delta}$): This value should be consistent across all participating clients and dictates the precision of the representation for the real-valued data. It is determined by a power-of-2 scale factor. A higher value of $\boldsymbol{\Delta}$ allows for a greater precision in representing fractional numbers but reduces the number of bits available for the integer part of the number, which could potentially introduce overflows.

Heuristic: we use 2^{16} as our scale factor, splitting the ring into 2 equal parts: one for integer and one for fractional representation.

7.1.3 k-Means Parameters

We enumerate and discuss the parameters integral to the functionality of the k-Means algorithm in the multiparty differentially-private setting. We specifically focus on the parameters pertaining to the Constrained-E-Exclusive variant of the k-Means algorithm, as it forms the basis of our multiparty system. The discussion encompasses the following parameters:

Number of clusters (k): This is the main parameter of the k-Means algorithm. It specifies the number of centroids/clusters needed for the dataset.

Assumption: we assume the optimal number of clusters is known beforehand for ever dataset. This is an accepted assumption over all previous works.

Initialization Method: between Random and Sphere-packing initialization methods, instantiated as discussed earlier in section 5.1.

Experiment: we show in the evaluation section that sphere-packing initialization outperforms random initialization in most settings, so we fix it for our evaluations afterwards.

Minimum Cluster Size (c_{min}): This parameter specifies the lower limit for the number of data points that a cluster can possess. Given a multiparty context, it becomes necessary to divide the c_{min} across the clients. To guarantee that all the contributions add up to the constraint, we use the ceiling function: $\lceil \frac{c_{min}}{M} \rceil$. The smaller the value of c_{min}, the higher the likelihood that all clients will be able to contribute data points. However, setting c_{min} too close to the actual cluster sizes enables a lower error in the execution of the centroid-level DP. This constraint also helps in evading scenarios where the output of the algorithm contains empty clusters which is not a desirable output.

Heuristic: For balanced datasets, we found that using $c_{min} = \lceil \frac{N}{1.5*k} \rfloor$ provides best consistent results. We fix this parameter in our evaluations. The intuition is that, in a balanced dataset, clusters should be around $\lceil \frac{N}{k} \rfloor$ in size, but since this is a hard constraint, and because we work in a distributed setting where clients do not necessarily have the same data sizes, we relax the constraint to $\lceil \frac{N}{1.5*k} \rfloor$.

Maximum Cluster Size (c_{max}): This parameter specifies the upper limit for the number of data points in a cluster. Similar to c_{min}, c_{max} is divided across the clients, but instead using the floor function: $\lfloor \frac{c_{max}}{M} \rfloor$. Lower values of c_{max}, especially those closer to the actual cluster sizes, yield more defined clusters, and subsequently, a better quality of clustering. However, if c_{max} is set too low, the outcome could be a suboptimal clustering configuration as big clusters would be divided.

Heuristic: For balanced datasets, we found that using $c_{max} = \lceil \frac{3*N}{k} \rfloor$ provides best consistent results. We fix this parameter in our evaluations. The intuition is similar to that of c_{min}.

Maximum Datapoint Contribution (E): This parameter dictates the number of clusters to which a single data point can contribute. This value could differ between clients as long as every client respects their value in the sensitivity of their differentially-private releases. In cases where a third party like a semi-honest server is sampling the differentially-private noise, this value has to be the same across clients (or the maximum value will be chosen for the noise). When E is set to 1, the algorithm reduces to the Constrained Exclusive assignment. Higher values of E permit clients with smaller number of records to satisfy the constraints, however, this comes at the expense of increasing the sensitivity of the DP mechanisms, consequently introducing more noise.

Remark. This parameter is specifically useful in serverless approaches. A client could possibly not satsify the minimum constraint c_{min}, but bluff their contribution by using a higher E parameter than the rest, scaling their DP noise appropriately. The evaluation of the effect of this parameter is left to future work, and we fix it to 1 for our evaluations. This means that using Constrained E-Exclusive assignment is equivalent to using Constrained Exclusive assignment.

Number of Iterations (t_{max}): This parameter is responsible for controlling the number of iterations the k-Means algorithm undergoes. A lower value of t_{max} allows for more privacy budget per iteration, hence producing higher quality clustering. In fact, as shown in our error analysis, the error grows quadratically with the number of iterations. However, more iterations potentially allow for greater convergence of the underlying k-Means algorithm. It is worth noting that, due to the server being oblivious to the data and the centroids, it cannot check the convergence. The clients, although possessing the centroids and hence technically capable of evaluating convergence, would risk additional data leakage. This is because some clients may converge faster than others, revealing information about their specific data distributions. Therefore, we only use number of iterations as our stopping criteria.

Calculation: Since we established the optimal number of iterations in all the DP methods discussed earlier, and the server has the number of datapoints N readily available at the start of the protocol, the server calculates the optimal number of iterations and broadcasts it to all clients. In non-private scenarios, we assume the maximum 7 iterations to provide fair comparison.

7.1.4 DP Parameters

This category comprises parameters that are specific to the operation of the differentially private mechanisms used in the k-Means algorithm.

Privacy Budget (ϵ): The privacy budget, denoted by ϵ, is a crucial parameter in differential privacy. It quantifies the maximum amount of privacy loss that is acceptable in a differentially private algorithm. The smaller the privacy budget, the greater the level of privacy preservation, but this often comes at the expense of the utility or accuracy of the algorithm's output.

Assumption: In the context of our evaluation, ϵ is treated as an independent variable. This is because we measure the quality of the instantiated protocols against the privacy budget, enabling us to observe how changes in ϵ impact the utility-privacy tradeoff. However, for ablations and maximum utility (almost same as non-private), we use $\epsilon = 4$.

Differentially-private Post-processing Strategies: After aggregating the values over all clients, each client ends up having either the noisy sums ($\widehat{S}_i$) and counts ($\widehat{C}_i$), or the noisy centroid ($\widehat{\mu}_i$) for every cluster. Post-processing happens at three levels: sums, counts, and centroids. At each level, we have a choice between two strategies: truncation and folding.

- Truncation strategy: This strategy involves cutting off the values that lie outside the data domain. The result is that any values that exceed the boundaries of the data domain are set to the nearest boundary value.
- Folding strategy: This strategy folds the values that lie outside the data domain back into the domain. It is akin to considering the data domain as a circular space, where exceeding the boundary from one side results in re-entry from the opposite side.

These strategies, described in detail in Subsection 6.1.3, are designed to ensure that the calculated centroids after noise addition stay within the data domain.

Experiment: we show in the evaluation section that doing only folding at the centroid-level provides the best consistent results across most settings. Therefore, we fix this strategy in our evaluations afterwards.

7.1.5 Parameter Summary

In Table 7.1, we show all the parameters for our protocols. We note that the first group has no default values as the aggregation method is set per experiment and the rest of the parameters are all dataset-dependent.

Parameter	**Symbol**	**Default Value**
Aggregation Method	**Agg**	-
Dataset Size	N	-
Number of clusters	k	-
Number of iterations	t_{max}	-
Number of Clients	M	2
Normalization bound	B	1
Ring	$\mathcal{R}$	$\mathcal{Z}_{2^{32}}$
Precision Factor	$\mathbf{\Delta}$	2^{16}
Initialization Method	**Init**	Sphere-packing
Assignment Method	**Ass**	Constrained E-Exclusive
Minimum Cluster Size	c_{min}	$\lceil \frac{N}{1.5*k} \rfloor$
Maximum Cluster Size	c_{max}	$\lceil \frac{3*N}{k} \rfloor$
Maximum Datapoint Contribution	E	1
Privacy Budget	ϵ	4
DP-Sum postprocessing method	**SumPost**	None
DP-Count postprocessing method	**CountPost**	None
DP-Centroid postprocessing method	**CentPost**	Fold

Table 7.1: Summary of the parameters, their symbols, and default values.

7.2 Multiparty DP k-Means with P_S

In this part, we discuss the scenario involving the presence of a semi-honest server, denoted P_S. This server plays a critical role in the computational process. To clarify this process, we break it down into three parts: Multiparty Protocols, Noise Application and Postprocessing.

7.2.1 Multiparty Protocols: Masked

First, we utilize masked multiparty k-Means protocols. These protocols, by design, hide all information from the server P_S. A critical aspect of these protocols is that they can be straightforwardly implemented given that all the participating clients agree on a common seed beforehand. This process removes the need for the server to know any private information while facilitating the computation.

7.2.2 Noise Application: Server

To ensure differential privacy (DP), we need to incorporate Laplace noise in the calculations as per our analysis earlier. This step can be applied by the optional plain addition step in the Masked Secure Aggregation (MSA) protocol detailed in Section 2.2. Specifically, the noise addition is performed by the server, who takes responsibility for this function. In our case, the noise should be converted to fixed-point format to be compatible with the MSA protocol. The server applies this step to every variable v it receives as follows:

1. **Noise Generation:** Generate the Laplace noise (η) according to the differential privacy parameters for v.
2. **Noise Conversion:** Convert the noise to the fixed-point format: $\tilde{\eta} = \lceil \eta \times \mathbf{\Delta} \rfloor$, where $\mathbf{\Delta}$ is the fixed point scale factor.
3. **Noise Addition:** Perform the noise addition to the sum calculation step as a part of the MSA protocol: $Enc(v) \equiv (\tilde{v} + \tilde{\eta}) + r$.

Remark (Utility in Multiparty Setting). In the multiparty context, our algorithm exploits the server's data-oblivious operation. The server, tasked with noise addition, functions as if the algorithm were locally executed on a union of all clients' datasets. This setup boosts the algorithm's utility. Instead of each client individually obfuscating their data

contribution – a process that could reduce overall utility due to cumulative noise – the server's role in introducing noise to the collective data allows us to attain utility at a level of central differential privacy (CDP). This arises from the noise being added once at an aggregate level, limiting total noise in comparison to individual noise addition. However, the server's obliviousness to the differential privacy mechanism's output affords a privacy level similar to local differential privacy (LDP). Hence, we manage to leverage the superior utility intrinsic to CDP, while preserving an LDP-level of trust.

Remark (Quantizing Noise). It is crucial to remark here that converting (or quantizing) the noise to fixed-point format might seem to be contradictory to the requirements needed by differential privacy. This assumption stems from the fact that the postprocessing (the noise quantization in our case) is applied separately to the noise, independent of the original value. One could think of an extreme case where the Laplace noise is quantized into a binary variable and always rounds to zero, effectively destroying all the information-hiding properties of the noise. However, we show why it would be acceptable to quantize the noise to the same level of quantization as the MSA protocol in our case.

Theorem 7.2.1 (Quantization and Differential Privacy)**.** *Quantizing Laplace noise at the same level of quantization as that used in the Masked Secure Aggregation (MSA) protocol does not violate the privacy guarantees offered by differential privacy.*

Proof. We begin by observing that any value, say v, can be split into two parts upon quantization: the integral part, $\bar{v}$, and the fractional part, $\hat{v}$, such that $\bar{v} \times \mathbf{\Delta}$ is the (rounded) integral part of $v \times \mathbf{\Delta}$ (i.e. it is $\lceil v \times \mathbf{\Delta} \rfloor$), and $\hat{v}$ denotes the fractional part that gets lost due to quantization. This notation can similarly be applied to the noise variable, η.

In our context, the sensitive data we wish to protect is $\bar{v}$ since the entire protocol operates in a quantized environment, i.e., v is never transmitted. We aim to demonstrate the following equivalence:

$$\tilde{v} + \tilde{\eta} = \lceil v \times \mathbf{\Delta} \rfloor + \lceil \eta \times \mathbf{\Delta} \rfloor = \lceil (\bar{v} + \eta) \times \mathbf{\Delta} \rfloor$$

The right-hand side represents quantization applied as a postprocessing to the sum of the sensitive data and the noise, which adheres to the rules of differential privacy.

After expanding the right-hand side, we obtain:

$$\lceil (\bar{v} + \eta) \times \mathbf{\Delta} \rfloor = \lceil \bar{v} \times \mathbf{\Delta} + \bar{\eta} \times \mathbf{\Delta} + \hat{\eta} \times \mathbf{\Delta} \rfloor$$

We can take out $\bar{v} \times \mathbf{\Delta}$ and $\bar{\eta} \times \mathbf{\Delta}$ from the rounding operator because they are exact integers (indeed $\bar{v} \times \mathbf{\Delta}$ is $\lceil v \times \mathbf{\Delta} \rfloor$). We then obtain:

$$\lceil (\bar{v} + \eta) \times \mathbf{\Delta} \rfloor = \lceil v \times \mathbf{\Delta} \rfloor + \lceil \eta \times \mathbf{\Delta} \rfloor + \lceil \hat{\eta} \times \mathbf{\Delta} \rfloor$$

However, $\hat{\eta} \times \mathbf{\Delta}$ will be lost due to quantization, hence:

$$\lceil (\bar{v} + \eta) \times \mathbf{\Delta} \rfloor = \lceil v \times \mathbf{\Delta} \rfloor + \lceil \eta \times \mathbf{\Delta} \rfloor = \tilde{v} + \tilde{\eta}$$

In essence, because the sensitive data is already quantized, adding quantized noise to it is equivalent to adding the noise first, then performing the quantization, which aligns with the postprocessing lemma in differential privacy.

This can be intuitively understood by noting that:

- The data before noise addition is already quantized.
- The sum of a quantized value and a non-quantized value will only have information of the non-quantized variable in the less significant bits (i.e., those lost to quantization).
- Hence, quantizing the sum is equivalent to dropping this lower bit information, which is similar to quantizing the second variable prior to addition.

□

7.2.3 Postprocessing: Clients

After the clients receive the noisy masked variables from the server, they unmask as usual, but they also apply the postprocessing strategies needed to ensure that the centroids are within the domain bounds.

We now demonstrate two specific instantiations of Multiparty DP k-Means in the presence of a semi-honest server: one that aggregates the contributions on the level of sums and counts of every cluster, while the other works on the centroid level to completely hide the counts from the clients.

7.2.4 Mask, Noise then Average Protocol (MNA)

In this variant, we propose a combination of the *Masked Multiparty Lloyd's Algorithm (Sum/Count)* and DPLloyd. The key objective here is to improve the privacy guarantees of the former while preserving the clustering accuracy.

In particular, we modify the masked local update phase to inject noise, inspired by the differentially-private mechanism used in DPLloyd. This noise is added by the server and is designed to protect the counts and sums of the local clusters from the participating clients while maintaining the accuracy of the computations.

Protocol Description

The Mask, Noise then Average protocol is executed as *Masked Multiparty Lloyd's Algorithm (Sum/Count)*, but with the following modifications:

- **Noisy Global Update:** Upon receiving the masked values, P_S adds Laplace noise to both the sum and the count, as described in the DPLloyd. The noise for the sum and count is sampled as follows:

  $$\eta^{\Sigma} = L(0, \Delta^{\Sigma}/\epsilon^{\Sigma}), \quad \eta^{C} = L(0, \Delta^{C}/\epsilon^{C})$$

  where $\Delta^{\Sigma} = BE$ and $\Delta^{C} = E$ are the sensitivities of the sum and count queries respectively as discussed in subsection 6.2.1.

  The server then sends the masked and noisy values $(Enc(S_i + \eta^{\Sigma}),\ Enc(C_i + \eta^{C}))$ back to the clients. Each party then unmasks the sums and counts, does the DP postprocessing (whether truncation or folding) and performs the division themselves to compute the new centroids (they can also do postprocessing on the centroid-level):

  $$\widehat{\mu}_i^{(t+1)} = \frac{S_i + \eta^{\Sigma}}{C_i + \eta^{C}}$$

Information Leakage

This protocol essentially follows the *Masked Multiparty Lloyd's Algorithm (Sum/Count)* in terms of leakage. However, all the released variables are now differentially-private, which enhances the privacy of the clients. The information shared among the server and the clients in this protocol is illustrated in Table 7.2.

Entity	Local Variables		Global Variables		
	C_{li}	S_{li}	C_i	S_i	μ_i
P_S	$\times$	$\times$	$\times$	$\times$	$\times$
$P_{l'}$	$l = l'$	$l = l'$	DP	DP	DP

Table 7.2: Leakage analysis of the Mask, Noise then Average protocol

7.2.5 Average, Mask then Noise Protocol (AMN)

To devise a variant of the *Masked Centroid-Level Multiparty Lloyd's Algorithm* that incorporates differential privacy, we introduce the protocol called "Average, Mask then Noise". This protocol follows all the steps of the masked centroid-level algorithm, with an additional step of adding differentially-private noise to the centroids.

Protocol Description

The Average, Mask then Noise Protocol can be summarized in the following steps:

1. **Initialization and Assignment:** These steps are identical to those in the *Masked Multiparty Lloyd's Algorithm (Centroid).*

2. **Masked Local Update:** As in the masked centroid-level algorithm, each party P_l computes and sends the masked version of their (scaled) local centroids, $Enc_l(\tilde{\mu}_{li})$, to the server.

3. **Global Update:** Upon receiving the masked centroids from all clients, the server aggregates them, adds Laplace noise, and returns the masked noisy centroids to the clients:

$$Enc(\widehat{\mu}_i^{(t+1)}) = Enc(\sum_{l=1}^{M} \tilde{\mu}_{li}^{(t)} + \eta^{\mu})$$

   where $\eta^{\mu} = L(0, \Delta^{\mu}/\epsilon^{\mu})$, with Laplace noise generated based on the sensitivity Δ^{μ} and privacy budget per dimension ϵ^{μ} as discussed in subsection 6.3.1.

This additional step ensures differential privacy of the centroids calculated. The "Average, Mask then Noise" protocol reduces the amount of information each party can extract from the global centroid and, in turn, from the other clients' data.

Information Leakage

In this protocol, the P_S receives the masked centroids and does not have access to the local counts, sums or true centroids of any party P_l. This design considerably limits the information leakage to the P_S and among clients. Additionally, the revealed centroids to the clients are now also differentially-private. The shared variables in this protocol are depicted in table 7.3.

	Local Variables		**Global Variables**		
Entity	C_{li}	S_{li}	C_i	S_i	μ_i
P_S	$\times$	$\times$	$\times$	$\times$	$\times$
$P_{l'}$	$l = l'$	$l = l'$	$\times$	$\times$	DP

Table 7.3: Leakage analysis of the Average, Mask then Noise protocol.

7.3 Serverless Multiparty DP k-Means

In order to remove the necessity of a centralized server in multiparty differentially private k-Means, we can design protocols to perform two crucial functions: secure aggregation and addition of Laplace noise. This section outlines the main strategies and the corresponding trade-offs for achieving these functions. A detailed examination and evaluation of these methodologies is considered as part of future work.

7.3.1 Local Privacy

In the context of local privacy, the key is to ensure that for any private variable v, it remains differentially private concerning the client's dataset. One of the straightforward strategies to implement this is to use a secure sum protocol.

In the simplest instance, P_1 could send their value added to a mask $(v_1 + r)$ to P_2. Each successive party then adds their value and forwards the result until it is returned to P_1 again, who unmasks and broadcasts the sum. The critical modification to this procedure for ensuring privacy is that instead of just adding their value, each client adds noise to their variable. This means that the variable remains differentially private on its own.

This approach is simple and fast, with no overhead required for adding differential privacy. However, a significant drawback is that the noise is scaled up by the number of clients, which quickly diminishes the utility of the output.

7.3.2 MPC Privacy

A different approach involves the use of secure multi-party computation (MPC) protocols. Here, clients participate in an MPC protocol to generate additive shares of the Laplace random variable. In the secure sum protocol, each party then adds their share of noise, revealing only the differentially-private sum across all clients.

The advantage of this approach is that it provides the tightest noise addition, emulating the central model with a trusted curator. However, it requires substantial communication and computation between clients, making it expensive in terms of resources.

7.3.3 Splitting Noise

A third approach aims to combine the efficiency of local privacy with a better noise rate, resulting in a compromise between the previous two methods. The principle is to split the Laplace random variable into variables drawn from other distributions such as Gamma, Gaussian, or Laplace distributions. This results in some redundancy in the noise, based on the assumed proportion of honest clients, but it is significantly less than the local privacy approach.

This approach offers an efficient method for adding differential privacy while maintaining a better noise rate. Goryczka et al. [49] provide an extensive study on secure sum with differential privacy protocols that rely completely on this idea. Their work can serve as a guide for implementing this method.

Chapter 8

Evaluation

8.1 Experimental Setup

The experiments were carried out on a hardware system comprising a Macbook Pro M2 Max. This device is equipped with a 30-core CPU, a 38-core GPU and 64 GB of RAM. For the software, the Python programming language (version 3.8) was utilized. Our code is available online at https://git.uwaterloo.ca/a2diaa/privateclustering.

To emulate the conditions of a multi-party computation, the Open MPI [50] (Open Source High-Performance Computing) library was employed. Open MPI is a high-performance message-passing library that enables the simulation of multi-party computation environments, providing functionalities for developing parallel applications using the Message Passing Interface (MPI) standard.

Network conditions were also taken into account during the setup. A custom delay was integrated into the system to simulate both Local Area Network (LAN) and Wide Area Network (WAN) conditions. For the LAN simulation, a round trip delay of 1 millisecond was set, whereas for the WAN simulation, the round trip delay was set at 100 milliseconds. These settings enable us to account for network latency in real-world scenarios, thus increasing the practical relevance of the experiment.

8.2 Dataset Descriptions

The following datasets were used in the evaluation of the multiparty clustering algorithm. A summary of these datasets is also presented in Table 8.1.

- **Image** [51]**:** This dataset is derived from an image of a house and contains 34112 RGB vectors forming three clusters: house, sky, and frames.
- **House** [51]**:** This is a quantized variant of the 'Image' dataset. It consists of 1837 RGB pixels and the data forms three clusters. This dataset allows for comparison of the algorithm's performance on different sizes/transformations of the same raw data.
- **Iris** [52]**:** This classic dataset from Fisher (1936) contains measurements for 150 samples of iris flowers, across four dimensions: petal length, petal width, sepal length, and sepal width. The samples are expected to form three clusters, corresponding to the Iris Setosa, Iris Versicolour, and Iris Virginica species. The dataset allows for evaluation of the algorithm's performance on well-studied, multi-dimensional data with clear ground truth.
- **Adult** [53]**:** This is a subset of the Census Income dataset, used to predict whether an individual's income exceeds \$50K/yr based on census data. It includes six numerical features, with the data expected to form three clusters. The high dimensionality of this dataset poses challenges for clustering, which can be used to evaluate the algorithm's effectiveness in higher-dimensional spaces.
- **S1** [54]**:** This synthetic dataset serves as a benchmark for studying clustering schemes. It consists of 5000 two-dimensional data points grouped into 15 Gaussian clusters. Since the dataset is two-dimensional, it can be visualized easily, enabling the qualitative assessment of the clustering results.
- **Birch2** [55]**:** A synthetic dataset comprised of 100 Gaussian clusters forming a sine wave. This dataset initially contains 100K data points, however, a random sample of 25K points is used for demonstration purposes. The complex structure of this dataset is useful for evaluating the algorithm's performance on complex and synthetic geometric shapes.
- **Synth**: This is a synthetic dataset of 5000 2D datapoints, to be clustered into 2 clusters. We use this dataset for timing evaluations with prior work.

When distributing the dataset across multiple clients, we shuffle and split the dataset equally. Evaluations are done on the union of all these splits (as if it was done centrally).

Table 8.1: Summary of Datasets Used in Evaluation

Dataset	N	d	k
Image [51]	34112	3	3
House [51]	1837	3	3
Iris [52]	150	4	3
Adult [53]	48842	6	3
S1 [54]	5000	2	15
Birch2 [55]	25000	2	100
Synth	5000	2	2

8.3 Implementation Details

The implementation of the experiment required the use of several software libraries and tools to ensure the accuracy and efficiency of the computations.

The differential privacy mechanism, specifically the Laplace mechanism, was implemented using the IBM's diffprivlib library [56]. This library is designed to provide a robust and user-friendly interface for the application of differential privacy in data analysis.

The Google's ortools [57] library was employed for solving the min-cost flow problem, which is integral to the Constrained-E-Exclusive and Constrained Exclusive assignments in our experiment. Ortools is renowned for its efficient and high-level interface to several combinatorial optimization solvers, and its use here allowed for efficient optimization of the min-cost flow problems.

The sphere-packing initialization and tuncation and folding postprocessing methods were adapted from the implementation provided in the diffprivlib library, as described by Su et al. [18]. These methods were necessary for the initialization and postprocessing stages of differentially-private instantiations, and their adaptation from an established implementation helped ensure that these stages were carried out correctly and efficiently.

8.4 Parameter Selection

To select the parameters for our quality experiments, we fix the default parameters in Table 7.1, but ablate over the parameter's choices, trying them over all the datasets and aggregation methods, to select the best initialization method and postprocessing strategy. To allow for randomness, we run every experiment 10 times and take the average.

	s1	house	iris	image	adult	birch2	BC
(none, none, fold)	**82.8**	**286.3**	33.7	3559.5	12295.4	**45.7**	3
(fold, none, fold)	83.0	287.1	33.9	**3559.4**	**12294.6**	482.5	2
(none, none, trunc)	**82.8**	288.6	33.3	3559.8	12295.2	**45.3**	2
(trunc, fold, trunc)	103.2	288.1	**32.8**	3559.6	**12294.8**	181.3	2
(fold, fold, trunc)	104.4	**286.4**	33.1	3559.5	12295.0	175.7	1
(fold, none, none)	82.9	**286.3**	33.1	3559.6	12295.3	475.4	1
(fold, none, trunc)	83.0	287.9	**32.9**	3559.7	12295.2	477.7	1
(none, fold, fold)	102.8	291.0	34.7	3559.6	**12294.9**	325.0	1
(none, fold, trunc)	102.8	289.2	**32.9**	3559.7	12295.1	179.7	1
(none, none, none)	83.2	294.8	33.0	3559.6	12295.5	**45.5**	1
(trunc, fold, none)	112.6	289.8	33.0	**3559.4**	12295.3	174.6	1
(trunc, none, none)	82.9	288.1	33.2	**3559.4**	12295.5	298.7	1
(trunc, none, trunc)	**82.8**	287.0	33.5	3559.5	12295.3	295.9	1

Table 8.2: Ablation of postprocessing strategies for NMA aggregation

8.4.1 Postprocessing Methods

For Sum/Count-level (NMA) aggregation, postprocessing can happen on three variables: noisy sums ($\widehat{S}_i$) and counts ($\widehat{C}_i$), or the noisy centroid ($\widehat{\mu}_i$). For each of these variables, there are three choices: no postprocessing (none), folding (fold) and truncating (trunc). These constitute 27 experiments for every dataset under every privacy budget ϵ. We calculate the Best Count (BC) for every method; which is the number of times the method was in the top three methods across all six datasets. The results are show in Table 8.2 for the top 13 strategies (the ones that had $BC > 0$). We observe that the (none, none, fold) strategy behaves most consistently and outperforms other methods. For non-default values of ϵ, we show the plots for the top ten strategies for every dataset in Figure B.1.

Similarly, for Centroid-level (AMN) aggregation, we do the same experiments but only for the centroid-level postprocessing methods. For this experiment, BC denotes the number of times every method was the top method across all datasets. As we can see from the results in Table 8.3, folding on the centroid-level still performs best. For non-default values of ϵ, we show the plots for every dataset in Figure B.2.

These experiments allow us to fix the strategy of only folding on the centroid-level. Intuitively, folding the centroids back into the domain would allow the centroid to be exposed to more datapoints in the next iteration which allows it to provide a better performance. This is opposed to truncation which would throw the centroid to the edge of the domain

	s1	house	iris	image	adult	birch2	BC
fold	81.7	**284.4**	**32.1**	3560.2	12296.8	**42.0**	3
trunc	**81.6**	284.6	32.5	**3560.0**	12296.8	42.1	2
none	81.7	284.7	32.7	**3560.0**	**12296.7**	42.3	1

Table 8.3: Ablation of postprocessing strategies for AMN aggregation

	s1	house	iris	image	adult	birch2	TNW	BC
sphere-packing	**83.2**	294.8	**32.8**	**3558.8**	12178.3	**47.0**	0.12522	3
random	117.3	**285.0**	36.7	**3558.8**	**12145.4**	96.7	0.13002	3

Table 8.4: Ablation of Initialization methods for NMA aggregation

where it's more susceptible to outliers.

8.4.2 Initialization Methods

We do a similar experiment to evaluate which of the initialization methods would be best: Random or Sphere-packing. However, we notice that the BC measure does not favor a method over the other. Therefore, we calculate the Total Normalized WCSS (TNW), which is calculated as the sum of $\frac{WCSS(X)}{|X|}$ for every dataset X. The idea is that the normalization allows us to get one score that governs all datasets at the same time. With this metric, we notice in both Tables 8.4 and Table 8.5 that the sphere-packing method has a better TNW, so we fix it in our evaluations.

For non-default values of ϵ, we show the plots for every dataset in Figure A.1 and Figure A.2.

	s1	house	iris	image	adult	birch2	TNW	BC
sphere-packing	**81.7**	284.7	**32.4**	3558.9	12180.1	**43.5**	0.12379	3
random	115.6	**283.8**	33.3	**3557.9**	**12145.7**	92.8	0.12605	3

Table 8.5: Ablation of Initialization methods for AMN aggregation

8.5 Quality Evaluation

For the purpose of quality evaluations, we adhere to the default parameters as specified in table 7.1, with specific exceptions as follows: We delineate the evaluation metrics in correspondence with the parameter ϵ, which is systematically varied within the range of 0.25 to 4. Our evaluation encompasses four distinct settings:

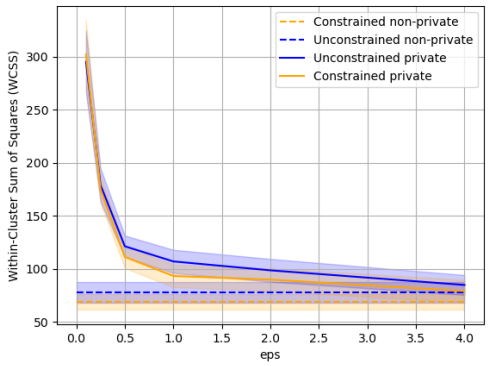
1. **Unconstrained Non-private:** As a benchmark method, the default multiparty Lloyd's algorithm is utilized, with unconstrained assignment and no differential privacy.
2. **Constrained Non-private:** This approach is designed to demonstrate how constraining the cluster sizes can even enhance the performance of the non-private method.
3. **Unconstrained Private:** The state-of-the-art DP method by Su et al. [18] is instantiated in our multiparty settings, representing the current forefront in differentially private multiparty clustering.
4. **Constrained Private:** A specialized setup to illustrate how imposing constraints on the cluster sizes significantly improves upon the current state-of-the-art in differentially private k-Means clustering.

All the settings are tested for both the NMA and AMN protocols. For the sake of statistical robustness, the evaluation is repeated 20 times. Subsequently, the mean values are computed and plotted with the shaded area signifying the corresponding 95% confidence intervals.

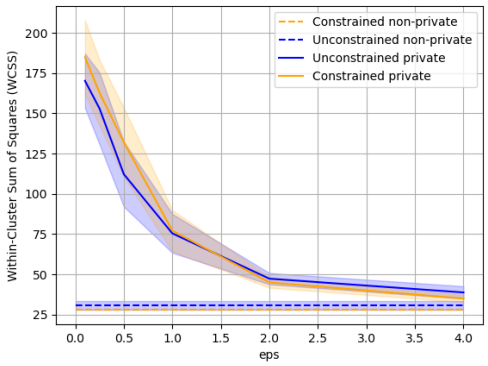
8.5.1 Within-Cluster Sum of Squares

NMA Approach: In our investigation of the NMA aggregation protocol, the observations can be categorized based on the size of the datasets:

- **Small Datasets** (s1, iris, house): The constrained methods manifest very similar or slightly superior performance when compared with the unconstrained versions as seen in Figure 8.1



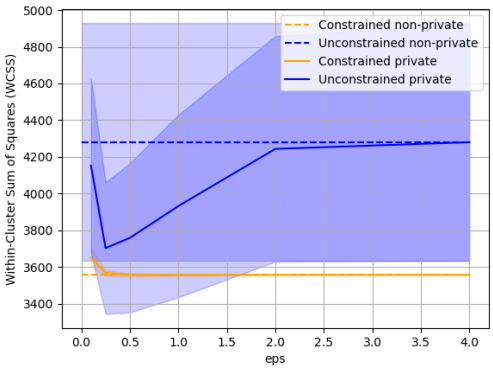
(a) S1



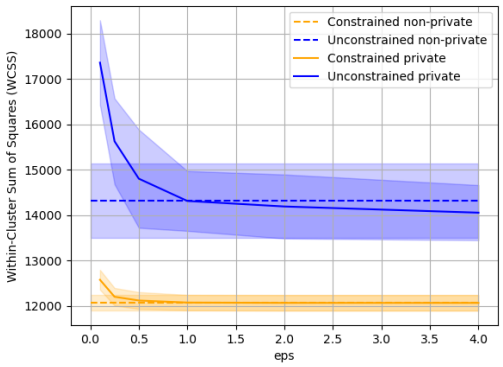
(b) Iris

Figure 8.1: WCSS against ϵ for NMA aggregation: small datasets

- **Large Datasets** (birch2, adult, image): The constrained methods exhibit a statistically significant enhancement in performance over the unconstrained counterparts. In particular, for the adult and image datasets, the private constrained method surpasses even the non-private unconstrained method as seen in Figure 8.2. For the image dataset, it should be noted that the performance of unconstrained versions is characterized by high variance, rendering it highly unstable.



(a) Image



(b) Adult

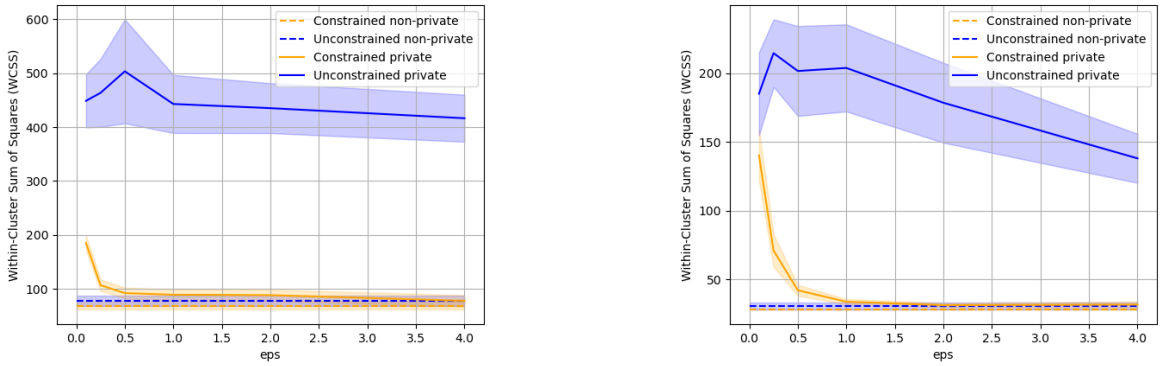
Figure 8.2: WCSS against ϵ for NMA aggregation: large datasets

A comprehensive visualization of the WCSS against ϵ for the NMA aggregation for all

datasets can be found in Figure C.1.

AMN Approach: The observations pertaining to the AMN approach reveal:

- The private unconstrained approach exhibits an extremely subpar performance across all datasets. This is an anticipated outcome, as the differentially-private noise introduced at the centroid level can hide all the information and consequently diminish the utility.
- The constrained methods consistently outperform the corresponding unconstrained methods across all datasets. In some instances, the private constrained method even surpass the non-private unconstrained method, underscoring the efficacy of constraining cluster sizes.

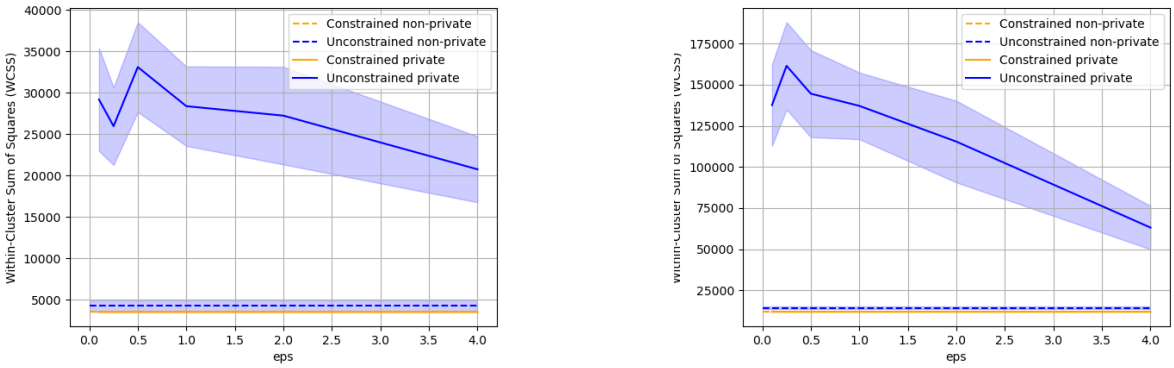


(a) S1 (b) Iris

Figure 8.3: WCSS against ϵ for AMN aggregation: small datasets

We show results of the same datasets in Figure 8.3 and Figure 8.4. For a detailed graphical representation of the WCSS for all datasets, check Figure C.2.

In summary, the findings of our comprehensive evaluation reveal the palpable improvements conferred by constrained methods. These improvements are particularly pronounced in large datasets, where they lead to significant enhancements in WCSS, thereby reaffirming the utility of constraints in both private and non-private multiparty clustering scenarios.



(a) Image (b) Adult

Figure 8.4: WCSS against ϵ for AMN aggregation: large datasets

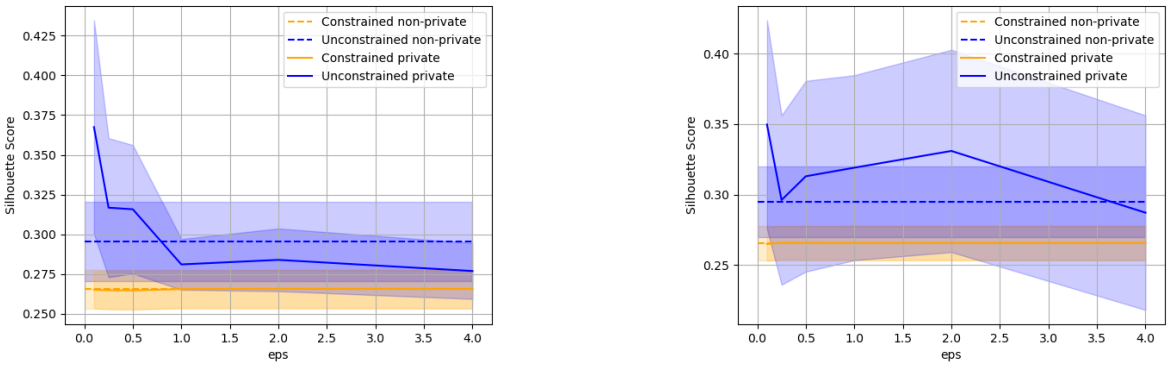
8.5.2 Silhouette Score

The Silhouette Score, employed as another crucial metric, largely reflects the observations made in the analysis of the Within-Cluster Sum of Squares (WCSS) for both NMA and AMN approaches. However, there exists a notable exception:

Adult Dataset: The results pertaining to the adult dataset exhibit inconsistency with regard to the Silhouette Score. Since the Silhouette Score serves as a measure of cluster separation, it is sensitive to the specific characteristics of the dataset. In the adult dataset, the clusters are not distinct but rather overlapping and in close proximity. Consequently, solutions that may appear more optimal in terms of WCSS might render a less favorable Silhouette Score. We can observe this in Figure 8.5.

The observations for the Silhouette Score against ϵ for both the NMA aggregation for other datasets largely parallel those outlined in the WCSS subsection. We show here the resuls for Birch2 dataset in Figure 8.6 and Iris dataset in Figure 8.7. For all other datasets, check Figure D.1 and Figure D.2.

In conclusion, the evaluation of the Silhouette Score underscores the efficacy of the constrained methods in enhancing multiparty clustering. Yet, it also brings to light the nuanced complexity inherent in evaluating clustering algorithms, as evidenced by the adult dataset, where traditional correlations between WCSS and Silhouette Score do not necessarily hold. Such insights emphasize the necessity of a multifaceted evaluation strategy that takes into account the specific characteristics of individual datasets.



(a) NMA (b) AMN

Figure 8.5: Silhouette Score against ϵ for Adult dataset

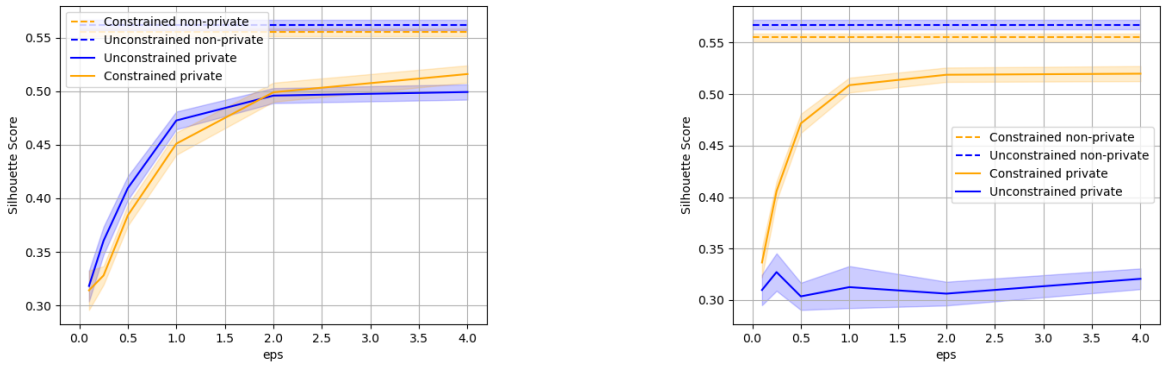
8.5.3 Empty Clusters

The k-Means clustering algorithm's intrinsic characteristic is its design to yield exactly k clusters. However, the optimization of standard objective functions such as WCSS or the Silhouette Score, while conducive to quality clustering and separation, does not necessarily guarantee the primary objective: the precise formation of k clusters.

This discrepancy is particularly pronounced in datasets demanding a high number of clusters, where the application of differential privacy methods can lead to significant perturbations.

Evaluation on Birch2 Dataset: To illustrate this phenomenon, we present an evaluation focusing on the average number of empty clusters within the Birch2 dataset. It is pertinent to highlight that our study is the pioneering effort to evaluate differentially-private clustering on the entirety of the Birch2 cluster space (with $k = 100$) — a feat previously unexplored in other works.

- **Unconstrained Non-private:** In this approach, the average number of empty clusters is approximately 15 and 20 for the NMA and AMN protocols, respectively.
- **Constrained Non-private:** In this domain, the constrained method intrinsically results in 0 empty clusters.



(a) NMA

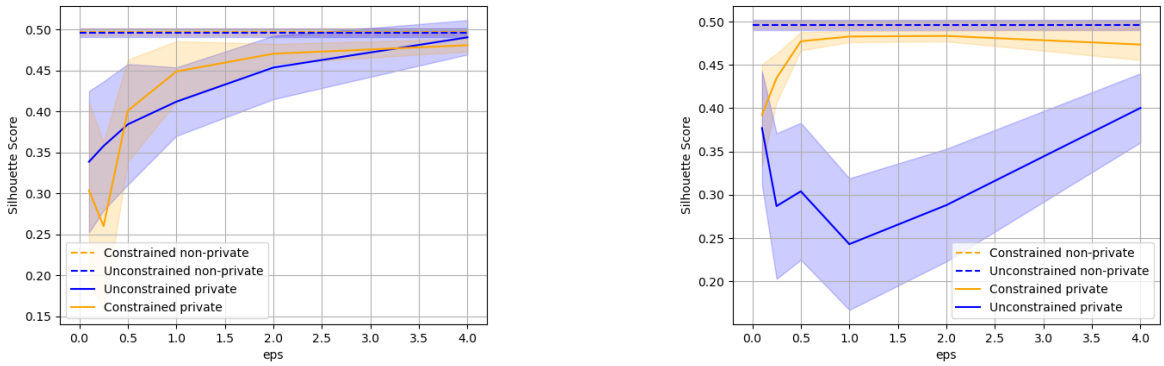
(b) AMN

Figure 8.6: Silhouette Score against ϵ for Birch2 dataset

- **Unconstrained Private:** This approach results in approximately 10 empty clusters in NMA and sustains around 22 to 25 empty clusters for AMN, thereby comparing unfavorably with the constrained method.
- **Constrained Private:** The introduction of differentially-private noise during the final step of the protocol (the global update phase) prevents this method from achieving 0 empty clusters. Nevertheless, it manifests an impressive performance, averaging around 4 empty clusters for the NMA protocol at $\epsilon = 1$, which swiftly declines to just 1 empty cluster. An even more favorable performance is observed in the AMN protocol, commencing with around 2 empty clusters at $\epsilon = 1$ and rapidly diminishing as well.

The variations in the average number of empty clusters across the different methods and protocols for the Birch2 dataset are graphically represented in Figure 8.8.

In summary, our analysis uncovers the critical nuances of empty cluster formation in multiparty clustering, especially within large cluster spaces. The findings accentuate the strength of constrained methods in achieving a more faithful representation of k clusters, while also delineating the impact of differentially-private noise.



(a) NMA

(b) AMN

Figure 8.7: Silhouette Score against ϵ for Iris dataset

8.6 Timing Evaluation

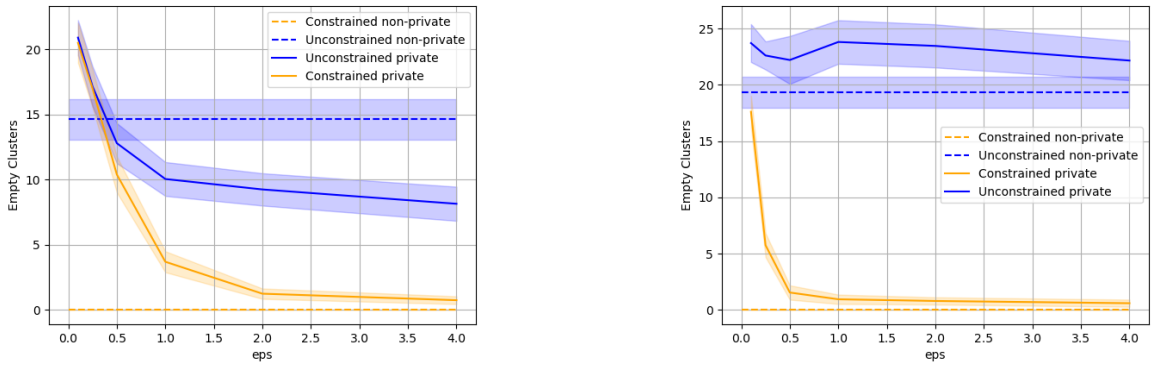
In this section, we conduct a rigorous timing evaluation of our proposed protocols on various datasets, and compare them with recent state-of-the-art methods in the serverless and outsourced settings. For these experiments, we ran the protocol 20 times and took the average, and computed the 95% confidence interval.

8.6.1 Comparison with Mohassel et al. (serverless)

Mohassel et al. [8] processed the S1 dataset in 1472.6 seconds over 30 iterations, leading to an average time of 49.2 seconds per iteration in a very fast local area network (LAN) with 0.02 ms latency. In comparison, our protocols take between 31 and 33 ms per iteration in a LAN with 0.1 ms latency, translating to an approximate speedup of 1500 times, a significant improvement of three orders of magnitude.

8.6.2 Comparison with Jiang et al. (outsourced)

Jiang et al. [14] use synthetic datasets like Synth. For 5000 datapoints, their approach takes 32.64 seconds per iteration in a LAN setting. In contrast, our protocols perform the same computations in between 7 and 9 ms per iteration, yielding an approximate speedup of 4000 times, and achieving an improvement of three orders of magnitude.



(a) NMA (b) AMN

Figure 8.8: Average Number of Empty Clusters in Birch2 Dataset

Dataset	LAN Time/iter (ms)	WAN Time/iter (ms)
iris	60 ± 16	155 ± 15
image	75 ± 8	197 ± 6
house	46 ± 8	148 ± 9
adult	263 ± 10	389 ± 7
birch2	1029 ± 11	1126 ± 9
s1	33 ± 6	141 ± 3
synth	9 ± 2	122 ± 1

Table 8.6: Time per iteration for NMA aggregation

8.6.3 Other Runtime Comparisons

The other runtimes and comparisons are presented in Table 8.6 and Table 8.7. These tables contain the datasets, the number of iterations, and the corresponding time per iteration in both LAN and wide area network (WAN) settings for two methods: NMA and AMN.

The timing evaluation demonstrates that our protocols achieve remarkable speedup over existing methods, in both LAN and WAN settings. This significant efficiency makes the protocols suitable for real-world deployment in various data-intensive applications.

Dataset	LAN Time/iter (ms)	WAN Time/iter (ms)
iris	68 ± 15	143 ± 14
image	71 ± 9	197 ± 5
house	42 ± 5	150 ± 5
adult	259 ± 8	387 ± 8
birch2	1045 ± 23	1135 ± 17
s1	31 ± 4	140 ± 2
synth	7 ± 1	123 ± 1

Table 8.7: Time per iteration for AMN aggregation

Chapter 9

Conclusion

This dissertation has provided a systematic investigation into privacy-preserving techniques for the Euclidean k-Means problem in federated settings. The results represent methodical advancements in the area of machine learning with implications for real-world applications. The key contributions are summarized as follows:

- **Integration of Differential Privacy:** We introduced a framework for integrating Differential Privacy (DP) into Horizontally-Federated k-Means, extending existing methodologies. The approach avoided the direct combination of existing methods, leading to a design that improved privacy without sacrificing accuracy.
- **Lightweight Aggregation Protocol:** The introduction of a lightweight aggregation protocol offered a staggering three orders of magnitude speedup over existing multiparty approaches. Such efficiency propels our methodology to the forefront of practical applicability, setting a new benchmark for federated k-Means solutions.
- **Application of Constraints:** The incorporation of cluster-size constraints in DP k-Means resulted in improved utility. This novel application provides an additional perspective on the role of constraints in the centralized differential privacy model.
- **Detailed Analysis:** A comprehensive examination of various aggregation methods and their applications in the protocol was conducted, providing insights into the behavior and performance of different methods in multiparty clustering.
- **Evaluation of Performance:** Through rigorous evaluation, the research demonstrated improvements in large datasets, and the timing evaluation confirmed the pro-

tocols' efficiency in different network settings. These results suggest the feasibility of the proposed protocols in real-world scenarios.

In conclusion, the findings presented in this dissertation contribute to the ongoing discourse in privacy-preserving machine learning, with specific emphasis on the k-Means problem in federated settings. The work builds on existing literature by proposing new methodologies and offers an analytical evaluation of these methods, contributing to both theoretical understanding and practical implementation. The results open new paths for further investigation and refinement in this complex and essential area of research.

References

[1] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, (New York, NY, USA), p. 206–215, Association for Computing Machinery, 2003.

[2] S. Jha, L. Kruger, and P. McDaniel, "Privacy preserving clustering," in *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, (Berlin, Heidelberg), p. 397–417, Springer-Verlag, 2005.

[3] G. Jagannathan and R. N. Wright, "Privacy-preserving distributed k-means clustering over arbitrarily partitioned data," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, (New York, NY, USA), p. 593–599, Association for Computing Machinery, 2005.

[4] G. Jagannathan, K. Pillaipakkamnatt, R. Wright, and D. Umano, "Communication-efficient privacy-preserving clustering," *Transactions on Data Privacy*, vol. 3, pp. 1–25, 04 2010.

[5] Z. Gheid and Y. Challal, "Efficient and privacy-preserving k-means clustering for big data mining," pp. 791–798, 08 2016.

[6] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, (New York, NY, USA), p. 486–497, Association for Computing Machinery, 2007.

[7] A. Jäschke and F. Armknecht, "Unsupervised machine learning on encrypted data," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 411, 2018.

[8] P. Mohassel, M. Rosulek, and N. Trieu, "Practical privacy-preserving k-means clustering," *Proceedings on Privacy Enhancing Technologies*, vol. 2020, pp. 414 – 433, 2020.

[9] S. Patel, S. Garasia, and D. Jinwala, "An efficient approach for privacy preserving distributed k-means clustering based on shamir's secret sharing scheme," in *IFIP Advances in Information and Communication Technology*, pp. 129–141, Springer Berlin Heidelberg, 2012.

[10] J. Yuan and Y. Tian, "Practical privacy-preserving mapreduce based k-means clustering over large-scale dataset," *IEEE Transactions on Cloud Computing*, vol. 7, no. 2, pp. 568–579, 2019.

[11] F.-Y. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, "Privacy-preserving and outsourced multi-user k-means clustering," in *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*, pp. 80–89, 2015.

[12] L. Xiaoyan, Z. Jiang, S. Yiu, X. Wang, C. Tan, L. Ye, Z. Liu, Y. Jin, and J. Fang, "Outsourcing two-party privacy preserving k-means clustering protocol in wireless sensor networks," pp. 124–133, 12 2015.

[13] A. Silva and G. Bellala, "Privacy-preserving multi-party clustering: An empirical study," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 326–333, 2017.

[14] Z. L. Jiang, N. Guo, Y. Jin, J. Lv, Y. Wu, Z. Liu, J. Fang, S. Yiu, and X. Wang, "Efficient two-party privacy-preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing," *Information Sciences*, vol. 518, pp. 168–180, 2020.

[15] S. Yaji and N. Bayyapu, "Result attack: a privacy breaching attack for personal data through k-means algorithm," *Cyber-Physical Systems*, vol. 7, pp. 11–40, Aug. 2020.

[16] A. Blum, C. Dwork, F. McSherry, and K. Nissim, "Practical privacy: the SuLQ framework," in *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pp. 128–138, Association for Computing Machinery, 2005.

[17] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, vol. 54, no. 1, pp. 86–95, 2011.

[18] D. Su, J. Cao, N. Li, E. Bertino, and H. Jin, "Differentially private k-means clustering," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY '16, pp. 26–37, Association for Computing Machinery, 2016.

[19] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "Gupt: privacy preserving data analysis made easy," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 349–360, ACM, 2012.

[20] J. Zhang, X. Xiao, Y. Yang, Z. Zhang, and M. Winslett, "Privgene: differentially private model fitting using genetic algorithms," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 665–676, ACM, 2013.

[21] M.-F. Balcan, T. Dick, Y. Liang, W. Mou, and H. Zhang, "Differentially private clustering in high-dimensional Euclidean spaces," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, pp. 322–331, PMLR, 06–11 Aug 2017.

[22] Z. Lu and H. Shen, "Differentially private k-means clustering with guaranteed convergence," pp. 1–1.

[23] M. Park, J. Foulds, K. Choudhary, and M. Welling, "Dp-em: Differentially private expectation maximization," in *Artificial Intelligence and Statistics*, pp. 896–904, 2017.

[24] T. Ni, M. Qiao, Z. Chen, S. Zhang, and H. Zhong, "Utility-efficient differentially private k-means clustering based on cluster merging," *Neurocomputing*, vol. 424, pp. 205–214, 2021.

[25] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), pp. 1175–1191, Association for Computing Machinery, 2017.

[26] K. Bennett, P. Bradley, and A. Demiriz, "Constrained k-means clustering," Tech. Rep. MSR-TR-2000-65, May 2000.

[27] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of euclidean sum-of-squares clustering," *Machine Learning*, vol. 75, pp. 245–248, Jan. 2009.

[28] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, Mar. 1982.

[29] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[30] J. C. Dunn†, "Well-separated clusters and optimal fuzzy partitions," *Journal of Cybernetics*, vol. 4, no. 1, pp. 95–104, 1974.

[31] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979.

[32] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, pp. 846–850, Dec. 1971.

[33] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *Journal of Machine Learning Research*, vol. 11, no. 95, pp. 2837–2854, 2010.

[34] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, pp. 553–569, Sept. 1983.

[35] A. Vattani, "k-means requires exponentially many iterations even in the plane," *Discrete & Computational Geometry*, vol. 45, pp. 596–616, Mar. 2011.

[36] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 881–892, July 2002.

[37] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), p. 1177–1178, Association for Computing Machinery, 2010.

[38] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography* (S. Halevi and T. Rabin, eds.), (Berlin, Heidelberg), pp. 265–284, Springer Berlin Heidelberg, 2006.

[39] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, (New York, NY, USA), p. 19–30, Association for Computing Machinery, 2009.

[40] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?," *SIAM Journal on Computing*, vol. 40, no. 3, pp. 793–826, 2011.

[41] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 429–438, 2013.

[42] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, p. 612–613, nov 1979.

[43] D. Liu, E. Bertino, and X. Yi, "Privacy of outsourced k-means clustering," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '14, (New York, NY, USA), p. 123–134, Association for Computing Machinery, 2014.

[44] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, pp. 223–238, Springer Berlin Heidelberg.

[45] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, IEEE, Oct. 2007.

[46] Z. Li, T. Wang, and N. Li, "Differentially private vertical federated clustering," *Proc. VLDB Endow.*, vol. 16, p. 1277–1290, apr 2023.

[47] E. Zhang, H. Li, Y. Huang, S. Hong, L. Zhao, and C. Ji, "Practical multi-party private collaborative k-means clustering," *Neurocomputing*, vol. 467, pp. 256–265, 2022.

[48] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," vol. 8, pp. 1027–1035, 01 2007.

[49] S. Goryczka and L. Xiong, "A comprehensive comparison of multiparty secure additions with differential privacy," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 5, pp. 463–477, 2017.

[50] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, (Budapest, Hungary), pp. 97–104, September 2004.

[51] P. Fränti and S. Sieranoja, "K-means properties on six clustering benchmark datasets," 2018.

[52] R. A. Fisher, "Iris." UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[53] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[54] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems," *Pattern Recognition*, vol. 39, no. 5, pp. 761–765, 2006.

[55] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 141–182, 1997.

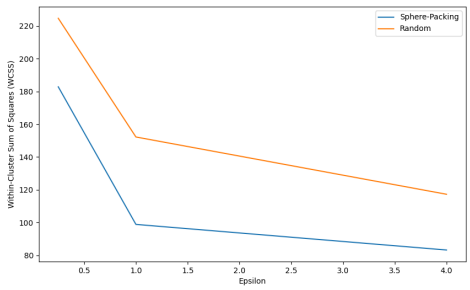
[56] N. Holohan, S. Braghin, P. Mac Aonghusa, and K. Levacher, "Diffprivlib: the IBM differential privacy library," *ArXiv e-prints*, vol. 1907.02444 [cs.CR], July 2019.

[57] L. Perron and V. Furnon, "Or-tools," Mar 2023.

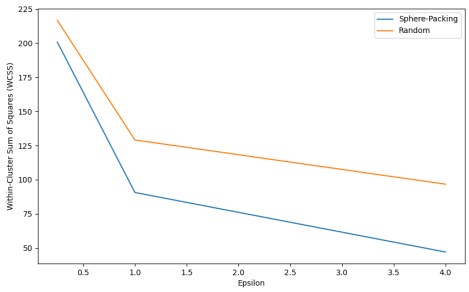
APPENDICES

Appendix A

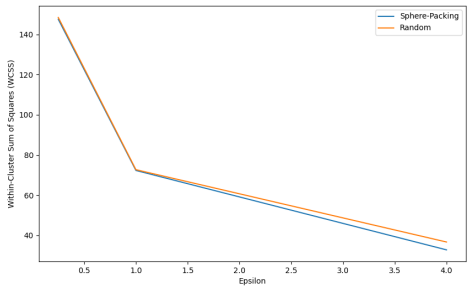
Ablation Plots: Initialization



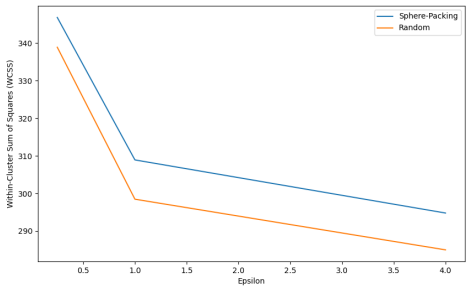
(a) S1



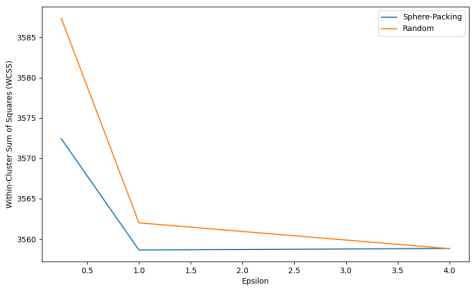
(b) Birch2



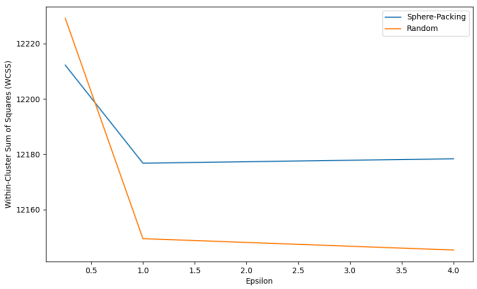
(c) Iris



(d) House

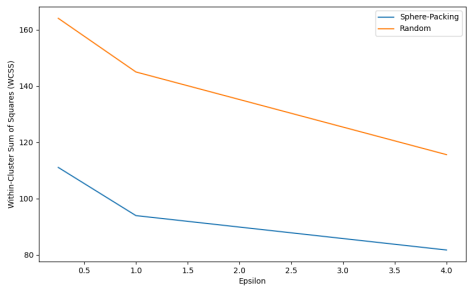


(e) Image

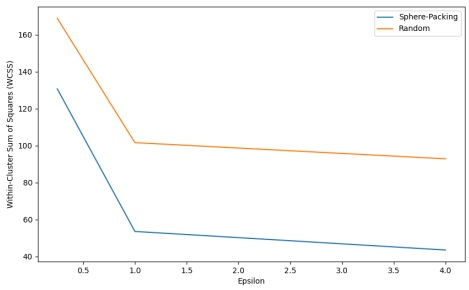


(f) Adult

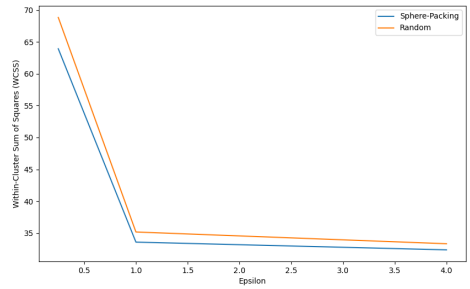
Figure A.1: WCSS against ϵ for NMA aggregation under different Initialization strategies



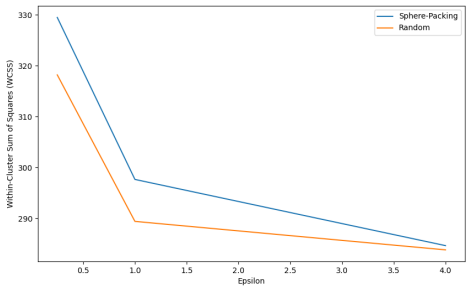
(a) S1



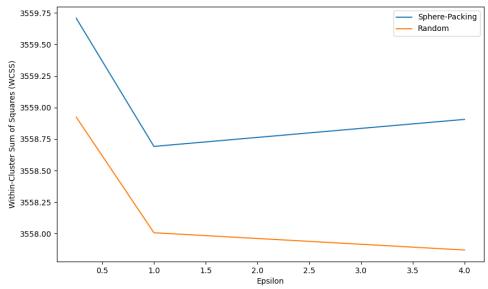
(b) Birch2



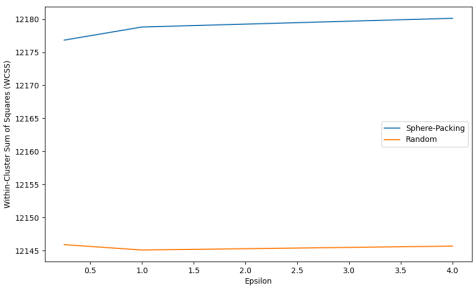
(c) Iris



(d) House



(e) Image

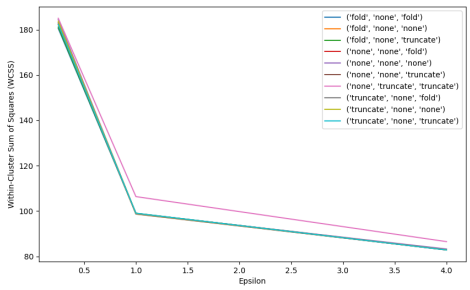


(f) Adult

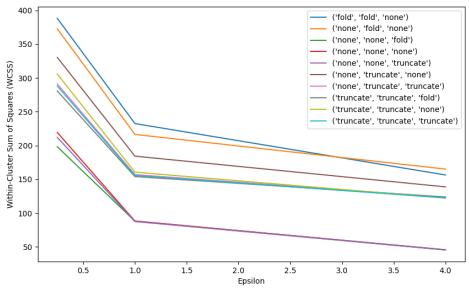
Figure A.2: WCSS against ϵ for AMN aggregation under different Initialization strategies

Appendix B

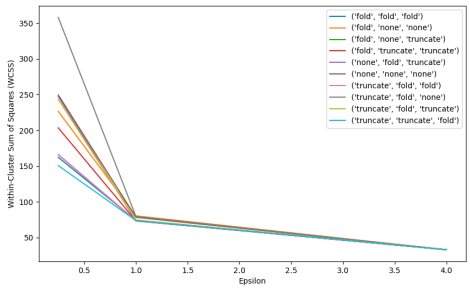
Ablation Plots: Postprocessing



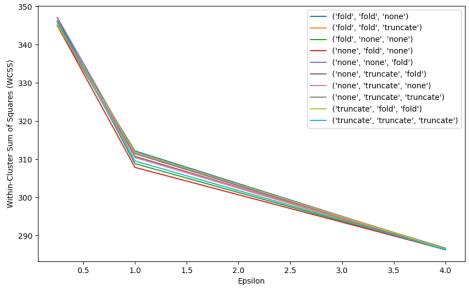
(a) S1



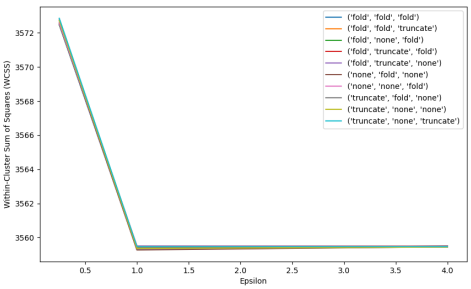
(b) Birch2



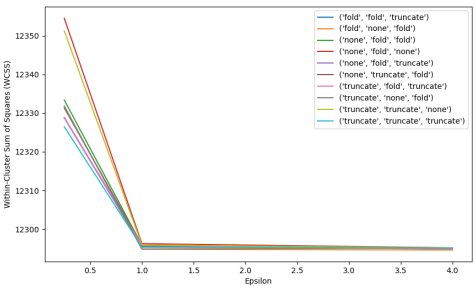
(c) Iris



(d) House

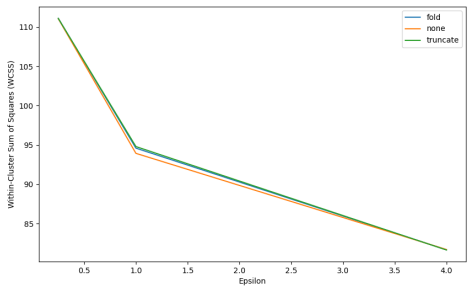


(e) Image

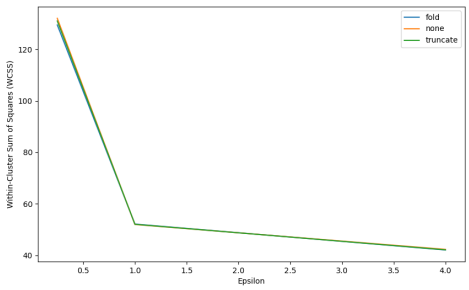


(f) Adult

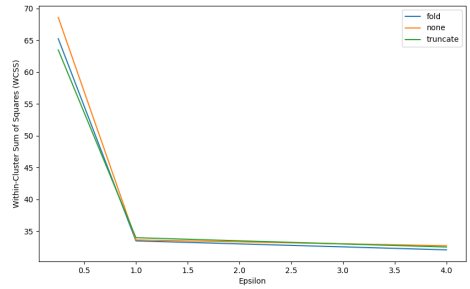
Figure B.1: WCSS against ϵ for NMA aggregation under different postprocessing strategies



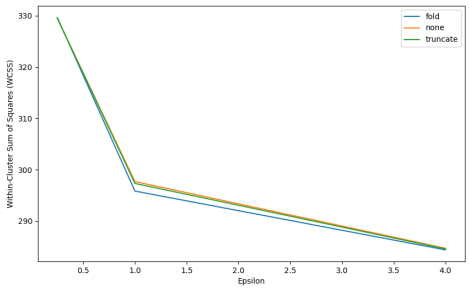
(a) S1



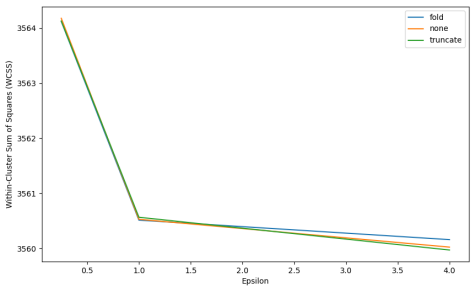
(b) Birch2



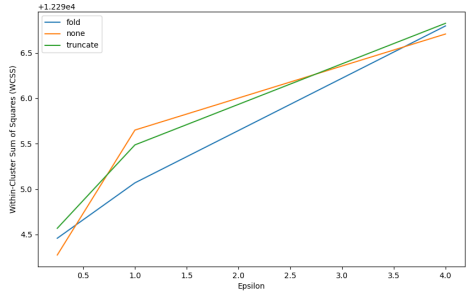
(c) Iris



(d) House



(e) Image

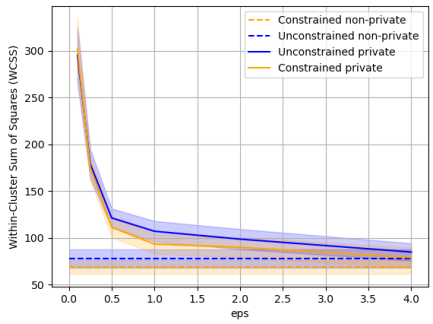


(f) Adult

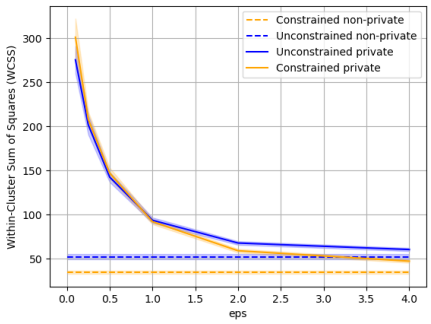
Figure B.2: WCSS against ϵ for AMN aggregation under different postprocessing strategies

Appendix C

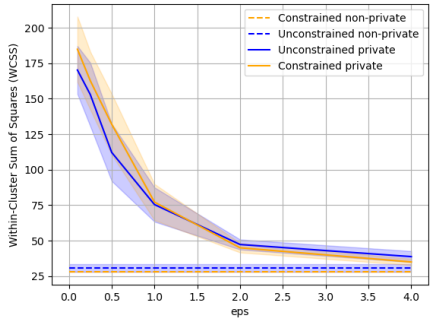
Quality Plots: WCSS



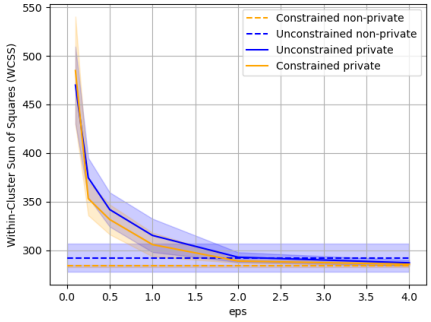
(a) S1



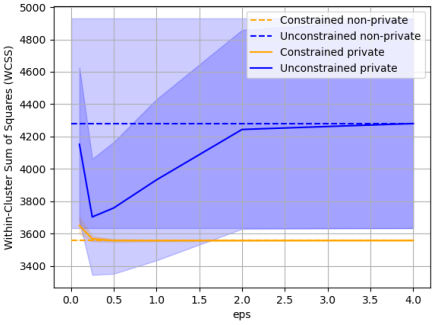
(b) Birch2



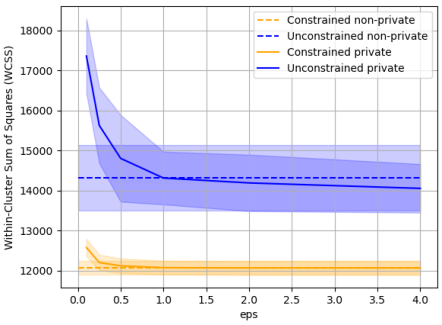
(c) Iris



(d) House

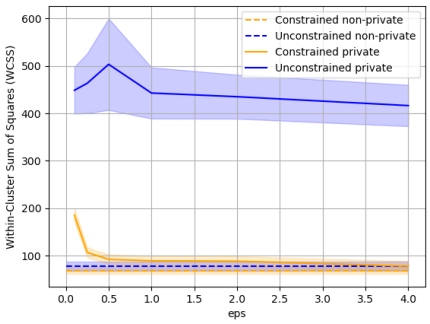


(e) Image

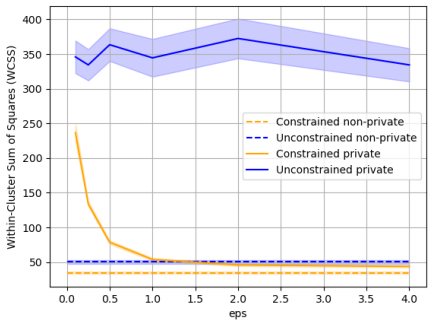


(f) Adult

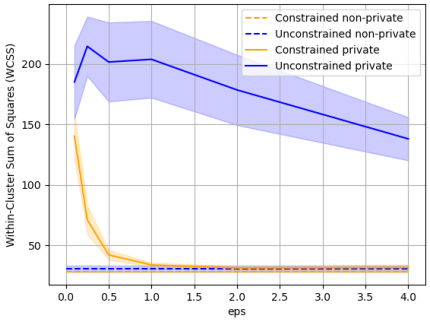
Figure C.1: WCSS against ϵ for NMA aggregation



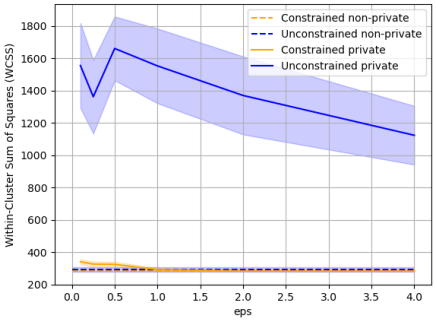
(a) S1



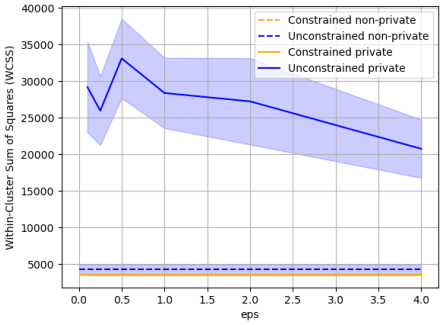
(b) Birch2



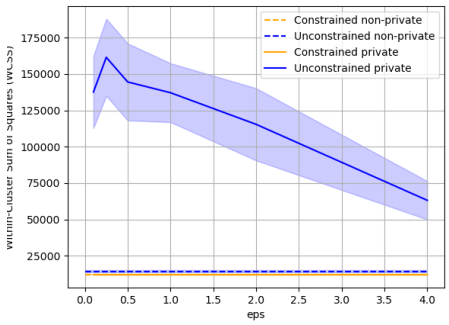
(c) Iris



(d) House



(e) Image

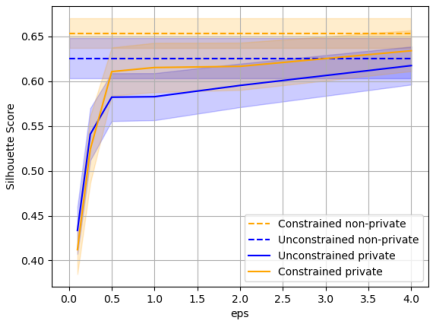


(f) Adult

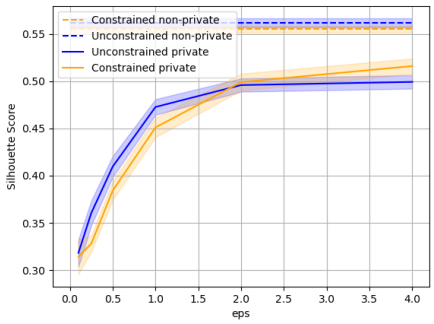
Figure C.2: WCSS against ϵ for AMN aggregation

Appendix D

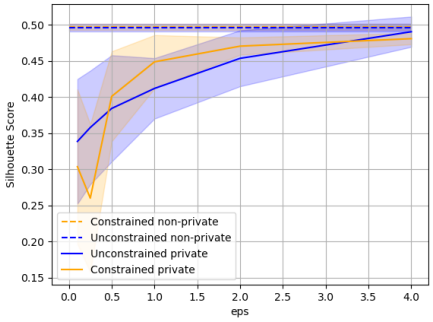
Quality Plots: Silhouette Score



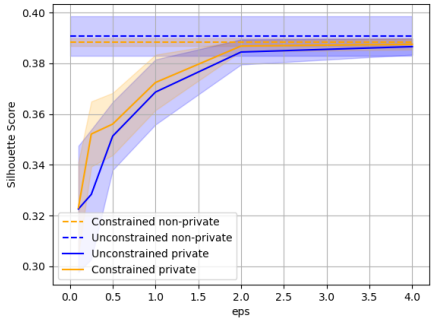
(a) S1



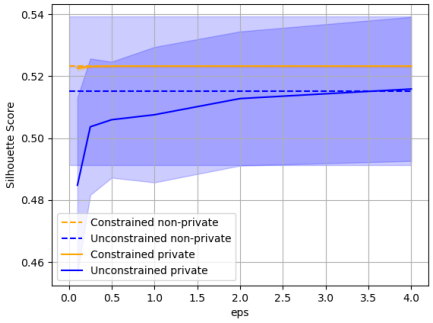
(b) Birch2



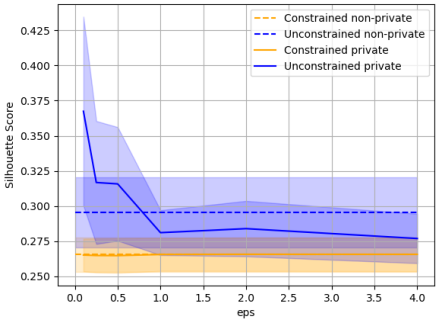
(c) Iris



(d) House

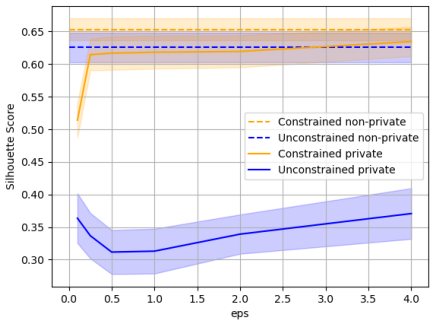


(e) Image

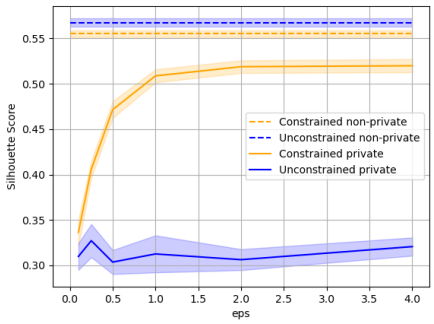


(f) Adult

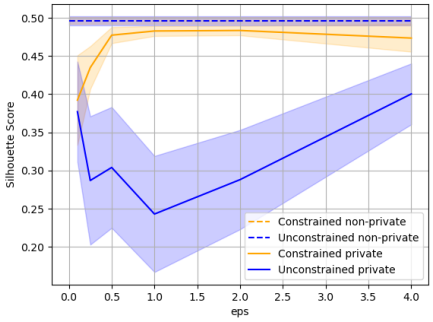
Figure D.1: Silhouette Score against ϵ for NMA aggregation



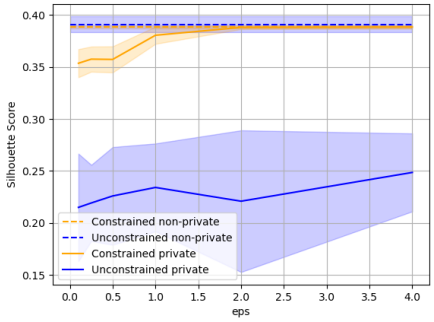
(a) S1



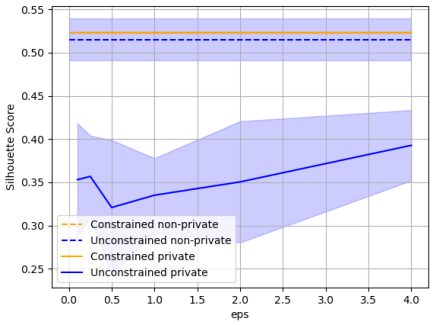
(b) Birch2



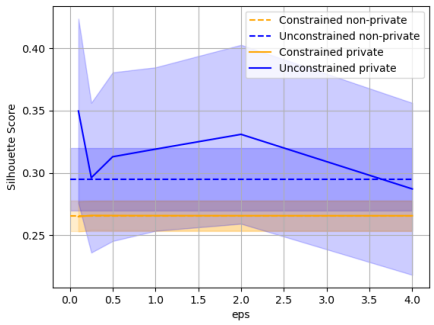
(c) Iris



(d) House



(e) Image



(f) Adult

Figure D.2: Silhouette Score against ϵ for AMN aggregation