ABSTRACT
Partition selection, or set union, is an important primitive in differentially private mechanism design: in a database where each user contributes a list of items, the goal is to publish as many of these items as possible under differential privacy.

In this work, we present a novel mechanism for differentially private partition selection. This mechanism, which we call DP-SIPS, is very simple: it consists of iterating the naive algorithm over the data set multiple times, removing the released partitions from the data set while increasing the privacy budget at each step. This approach preserves the scalability benefits of the naive mechanism, yet its utility compares favorably to more complex approaches developed in prior work.

KEYWORDS
Differential privacy, partition selection, scalable algorithms

1 INTRODUCTION
Group-level aggregation is a fundamental building block for many data analysis tasks. For example, doctors may want to understand mortality rates for patients with different underlying conditions, and search engine developers may want to study the frequency of different search queries that users are making.

With increased awareness of the risks of data disclosure, increasingly data analysts are using differentially private mechanisms to answer queries on sensitive data sets. In both of the given examples, the aggregate values (e.g., the mortality rates, or the frequencies of search queries) are sensitive, but also the groups themselves are sensitive. Revealing that some user in the data set has a particular disease or made a particular search query could itself violate privacy—even if it is never revealed who that person is. Additionally, the set of groups may be a priori unknown or impractical to enumerate, like in vocabulary extraction [11] or the private release of search queries [14].

In order to privately do a group-level aggregation in such settings, one must first privately compute the set of groups represented in the data set—ideally, releasing as many as possible while still maintaining privacy. More formally, the mechanism $M$ gets a data set $\mathbf{x} = (W_1, \ldots, W_N)$ where each user $i$ has a list of items $W_i$ and $M$ differentially privately releases a subset $S \subseteq \bigcup_i W_i$ of the partitions that is as large as possible.

We study this problem, called differentially private partition selection (also called key selection or set union), with a focus on approaches that can be incorporated into general-purpose differentially private tooling. When designing such frameworks, scalability is of paramount importance [1, 2, 21]: underlying mechanisms must be able to run even when the input or the output is too large to fit on a single machine. In such cases, the computation itself must also be parallelizable, so it can run across multiple machines and avoid unacceptably long running times.

Differentially private partition selection mechanisms have been proposed as early as 2009 [14], but recent work has shed a new light on this problem, and proposed alternative approaches that bring significant utility gains [6, 11]. To obtain these utility improvements, these newer mechanisms use a greedy approach: each user considers what items have been contributed by previous users so far, and “chooses” which items to contribute according to a policy, chosen carefully to maintain a sensitivity bound. Unfortunately, these mechanisms do not scale: each user chooses their contribution based on the contribution of all previous users, so the data has to be processed one user after another, and the overall algorithm cannot be parallelized. This intuition can be verified experimentally: we show that such algorithms eventually time out or run out of memory when the input is very large, even on clusters of multiple machines. Because these greedy algorithms do not scale, differential privacy tools that need to handle large datasets cannot use these smarter approaches, and instead must rely on the naive algorithm and its underwhelming utility [9, 10, 15].

This raises a natural question: can we achieve the utility benefits of policy-based approaches, while preserving the scalability of more naive approaches? In this work, we introduce a new approach that combines both benefits: DP-SIPS, short for scalable, iterative partition selection.

DP-SIPS relies on a simple idea: rather than having to process the data of each user sequentially, it runs the naive, massively-parallelizable algorithm multiple times, splitting the privacy budget between each step and removing items that were previously discovered. It uses a minuscule amount of privacy budget to publish and then remove the “heavy hitters”—those which have an enormous amount of weight in the histogram—and allocates the majority of the budget to discovering items that remain after heavy hitters are removed from the dataset. On skewed datasets, the removal of heavy hitters allows users to allocate proportionally more weight in subsequent histograms to less-frequent items while the increased privacy budget on later iterations also lowers the threshold for releasing an item.
As we show experimentally on multiple real-world and synthetic datasets, this mechanism has a similar utility to greedy approaches, but scales horizontally: increasing the number of cluster nodes significantly reduces runtime. This makes it a suitable choice for implementation in general-purpose DP infrastructure with high scalability requirements.

The rest of this paper is organized as follows:

- In Section 2, we formally define the problem and the building blocks we use for DP-SIPS and its privacy accounting.
- In Section 3, we detail existing approaches to differentially private partition selection.
- In Section 4, we introduce our algorithm and the proof of its privacy guarantees.
- In Section 5, we report on the experimental evaluation of DP-SIPS.
- Finally, in Section 6, we discuss our results, report on some unsuccessful approaches that we tried, and outline directions for future work.

2 PRELIMINARIES

A data set $x = (W_1, \ldots, W_N)$ contains a set of user lists $W_i \in U_i$. We refer to elements in $U$ as items or partitions, and define partition selection (also called key selection or set union) as follows:

**Definition 1 (Partition Selection Problem).
**Given a (possibly unbounded) universe $U$ of items and a data set $x = (W_1, \ldots, W_N)$ of user lists $W_i \in U_i$, an algorithm $M$ solves the partition selection problem if it is differentially private, and $M$ outputs a set $S \subseteq \cup_i W_i$.

We begin by presenting the standard notion of differential privacy. Two data sets $x, x'$ are neighbors if they differ on one user’s list: $x = x' \cup W_f$. Informally, differential privacy requires that an algorithm’s output is distributed similarly on every pair of neighboring data sets.

**Definition 2 (Differential Privacy [7, 8]).
**A randomized algorithm $M: U^* \rightarrow Y$ is $(\epsilon, \delta)$-differentially private if for every pair of neighboring datasets $x, x' \in U^*$ and for all subsets $Y \subseteq Y$,

$$\Pr[M(x) \in Y] \leq e^\epsilon \cdot \Pr[M(x') \in Y] + \delta.$$  

A common variant of differential privacy, called zCDP, is useful for analyzing algorithms that sample noise from a Gaussian distribution (as ours will). The definition of zCDP uses the Rényi Divergence:

**Definition 3 (Rényi Divergence).
**Fix two probability distributions $P$ and $Q$ over a discrete domain $S$. Given a positive $\alpha \neq 1$, Rényi divergence of order $\alpha$ of distributions $P$ and $Q$ is

$$D_\alpha(P||Q) = \frac{1}{1 - \alpha} \log \left( \sum_{s \in S} P(s)^\alpha Q(s)^{1-\alpha} \right).$$  

**Definition 4 ($\rho$-zCDP [4]).
**A randomized mechanism $M: X^* \rightarrow Y$ satisfies $\rho$-zCDP if, for all $x, x' \in X^*$ differing on a single entry,

$$D_\alpha(M(x)||M(x')) \leq \rho \cdot \alpha \quad \forall \alpha \in (1, \infty).$$  

This definition can also be relaxed to approximate zCDP.

**Definition 5 (Approximate zCDP [4]).** A randomized mechanism $M: X^* \rightarrow Y$ is $\delta$-approximately $\rho$-zCDP if, for all $x, x' \in X^*$ differing on a single entry, there exist events $E = E(M(x))$ and $E' = E'(M(x'))$ such that, for all $\alpha \in (1, \infty)$,

$$D_\alpha(M(x)|E)||M(x')|E') \leq \rho \cdot \alpha \quad \text{and}$$  

$$D_\alpha(M(x')|E')||M(x)|E) \leq \rho \cdot \alpha,$$

and $\Pr[E] \geq 1 - \delta$ and $\Pr[E'] \geq 1 - \delta$.

Approximate zCDP satisfies composition and post-processing properties.

**Lemma 6 ([4], Lemma 8.2).** Let $M: X^* \rightarrow Y$ and $M': X^* \rightarrow Y \rightarrow Z$. Suppose $M$ satisfies $\delta$-approximate $\rho$-zCDP and for all $y \in Y, M'(y, \cdot): X^* \rightarrow Z$ satisfies $\delta'$-approximate $\rho'$-zCDP. Define $M'' : X^* \rightarrow Z$ by $M''(x) = M'(x, M(x))$. Then, $M''$ satisfies $(\delta + \delta' - \delta \cdot \delta')$-approximate $(\rho + \rho')$-zCDP.

Next we show a conversion from approximate zCDP to approximate DP. We rewrite the conversion lemma from [4] to be slightly more general (i.e., it uses an existing conversion from zCDP to approximate DP), and then apply the tight zCDP-to-approximate DP conversion given in [5]. Overall, this gives a tighter approximate zCDP-to-approximate DP conversion than what is stated in Lemma 8.8 of [4].

**Lemma 7 (Generalized Version of Lemma 8.8 from [4]).** Suppose we can show that every mechanism that satisfies $\rho$-zCDP must satisfy $\epsilon^*(\rho), \delta^*(\rho)$-approximate DP. That is, $(\epsilon^*, \delta^*)$ is a function converting a (pure) zCDP guarantee to an approximate DP guarantee. Suppose $M : X^N \rightarrow Y$ satisfies $\delta$-approximate $\rho$-zCDP. Then, $M$ satisfies $(\epsilon^*(\rho), \delta + (1 - \delta)\delta^*(\rho))$-DP.

**Lemma 8 ([5], Proposition 7).** Suppose $M : X^N \rightarrow Y$ satisfies $\rho$-zCDP. Then $M$ satisfies $(\epsilon, \delta)$-approximate DP for any $\epsilon > 0$ and

$$\delta = \inf_{\alpha \in (1, \infty)} \frac{\exp((\alpha - 1)(\alpha \cdot \rho - \epsilon))}{\alpha - 1} \left(1 - \frac{1}{\alpha}\right).$$  

Combining Lemma 7 and Lemma 8 gives us the following.

**Corollary 9.** Suppose $M : X^N \rightarrow Y$ satisfies $\delta$-approximate $\rho$-zCDP. Then $M$ satisfies $(\epsilon, \delta + (1 - \delta)\delta^*)$-approximate DP for any $\epsilon > 0$ and

$$\delta^* = \inf_{\alpha \in (1, \infty)} \frac{\exp((\alpha - 1)(\alpha \cdot \rho - \epsilon))}{\alpha - 1} \left(1 - \frac{1}{\alpha}\right).$$  

A common primitive in building private algorithms, the Gaussian Mechanism, satisfies $\rho$-zCDP.

**Definition 10 (Gaussian Distribution).
**The Gaussian distribution with parameter $\sigma$ and mean 0, denoted $N(0, \sigma^2)$ is defined for all $t \in \mathbb{R}$ and has probability density

$$h(t) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{t^2}{2\sigma^2}}.$$  

**Definition 11 (f2-Sensitivity).
**Let $f : U^* \rightarrow \mathbb{R}^d$ be a function. Its $f_2$-sensitivity is

$$\Delta_f = \max_{x,x' \in U \text{ neighbors}} \|f(x) - f(x')\|_2.$$
3 PRIOR APPROACHES TO PARTITION SELECTION

In this section we discuss three existing algorithms for differentially private partition selection. We begin with the naive algorithm, called Weighted Gaussian, in Section 3.1. In Section 3.2, we present Policy Gaussian [11] and Greedy updates Without sampling [6].

The algorithms all have three main steps: first, they compute a weighted histogram, which is simply a mapping from an item $i \in \mathcal{U}$ to a weight $H[u] \in \mathbb{R}$; second, they add calibrated noise to each item in the histogram; and thirdly, they release items that are above some appropriately-chosen threshold $T$. The primary difference between the algorithms is in how they compute the weighted histogram. This high-level algorithm for private partition selection is described in Algorithm 1, which can be composed with different weighted histogram algorithms.

**Algorithm 1 High-Level Partition Selection Algorithm**

**Input:** Data set of user partitions $x = (W_1, \ldots, W_N)$
- Weighted Histogram Algorithm $\text{WeightedHist}$
- Threshold $T$

**Noise distribution $\mathcal{D}$**

**Output:** Partitions $S \subseteq \cup_i W_i$

1: Initialize empty set $S \leftarrow \{\}$
2: $H \leftarrow \text{WeightedHist}(x)$ \text{ Compute weighted histogram}
3: for $u \in \text{Supp}(H)$ do
4: \quad $Z_u \sim \mathcal{D}$
5: \quad $\hat{H}[u] \leftarrow H[u] + Z_u$
6: \quad if $\hat{H}[u] \geq T$ then
7: \quad \quad $S \leftarrow S \cup \{u\}$
8: return $S$

3.1 Baseline: Weighted Gaussian

The Weighted Gaussian algorithm (Algorithm 2) is one of the simplest and first algorithms for private partition selection [14]. To build a weighted histogram, the algorithm first pre-processes the data set by removing any duplicates within a user’s set and truncating each user’s set to have at most $\Delta_0$ items. Then, the users compute a histogram with bounded $\ell_2$-sensitivity as follows: each user $i$ updates the weight $H[u]$ for each of the items $u$ in their set $W_i$ with the following rule:

$$H[u] \leftarrow H[u] + \frac{1}{\sqrt{|W_i|}}$$

The resulting weighted histogram has an $\ell_2$-sensitivity of 1, so it can be composed with the high-level algorithm using calibrated Gaussian noise and an appropriate threshold to ensure that the overall algorithm satisfies $\delta$-approximate $\rho$-zCDP. In the statement of Algorithm 2, $\Phi(\cdot)$ is used to denote the cumulative density function of the standard Gaussian distribution and $\Phi^{-1}(\cdot)$ is its inverse.

**Algorithm 2 Weighted Gaussian**

**Input:** Data set $x = (W_1, \ldots, W_N)$
- Privacy parameters $(\rho, \delta)$
- Maximum per-user contribution $\Delta_0$

**Output:** Partitions $S \subseteq \cup_i W_i$

1: Initialize empty histogram $H \leftarrow \{\}$
2: Initialize empty set $S \leftarrow \{\}$
3: for $i = 1, \ldots, N$ do
4: \quad $W_i \leftarrow$ get rid of duplicate items from $W_i$
5: \quad $\mathcal{W}_i \leftarrow$ uniformly sample at most $\Delta_0$ items from $W_i$
6: \quad for $u \in \mathcal{W}_i$ do
7: \quad \quad $H[u] \leftarrow H[u] + \frac{1}{\sqrt{|W_i|}}$
8: \quad $\sigma \leftarrow \frac{1}{\sqrt{2\rho}}$
9: \quad $T \leftarrow \max_{u \in \text{Supp}(H)} \left\{ \frac{1}{\sqrt{2\rho}} + \sigma \cdot \Phi^{-1}\left((1 - \delta)^{1/k}\right) \right\}$
10: \quad for $u \in \text{Supp}(H)$ do
11: \quad \quad $\hat{H}[u] \leftarrow H[u] + \mathcal{N}(0, \frac{1}{2\rho})$
12: \quad if $\hat{H}[u] \geq T$ then
13: \quad \quad $S \leftarrow S \cup \{u\}$
14: return $S$

**Theorem 13.** Fix any $\rho > 0$, any $\delta \in (0, 1)$, and any $\Delta_0 \in \mathbb{N}$. The Weighted Gaussian algorithm (Algorithm 2) satisfies $\delta$-approximate $\rho$-zCDP.

See Appendix A for the proof of Theorem 13.

The Weighted Gaussian algorithm benefits from being highly scalable. In particular, it lends itself well to parallel computation across several computers within a cluster, because the weights on each histogram item can be computed in parallel as well as the noise addition and thresholding steps (see Figure 8). Thus, Algorithm 2 is the standard approach for doing partition selection on data sets that are too large to fit in a single machine’s memory. Unfortunately, Weighted Gaussian suffers from poor accuracy compared to the greedy approaches we discuss next.

3.2 Greedy Approaches

One problem with the Weighted Gaussian algorithm is users waste their sensitivity budget on histogram items that are already well above the threshold. Most real-world data has highly skewed item frequencies, but Weighted Gaussian increments all items in a user’s set by the same amount.

The two greedy algorithms we discuss next solve this problem by iterating through the users one-by-one and using an update policy and the current state of the histogram to decide how to allocate weight across the items in their set. That is, each user’s update depends on previous users’ updates. For example, in both algorithms, users do not contribute to items in $H$ that have already reached $T^*$, the threshold $T$ plus some positive buffer; items that have reached the buffered threshold are very likely to be returned after noise is added and thus do not need more weight. These adaptive update rules are carefully chosen so that the overall algorithm has
bounded global sensitivity. As we will discuss in later sections, the main downside of these algorithms is their sequential nature. By design, they require iterating over the entire data set, which may be prohibitively slow for industrial data sets.

3.2.1 Policy Gaussian [11] (DPSU). As with the Weighted Gaussian algorithm, the data set is pre-processed by removing duplicate items from each user’s set and truncating the user sets to some fixed maximum size $\Delta_0$. Then, the algorithm iterates sequentially over the users and, for each item $u$ in user $i$’s set $W_i$, the user increments $H[u]$ by a weight that is proportional to $T^i - H[u]$, where $T^i$ is equal to $T$ plus some positive buffer. Essentially, items that are further from the buffered threshold get more weight added to them, and those that have already reached the buffered threshold get none. The weight that a user adds to each item is normalized so a single user’s update to $H$ has an $\ell_2$-norm of at most 1. We refer to this algorithm as DPSU.

Gopi et al. prove that the global $\ell_2$-sensitivity of the entire Policy Gaussian algorithm is bounded by 1, so applying the high-level algorithm (Algorithm 1) with appropriately-scaled Gaussian noise to each item and thresholding are sufficient for satisfying differential privacy.

3.2.2 Greedy updates Without sampling (GW) [6]. Carvalho et al. observe that, rather than removing duplicate items in a user’s set, one can use this frequency information to decide where to allocate sensitivity budget. GW iterates over the users, and each user computes $u^*$: the most frequent item in their list such that $H[u^*]$ is below the buffered threshold $T^i$. Then, the user increments $H[u^*]$ by $\min(1, T^i - H[u^*], \text{budget})$ where budget is the user’s remaining $\ell_1$ budget. This process is repeated until the user’s initial budget of 1 is consumed or the user has no items left that are below $T^i$. Carvalho et al. prove that their GW algorithm for building a weighted histogram has a global $\ell_1$-sensitivity bound of 1, so running the high-level algorithm (Algorithm 1) with Laplace noise and thresholding is sufficient for satisfying differential privacy.

One notable feature of their algorithm is that it can use item frequency information from a public data set to increase the accuracy of the frequency estimates. We do not use this feature when comparing it to other algorithms, since our goal is to create a general-purpose algorithm that could be incorporated into a privacy framework without requiring a data analyst to input a public data set.

4 DP-SIPS

In this section we present DP-SIPS: Differentially Private Scalable, Iterative Partition Selection, detailed in Algorithm 3. The basic structure is quite simple: it runs Weighted Gaussian on the data set multiple times with increasing privacy budget (and corresponding decreasing thresholds); on each iteration, the partitions returned by Weighted Gaussian are removed from each of the users’ sets.

Because Weighted Gaussian updates the histogram uniformly across a user’s items, when returned partitions are removed from the user’s set, they can allocate more weight to their remaining items (see Figure 1). The first iteration returns the very popular items using only a tiny fraction of the overall privacy budget, and subsequent iterations yield less frequent items. The action of the algorithm is twofold: in each iteration, the threshold is lowered at the same time as users allocate more weight to each of the items that remain in their sets (since the user sets get smaller when previously-returned items are removed).

Furthermore, the user sets are re-truncated on each iteration after the previously returned partitions are removed. For data sets that are both skewed in the item frequencies and in the sizes of users’ sets, this allows additional items to be included on each iteration.

Algorithm 3 DP-SIPS: Scalable, Iterative Partition Selection

**Input:** Data set $x = (W_1, \ldots, W_N)$
- Maximum per-user contribution $\Delta_0$
- Privacy parameters $\rho, \delta$
- Number of iterations $I$
- Privacy budget allocation factor $r > 0$

**Output:** Subset $S \subseteq \bigcup_{i=0}^{I} W_i$

1. Initialize $S \leftarrow {}$
2. for $i = 0, \ldots, I - 1$
3. if $r = 1$
4. then $(\rho_i, \delta_i) \leftarrow \left( \frac{\rho}{I}, \frac{\delta}{I} \right)$
5. else $(\rho_i, \delta_i) \leftarrow \left( \rho \cdot \frac{1}{1 - r} \cdot r^{i - 1}, \frac{\delta}{1 - r} \cdot r^{i - 1} \right)$
6. $P_i \leftarrow \text{Weighted} \cdot \text{Gauss}(x = (W_1, \ldots, W_N), \rho_i, \delta_i, \Delta_0)$
7. for $j \in N$
8. do $W_j \leftarrow W_j \setminus P_i$ $\quad \triangleright \text{Remove already-found partitions}$
9. $S \leftarrow S \cup P_i$
10. return $S$

*Theorem 14.* For any $\rho > 0$ and any $\delta \in (0, 1]$, Algorithm 3 satisfies $\delta$-approximate $\rho$-$\varepsilon$CDP.

*Proof.* By Theorem 13, each call to Weighted-Gauss satisfies $\delta_i$-approximate $\rho_i$-$\varepsilon$CDP. Applying composition and postprocessing (Lemma 6), Algorithm 3 satisfies $\sum_{i=0}^{I-1} \delta_i$-approximate $\sum_{i=0}^{I-1} \rho_i$-$\varepsilon$CDP.

If $r = 1$, then clearly $\sum_{i=0}^{I-1} \rho_i / I = \rho$ and similarly $\sum_{i=0}^{I-1} \delta_i / I = \delta$.

Now, for $r \neq 1$ we will solve for these summations using the closed-form formula for geometric sums.

$$
\sum_{i=0}^{I-1} \rho_i = \frac{\sum_{i=0}^{I-1} 1 - r}{1 - r} = \rho \cdot \frac{1 - r}{1 - r^I} \cdot \sum_{i=0}^{I-1} r^{i-1}
$$

$$
= \rho \cdot \frac{1 - r}{1 - r^I} \cdot \sum_{j=0}^{I-1} r^j = \rho \cdot \frac{1 - r}{1 - r^I} \cdot \frac{1 - r^I}{1 - r} = \rho.
$$

An identical calculation holds for $\sum_{i=0}^{I-1} \delta_i$. Thus, Algorithm 3 satisfies $\delta$-approximate $\rho$-$\varepsilon$CDP. \hfill \square
Figure 1: Depiction of Weighted Gaussian noisy histogram (left) compared to intermediate SIPS noisy histograms (right three diagrams) on a skewed data set. Solid blue bars represent partitions that will be returned, and yellow outlines represent the weight of each item on the previous iteration. Although the threshold for Weighted Gaussian is lower than each of the SIPS thresholds, SIPS benefits from less-frequent items getting increased weight as returned partitions are removed from user sets.

a few iterations are required to achieve good accuracy (see Section 5.2.3), so its runtime is still reasonable even on large datasets. A visual explanation of the scalability property of DP-SIPS in comparison with other approaches can be found in Figure 8, and an experimental performance and scalability comparison can be found in Section 5.3.

Relation to Algorithm 7 in [13]. Private Product-Distribution Estimator in [13] privately estimates a product of $d$ Bernoulli distributions, and has some structural similarities to DP-SIPS: it also uses weighted Gaussian iteratively with decreasing thresholds, removing items above the threshold in each round to estimate the frequencies of each coordinate. Aside from the problem setting (bounded vs. unbounded domain) and goal (distribution estimation vs. partition selection), a major difference is that [13] partitions the dataset horizontally into $\log(d/2)$ groups, and uses different subsets of user records on each round. This partitioning step is key to their privacy and accuracy arguments and leads to $\log(d/2)$ rounds rather than the constant number of rounds that DP-SIPS uses.

4.1 Scalability Analysis in the MapReduce Framework

We analyze DP-SIPS in the MapReduce model of distributed computing to better understand its scaling behavior on a cluster. The model proceeds in rounds that are synchronized across worker nodes running in parallel. The data records begin arbitrarily partitioned among the worker nodes. Each round begins with a map phase in which the worker node applies a map to each record in its partition. Next, the records are shuffled among the workers so that records with the same key are located on the same worker node. Lastly, each worker processes the records in the reduce phase. The three phases (map, shuffle, and reduce) constitute one round in the MapReduce framework. Typically, the shuffle phase is the most time intensive since large amounts of data must move among the worker nodes, so the overall number of rounds usually reflects an algorithm’s efficiency on a real system.

DP-SIPS proceeds in the following steps (see Figure 2):

- (1) Shuffle records by user_id
- (2) Bound the number of contributions per user
- (3) Shuffle records by item. Call this dataset $D$.
- (4) Count the number of items, add noise, and store the set of items $S$ above the threshold
- (5) Do a join to remove items in $S$ from $D$. The records now consist of $D \setminus S$. If this is the last iteration, this step can be skipped.
- (6) Return to Step 1 for $I$ iterations in total.

All but the last iteration of DP-SIPS involves three MapReduce rounds: first shuffling records by user to truncate; then shuffling by item to count, add noise, and threshold; lastly, doing a join with the original data to remove items that were released. In the last iteration, the join is not necessary. So, in total the algorithm does $3(I - 1) + 2$ rounds in the MapReduce framework, where $I$ is the number of iterations of DP-SIPS. In our experiments we set $I = 3$, so DP-SIPS performs 8 MapReduce rounds. Weighted Gaussian only uses 2 MapReduce rounds, since $I = 1$. The greedy algorithms cannot be parallelized--specifically the greedy methods for histogram computation—so each record must be loaded into the one worker node to make its contribution to the histogram. This requires $O(N)$ rounds of MapReduce, where $N$ is the number of users. Indeed, our experimental results in Section 5.3 (Figures 10 and 11) show that the runtime of DP-SIPS decreases with the number of cores while the greedy algorithms stay constant.
5 EXPERIMENTAL RESULTS

We use data sets of several different sizes from varied text domains to validate the empirical performance of the DP-SIPS algorithm, and we find that in general DP-SIPS has comparable accuracy to DPSU and GW while scaling to large data sets that DPSU and GW time out on. As with prior works, we focus on the problem of vocabulary discovery under the constraint of user-level privacy.

We begin by describing the data sets in Section 5.1 and then, in Section 5.2 we discuss how the empirical accuracy of DP-SIPS compares to that of existing algorithms, and in Section 5.3 we discuss the results from our scalability experiments.

5.1 Data sets

We use six publicly-available data sets and one synthetically-generated data set to study the accuracy of DP-SIPS against existing partition selection algorithms, and we use the largest two of the seven for scalability experiments on Amazon Elastic Map Reduce clusters. Reddit [18] is a data set of text posts collected from r/AskReddit which appeared as a benchmark in [6, 11]. We also use four data sets that appeared in [6]: Twitter [19], comprising customer support tweets to and from large corporations; Finance [3], financial headlines for stocks; IMDb [17], a set of movie reviews scraped from IMDb; Wikipedia [20], a set of Wikipedia abstracts (where we treat each abstract as a separate user set).

For the scalability experiments, we use Amazon [12], a publicly-available text data set from Kaggle of 4 million Amazon product reviews, and a synthetically generated data set Synthetic_80M with 80 million users and 4.6 billion observations. The synthetic data set, Synthetic_80M, is generated as follows. First, each user draws the number of items in their set from a Pareto distribution of scale 10 and shape 1.16; this captures the common "80-20 principle" observed on many empirical data sets that states that 80% of outcomes are due to 20% of causes. Then, each item in each user’s set is generating by sampling from a zeta distribution of parameter 1.1; this captures Zip’s law, which states that in many types of data (including words in natural languages), the rank-frequency distribution follows an inverse relation.

Table 1 lists the number of users, number of observations, and the vocabulary size of each data set. Using the same methods as [6, 11], we preprocess all non-synthetic data sets using tokenization with NLTK word_tokenize, removing URLs and symbols, and lower casing all words.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Users</th>
<th>Observations</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>223,388</td>
<td>373,983</td>
<td>155,701</td>
</tr>
<tr>
<td>Twitter</td>
<td>702,682</td>
<td>2,811,774</td>
<td>1,300,123</td>
</tr>
<tr>
<td>Finance</td>
<td>1,400,465</td>
<td>1,400,465</td>
<td>267,256</td>
</tr>
<tr>
<td>IMDb</td>
<td>49,999</td>
<td>49,999</td>
<td>194,532</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>245,103</td>
<td>245,103</td>
<td>631,866</td>
</tr>
<tr>
<td>Amazon</td>
<td>4,000,000</td>
<td>4,000,000</td>
<td>4,250,427</td>
</tr>
<tr>
<td>Synthetic_80M</td>
<td>80,000,000</td>
<td>4,643,596,660</td>
<td>741,129,124</td>
</tr>
</tbody>
</table>

Table 1: Number of users, number of observations, and true vocabulary size for each data set we consider.

5.2 Accuracy results

Because the goal of partition selection is to privately output as many partitions as possible, we measure accuracy as the number of partitions released. To date, there are no analytical accuracy guarantees for any prior partition selection algorithms (in the setting where users contribute multiple items), so we must use experimental validation to understand the accuracy of each algorithm.

We test the accuracy of the four algorithms: Weighted Gaussian (Algorithm 2), SIPS (our Algorithm 3), Policy Gaussian (DPSU) from [11], and Greedy updates Without sampling (GW) from [6].

Table 2 shows the following accuracy trends for SIPS on the datasets described in Section 5.1:

- SIPS only performs slightly worse than both GW and DPSU on one dataset (Finance); in general, SIPS’ performance is on par with DPSU’s.
- The accuracy of SIPS is consistently approximately double that of Weighted Gaussian.

In addition, Figures 3, 4, and 5 demonstrate that the relative accuracy of each algorithm on a given data set remains consistent across different choices of privacy parameters.

5.2.1 Accuracy Inconsistencies. While GW tends to perform well, its accuracy on the IMDb and Wikipedia data sets is below even that of Weighted Gaussian (see Figure 5). To further investigate this phenomenon, we modify these two data sets in addition to Finance to remove repeated items within each user’s set (see the data sets marked as “deduplicated” in Table 2). Note also that the accuracy of the other three algorithms is unaffected by deduplication since they all perform this preprocessing step to the data sets.

Without any item frequency information, the algorithm adds all of its weight to a randomly-selected item, so one would expect the performance on all three data sets to be worse than Weighted Gaussian. To the contrary, GW’s accuracy on the deduplicated Finance data set is still significantly higher than the others, and GW’s accuracy on the deduplicated IMDb and Wikipedia is again worse than Weighted Gaussian. Figure 5 shows that GW’s poor relative accuracy on IMDb worsens at higher levels of epsilon.

We believe that GW’s inconsistent accuracy depends on the ratio between vocabulary size and number of observations: GW performs very well on datasets where this ratio is small (Finance: 0.19), and very poorly on datasets with large relative vocabulary (IMDb: 3.8). Other factors, such as the short length of Finance headlines compared to IMDb reviews and Wikipedia abstracts, might also play a role. We did not attempt to fully resolve this discrepancy in GW’s accuracy; however, we present these results as a caution when selecting a partition selection algorithm.

5.2.2 Comparing zCDP to DP. We implement our algorithm to satisfy approximate zCDP to take advantage of its simpler composition properties over standard DP. In [11], the Policy Gaussian algorithm satisfies $(\epsilon, \delta)$-DP, but the threshold and Gaussian noise can easily be recalibrated to satisfy approximate zCDP instead. We do so in this work to facilitate the comparison to our algorithm. Unfortunately, the GW algorithm from [6] has a bounded $\ell_1$-sensitivity and uses Laplace noise, so it is not easily converted to satisfy approximate zCDP without a significant loss in accuracy. Instead, we use Corollary 9 to choose appropriate $(\epsilon, \delta)$ parameters for the given $\rho$.
Table 2: Number of partitions returned by Wt. Gauss (Algorithm 2), SIPS (Algorithm 3), DPSU (Policy Gaussian from [11]), and GW (from [6]) on six data sets. Additionally, we run GW on three data sets where duplicates within user lists have been removed. Note that the other three algorithms already deduplicate user sets. For SIPS we use 3 iterations and $r = 1/3$. For Wt. Gaussian, SIPS, and DPSU, the privacy budget is set to $\rho = 0.1$, $\delta = 10^{-5}$, and the user contributions are truncated to $\Delta_0 = 100$. For GW, $\varepsilon = 1.7$ and $\delta = 8.1142 \times 10^{-5}$, which implies $10^{-5}$-approximate 0.1-zCDP. On Synthetic_80M, DPSU and GW ran out of memory before completing the computation.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Wt. Gauss</th>
<th>SIPS</th>
<th>DPSU</th>
<th>GW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reddit</td>
<td>6,160</td>
<td>11,392</td>
<td>11,186</td>
<td>11,984</td>
</tr>
<tr>
<td>Twitter</td>
<td>12,632</td>
<td>23,649</td>
<td>23,576</td>
<td>27,184</td>
</tr>
<tr>
<td>Finance</td>
<td>17,350</td>
<td>27,559</td>
<td>29,005</td>
<td>37,503</td>
</tr>
<tr>
<td>IMDb</td>
<td>3,728</td>
<td>7,759</td>
<td>5,845</td>
<td>3,133</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>11,340</td>
<td>21,037</td>
<td>18,129</td>
<td>11,251</td>
</tr>
<tr>
<td>Amazon</td>
<td>67,522</td>
<td>144,805</td>
<td>143,997</td>
<td>185,563</td>
</tr>
<tr>
<td>Synthetic_80M</td>
<td>711,601</td>
<td>1,137,467</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Finance (deduplicated)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>37,563</td>
</tr>
<tr>
<td>IMDb (deduplicated)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3,005</td>
</tr>
<tr>
<td>Wikipedia (deduplicated)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>9,802</td>
</tr>
</tbody>
</table>

Figure 3: Accuracy on Reddit as a function of varying epsilon (and associated $\rho$), for $\delta = 10^{-5}$.

Figure 4: Accuracy on Reddit as a function of varying delta for $\varepsilon = 1.7$.

and $\delta_{\text{CDP}}$. The tables in Appendix B give the conversions derived from Corollary 9 used for our experiments.

The conversion from approximate zCDP to approximate DP is not exactly tight (meaning the given $(\varepsilon, \delta)$ may be higher than the true privacy guarantee given by the approximate zCDP parameters). Because of the looseness of the conversion, the GW algorithm’s accuracy may be slightly inflated when compared to the other algorithms.

Doing the privacy budget accounting using approximate zCDP is not exactly tight either; instead, one could use numerical methods to compute the total privacy budget spent [16]. At the time of writing, we could not find a working implementation of such methods that could support the privacy property of Weighted Gaussian. Doing so could slightly improve the privacy analysis of DP-SIPS, leading to more favorable comparisons with other approaches. Furthermore, using approximate zCDP has the advantage of being easier to integrate with existing differential privacy software [2, 15].

Doing the privacy budget accounting with zCDP is not exactly tight either: numerical methods using

5.2.3 Selecting hyperparameters. Our algorithm has several hyperparameters (aside from the privacy parameters $\rho$ and $\delta$) that need to be set by the data analyst: $\Delta_0$ the maximum number of items per user, $r$ the ratio between the privacy budget for iteration $i + 1$ and $i$, and $I$ the number of iterations in the algorithm. One option is to divide the privacy budget and try several hyperparameter settings and select the setting with the highest accuracy; however, this wastes a lot of privacy budget. We find that the accuracy of DP-SIPS is largely invariant to reasonable settings of the hyperparameters, and we provide general rules of thumb for selecting them.
Figure 5: Accuracy on IMDb as a function of varying epsilon for fixed $\delta = 10^{-5}$.

<table>
<thead>
<tr>
<th>Iterations $I$</th>
<th>Partitions Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5,464</td>
</tr>
<tr>
<td>2</td>
<td>10,041</td>
</tr>
<tr>
<td>3</td>
<td>11,126</td>
</tr>
<tr>
<td>4</td>
<td>11,182</td>
</tr>
<tr>
<td>5</td>
<td>11,541</td>
</tr>
<tr>
<td>6</td>
<td>11,061</td>
</tr>
<tr>
<td>8</td>
<td>11,585</td>
</tr>
<tr>
<td>10</td>
<td>11,637</td>
</tr>
</tbody>
</table>

Table 3: SIPS's accuracy as a function of the number of iterations on Reddit with parameters $\rho = 0.1$, $\delta = 10^{-5}$, $r = 1/3$, and $\Delta_0 = 50$.

Figures 6 and 7 display the accuracy of our algorithm on five data sets with varying $r$ and $\Delta_0$, respectively. For the given setting of $\delta$ and $\rho$, the accuracy of DP-SIPS is largely unaffected by different choices of $r$ and $\Delta_0$ across 5 data sets, and Table 3 shows that the accuracy of DP-SIPS only slightly increases with the number of iterations $I$ on the Reddit data set (for the given settings of the other parameters). Furthermore, Figure 6 (privacy ratio $r = 1$) suggests that evenly splitting the budget among the rounds yields lower accuracy than geometrically increasing the budget at each round.

The results suggest that DP-SIPS has good accuracy for a large range of hyperparameters (aside from the privacy parameters). We recommend using the following settings: $r \in [0.2, 0.4]$, $I \geq 3$, and $\Delta_0$ set to an overestimate of the true maximum number of per-user contributions. If the true maximum number of contributions per user is public, $\Delta_0$ should be set to that value.

5.3 Scalability results

We benchmark the algorithms in several ways, and we find that DP-SIPS and Weighted Gaussian scale well with large data sets, while DPSU and GW do not.
in PySpark) and return the number of partitions that were discovered, in order to ensure PySpark’s lazy evaluation executed the algorithm during the timing phase. See Appendix C for information about the machine specifications.

For our first scalability experiment, we run the algorithms on subsets of Synthetic_80M, a synthetic data set with 80 million users and 4.6 billion items in total, and benchmark the algorithms as the number of users increases. Figure 9 shows the results of this experiment. DP-SIPS and Weighted Gaussian scale well to large data sets while DPSU and GW timeout or run out of memory on subsets with just 2 million and 10 million users, respectively. We ran this experiment on an EMR cluster with 8 nodes, each of size m5a.8xlarge (each node has 32 processor cores and 128 GB of memory).

To further investigate the scaling behaviors of the algorithms, we ran more scalability experiments on a smaller data set, Amazon, which consists of product reviews from 4 million users. Figure 10 shows the results from the first experiment, in which we measure the runtimes of the algorithms on clusters with a single master node and varying numbers of core nodes. All nodes are of type m5a.2xlarge, which has 8 processor cores and 32 GB of memory. Figure 10 illustrates the following runtime trends:

- The runtimes of Weighted Gaussian and SIPS both decrease as the number of core nodes increases.
- The runtimes of DPSU and GW remain approximately the same even as the number of core nodes increases.

These results confirm the intuition that Weighted Gaussian and SIPS are both parallelizable and thus scale well with increased cluster sizes, even on large data sets. Additionally, it confirms our observation that since DPSU and GW need to iterate sequentially over each user, increasing the cluster size does little to improve their running times.

We run an additional scalability experiment on Amazon: instead of increasing the number of core nodes, we increase the sizes of the core nodes. We use a single master node and two core nodes, and increase the sizes of all three nodes. Table 11 shows similar trends as the previous experiment: Weighted Gauss and SIPS scale with
increased node sizes while DPSU and GW do not. So, even when the sizes of the machines increase, this makes little difference in the runtimes of DPSU and SIPS.

Comparing Figures 10 and 11, we see that for DP-SIPS, the communication overhead between machines is small. Specifically we can compare SIPS’s performance on 8 cores in Figure 10, a cluster with 1 master node and 8 core nodes each of type m5a.2xlarge (8 CPU cores, 32 GB memory) to m5a.8xlarge (32 CPU cores, 128 GB memory) in Figure 11, a cluster with 1 master node and 2 core nodes. In both, the core nodes have in total $8 \cdot 8 = 2 \cdot 32$ CPU cores and $8 \cdot 32 = 2 \cdot 128$ GB of memory among them. The runtimes of SIPS in the first case is 58.41 seconds while in the second case, it is 66.28 seconds, which suggests that the communication overhead is small. In fact, SIPS runs faster on a cluster of 9 smaller machines than a cluster of 3 larger machines. This discrepancy could be the result of any number of factors in the machine specifications.

Our takeaway is clear: the scalability experiments confirm that DPSU and GW are not suitable for massive data sets and the prohibitively poor run times of DPSU and GW on industrial-scale data sets cannot be overcome by increasing the computational power of the cluster (either in number of machines or even machine size) because the algorithms are not parallelizable on a fundamental level. For larger data sets, SIPS and Weighted Gaussian are the only feasible algorithms and SIPS consistently has improved accuracy over Weighted Gaussian.

### 6 DISCUSSION

Differentially private partition selection (or set union) is a fundamental problem for many private data analysis tasks. Prior approaches to this problem either suffer from poor accuracy or are prohibitively slow on large data sets because they are designed to sequentially iterate over the users. We present a simple algorithm for differentially private partition selection that achieves accuracy that is comparable to DPSU and runtimes that scale well with increased computational resources.

#### 6.1 Unsuccessful Attempts

One natural question we explored is: can we boost the accuracy of DPSU using our method of iterating several times and removing partitions from the data set that were previously returned? We found that empirically this did not boost the accuracy on the data sets that we tested. Intuitively, this makes sense because the per-user Gaussian update policy in DPSU prevents users from adding weight to items that have already reached the buffered threshold. Such items will very likely be released after the noise addition and thresholding steps. The DPSU approach achieves the same goal as iterating, albeit with more precision due to the greedy nature of the Policy Gaussian per-user updates.

#### 6.2 Future work

This work raises several natural questions. From a practical perspective, one may wonder whether there is a scalable algorithm with higher accuracy than DP-SIPS. Another interesting line of inquiry could consider theoretical guarantees on the number of partitions released, possibly under some distributional assumptions on the data. We believe these lines of inquiry would give important insights into a fundamental problem in private data analysis.

### ACKNOWLEDGMENTS

We are thankful to the anonymous reviewers for the helpful feedback. This work was supported by Tumult Labs.

### REFERENCES

Now, it remains to show that $\Pr[E] \geq 1 - \delta$. First, note that each user contributes to most $\delta \alpha$ items in the histogram, and further note that event $E^c$ occurs when at least one item from $W^*$ that is not in $x$ is released. Let $W^*$ denote the set of items in $W^*$ that do not appear in $x$. Then,

$$\Pr[E] = \Pr[\forall u \in W^* \cup W, u \notin M(x)]$$

$$= \Pr[\bigcap_{u \in W^*} \bigcap_{W \in W^*} \bigcup_{u \in W^*} \bigcap_{u \in W^*} H(u)] \leq T$$

For $Z_u \sim N(0, 1/2 \rho)$

$$= \prod_{u \in W^*} \Pr[H(u) + Z_u \leq T]$$

By independence of $Z_u$'s

$$= \left( \prod_{u \in W^*} \Pr \left( \frac{1}{\sqrt{|W^*|}} + Z_u \leq T \right) \right)^{|W^*|}$$

Since $Z_u$'s are i.i.d.

$$\geq \min_{k \in [\Delta_0]} \left\{ \left( \Pr \left( \frac{1}{\sqrt{k}} + Z_u \leq T \right) \right)^k \right\}$$

Since $|W^*| \leq \Delta_0$

$$\geq 1 - \delta$$

By definition of $T$.

Therefore, $M$ is $\delta$-approximate $\rho$-zCDP. □

### B PRIVACY PARAMETER CONVERSIONS

Tables 4 and 5 give the conversions between zCDP and DP that we use for the experiments, as well as the values of $\sigma$ that are used in Corollary 9 to do the conversions.

#### Table 4: Values of $\rho$, $\delta_{\text{CDP}}$, $\epsilon$, and $\delta_{\text{DP}}$ used for experiments with fixed $\delta_{\text{DP}}$ and varying $\epsilon$.

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\delta_{\text{CDP}}$</th>
<th>$\epsilon$</th>
<th>$\delta_{\text{DP}}$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td>$1 \times 10^{-3}$</td>
<td>0.14</td>
<td>5.00 $\times 10^{-2}$</td>
<td>77.033</td>
</tr>
<tr>
<td>0.005</td>
<td>$1 \times 10^{-5}$</td>
<td>0.338</td>
<td>5.08 $\times 10^{-5}$</td>
<td>37.037</td>
</tr>
<tr>
<td>0.01</td>
<td>$1 \times 10^{-5}$</td>
<td>0.495</td>
<td>4.99 $\times 10^{-5}$</td>
<td>27.128</td>
</tr>
<tr>
<td>0.05</td>
<td>$1 \times 10^{-5}$</td>
<td>1.2</td>
<td>4.99 $\times 10^{-5}$</td>
<td>13.283</td>
</tr>
<tr>
<td>0.1</td>
<td>$1 \times 10^{-5}$</td>
<td>1.765</td>
<td>4.96 $\times 10^{-5}$</td>
<td>9.86</td>
</tr>
<tr>
<td>0.5</td>
<td>$1 \times 10^{-5}$</td>
<td>4.41</td>
<td>4.90 $\times 10^{-5}$</td>
<td>5.127</td>
</tr>
</tbody>
</table>

#### Table 5: Values of $\rho$, $\delta_{\text{CDP}}$, $\epsilon$, and $\delta_{\text{DP}}$ used for experiments with fixed $\epsilon$ and varying $\delta_{\text{DP}}$.

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\delta_{\text{CDP}}$</th>
<th>$\epsilon$</th>
<th>$\delta_{\text{DP}}$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.005</td>
<td>$1 \times 10^{-8}$</td>
<td>0.62</td>
<td>1.04 $\times 10^{-3}$</td>
<td>64.073</td>
</tr>
<tr>
<td>0.0055</td>
<td>$1 \times 10^{-8}$</td>
<td>0.62</td>
<td>1.02 $\times 10^{-8}$</td>
<td>58.443</td>
</tr>
<tr>
<td>0.006</td>
<td>$1 \times 10^{-7}$</td>
<td>0.62</td>
<td>1.01 $\times 10^{-3}$</td>
<td>53.732</td>
</tr>
<tr>
<td>0.007</td>
<td>$1 \times 10^{-6}$</td>
<td>0.62</td>
<td>1.01 $\times 10^{-3}$</td>
<td>46.334</td>
</tr>
<tr>
<td>0.0083</td>
<td>$1 \times 10^{-5}$</td>
<td>0.62</td>
<td>1.01 $\times 10^{-3}$</td>
<td>39.398</td>
</tr>
<tr>
<td>0.01</td>
<td>$1 \times 10^{-4}$</td>
<td>0.62</td>
<td>1.01 $\times 10^{-3}$</td>
<td>33.037</td>
</tr>
<tr>
<td>0.013</td>
<td>$1 \times 10^{-3}$</td>
<td>0.62</td>
<td>1.01 $\times 10^{-3}$</td>
<td>25.863</td>
</tr>
</tbody>
</table>
C  AMAZON EMR CLUSTER SPECIFICATIONS

Table 6 shows the number of CPU cores and amount of memory available to each type of machine on Amazon EMR that we use in our experiments.

<table>
<thead>
<tr>
<th>Machine Size</th>
<th>Cores</th>
<th>Memory (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m5a.xlarge</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>m5a.2xlarge</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>m5a.4xlarge</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>m5a.8xlarge</td>
<td>32</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 6: Number of cores and memory per EMR machine of each size.