FAST STREAMING ALGORITHMS FOR HIGH-DIMENSIONAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

A fundamental problem in machine learning is to design algorithms that extract essential structure in large, dynamically growing datasets with provable statistical guarantees. The streaming model captures this challenge by requiring algorithms to process data sequentially and irreversibly using memory sublinear in the input size. While prior work has achieved near-optimal space bounds for fundamental problems such as clustering and dimensionality reduction, these guarantees often rely on costly per-update computation, limiting their relevance for large datasets.

We study whether *both* fast update time and optimal space can be achieved simultaneously for high-dimensional geometric problems in insertion-only streams. We present a general streaming framework that combines coarse and refined sampling schemes to accelerate the update time. Applying this framework, we obtain new streaming algorithms for Euclidean (k, z)-clustering, L_p subspace embeddings, and projection-cost preservation with near-linear update time and space independent of the stream length. In particular, we achieve an *exponential* improvement in k in the update time for (k, z)-clustering compared to previous state-of-the-art streaming algorithms. For subspace embedding and projection-cost preservation, we show that the total update time achieves input-sparsity time, demonstrating that there does not necessarily exist a separation between the runtime of streaming and offline algorithms. Moreover, we do not lose additional factors in space complexity compared to previous algorithms that give the best space.

1 INTRODUCTION

A central challenge in machine learning is how to efficiently extract essential structure from massive, evolving datasets, as the scale of modern datasets have grown far beyond the memory capacity of standard CPUs. This is formalized by the *streaming model* of computation, where data points arrive sequentially, and algorithms must maintain compact summaries while processing each data point irreversibly, without the ability to revisit past observations. The main objective in this setting is to develop algorithms that can quickly and accurately approximate or track key properties of the data using minimal memory. This setting captures fundamental questions in machine learning and is particularly well-suited for applications in data summarization.

In this paper, we study the construction of *coresets* for high-dimensional optimization problems in the streaming model. Coresets are small-scale weighted summaries of datasets that approximately preserve key objective values, allowing streaming algorithms to operate on a much smaller representation of the data while maintaining provable approximation guarantees. For instance, in Euclidean k-means, a coreset preserves the clustering cost for every candidate set of k centers, so that running k-means algorithms on the coreset yields a near-optimal solution for the original dataset. Therefore, more efficient coreset construction enables downstream applications with reduced space usage and faster update time. Other examples include L_p subspace embeddings, which preserve the norm of a matrix projected by any vector, and projection-cost preserving sketches, which approximately maintain the residual error of a matrix to every low-rank subspace. These dimensionality-reduction sketches are widely applied to regression, random matrix algorithms, and low-rank approximation.

In many of these applications, however, the data arrive not only at massive scale but also at high velocity. Modern learning systems routinely process streams consisting of millions or billions of high-dimensional data points, often generated continuously and in real time. In such regimes, memory

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

efficiency alone is insufficient: algorithms must also support extremely fast per-update computation in order to keep pace with the stream. Ensuring that each incoming data point is processed with minimal latency is therefore essential for deploying streaming algorithms in practice, and raises fundamental questions about the tradeoffs between space, update time, and approximation guarantees.

Unfortunately, although existing approaches for coreset construction use near-optimal space, they generally do not achieve fast update time. For example, one common approach is based on online importance sampling, which informally selects each arriving element based on its so-called sensitivity, which quantifies how "important" the element appears to be. However, naïvely computing or even approximating the sensitivity for high-dimensional problems like clustering and subspace embedding can take polynomial time (Braverman et al., 2020; Cohen et al., 2020; Woodruff & Yasuda, 2023; Woodruff et al., 2023) per arriving element. Another approach is based on merge-and-reduce Bentley & Saxe (1980); Har-Peled & Mazumdar (2004), which iteratively adds new elements to a coreset maintained by a binary tree data structure, but it periodically performs expensive operations to merge multiple coresets. As a result, the amortized update time has extraneous polylogarithmic terms over simply reading the arriving element. Moreover, the space complexity of merge-and-reduce also incurs polylogarithmic terms, which is prohibitively large for many streaming applications under strict space constraints. Motivated by these shortcomings, we ask:

Is it possible to achieve linear update time with optimal space complexity for high-dimensional optimization problems on data streams?

1.1 Our Contributions

In this paper, we introduce a unified framework to obtain streaming algorithms for coreset construction that are *both* fast and space-optimal, providing an affirmative answer to the above problem. Our framework is applied to (k, z)-clustering, L_p subspace embedding, and projection-cost preservation. Essentially, our framework first down-samples the stream to $\mathcal{O}(n^\alpha) \ll n$ elements using a coarse sampling scheme, and then apply existing space optimal refined sampling schemes (with slow update time) on the coarse sampling sub-stream. This reduces the time-consuming refined sampling steps into lower order terms and significantly improves their update time, with *no* additive factors in space compared to the refined sampling schemes, and thus we achieve *both* fast time and optimal space.

(k, z)-Clustering. Clustering is a fundamental problem that seeks to partition a dataset so that similarly structured points are grouped together and differently structured points are separated. The problem has been studied since the 1950s (Steinhaus et al., 1956; MacQueen, 1967) and variations of clustering are used across a wide range of applications, including bioinformatics, combinatorial optimization, computational geometry, computer graphics, data science, and machine learning. In the metric (k, z)-clustering problem, the input is a set X of n points $x_1, \ldots, x_n$ in metric space $(\mathcal{M}, \text{dist})$, a cluster parameter $k > 0$, and a positive integer exponent $z > 0$, and the goal is to minimize the clustering cost across all sets $\mathcal{C}$ of at most k centers: $\min_{\mathcal{C}\subset\mathcal{M},|\mathcal{C}|\le k} \text{cost}(X, \mathcal{C}) := \min_{\mathcal{C}\subset\mathcal{M},|\mathcal{C}|\le k} \sum_{i=1}^{n} \min_{c\in\mathcal{C}} \text{dist}(x_i, c)^z$. In the Euclidean (k, z)-clustering problem, the input set X is a subset of $\mathbb{R}^d$ and the metric dist is simply the L_2 norm. For $z = 1$ and $z = 2$, respectively, the (k, z)-clustering problem corresponds to the well-known k-median and k-means clustering problems.

For clustering in the standard insertion-only streaming model, the points $x_1, \ldots, x_n$ of X arrive sequentially along with an input accuracy parameter $\varepsilon > 0$. The goal is for each time $t \in [n]$ to find a clustering $\mathcal{C}_t$ on the prefix X_t of the first t points of X with cost that is a $(1 + \varepsilon)$-multiplicative approximation of the optimal clustering of X_t. To that end, note that a set $\mathcal{C}$ of k centers implicitly defines the overall clustering, since each point is assigned to the closest cluster center. In fact, it is not feasible to directly output a label for every point in X using space sublinear in $n = |X|$, and so in some sense, outputting $\mathcal{C}$ is the most reasonable objective we can hope for. We also remark that due to bit representation, input points cannot be represented in arbitrary precision and thus we generally rescale the points in X to lie within the grid $\{1, \ldots, \Delta\}^d$. Given a (k, z)-clustering problem, a (strong) coreset Z_t at time t is defined as a data structure that preserves the clustering cost for any choice $\mathcal{C}$ of k centers, i.e., $\text{cost}(X_t, \mathcal{C}) \in (1 \pm \varepsilon)\,\text{cost}(Z_t, \mathcal{C})$. Given a coreset Z_t, we can then conduct k-means algorithm on Z_t and obtain a set of near-optimal solutions without looking at the original dataset X_t, thus constructing more efficient coresets is essential. Applying our framework, we have the following result on coreset construction for (k, z)-clustering:

Theorem 1.1 (Restated in Theorem 3.2). *Given a set X of n points on $[\Delta]^d$ and an accuracy parameter $\varepsilon \in (0,1)$, there is a one-pass insertion-only streaming algorithm that uses $\mathcal{O}(d\log k) + \log^2(k)\cdot \mathrm{polylog}(\log(n\Delta)) + o_{d,k}(1)$ amortized update time and $\mathcal{O}\left(\frac{dk}{\min(\varepsilon^4,\varepsilon^{2+z})}\right)$ words of space and outputs a $(1+\varepsilon)$-coreset of X for (k,z)-clustering.*

We remark that the lower-order term $o_{d,k}(1)$ has the form of $\frac{dk}{n^{\Omega(1)}}$, and so it is constant assuming that $n \gg \mathrm{poly}(d,k)$, which holds for massive datasets in modern applications. Our results for L_p subspace embedding and projection-cost preserving sketches have similar expressions. In addition, we allow each word of space to use $\log(nd\Delta)$ bits, as is standard in the streaming literature. Finally, we note that amortized update time refers to the computational time required to process each incoming item and update the coreset, whereas reporting the coreset takes time proportional to its size.

Theorem 1.1 has several important implications. Most notably, it provides an *exponential* improvement in the dependence on k and $\log n$ over prior work (Henzinger & Kale, 2020; Bhattacharya et al., 2023; 2024b). The fastest previous algorithm (Bhattacharya et al., 2023; 2024b) achieves amortized update time $k\cdot\mathrm{polylog}(n)$ for constant d, while a concurrent result (Bhattacharya et al., 2026) obtains $k^{1/c}\cdot\mathrm{polylog}(n)$ time for approximation factor $\mathcal{O}(c)$ for a trade-off constant $c \geq 1$. In contrast, our algorithm uses only $\log^2(k)\cdot\mathrm{polylog}(\log(n\Delta))$ amortized time. Moreover, although the previous results can handle the fully-dynamic setting, i.e., insertions and deletions, they must store the entire dataset at all times and are therefore not suitable for the streaming setting. Thus, Theorem 1.1 is the first result to show that (k,z)-clustering can be performed in the insertion-only streaming model with *both* amortized update time and memory sublinear in the number k of clusters.

In addition, metric (k,z)-clustering requires $\Omega(nk)$ total runtime for constant-factor approximations (Mettu & Plaxton, 2004; Bhattacharya et al., 2023), implying an amortized $\Omega(k)$ update time over the stream. Theorem 1.1 shows that such a lower bound does not hold for Euclidean clustering, thereby establishing a separation between metric and Euclidean (k,z)-clustering on data streams.

Streaming algorithm	Amortized Update Time
(Henzinger & Kale, 2020)	$k^2\cdot\mathrm{polylog}(n\Delta)$
(Bhattacharya et al., 2024b)	$k^{1+\Omega(1)}\cdot\mathrm{polylog}(n\Delta)$
(Bhattacharya et al., 2023; 2024a)	$k\cdot\mathrm{polylog}(n\Delta)$
Theorem 1.1 (this work)	$\log^2(k)\cdot\mathrm{polylog}(\log(n\Delta))$

Figure 1: Table of (k,z)-clustering algorithms on data streams, omitting linear dependencies in the dimension d. We remark that (Henzinger & Kale, 2020; Bhattacharya et al., 2023; 2024b;a) can handle the fully-dynamic setting, whereas ours cannot. However, our algorithm uses sublinear space while theirs does not.

Theorem 1.1 also implies that we can efficiently compute the approximately optimal centers for (k,z)-clustering in the incremental setting. In particular, given an insertion-only stream of n points that defines a dataset X on $[\Delta]^d$, our algorithm uses $\mathcal{O}(d\log k) + \log^2(k)\cdot\mathrm{polylog}(\log(n\Delta))$ amortized update time and outputs an $\mathcal{O}(z)$-approximation to (k,z)-clustering at all times in the stream.

Theorem 1.2 (Restated in Theorem E.11). *Given an dataset X of n points in $[\Delta]^d$ arriving sequentially, there is an algorithm that uses $\mathcal{O}(nd\log(k)) + n\log k\cdot\mathrm{polylog}(\log(n\Delta)) + o_{d,k}(n)$ total runtime and outputs an $\mathcal{O}(z)$-approximation to the (k,z)-clustering problem.*

In fact, Theorem 1.2 surprisingly even gives the best known runtime for (k,z)-clustering in the *offline* setting. Previously, the best runtime has been $(nd+nk)\cdot\mathrm{polylog}(n\Delta)$, by using bicriteria approximations (Arthur & Vassilvitskii, 2007) in conjunction with dimensionality reduction (Cohen et al., 2015; Becchetti et al., 2019; Makarychev et al., 2019; Izzo et al., 2021) and importance sampling (Huang & Vishnoi, 2020; Cohen-Addad et al., 2021; 2022a) to construct a coreset, which is then given as input to any polynomial-time approximation algorithm such as local search (Gupta & Tangwongsan, 2008). By comparison, Theorem 1.2 has exponentially better dependencies on both k and $\log(n\Delta)$. In more recent works, techniques by (Draganov et al., 2024; la Tour et al., 2024) achieve offline algorithms for (k,z)-clustering with $\mathcal{O}(nd\log(n\Delta k))$ runtime, while (la Tour & Saulpic, 2024) achieved $\mathcal{O}\left(n^{1+\Omega(1)}d\log(n\Delta k)\right)$ runtime, for the setting of $\Delta = \mathrm{poly}(n)$. In the

same spirit as Theorem 1.1, these works observed that the $\Omega(nk)$ lower bound of (Bhattacharya et al., 2023) is not necessarily for Euclidean clustering. However, our result is faster across all settings.

Subspace embeddings. Subspace embedding is a fundamental tool for dimensionality reduction and numerical linear algebra. The input is a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and given an accuracy parameter $\varepsilon \in (0,1)$, the goal is to construct a matrix $\mathbf{M} \in \mathbb{R}^{o(n)\times d}$ such that for all vectors $\mathbf{x} \in \mathbb{R}^d$, we have $\|\mathbf{Mx}\|_p \in (1\pm\varepsilon)\|\mathbf{Ax}\|_p$. This property ensures that linear computations like regression and low-rank approximation can be performed on $\mathbf{M}$ with nearly the same accuracy as on $\mathbf{A}$. Since their introduction in the context of randomized linear algebra (Frieze et al., 2004; Woodruff, 2014), subspace embeddings have become a cornerstone of big data analysis, enabling fast algorithms for large-scale optimization, statistics, and machine learning. Note that different choices of the norm parameter p capture distinct notions of stability: for example, $p = 2$ preserves Euclidean structure; $p = 1$ provides robustness to outliers; and when p approaches infinity, it provides a smooth interpolation towards the ℓ_∞ norm. In insertion-only streams, the rows of $\mathbf{A}$ arrive sequentially and at each time $t \in [n]$, the goal is to find a subspace embedding for the matrix $\mathbf{A}_t$ consisting of the first t rows of $\mathbf{A}$. We next provide our main results for L_p subspace embedding:

Theorem 1.3 (Restated in Theorem 4.2)**.** *Given a constant p and a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ on $[-M,\ldots,-1,0,1,\ldots,M]^d$ with $M = \mathrm{poly}(n)$ and online condition number κ (defined as $\max_{t\in\{1,\ldots,n\}} \|A_t\|_2 \left\|A_t^+\right\|_2$), there exists a one-pass streaming algorithm that outputs a $(1+\varepsilon)$-L_p subspace embedding of $\mathbf{A}$ at all times using $\mathcal{O}(\mathrm{nnz}(\mathbf{A})) + o_{d,1/\varepsilon}(n)$ total runtime. The algorithm uses $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\right)$ words of space for $p \in [1,2]$ and $\mathcal{O}\left(\frac{d^{p/2+1}}{\varepsilon^2}\right)$ words of space for $p \in (2,4)$.*

We emphasize that the amortized runtime of Theorem 1.3 translates to an offline algorithm with $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ runtime, omitting the lower order term. Since any offline algorithm requires $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ time to read the input, without additional assumptions, our result is tight and shows that there is no inherent time complexity separation between the offline and streaming settings. For comparison with previous space-optimal streaming algorithms, Cohen-Addad et al. (2023); Wang et al. (2026) run online leverage score (Lewis weight) sampling and then post-filter by merge-and-reduce, improving the $\log(n\kappa))$ factor in space in both pure online sampling (Cohen et al., 2020; Woodruff & Yasuda, 2023) and pure merge-and-reduce. However, these algorithms have slow update time: as we need to estimate the online Lewis weight of each row in the matrix, the update time for each step in these methods is $\mathrm{poly}(d)$ (e.g., the term involves the matrix multiplication time). On the contrary, our total runtime is $\mathcal{O}(\mathrm{nnz}(\mathbf{A})) = \mathcal{O}(nd)$, which leads to a $\mathcal{O}(d)$ amortized update time, and hence we significantly improve on the runtime of previous algorithms that achieve near-optimal space.

Projection-cost preservation. projection-cost preserving sketches (Cohen et al., 2015) relax the guarantees of L_p subspace embeddings and instead require preserving the residual cost of projecting onto any rank-k subspace. Formally, given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and an accuracy parameter $\varepsilon \in (0,1)$, a matrix $\mathbf{B} \in \mathbb{R}^{m\times d}$ of $m < n$ rescaled rows is a rank-k $(1+\varepsilon)$ projection-cost preservation if, for all rank-k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d\times d}$, we have $\|\mathbf{B}-\mathbf{BP}\|_F^2 \in (1\pm\varepsilon)\|\mathbf{A}-\mathbf{AP}\|_F^2$, where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Importantly, projection-cost preserving sketches can be applied to compute the rank-k approximation of matrix $\mathbf{A}$, and thus has wide applications in random matrix algorithms and dimensionality reductions. We next present our main result for projection-cost preserving sketches:

Theorem 1.4 (Restated in Theorem 5.2)**.** *Given a matrix $\mathbf{A}$ of n rows on $[-M,\ldots,-1,0,1,\ldots,M]^d$ with $M = \mathrm{poly}(n)$ and online condition number κ, there exists a one-pass streaming algorithm in the row arrival model that outputs a $(1+\varepsilon)$ rank-k projected-cost preservation sketch of $\mathbf{A}$ with probability 0.99. The algorithm uses $\frac{k^2}{\varepsilon^2}\cdot\mathrm{polylog}\left(\frac{1}{\varepsilon},\log(nd\kappa)\right) + \tilde{\mathcal{O}}\left(\frac{dk}{\varepsilon^2}\right)$ words of space and $\mathcal{O}(\mathrm{nnz}(\mathbf{A})) + o_{d,k,1/\varepsilon}(n)$ total runtime.*

Similarly, we note that the total runtime in Theorem 1.4 translates to an offline algorithm with input sparsity time, omitting the lower order term. For comparison, the online sampling method in Braverman et al. (2020) achieves nearly input-sparsity time, i.e., $\mathrm{polylog}(n,\frac{1}{\varepsilon})\cdot\mathrm{nnz}(\mathbf{A})$, on the contrary, we reduce the $\mathrm{polylog}(n,\frac{1}{\varepsilon})$ terms and avoid the undesirable poly-logarithmic terms in their space complexity, showing better effectiveness for streaming applications. Again, directly implementing merge-and-reduce also incurs additional poly-logarithmic terms in space, which is prohibitively large for our purpose, so we achieve the best update time and space complexity.

2 Framework for Fast and Space-Optimal Streaming Algorithms

In this section, we formally describe our fast streaming framework. The standard way to construct coresets in the streaming setting is to samples each item x_t with probability proportional to its *online sensitivity*, which measures the importance of x_t with respect to $X_t = \{x_1, \ldots, x_t\}$, i.e., the points arrived before x_t. Unfortunately, the online sensitivities always over-estimate the actual sensitivities, and moreover, the samples are not mutually independent since we utilize previous sampled items to estimate the online sensitivity of the future incoming items, so standard concentration bounds, which requires independence, cannot be applied. These limitations lead to extra $\log n$ factors in the number of samples. Recent insight by (Cohen-Addad et al., 2023) observed that online sensitivity sampling gives an insertion-only stream $\mathcal{S}'$ of length $o(n)$ that forms a $(1+\varepsilon)$-coreset, and then by running merge-and-reduce on $\mathcal{S}'$, we reduce the $\log n$ factors to $\mathrm{poly}(\log\log n)$ factors. Additionally, Wang et al. (2026) further improves the space via efficiently encoding the coresets, avoiding dependence on n in the words of space. Although their methods achieve optimal space complexity, it is slow to process at each arrival in the stream, e.g., it requires exponential time (in d and k) for (k,z)-clustering to process each item. Also, running merge-and-reduce directly on the input stream may have fast update time, but we instead lose $\mathrm{polylog}(n)$ factors in the space complexity, so none of the existing methods meets our goal of achieving *both* optimal space and fast update time.

We observe that we can pre-filter the input stream by a *crude* sampling scheme to obtain a stream $\mathcal{S}'$ with length $o(n)$ that gives a $(1+\varepsilon)$-coreset. Then, we apply a refined sampling scheme on $\mathcal{S}'$, e.g., black-boxedly implementing the algorithms in Cohen-Addad et al. (2023); Wang et al. (2026), which provides a significantly more efficient compression of the input but requires longer time to compute, and we result in a stream $\mathcal{S}''$. However, because these operations are performed on the input stream $\mathcal{S}'$ of length $o(n)$, then the slower operations can be amortized into lower-order terms that do not multiply a linear term in n. That is, if the crude and refined sampling scheme uses T_1 and T_2 time to compute the sampling probability, respectively, then the amortized update time is $T_1 + n^{-\Omega(1)}T_2$. The former term is always the dominant term since T_2 does not have a dependence on n for our problems, so it remains to construct crude sampling schemes with efficient update time. The approach is summarized in Figure 2 and the formal theorem statement is postponed to Framework B.3.

(1) Given a stream $\mathcal{S}$ of length n, use a crude sampling scheme to generate an stream $\mathcal{S}'$ of length $n^{1-\Omega(1)}$.

(2) Use $\mathcal{S}'$ and a refined sampling scheme that takes longer to compute but generates a stream $\mathcal{S}''$ of optimal length.

(3) Use $\mathcal{S}''$ to compute the crude sampling probability of the next item in step (1).

Figure 2: High-level summary of our approach

Notably, since the crude sampling scheme returns an online sub-stream, our framework might also provide algorithms in the online setting, in which we are not allowed to discard sampled items, e.g., the algorithmic output must be passed on to down-stream structures immediately.

3 (k,z)-Clustering

In this section, we introduce our fast streaming algorithm for (k,z)-clustering. Please refer to Appendix C, Appendix D, and Appendix E for a detailed discussion. In this problem, the online sensitivity of a point x_t is defined as $\max_{C\subset\mathbb{R}^d:|C|\le k}\frac{\mathrm{cost}(x_t,C)}{\mathrm{cost}(X_t,C)}$, where $X_t = \{x_1,\ldots,x_t\}$ is the subset arrived before x_t. Our fast algorithm relies heavily on the relevant (k,z)-*medoids sensitivity* defined as $\max_{C\subset X:|C|\le k}\frac{\mathrm{cost}(x,C)}{\mathrm{cost}(X,C)}$. We remark that the k centers in this formulation must be inside the dataset X; in fact, they can be inside the coreset obtained by the algorithm for our purpose. Our core technical lemma shows that the (k,z)-medoids sensitivity is a constant-factor approximation of the (k,z)-clustering sensitivity (see Appendix C.3), reducing the problem to estimating the (k,z)-medoids sensitivity. However, it is still not clear how we can achieve this with fast update time. Next we present a fast algorithm that obtains a constant-approximation, which gives a time-efficient streaming algorithm, reducing the $\mathrm{polylog}(n\Delta)$ terms of previous work Henzinger & Kale (2020); Bhattacharya et al. (2023; 2024a;b) to $\mathrm{poly}(\log\log(n\Delta))$ and achieving linear dependency in k.

3.1 Constant Approximation with Fast Update Time

In this section, we present a streaming algorithm with $dk \cdot \text{poly}(\log\log(n\Delta))$ update time, which is exponentially smaller in terms of $n\Delta$. We consider the constraint medoids clustering problem: fix a point p, we minimize $\text{cost}(X, C)$ across all k centers $C \subset X$ containing the center at p. Intuitively, suppose that we want to compute the sensitivity of x and we guess p to be the center that realizes the sensitivity, so we need to compute the solution to the constrained problem. Given a constant-factor approximation S to the optimal unconstrained (k, z)-medoids clustering on X, the best clustering obtained by switching a center in S with $\{p\}$ give a valid approximation. In particular, let W be the set of $k-1$ centers such that $C' = W \cup \{p\}$ minimize the k-medoids clustering cost, then we can output C' as a approximated solution. Unfortunately, it is challenging to find W with efficient time. We observe that for each center $s \in S$, let n_s be the number of points in X served by s (i.e., s is their closest center in S) and let r_s be the distance from s to its nearest neighbor in $S \backslash \{s\}$, then removing s would increase the cost by roughly $n_s \cdot (r_s)^z$. Then it is natural to update this information for p. However, to compute the sensitivity, we traverse through all possible queries p that serve x in the optimal constrained solution, and we do not afford to update this information for all of them. Therefore, instead of updating the information for each center $s \in S$, we only update the information for the nearest neighbor $s^* \in S$ to each query p, which suffices for our purpose. Finally, we remove the center s with the smallest increment to find both W and C'. We present this process in Algorithm 1 and show that C' is a constant-factor approximation to the optimal constrained solution.

Algorithm 1 Finding an approximated (k, z)-medoids clustering containing a fixed center

Require: Constant-factor approximation $C \subset X \subset [\Delta]^d$ for (k, z)-medoids clustering on X; query point $x \in X$
Ensure: Constant-factor approximation for (k, z)-medoids clustering on X with a center at x
Initialize $\psi \leftarrow nd\Delta$ and $\psi_x \leftarrow \emptyset$
1: **for** $c \in C$ **do**
2: Let n_c be the number of points in X assigned to c under C
3: **if** $n_c \cdot \|c - x\|_2^z < \psi$ **then**
4: Update $\psi \leftarrow n_c \cdot \min_{p \in C \cup \{x\} \backslash \{c\}} \|c - p\|_2^z$ and set $\psi_x \leftarrow c$
5: **end if**
6: **end for**
7: **return** $(\psi_x, \text{cost}(X, C) + \psi)$

Now, it could happen that C' is a set of k centers that contains our guess p, but p does not serve x in C'. Then we need to further use C' to solve a more restrictive constrained problem: we minimize $\text{cost}(X, C)$ across all k centers $C \subset X$ containing the center at p *and p is the closest center to* x. Let C be the optimal solution and let $r := \|x - p\|_2$ We notice that any points that are served by some $c \in C'$ in the ball $B_{r/2}(x)$ must be served by a center in C outside the interior of ball $B_r(x)$, since p is the closest center to x in C but not in C'. Intuitively, let n_b be the number of such points described in the previous sentence, one would expect them to increase the cost by $n_b \cdot r^z$ in $\text{cost}(X, C)$. However, $\text{cost}(X, C') + n_b \cdot r^z$ is not a good approximation to $\text{cost}(X, C)$, because there could be a lot of points served by a center q at distance $[\frac{r}{2}, r)$ to x, which are not included in n_b. Then, $\text{cost}(X, C') + n_b \cdot r^z$ could significantly under-estimates $\text{cost}(X, C)$. To solve this issue, we instead look at n_b' that is the number of points served by some $c \in C'$ in the ball $B_{r/2}(x)$. Then we are able to show that $\text{cost}(X, C') + n_b' \cdot r^z$ gives a constant approximation to $\text{cost}(X, C)$, which leads to a good approximation to the sensitivity of x. The analysis involves some subtle case analysis and is deferred to Appendix D. We summarize our algorithm in Algorithm 2.

The amortized runtime of Algorithm 2 is $dk \cdot \text{polylog}(\log(n\Delta)$ for each batch with size k. We recall that the framework in Cohen-Addad et al. (2023) filters the input stream by an online sampling subroutine and then passes on to a merge-and-reduce tree, and computing the online sensitivities is the primary runtime bottleneck. Then applying our algorithm to their framework achieves the following guarantee.

Theorem 3.1. *Given a set X of n points on $[\Delta]^d$ and an accuracy parameter $\varepsilon \in (0, 1)$, there is a one-pass insertion-only streaming algorithm that uses $dk \cdot \text{polylog}(\log(n\Delta)$ amortized update time and $\frac{dk}{\min(\varepsilon^4, \varepsilon^{2+z})} \cdot \text{polylog}\left(\frac{1}{\varepsilon}, \log(n\Delta)\right)$ words of space and outputs a $(1+\varepsilon)$-coreset of X.*

Algorithm 2 BATCHSENS: Fast batch approximation of sensitivities

Require: Input set $Z \subset [\Delta]^d$ for (k, z)-medoids clustering; batch B of k query points
Ensure: Constant-factor approximation of the sensitivity of x for (k, z)-medoids clustering on $Z \cup B$, for all $x \in B$

Compute S as a constant-factor solution on $Z \cup B$ for (k, z)-medoids clustering
for $x \in B$ **do**
  Initialize $\widehat{s(x)} \leftarrow 0$
  for $p \in Z \cup B$ **do**
    Let Q be a constant-factor approximation of the optimal (k, z)-medoids clustering on X that contains a center at p {Only needs to be preprocessed and computed once per p across all x}
    Use S to compute a constant-factor approximation Ψ of $\mathrm{cost}(X, Q)$ {Algorithm 1}
    Set $r \leftarrow \|x - p\|_2$
    Let n_b be the number of points assigned by Q to centers of Q lying in the ball $B_{r/2}(x)$
    Update $\widehat{s(x)} \leftarrow \max\left(\widehat{s(x)}, \frac{r^z}{\Psi + n_b \cdot r^z}\right)$.
  end for
end for
return $\widehat{s(x)}$

3.2 CRUDE APPROXIMATION WITH FASTER UPDATE TIME

In this section, we present a streaming algorithm with $\mathcal{O}(d \log k) + \log^2(k) \cdot \mathrm{polylog}(\log(n\Delta))$ update time via our fast streaming framework, which further improves on the algorithm in the previous section. To construct the crude sampling scheme, we create a quadtree for the input set, which is often used in computing sensitivities Cohen-Addad et al. (2020). In a standard quadtree, each cell of a level has half the side length of a cell of an adjacent level. However, we only need a n^α-approximation for crude sampling since it suffices to filter down to n^α items before passing to the refined sampling scheme, so the quadtree we create allows each cell to have a $\frac{1}{n^\alpha}$ fraction of the side length of a cell of an adjacent level. We then implement the algorithm in Section 3.1 searching up the levels of the quadtree, which only has a constant number of levels due to the crude construction. In particular, for each level β, representing a candidate distance r, we compute a crude estimate of the optimal (k, z)-clustering cost constrained to having a center in the same cell as x. We then move all centers in the same cell as x by computing the number n_β of points served by these centers and increase the estimate by $n_\beta \cdot r^z$, where r is the estimated distance by the quadtree (see the full discussion in Appendix E). Combining our crude sampling scheme with the refined sampling scheme in Wang et al. (2026), we achieve a streaming algorithm with both near-optimal space and fast update time via our framework in Figure 2.

Theorem 3.2. *Given a set X of n points on $[\Delta]^d$ and an accuracy parameter $\varepsilon \in (0, 1)$, there is a one-pass insertion-only streaming algorithm that uses $\mathcal{O}(d \log k) + \log^2(k) \cdot \mathrm{polylog}(\log(n\Delta))$ amortized update time and $\mathcal{O}\left(\frac{dk}{\min(\varepsilon^4, \varepsilon^{2+z})}\right)$ words of space and outputs a $(1 + \varepsilon)$-coreset of X.*

4 L_p SUBSPACE EMBEDDING

In this section, we introduce our L_p subspace embedding algorithm with fast runtime. Our key observation here is that the square roots of the *leverage scores* are within a n^α-factor to the L_p sensitivities for $p < 4$, which describes the importance of a row for L_p subspace embedding, due to the relation between the 2-th norm and the p-th norm. Thus, it suffices to produce crude but fast n^α-approximations to the root leverage scores. We utilize a quadratic form of the constant-factor approximation maintained by the algorithm to estimate these quantities. That is, suppose that we have a constant-factor L_2 subspace embedding $\mathbf{Z}$ to $\mathbf{A}$, the leverage score of a row $\mathbf{a}_i$ with respect to $\mathbf{A}$ is a constant factor multiple of $\mathbf{a}_i^\top (\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{a}_i$. Then we can approximate this quantity using $\|\mathbf{g}\mathbf{J}\mathbf{a}_i\|_2^2$, where $\mathbf{g}$ is a random Gaussian vector and $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z})^{-1/2}$. Thus by sampling each row with the crude approximations, we then induce a stream of length $o(n)$ for which we can amortize existing approaches that use $\mathrm{poly}(d)$ update time. Note that we only need to store $\mathbf{g}\mathbf{J} \in \mathbb{R}^d$ using roughly $\mathcal{O}(d)$ space to implement the crude sampling scheme, so this incurs no additive factors to

the space complexity. Moreover, since the constant-factor approximation $\mathbf{Z}$ only samples $\mathrm{poly}(d)$ rows of $\mathbf{A}$, we only need to compute $\mathbf{gJ}$ for $\mathrm{poly}(d)$ times, which is also amortized to lower-order terms. We formally describe this method and its guarantees in Algorithm 3 and Lemma 4.1. For more details, see the full analysis in Appendix F.

Algorithm 3 Crude Lewis weight approximation

Require: Matrix $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \in \mathbb{R}^{n\times d}$ that arrives as a data stream, matrix $\mathbf{Z} \in \mathbb{R}^{m\times d}$ satisfying $\frac{1}{C} \cdot \mathbf{A}^\top\mathbf{A} \preceq \mathbf{Z}^\top\mathbf{Z} \preceq C \cdot \mathbf{A}^\top\mathbf{A}$ for some constant C, vector $\mathbf{a} \in \mathbb{R}^d$, parameter $p \geq 1$
Ensure: n^α approximation to the L_p Lewis weight of $\mathbf{a}$

1: Generate $Q = \mathcal{O}\left(\frac{1}{100p}\right)$ independent random Gaussian vectors $\mathbf{g}^{(1)}, \ldots, \mathbf{g}^{(Q)} \in \mathbb{R}^d$
2: $\mathbf{J} \leftarrow (\mathbf{Z}^\top\mathbf{Z})^{-1/2}$
3:
4: **return** $\widehat{\sigma(\mathbf{a})} \leftarrow \mathrm{median}_{q\in[Q]} \left|\mathbf{g}^{(q)}\mathbf{J}\mathbf{a}\right|$

Lemma 4.1. *Given $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\mathbf{Z} \in \mathbb{R}^{m\times d}$ be a matrix such that $\frac{1}{C} \cdot \mathbf{A}^\top\mathbf{A} \preceq \mathbf{Z}^\top\mathbf{Z} \preceq C \cdot \mathbf{A}^\top\mathbf{A}$, for some constant C, there exists an algorithm that computes an n^α approximation to the L_p Lewis weight using $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ runtime.*

For the refined sampling scheme, we note that the online Lewis weight sampling method obtains a $(1+\varepsilon)$-approximation using $\frac{d^{\min(p,2)/2}}{\varepsilon}\mathrm{polylog}(n\kappa)$ rows (Woodruff & Yasuda, 2023). We can then apply merge-and-reduce on the online sub-stream to reduce the sampling complexity to $\frac{d^{\min(p,2)/2}}{\varepsilon}\mathrm{poly}(\log\log(n\kappa))$ (Cohen-Addad et al., 2023). The $\mathrm{poly}(\log\log(n\kappa))$ term can be further redued by the efficient encoding framework in Wang et al. (2026). Then, combining our crude sampling scheme in Algorithm 3 and the refined sampling scheme, we achieve the following result:

Theorem 4.2. *Given a constant p and a matrix $\mathbf{A}$ of n rows on $[-M, \ldots, -1, 0, 1, \ldots, M]^d$, with $M = \mathrm{poly}(n)$ and online condition number κ, there is a one-pass streaming algorithm that outputs a $(1+\varepsilon)$-L_p subspace embedding of $\mathbf{A}$ at all times using $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ total runtime. For $p \in [1, 2]$, the algorithm uses $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\right)$ words of space, while for $p \in (2, 4)$, it uses $\mathcal{O}\left(\frac{d^{p/2+1}}{\varepsilon^2}\right)$ words of space.*

5 Projection-Cost Preserving Sketches

In this section, we discuss our fast algorithm to obtain rank-k projection-cost preserving (PCP) sketches. Online PCP sketches sample each row in the matrix with respect to the online ridge leverage scores $\mathbf{a}_i^\top(\mathbf{A}^\top\mathbf{A} + \lambda\mathbb{I})^{-1}\mathbf{a}_i$. Thus we provide n^α-approximations to this quantity. We remark that a $(1+\varepsilon)$-PCP sketch obtained by online ridge leverage score sampling is also a constant-factor L_2 subspace embedding of $\mathbf{A}$ with small additive distortion. To this end, we again use the approach of random Gaussian projection described in Section 4 to estimate the quadratic form of the PCP sketch $\mathbf{Z}$. In particular, we can approximate the online ridge leverage score using $\|\mathbf{gJa}_i\|_2^2$, where $\mathbf{g}$ is a random Gaussian vector and $\mathbf{J} = (\mathbf{Z}^\top\mathbf{Z} + \Theta(\lambda)\mathbb{I})^{-1/2}$, where $\lambda = \frac{\|\mathbf{A}-\mathbf{A}_k\|_F^2}{k}$ is the residual value and can be approximated by $\frac{\|\mathbf{Z}-\mathbf{Z}_k\|_F^2}{k}$. We then sample each row with the crude approximations to obtain a stream of length $o(n)$. We formally describe this method and its guarantees in Algorithm 10 and Lemma 5.1. For more details, see the full analysis in Appendix G.

Lemma 5.1. *Given $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $\widehat{\lambda}$ being a 2-approximation of $\lambda_\mathbf{A} := \frac{\|\mathbf{A}-\mathbf{A}_k\|_F^2}{k}$, let $\mathbf{Z} \in \mathbb{R}^{m\times d}$ be a matrix such that $\frac{1}{C} \cdot (\mathbf{A}^\top\mathbf{A} - \lambda_\mathbf{A} \cdot \mathbb{I}_d) \preceq \mathbf{Z}^\top\mathbf{Z} \preceq C \cdot (\mathbf{A}^\top\mathbf{A} + \lambda_\mathbf{A} \cdot \mathbb{I}_d)$ for some constant C, there exists an algorithm that computes an n^α approximation to the $\lambda_\mathbf{A}$-ridge leverage score using $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ runtime.*

Similarly as L_p subspace embedding, combining our crude sampling scheme and the refined sampling scheme (Wang et al., 2026), we achieve the following result:

Theorem 5.2. *Given a matrix $\mathbf{A}$ of n rows on $[-M, \ldots, -1, 0, 1, \ldots, M]^d$ with $M = \mathrm{poly}(n)$ and online condition number κ, there exists a one-pass streaming algorithm in the row arrival model that outputs a $(1+\varepsilon)$ rank-k projected-cost preservation sketch of $\mathbf{A}$ with probability 0.99. The algorithm uses $\frac{k^2}{\varepsilon^2} \cdot \mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\kappa)\right) + \tilde{\mathcal{O}}\left(\frac{dk}{\varepsilon^2}\right)$ words of space and $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ total runtime.*

6 Experimental Results

We illustrate the computational benefit of our fast streaming framework for Euclidean k-means clustering, whose objective is $\mathrm{cost}(X, C) = \sum_{x \in X} \min_{c \in C} \|x - c\|_2^2$ for $|C| \leq k$. We compare construction time and clustering quality at fixed coreset sizes. Additional results of L_2-subspace embedding and implementation details are presented in Appendix H. We remark that our experiments prove our practical advantages, while our main contribution remains the theoretical fast streaming framework. The experiments are conducted using an Apple M2 CPU, with 16 GB RAM and 8 cores.

Real-world dataset. We use MNIST (LeCun et al., 2010) with $n = 60000$ training images of handwritten digits (0 through 9), where each image is of 28×28 pixels and is represented by a vector of $d = 784$ dimensions. The dataset is naturally associated with $k = 10$ centers since it contains 10 digits. MNIST is suitable for evaluating our high-dimensional coreset construction algorithm due to its large scale and clear clustering structure.

Baselines. We compare with (batched) online sensitivity sampling and the efficient-space streaming framework of Cohen-Addad et al. (2023), labeled *CWZ framework*, which post-filters the online sampled sub-stream by a merge-and-reduce method. Our *crude filter + CWZ framework* instead uses the crude sampling approach in Section 3.2 and sends the resulted sub-stream to merge-and-reduce. All methods use the same batch size ($B = 100$) for fair comparison. A common 64-dimensional projection is used to accelerate the coreset construction, while evaluation uses the original vectors.

Evaluation Metrics. In each test, we fix the coreset size $m \in \{64, 128, 256, 512\}$ and compare the amortized runtime and the coreset quality. For the latter, the standard evaluation is to apply k-means algorithms on the coreset and compute the cost. We implement the standard k-means algorithm by SK-LEARN on the reported coreset of each method to obtain a set of centers C^*. Next we obtain a set of centers C_{ref} by running k-means on the entire MNIST dataset and plot the ratio $\frac{\mathrm{cost}(X, C^*)}{\mathrm{cost}(X, C_{\mathrm{ref}})}$. The evaluation is not counted in the runtime. Appendix H gives the parameter choices and the tuning procedure to return m points for all methods, which ensures fair comparison.

Results and Discussion. Figure 3 reports means over 5 trials. Our crude-filter framework (orange line) is consistently faster than CWZ (blue line) across the tested budgets ($2\times$ at $m = 256$ and $4\times$ at $m = 512$), with closely matched clustering costs. While the pure online baseline (green line) is slightly faster than our method, its clustering cost is much higher than both our method and CWZ, i.e., to achieve the same clustering quality, the online coreset needs to have $2\times$ the size.

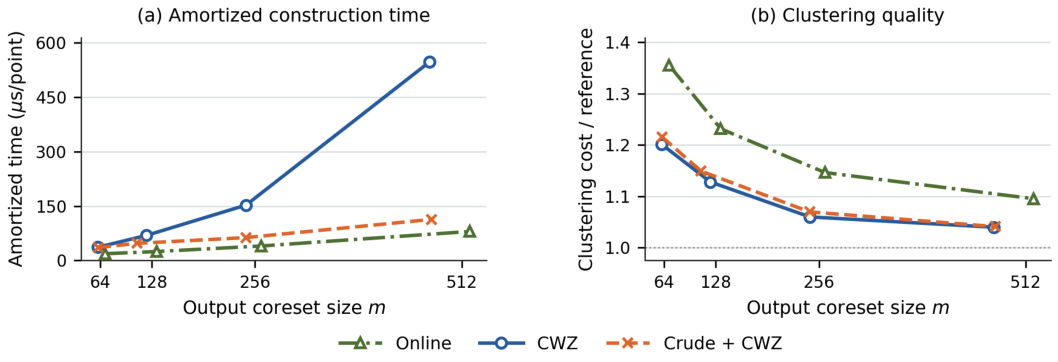


Figure 3: k-means coreset construction on MNIST.

These results match the theoretical analysis. Online only computes batch online sensitivities, so it is the fastest, however, it needs to maintain a large coreset to achieve good cost, which is highly restrictive in streaming applications. Instead, CWZ post-filters the online sub-stream with merge-and-reduce to reduce the coreset size, achieving better cost under the same budget, as shown in Figure 3. However, the time required to process each item becomes longer. Our method uses crude sampling to obtain the online sub-stream, accelerating the process without degrading the performance. This shows that a crude approximation to online sensitivities is sufficient for pre-filtering the stream and applying refined sampling later on, demonstrating the computational benefit of our fast streaming framework. Importantly, our method achieve *both* small space and fast update time in practice.

AI DISCLOSURE

In this work, we used generative AI tools to assist in writing the code for experiments, draft parts of the paper, provide feedback on experiments, implement methods, interpret results, and check the proofs. We have not used generative AI tools to help develop theoretical models, formulate mathematical claims, provide critical algorithmic ingredients, and design experiments, and the rest of the required disclosure tasks are not applicable to this work. LLM-generated code was carefully verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 1027–1035, 2007.

Arturs Backurs, Piotr Indyk, Ilya P. Razenshteyn, and David P. Woodruff. Nearly-optimal bounds for sparse recovery in generic norms, with applications to k-median sketching. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 318–337, 2016.

Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. Oblivious dimension reduction for k-means: beyond subspaces and the johnson-lindenstrauss lemma. In Moses Charikar and Edith Cohen (eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 1039–1050. ACM, 2019. doi: 10.1145/3313276.3316318. URL https://doi.org/10.1145/3313276.3316318.

Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

Sayan Bhattacharya, Martín Costa, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering in $\widetilde{O}(k)$ update time. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2023.

Sayan Bhattacharya, Martín Costa, and Ermiya Farokhnejad. Fully dynamic k-median with near-optimal update time and recourse. *CoRR*, abs/2411.03121, 2024a.

Sayan Bhattacharya, Martín Costa, Naveen Garg, Silvio Lattanzi, and Nikos Parotsidis. Fully dynamic k-clustering with fast update time and small recourse. In *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pp. 216–227, 2024b.

Sayan Bhattacharya, Martín Costa, Ermiya Farokhnejad, Shaofeng H-C Jiang, Yaonan Jin, and Jianing Lou. Fully dynamic euclidean k-means. In *67th IEEE Annual Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 2026.

Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pp. 517–528, 2020.

Vladimir Braverman, Dan Feldman, Harry Lang, Adiel Statman, and Samson Zhou. Efficient coreset constructions via sensitivity sampling. In *Asian Conference on Machine Learning, ACML*, pp. 948–963, 2021.

Xue Chen and Michal Derezinski. Query complexity of least absolute deviation regression via robust uniform convergence. In *Conference on Learning Theory, COLT*, volume 134, pp. 1144–1179, 2021.

Yeshwanth Cherapanamjeri, Sandeep Silwal, David P. Woodruff, and Samson Zhou. Optimal algorithms for linear algebra in the current matrix multiplication time. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 4026–4049, 2023.

Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC*, pp. 205–214, 2009.

Michael B. Cohen and Richard Peng. l_p row sampling by lewis weights. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pp. 183–192, 2015.

Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pp. 163–172, 2015.

Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 1758–1777, 2017.

Michael B. Cohen, Cameron Musco, and Jakub Pachocki. Online row sampling. *Theory Comput.*, 16:1–25, 2020.

Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Fast and accurate k-means++ via rejection sampling. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020.

Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In *STOC: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 169–182, 2021.

Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1038–1051, 2022a.

Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, Chris Schwiegelshohn, and Omar Ali Sheikh-Omar. Improved coresets for euclidean k-means. In *NeurIPS*, 2022b. URL http://papers.nips.cc/paper_files/paper/2022/hash/120c9ab5c58ba0fa9dd3a22ace1de245-Abstract-Conference.html.

Vincent Cohen-Addad, David P. Woodruff, and Samson Zhou. Streaming euclidean k-median and k-means with $o(\log n)$ space. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pp. 883–908, 2023.

Vincent Cohen-Addad, David P Woodruff, Shenghao Xie, and Samson Zhou. Nearly space-optimal graph and hypergraph sparsification in insertion-only data streams. *arXiv preprint arXiv:2510.18180*, 2025.

Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W. Mahoney. Sampling algorithms and coresets for l_p regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009.

Andrew Draganov, David Saulpic, and Chris Schwiegelshohn. Settling time vs. accuracy tradeoffs for clustering big data. *Proc. ACM Manag. Data*, 2(3):173, 2024.

Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX and 10th International Workshop on Randomization and Computation, RANDOM, Proceedings*, pp. 316–326, 2006a.

Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-row-based methods. In *Algorithms - ESA 2006, 14th Annual European Symposium, Proceedings*, pp. 304–314, 2006b.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pp. 569–578, 2011.

Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 1343–1354, 2012.

Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte carlo algorithms for finding low-rank approximations. *J. ACM*, 51(6):1025–1041, 2004.

Anupam Gupta and Kanat Tangwongsan. Simpler analyses of local search algorithms for facility location. *CoRR*, abs/0809.2554, 2008.

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pp. 291–300, 2004.

Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In *28th Annual European Symposium on Algorithms, ESA*, volume 173, pp. 57:1–57:21, 2020.

Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pp. 1416–1429, 2020.

Lingxiao Huang, Jian Li, and Xuan Wu. On optimal coreset construction for euclidean (k, z)-clustering, 2023.

Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *International Workshop on Statistical and Computational Theories of Vision, ICCV Workshop*, volume 120, 2003.

Zachary Izzo, Sandeep Silwal, and Samson Zhou. Dimensionality reduction for wasserstein barycenter. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2021.

William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. conference in modern analysis and probability (new haven, conn., 1982), 189–206. In *Contemp. Math*, volume 26, 1984.

Max Dupré la Tour and David Saulpic. Almost-linear time approximation algorithm to euclidean k-median and k-means. *CoRR*, abs/2407.11217, 2024.

Max Dupré la Tour, Monika Henzinger, and David Saulpic. Fully dynamic k-means coreset in near-optimal update time. In *32nd Annual European Symposium on Algorithms, ESA*, pp. 100:1–100:16, 2024.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

J MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pp. 281–297, 1967.

Sepideh Mahabadi, David P. Woodruff, and Samson Zhou. Adaptive sketches for robust regression with importance sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pp. 31:1–31:21, 2022.

Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pp. 1027–1038, 2019.

Ramgopal R. Mettu and C. Greg Plaxton. Optimal time bounds for approximate clustering. *Mach. Learn.*, 56(1-3):35–60, 2004.

Raphael A. Meyer, Cameron Musco, Christopher Musco, David P. Woodruff, and Samson Zhou. Fast regression for structured inputs. In *The Tenth International Conference on Learning Representations, ICLR, 2022*, 2022.

Raphael A. Meyer, Cameron Musco, Christopher Musco, David P. Woodruff, and Samson Zhou. Near-linear sample complexity for L_p polynomial regression. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 3959–4025, 2023.

Cameron Musco and Christopher Musco. Projection-cost-preserving sketches: Proof strategies and constructions. *CoRR*, abs/2004.08434, 2020.

Cameron Musco, Christopher Musco, David P. Woodruff, and Taisuke Yasuda. Active linear regression for l_p norms and beyond. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pp. 744–753, 2022.

Mehdi Neshat, Bradley Alexander, Nataliia Sergiienko, and Markus Wagner. Large-scale Wave Energy Farm. UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C5GG7Q.

Aditya Parulekar, Advait Parulekar, and Eric Price. L1 regression with lewis weights subsampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pp. 49:1–49:21, 2021.

Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), Proceedings*, pp. 143–152, 2006.

Hugo Steinhaus et al. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci*, 1(804): 801, 1956.

Liudeng Wang, David Woodruff, Shenghao Xie, and Samson Zhou. Space-optimal streaming algorithms via efficient encodings. In *Advances in Neural Information Processing Systems 39: 40th Annual Conference on Neural Information Processing Systems*, 2026.

David P. Woodruff. Sketching as a tool for numerical linear algebra. *Found. Trends Theor. Comput. Sci.*, 10(1-2):1–157, 2014.

David P. Woodruff and Taisuke Yasuda. Online lewis weight sampling. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 4622–4666, 2023.

David P. Woodruff, Peilin Zhong, and Samson Zhou. Near-optimal k-clustering in the sliding window model. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems, NeurIPS 2023*, 2023.

A PRELIMINARIES

Given an integer $n > 0$, we use the notation $[n]$ to represent the set $\{1, \ldots, n\}$. We use $\mathrm{poly}(n)$ to represent a fixed polynomial in n and $\mathrm{polylog}(n)$ to represent $\mathrm{poly}(\log n)$. We say an event occurs with high probability if it occurs with probability at least $1 - \frac{1}{\mathrm{poly}(n)}$.

In this paper, we will focus on the Euclidean clustering, as opposed to inputs from a general metric space. As is often standard in the streaming literature, we assume $\Delta = \mathrm{poly}(n)$ and thus we allow each word of space to use $\Theta(\log(nd\Delta))$ bits of storage, so that we can store the weight and each coordinate of each point using $\mathcal{O}(1)$ words of space. Thus for vectors $x, y \in \mathbb{R}^d$, we use $\mathrm{dist}(x, y)$ to represent the Euclidean distance $\|x - y\|_2$, so that $\|x - y\|_2^2 = \sum_{i=1}^d (x_i - y_i)^2$. More generally, given a point x and a set S, we abuse notation by representing $\mathrm{dist}(x, S) := \min_{s \in S} \mathrm{dist}(x, s)$. We also recall the L_z norm of x is defined by $\|x\|_z$, where $\|x\|_z^z = \sum_{i=1}^d x_i^z$. For a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, we use $\|\mathbf{A}\|_F$ to denote the Frobenius norm, so that

$$\|\mathbf{A}\|_F^2 = \sum_{i \in [n]} j \in [d] A_{i,j}^2.$$

For a fixed $z \geq 1$ and sets $X, C \subset \mathbb{R}^d$ with $X = \{x_1, \ldots, x_n\}$ we use $\mathrm{cost}(X, C)$ to denote $\sum_{i=1}^n \mathrm{dist}(x_i, C)^z$.

We first recall the generalized triangle inequality:

Fact A.1 (Generalized triangle inequality)**.** *For any $z \geq 1$ and $x, y, z \in \mathbb{R}^d$, we have*

$$\mathit{dist}(x, y)^z \leq 2^{z-1}(\mathit{dist}(x, w)^z + \mathit{dist}(w, y)^z).$$

We next recall the standard Johnson-Lindenstrauss transformation:

Theorem A.2 (Johnson-Lindenstrauss lemma). *(Johnson & Lindenstrauss, 1984) Let $\varepsilon \in \left(0, \frac{1}{2}\right)$ and $m = \mathcal{O}\left(\frac{1}{\varepsilon^2} \log n\right)$. Let $X \subset \mathbb{R}^d$ be a set of n points. There exists a family of random linear maps $\Pi : \mathbb{R}^d \to \mathbb{R}^m$ such that with high probability over the choice of $\pi \sim \Pi$,*

$$(1-\varepsilon)\|x-y\|_2 \leq \|\pi x - \pi y\|_2 \leq (1+\varepsilon)\|x-y\|_2,$$

for all $x, y \in X$.

We next recall the following standard concentration inequality:

Theorem A.3 (Hoeffding's inequality). *Let $X_1, \cdots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$, and let $S_n = \sum_{i=1}^n X_i$. Then*

$$\mathbf{Pr}\left[|S_n - \mathbb{E}\left[S_n\right]| > t\right] \leq 2\exp\left(-\frac{t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

We next state the definition of a coreset, which is an effective summarization of the entire dataset, preserving the cost of certain query functions.

Definition A.4 (Coreset). *Let $\mathcal{Q}$ be a family of query functions on a weighted dataset $X \subseteq [\Delta]^d$ with weight function $w : X \to \mathbb{R}_{>0}$. A (strong) $(1+\varepsilon)$-coreset for $(X, \mathcal{Q})$ is a weighted subset $Y \subseteq X$ such that for all queries $q \in \mathcal{Q}$,*

$$(1-\varepsilon)\operatorname*{cost}_q(X) \leq \operatorname*{cost}_q(Y) \leq (1+\varepsilon)\operatorname*{cost}_q(X),$$

where $\operatorname{cost}_q(Y) = \sum_{x \in Y} w(x)\, q(x)$.

The above definition generalizes the coresets in the streaming problems we study. In (k, z)-clustering, q denotes a group of k centers and $q(x)$ is the distance (raised to power of z) from point x to the group of k centers. In subspace embedding, q denotes a vector $\mathbf{v} \in \mathbb{R}^d$ and $q(\mathbf{x}) = |\mathbf{x}^\top \mathbf{v}|^p$, i.e., $\operatorname{cost}_q(Y)$ is $\|\mathbf{Y}\mathbf{v}\|_p^p$, where $\mathbf{Y}$ is the matrix containing the rows in Y. In PCP, q denotes a projection matrix $\mathbf{P}$ and $\operatorname{cost}_q(Y)$ is the Frobenius norm of the orthogonal part: $\|\mathbf{Y} - \mathbf{Y}\mathbf{P}\|_F^2$, where $\mathbf{Y}$ is the matrix containing the rows in Y.

B Framework for Fast and Space-Optimal Streaming Algorithms

In this section, we formally describe our framework in Figure 2, which is applied in the following sections to achieve fast and space-optimal streaming algorithms for (k, z)-clustering, subspace embedding, and projection-cost preservation (PCP).

The standard way to construct coresets of a dataset X in the offline setting is to sample each item $x \in X$ with probability proportional to its *sensitivity* and re-weight each sampled item by the inverse of the probability. The sensitivity of x is defined by $s(x) = \max_{q \in \mathcal{Q}} \frac{\operatorname{cost}_q(x, C)}{\operatorname{cost}_q(X, C)}$, where $\operatorname{cost}_q(X, C)$ is the weighted sum of all items in X, and it captures the importance of each item with respect to the queries $\mathcal{Q}$, leading to effective coresets with small sample complexity.

In the streaming setting, it is impossible to compute these sensitivities since we do not have access to the entire dataset. To bypass this issue, one instead samples each item x_t of the stream with probability proportional to its *online sensitivity*, which measures the importance of x_t with respect to $X_t = \{x_1, \ldots, x_t\}$, i.e., the points that arrived before x_t. Unfortunately, the online sensitivities always over-estimate the actual sensitivities, and moreover, the samples are not mutually independent since we utilize previous sampled items to estimate the online sensitivity of the future incoming items, so standard concentration bounds, which requires independence, cannot be applied. These limitations lead to extra $\log n$ factors in the number of sampled edges. Recent insight by (Cohen-Addad et al., 2023) observed that online sensitivity sampling gives an insertion-only stream $\mathcal{S}'$ of length $o(n)$ that forms a $(1+\varepsilon)$-coreset, and then by running merge-and-reduce on $\mathcal{S}'$, we reduce the $\log n$ factors to $\operatorname{poly}(\log \log n)$ factors. Here, merge-and-reduce partitions the stream into consecutive blocks, builds a binary tree on the sequence of blocks, and then maintains a coreset for the points in two neighboring blocks by "merging" them using offline subroutines. Although their method achieves near-optimal

space complexity, it is slow to process at each arrival in the stream, e.g., it requires exponential time (in d and k) for (k, z)-clustering to process each item. On the other hand, running merge-and-reduce directly on the input stream would have fast update time, but we instead lose $\mathrm{polylog}(n)$ factors in the space complexity, and thus none of the existing methods meets our goal of achieving *both* near-optimal space and fast update time.

Crude sampling and refined sampling. Previous algorithms based on online sensitivity sampling require computing a constant approximation to achieve an efficient space, which becomes the main bottleneck for fast update time. Moreover, since we need to perform this operation at each step, there is no gain to amortize the time to the number of arrivals. Our key observation here is that we can pre-filter the input stream by a crude sampling scheme to obtain a stream $\mathcal{S}'$ with length $o(n)$ that gives a $(1+\varepsilon)$-coreset. Then, we use a refined sampling scheme on $\mathcal{S}'$, which provides a significantly more efficient compression of the input, but requires longer time to compute, resulting in a stream $\mathcal{S}''$. However, because these operations are performed on the input stream $\mathcal{S}'$ of length $o(n)$, then these slower operations can be amortized into lower-order terms that do not multiply a linear term in n, i.e., if the crude and refined sampling scheme uses T_1 and T_2 time to compute the sampling probability, respectively, then the amortized update time is $T_1 + n^{-\Omega(1)}T_2$. The former term is always the dominant term since T_2 does not have a dependence on n for our problems, so it remains to construct crude sampling schemes with efficient update time. We next formalize the above procedure. First, we state the definition of the crude and refined sampling scheme

Definition B.1 (Crude sampling scheme)**.** *Given a stream $\mathcal{S}$ of length n and accuracy parameter ε, a crude sampling scheme down-samples it into $\mathcal{S}'$ of length n^α for $0 < \alpha < 1$. Moreover, the outputted sub-stream $\mathcal{S}'$ is a $(1+\varepsilon)$-coreset of $\mathcal{S}$.*

Definition B.2 (Refined sampling scheme)**.** *Given a stream $\mathcal{S}'$ of length n^α and accuracy parameter ε, a refined sampling scheme down-samples it into $\mathcal{S}''$ of length $S(n,\varepsilon) \ll n$. Moreover, the outputted sub-stream $\mathcal{S}''$ is a $(1+\varepsilon)$-coreset of $\mathcal{S}'$.*

We assembly the subroutines in Algorithm 4, which is a formal presentation of Figure 2.

Algorithm 4 Fast and Space-Optimal Streaming Framework

Require: Stream $\mathcal{S}$ of n objects in $\mathbb{R}^d$ (points, rows, or vectors); target query family $\mathcal{Q}$ defining $\mathrm{cost}_{\mathcal{Q}}(\cdot)$; accuracy parameter $\varepsilon \in (0,1)$; failure probability δ; streaming algorithm $\mathcal{A}$
Ensure: $(1+\varepsilon)$-coreset of $\mathcal{S}$
1: **for** element $x_t \in \mathcal{S}$ **do**
2: Sample x_t using the crude sampling scheme and pass it to a substream $\mathcal{S}'$ {Compute the sampling probability using the sketch maintained by the refined sampling scheme}
3: **if** x_t is sampled into $\mathcal{S}'$ **then**
4: Sample x_t using the refined sampling scheme to obtain a $(1+\varepsilon)$-coreset $\mathcal{S}''$ of $\mathcal{S}'$
5: **end if**
6: **end for**
7: **return** $\mathcal{S}''$

We provide the formal guarantees of Algorithm 4 in Framework B.3.

Framework B.3. *Given a stream $\mathcal{S} \subset [\Delta]^d$ of length n, a family of query functions $\mathcal{Q}$, and an accuracy parameter ε, suppose that we have access to the following subroutines:*

- *A crude sampling scheme that returns a $(1+\varepsilon)$-coreset of size n^α with probability $1-\delta$ and uses $\mathcal{T}_1\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ time to compute each sampling probability.*
- *A refined sampling scheme that returns a $(1+\varepsilon)$-coreset with probability $1-\delta$ using $\mathcal{S}\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ bits of memory and $\mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ update time.*

Then there is an algorithm for insertion-only streams that returns a $(1+\varepsilon)$-coreset with probability $1-\delta$. The algorithm uses $\mathcal{S}\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ bits of memory and $\mathcal{T}_1\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right) + \frac{1}{n^\alpha} \cdot \mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ amortized update time. Here, we assume that the space and time complexities scale at most polynomially with ε and δ, and so we omit the constant terms.

Proof. We first prove the correctness of Algorithm 4. Note that $\mathcal{S}'$ is a $(1+\varepsilon)$-coreset of $\mathcal{S}$ and $\mathcal{S}''$ is a $(1+\varepsilon)$-coreset of $\mathcal{S}'$, then by the transitivity property of coresets, $\mathcal{S}''$ is a $(1+2\varepsilon)$-coreset of $\mathcal{S}$. The correctness then follows from setting $\varepsilon' = \frac{\varepsilon}{2}$ and $\delta' = \frac{\delta}{2}$.

Since we do not store the stream $\mathcal{S}$ obtained from the crude sampling scheme, instead, we directly pass it on to the streaming algorithm $\mathcal{A}$, the space complexity of our algorithm follows from that of $\mathcal{A}$, which is $\mathcal{S}\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$. We next calculate the amortized update time. Upon each arrival of $x_t \in \mathcal{S}$, we sample it with the crude sampling scheme, using $\mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$ update time. Then we pass on the sub-stream $\mathcal{S}'$ of size n^α to the refined sampling scheme. The time to perform each refined sampling operation is $\mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$, however, there are at most n^α times we conduct these operations, so the amortized time of the refined sampling scheme is $\frac{1}{n^\alpha} \cdot \mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$, giving a total time of $\mathcal{T}_1\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right) + \frac{1}{n^{1-\alpha}} \cdot \mathcal{T}_2\left(\mathcal{Q}, n, d, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$. □

C (k, z)-Clustering I: Structural Properties for Clustering

In this section, we summarize the background information for (k, z)-clustering and provide a brief overview of our fast algorithms introduced in Appendix D and Appendix E. Finally, we show in Appendix C.3 that the (k, z)-clustering sensitivities and the (k, z)-mediods sensitivities are within constant factors of each other, which is essential in our fast sensitivity estimation. We begin with a number of preliminaries.

We first recall the definition of a strong coreset for (k, z)-clustering.

Definition C.1 (Coreset)**.** *Given an approximation parameter $\varepsilon > 0$, and a set X of points $x_1, \ldots, x_n \in \mathbb{R}^d$ with distance function dist, a* coreset *for (k, z) clustering is a set S with weight function w such that for any set C of k points, we have*

$$(1-\varepsilon)\sum_{t=1}^{n} \mathit{dist}(x_t, C)^z \le \sum_{q\in S} w(q)\mathit{dist}(q, S)^z \le (1+\varepsilon)\sum_{t=1}^{n} \mathit{dist}(x_t, C)^z.$$

We use the following coreset construction for (k, z)-clustering:

Theorem C.2. *(Cohen-Addad et al., 2022a; Huang et al., 2023; Cohen-Addad et al., 2022b) Given an accuracy parameter $\varepsilon \in (0, 1)$, there exists a coreset construction for Euclidean (k, z)-clustering that samples $\tilde{\mathcal{O}}\left(\min\left(\frac{1}{\varepsilon^2} \cdot k^{2-\frac{z}{z+2}}, \frac{1}{\min(\varepsilon^4, \varepsilon^{2+z})} \cdot k\right)\right)$ weighted points of the input dataset.*

We next formally define the sensitivity and online sensitivity of each point x in a dataset X for (k, z)-clustering.

Definition C.3 (Sensitivities for (k, z)-clustering)**.** *The* sensitivity *of a point $x \in X$ for (k, z)-clustering in a metric space equipped with metric dist is*

$$\max_{C:|C|\le k} \frac{\mathrm{cost}(x, C)}{\mathrm{cost}(X, C)} = \max_{C:|C|\le k} \frac{\mathit{dist}(x, C)^z}{\sum_{x\in X} \mathit{dist}(x, C)^z}.$$

Definition C.4 (Online sensitivity for (k, z)-clustering)**.** *Let $x_1, \ldots, x_n$ be a sequence of points with metric dist and define $X_t := \{x_1, \ldots, x_t\}$ for all $t \in [n]$. The* online sensitivity *of x_t for (k, z)-clustering is*

$$\max_{C:|C|\le k} \frac{\mathrm{cost}(x_t, C)}{\mathrm{cost}(X_t, C)} = \max_{C:|C|\le k} \frac{\mathit{dist}(x_t, C)^z}{\sum_{i=1}^{t} \mathit{dist}(x_i, C)^z}.$$

We have the following upper bound on the sum of the online sensitivities, i.e., the total online sensitivity.

Theorem C.5. *(Cohen-Addad et al., 2023) Let $X = \{x_1, \ldots, x_n\} \subset [\Delta]^d$ be a sequence of n points and let $\sigma(x_t)$ denote the online sensitivity of x_t for $t \in [n]$ for (k, z)-clustering, where $z \ge 1$. Then*

$$\sum_{t=1}^{n} \sigma(x_t) = \mathcal{O}\left(2^{2z} k \log^2(nd\Delta)\right).$$

We next recall the guarantees of online sensitivity sampling, in terms of both correctness and sample complexity.

Theorem C.6 (Online sensitivity sampling). *(Cohen-Addad et al., 2023) Given a sequence $x_1, \ldots, x_n$ of points, suppose each point x_t is sampled with probability $p_t \geq \min(1, \gamma \cdot \sigma(x_t))$, where σ_t is the online sensitivity of x_t and $\gamma = \mathcal{O}\left(\frac{dk}{\varepsilon^2} \log \frac{n\Delta}{\varepsilon}\right)$ and weighted $\frac{1}{p_t}$ if x_t is sampled. Then with high probability, the weighted sample is a $(1+\varepsilon)$-coreset for (k, z)-clustering that contains at most $\mathcal{O}\left(\frac{dk^2}{\varepsilon^2} \log^3 \frac{n\Delta}{\varepsilon}\right)$ points.*

C.1 Background for Optimal Space Clustering in the Streaming Model

We first describe common approaches for clustering in the streaming setting. Even though they incur logarithmic overheads in the size n of the dataset compared to offline coreset constructions, their intuition will nevertheless be helpful towards our main algorithm.

Merge-and-reduce. Merge-and-reduce (Bentley & Saxe, 1980; Har-Peled & Mazumdar, 2004) is a standard approach on insertion-only streams for (k, z)-clustering, as well as many other problems for which there exist coreset constructions. Given a dataset $X \subset [\Delta]^d$ of size n, the number k of clusters, an accuracy parameter ε, a failure probability δ, suppose there exists a coreset construction algorithm for (k, z)-clustering that samples and reweights $f(n, d, k, \varepsilon, \delta)$ points of X. The merge-and-reduce framework first partitions the stream into consecutive blocks of size $f(n, d, k, \varepsilon', \delta')$, where $\varepsilon' = \frac{\varepsilon}{\mathcal{O}(\log n)}$ and $\delta' = \frac{\delta}{\text{poly}(n)}$. It then creates a coreset with accuracy $(1+\varepsilon')$ and failure probability δ' for each block of the stream. Each of these coresets uses space $f(n, d, k, \varepsilon', \delta')$ and can be viewed as the leaves of a binary tree with height $\mathcal{O}(\log n)$. For each node in the tree at depth ℓ, the merge-and-reduce framework then takes the points that are in the coresets in its children nodes at depth $\ell + 1$ and constructs a coreset with accuracy $(1+\varepsilon')$ for these points. By setting the accuracy for each coreset as $\varepsilon' = \frac{\varepsilon}{\mathcal{O}(\log n)}$, the total distortion of the merge-and-reduce tree is $(1+\varepsilon')^{\mathcal{O}(\log n)} = (1+\varepsilon)$ since the height is at most $\mathcal{O}(\log n)$. Unfortunately, the optimal coreset constructions sample $\tilde{\mathcal{O}}\left(\frac{k}{\varepsilon^2}\right) \cdot \min\left(k, \frac{1}{\min(\varepsilon^2, \varepsilon^z)}\right)$ points (Cohen-Addad et al., 2021; 2022a) and in fact for certain regimes of ε, coreset constructions for (k, z)-clustering provably require $\Omega\left(\frac{k}{\varepsilon^{z+2}}\right)$ points (Huang et al., 2023; Cohen-Addad et al., 2022b). Therefore setting $\varepsilon' = \frac{\varepsilon}{\mathcal{O}(\log n)}$ in the merge-and-reduce approach would incur $\log n$ factors that are prohibitive for our goal.

Offline sensitivity sampling. Another common approach is to adapt offline coreset constructions to the streaming model. In particular, recent efforts (Cohen-Addad et al., 2023; Woodruff et al., 2023) have been made to adapt the sensitivity sampling framework (Feldman & Langberg, 2011; Feldman & Schulman, 2012; Braverman et al., 2021) to data streams. The *sensitivity* of each point x among a dataset $X \subset \mathbb{R}^d$ of n points measures the "importance" of the point x with respect to X, for the purposes of (k, z)-clustering and, as per Definition C.3, is defined by

$$\max_{C \subset \mathbb{R}^d : |C| \leq k} \frac{\text{cost}(x, C)}{\text{cost}(X, C)} = \max_{C \subset \mathbb{R}^d : |C| \leq k} \frac{\text{dist}(x, C)^z}{\sum_{x \in X} \text{dist}(x, C)^z}.$$

There exist many variants of the sensitivity sampling framework, but the most relevant approach is sampling each point independently without replacement with probability proportional to its sensitivity, a process that can be shown to admit a coreset construction with high probability.

In fact, the argument is relatively straightforward. The analysis first fixes a set C of k centers and shows that sensitivity sampling preserves the cost of clustering X with C in expectation. It then upper bounds the variance of the cost of clustering C on the sampled points to show concentration. Although there can be arbitrary number of choices for C, for k centers among $\mathbb{R}^d$, it suffices to only show correctness-of-approximation on a net of size $\left(\frac{n}{\varepsilon}\right)^{\mathcal{O}(kd)}$, which can be handled by adjusting the probability of failure for each choice of C and then applying a union bound.

Unfortunately, the sensitivity of a point x is defined with respect to the entire dataset X, which is unknown at the time of arrival of x in a data stream, and thus the sensitivity of x cannot be computed or even well-approximated at that point.

Online sensitivity sampling. Instead, recent works have focused on *online* sensitivity sampling, where the elements of the dataset X are ordered by the time of their arrival in the data stream, so that

$X = \{x_1, \ldots, x_n\}$, where x_1 is the first item in the stream and x_n is the last item in the stream. The online sensitivity of a point x_t is then, as per Definition C.4, defined by

$$\max_{C \subset \mathbb{R}^d : |C| \leq k} \frac{\text{cost}(x_t, C)}{\text{cost}(X_t, C)} = \max_{C \subset \mathbb{R}^d : |C| \leq k} \frac{\text{dist}(x_t, C)^z}{\sum_{i=1}^{t} \text{dist}(x_i, C)^z},$$

where X_t is the subset consisting of the first t points of X. Note that while the sensitivity of a point x_t measures the importance of x_t with respect to the entire dataset X, the online sensitivity of x_t measures the importance of x_t with respect to the prefix of size t, a quantity that can be computed at the time of arrival of x_t. In particular, since online sensitivity sampling produces a $(1+\varepsilon)$-coreset of X_{t-1}, then the previously sampled points can be used to approximate the online sensitivity of x_t.

The online sensitivity framework then samples each point with probability proportional to its online sensitivity. We remark that although this process seemingly induces dependencies in the analysis, a standard martingale and coupling argument shows correctness of online sensitivity sampling. Unfortunately, as a result of the sampling probability, the total number of points sampled is proportional to the sum of the online sensitivities of the points, which can be shown to be at least $\Omega(k \log(n\Delta))$. Thus online sensitivity sampling would again incur $\log n$ factors that are prohibitive for our goal.

C.2 Overview of Our Algorithm

To begin with, we present the fast update time algorithm for (k, z)-clustering. Recall that to efficiently implement Figure 2, we first compute a crude but fast approximation to the online sensitivity of each point. Note that since the total sensitivity sums to $\mathcal{O}(k)$, then even n^α-approximation to the sensitivities, where $\alpha \in (0, 1)$, will sample $n^\alpha \cdot \text{poly}\left(k, \frac{1}{\varepsilon}, \log(n\Delta)\right) = o(n)$ points, forming the new insertion-only stream $\mathcal{S}'$ of length $o(n)$. Unfortunately, it is not clear how to efficiently acquire even crude approximations to the online sensitivities; this is the main runtime bottleneck in achieving $o(k)$ amortized update time.

(k, z)-clustering sensitivity to (k, z)-medoids sensitivity. To that end, we first define the (k, z)-medoids sensitivity of x with respect to the dataset X for (k, z)-clustering sensitivity by $\tau(x) = \max_{C \subset X : |C| \leq k} \frac{\text{cost}(x, C)}{\text{cost}(X, C)}$; we define the online sensitivities analogously. Note that for the medoids formulation, the k centers must be among the input set X, which for our purposes will be the coreset maintained by the algorithm, rather than the original input. Hence, we can assume $|X| = \text{poly}\left(k, d, \log(n\Delta), \frac{1}{\varepsilon}\right)$. We show that the (k, z)-medoids sensitivity $\tau(x)$ is a constant-factor approximation to the (k, z)-clustering sensitivity $s(x)$ as follows.

First, note that the maximization of the ratio $\frac{\text{cost}(x, C)}{\text{cost}(X, C)}$ is over a smaller search space for the possible values of C in the medoids formulation and thus $\tau(x) \leq s(x)$. To show $s(x) \lesssim \tau(x)$, we first recall that the optimal (k, z)-clustering and (k, z)-medoids clustering costs are within a factor of $2^{\mathcal{O}(z)}$ due to the generalized triangle inequality. Hence if we fix C to be a set of centers that maximizes $s(x)$, let c be the closest center of C to x and let R be the set of points of X served by c, then it suffices to show that there exists $c' \in X$ such that $\frac{\text{cost}(x, c)}{\text{cost}(R, c)} \approx \frac{\text{cost}(x, c')}{\text{cost}(R, c')}$, as we can find a $2^{\mathcal{O}(z)}$-approximate clustering of $X \setminus R$ using $k - 1$ other medoids.

For k-medoids, we perform casework on the size of the set P of points that are closer to x than to c. If $|P| < 0.99|R|$, then by a simple Markov-type argument, we can show there exists some $y \in R \setminus P$ such that $\text{cost}(y, c) \leq \frac{100}{r} \cdot \text{cost}(R, c)$, but since $y \notin P$, then y is closer to c than y is to x. Thus by setting y to be a k-medoids center, it follows from the triangle inequality that the cost of clustering R with y instead of c cannot change by much. Moreover, the distance between x and y is similar to the distance between x and c, so that $\tau(x) \approx s(x)$. On the other hand, if $|P| \geq 0.99|R|$, then there is a large number of points which are closer to x than to c by definition of P and hence $\frac{\text{cost}(x, c)}{\text{cost}(R, c)} = \mathcal{O}(1)\,|R|$. However, we can choose the point c' of P farthest away from x, so that $\frac{\text{cost}(x, c')}{\text{cost}(R, c')} \geq \frac{1}{2|R|}$ and thus $\tau(x) \approx s(x)$ again.

Constrained (k, z)-medoids clustering with a fixed center. We next show how to approximately solve the constrained medoids clustering problem $\min \text{cost}(X, C)$ across all sets $C \subset X$ of k centers containing a center at p. Suppose C is the optimal such constrained clustering. Let S be a constant-factor approximation to the optimal unconstrained (k, z)-medoids clustering on X, which can be

efficiently computed by local search (Gupta & Tangwongsan, 2008). We show that the best clustering obtainable from swapping a center in S with $\{p\}$ is a good approximation to C.

Specifically, let W be the set of $k-1$ centers $W \subseteq S$ such that $W \cup \{p\}$ has the minimum (k,z)-medoids clustering cost on X, and let $C' = W \cup \{p\}$. Let Q be the set of at most k centers of $S \cup \{p\}$ consisting of the nearest neighbors to C. For each $y \in X$, let $\pi_C(y)$ be the center in C closest to y Then we can use the triangle inequality to charge $\text{cost}(y,Q)$ in terms of $\text{cost}(y,\pi_C(y))+\text{cost}(\pi_C(y),Q)$. Since $\text{cost}(\pi_C(y),Q) \leq \text{cost}(\pi_C(y), S\cup\{p\}) \leq \text{cost}(\pi_C(y),S)$, we can use the triangle inequality to charge $\text{cost}(\pi_C(y),Q)$ to $\text{cost}(y,C)+\text{cost}(y,S)$. Finally, because C' is the k centers among $S \cup \{p\}$, summing across all $y \in X$ we can upper bound $\text{cost}(X,C') \leq \text{cost}(X,Q)$ in terms of $\text{cost}(X,C)$ and $\text{cost}(X,S)$ and hence, simply in terms of $\text{cost}(X,C)$.

However, finding C' requires finding W, the best set of $k-1$ centers $W \subseteq S$, which does not immediately seem like an easy algorithmic task. To that end, we approximate $\text{cost}(X,C')$ by first performing the following bookkeeping on S. For each center $s \in S$, we compute the number n_s (or weight) of the points of X served by s. We then compute the distance r_s from s to its closest center in $S \setminus \{s\}$ and show the clustering cost after removing s would approximately increase by $n_s \cdot (r_s)^z$.

Next, we need to update this information for each query p that we guess to serve x in the optimal constrained solution. Rather than updating the information for each center $s \in S$, we instead show that it suffices to update the information for just the nearest neighbor $u \in S$ to p. We then remove the center s with the smallest $n_s \cdot (r_s)^z$ to find W and thus C'. Finally, we show that $\text{cost}(X,S) + n_c \cdot (r_c)^z$ is both efficiently computable given our pre-processing information and also a good approximation to $\text{cost}(X,C')$.

Constrained (k,z)-medoids clustering with a fixed closest/serving center. Although C' is a set of k centers that contains our guess p for the center realizing the sensitivity of x, p may not be the nearest center in C' to x. Thus we must further utilize C' to approximately solve the constrained clustering problem $\min \text{cost}(X,C)$ across all sets $C \subset X$ of k centers containing a center at p *and no other centers that are closer to x than p*. Let C be the optimal solution to this constrained problem and let $r := \|x-p\|_2$. Our first observation is that any points served by some center $c \in C'$ in the ball $B_{r/2}(x)$ of radius $\frac{r}{2}$ around x must be served in C by a center outside the interior of the ball $B_r(x)$ of radius r around x. In fact, if n_b is the number (or weight) of points served by centers of C' in $B_{r/2}(x)$, then we would expect these points to incur additional cost $n_b \cdot r^z$ in $\text{cost}(X,C)$.

Unfortunately, it is not true that $\text{cost}(X,C') + n_b \cdot r^z$ is a good approximation to $\text{cost}(X,C)$ because there may be a large number of points served by a single center q at some distance $d_q \in \left[\frac{r}{2}, r\right)$ from x. However, since we did not include these points in n_b, then they are effectively not moved to outside of $B_r(x)$. Hence, it is possible that $\text{cost}(X,C)$ is significantly larger than $\text{cost}(X,C') + n_b \cdot r^z$.

The natural approach would be to include these points in the computation of n_b, so that perhaps we instead compute n_b' to be the number of points served by centers of C' inside $B_r(x)$ and then effectively move them to a center $B_r(x)$ via $\text{cost}(X,C') + n_b' \cdot r^z$. This still does not work because there can be a large number of points served by a center q inside but arbitrarily close to the boundary of $B_r(x)$; in this case, $\text{cost}(X,C') + n_b' \cdot r^z$ is a significant over-estimate of $\text{cost}(X,C)$.

The crucial observation is that we do not need to handle either of these cases because in both of these cases, there exists a different point p' in the annulus between $B_r(x)$ and $B_{r/2}(x)$ that approximately realizes the (k,z)-medoids sensitivity of x, such that these bad cases do not hold for p'. In particular, these problematic centers would be outside the ball of radius $\|p'-x\|_2$ centered at x and so the resulting clustering is valid for the constraint of p' serving x.

On the other hand, we must still ensure our approximation $\text{cost}(X,C') + n_b' \cdot r^z$ does not significantly overestimate the sensitivity. Thus if n_a is number of points served by centers in the annulus between $B_r(x)$ and $B_{r/2}(x)$, we show that if $n_b \geq n_a$, then $\frac{r^z}{\text{cost}(X,C')}$ is a good estimate to $\frac{\text{cost}(x,C)}{\text{cost}(X,C)}$. Otherwise if $n_b < n_a$, then we show that $\frac{r^z}{\text{cost}(X,C')}$ is an overestimate of $\frac{\text{cost}(x,C)}{\text{cost}(X,C)}$ but still upper bounded by the desired (k,z)-medoids sensitivity $\tau(x)$, up to a constant factor.

Crude quadtree for crude approximations. Finally, to implement our crude sampling scheme in Figure 2, we create a quadtree for the input set, e.g. (Indyk & Thaper, 2003; Backurs et al., 2016;

Cohen-Addad et al., 2020). Traditionally, each cell of a level of a quadtree has half the side length of a cell of an adjacent level in the quadtree. However, because we only seek n^α-approximation, we permit each cell of a level to have side length $\frac{1}{n^\alpha}$ fraction of the side length of the adjacent level. This significantly decreases the number of levels in the quadtree to a constant number of levels overall, provided that $\Delta = \text{poly}(n)$, rather than $\mathcal{O}(\log \Delta)$ levels in the standard quadtree. We then run the same algorithm above searching up the levels of the quadtree rather than the radially outward ball.

Specifically, we estimate the distance of the center serving the point x whose sensitivity we wish to approximate. Each estimate of a possible distance corresponds to a separate level in the quadtree. For a fixed level β in the quadtree, we first take a constant-factor approximation to the unconstrained (k, z)-clustering problem to find a crude-approximation to the optimal (k, z)-clustering constrained to having a center in the same cell as x at level β. Our algorithm provides an estimate Ψ of this cost by deleting the center that increases the overall cost the least, among the centers in the constant-factor solution adjoined with a center in the same cell as x at level β. We then ensure that x is served by a center at level β by moving all centers in the same cell as x before level β "upward" until level β by computing the number (or weight) n_β of points served by these centers in Ψ and again increasing Ψ by $n_\beta \cdot r^z$, where r is the distance estimated by the quadtree for a center serving x at level β.

C.3 (k, z)-Clustering Sensitivities and (k, z)-medoids Sensitivities

In this section, we show that the sensitivity $\tau(x)$ for a point x with respect to the dataset X for k-medoids is a constant-factor approximation to the sensitivity $s(x)$ for a point x with respect to the dataset X for (k, z)-clustering. Thus to acquire a constant-factor approximation to the sensitivity $s(x)$, it suffices to approximate the (k, z)-medoids sensitivity $\tau(x)$, and vice versa.

We first recall the well-known fact that the optimal (k, z)-clustering and (k, z)-medoids costs are within a constant factor of each other. For the sake of completeness, we include the proof.

Lemma C.7. *For each $X \subset [\Delta]^d$, the optimal (k, z)-clustering cost is a 2^{z+1}-approximation of the optimal (k, z)-medoid clustering cost.*

Proof. Let C be an optimal (k, z)-clustering of X and for a fixed $c \in C$, let P be the subset of X served by c. Let $c' \in P$ be the point closest to c. Then we have for all $p \in P$,

$$\|c' - p\|_2^z \le 2^z(\|c' - c\|_2^2 + \|p - c\|_2^2) \le 2^{z+1}\|p - c\|_2^2.$$

Therefore, $\text{cost}(P, c') \le 2^{z+1} \cdot \text{cost}(P, c)$. The conclusion then follows by iterating over all $c \in C$. □

We now show that the (k, z)-clustering and (k, z)-medoids sensitivities are within a constant factor of each other.

Theorem C.8. *There exists a constant $\gamma \le 2^{z+1} \cdot 101$ such that the sensitivity of a point x with respect X for (k, z)-clustering is a γ-approximation to the sensitivity of x with respect to X for (k, z)-medoids.*

Proof. Let C be any set of k centers that achieves the sensitivity $s(x)$ for (k, z)-clustering and let S be any set of k centers that achieves the sensitivity $\tau(x)$ for (k, z)-medoids. Since $S \subseteq X \subseteq [\Delta]^d$ and $|S| \le k$, then by the maximality characterization of sensitivity, we have $\tau(x) \le s(x)$.

Now, let $c \in C$ be the closest center to x. Let $R \subset X$ be the set of points served by c and let $r = |R|$. Firstly, let $w \in R$ be the farthest point from x. Then

$$\max_{c' \in X} \frac{\text{cost}\, x, c'}{\text{cost}(R, c')} \ge \frac{\|w - x\|_2}{\sum_{y \in R} \|w - y\|_2} \ge \frac{\|w - x\|_2}{\sum_{y \in R} \|w - x\|_2 + \|y - x\|_2} \ge \frac{1}{2|R|},$$

since $\|w - x\|_2 \ge \|y - x\|_2$ for all $y \in R$. Thus, $\frac{\|w-x\|_2}{\sum_{y \in R} \|w-y\|_2} \ge \frac{1}{2r}$.

Let $P \subseteq R$ be the set of points y that are closer to x than to c. We perform casework on the size $|P|$ of P.

First, suppose $|P| < 0.99r$. Note that there are at most $0.01r$ points y such that $\text{cost}(y, c) > \frac{100}{r} \cdot \text{cost}(R, c)$. Thus there are at least $0.99r$ points y such that $\text{cost}(y, c) \le \frac{100}{r} \cdot \text{cost}(R, c)$. If

$|P| < 0.99r$, then there exists $y \in R$ such that $y \notin P$, so that $\text{cost}(y, c) \leq \frac{100}{r} \cdot \text{cost}(R, c)$ but $\|y - c\|_2 \leq \|y - x\|_2$. But then if we set $c' = y$, we have that $\|x - c'\|_2 \geq \frac{1}{2}\|x - c\|_2$ and moreover,

$$\text{cost}(R, c') \leq 2^z(\text{cost}(R, c) + |R| \cdot \text{cost}(C, c')) \leq 2^z(101 \cdot \text{cost}(R, c)).$$

Hence we have

$$\frac{\text{cost}(x, c')}{\text{cost}(R, c')} \geq \frac{1}{2^{z+1} \cdot 101} \frac{\text{cost}(x, c)}{\text{cost}(R, c)}.$$

The claim then follows by clustering $X \setminus R$ with the remaining $k - 1$ centers using Lemma C.7.

Otherwise, suppose $|P| \geq 0.99r$. Furthermore, it suffices to assume that there exist $0.99r$ points $y \in P$ such that $\text{cost}(y, c) \leq \frac{100}{r} \cdot \text{cost}(R, c)$, or else we could reach the same conclusion as the previous case where $|P| < 0.99r$ by moving c to such a point y. Since these points are closer to x than to c by definition of P, then we have $\|y - c\|_2 \geq \frac{1}{2} \cdot \|x - c\|_2$. Thus we have $\text{cost}(R, c) \geq 0.99r \cdot \frac{1}{2} \cdot \|x - c\|_2^2$. Hence, $\frac{\text{cost}(x,c)}{\text{cost}(R,c)} \leq \frac{4}{r}$. On the other hand, by the above argument, we have $\max_{c' \in X} \frac{\text{cost}\, x,c'}{\text{cost}(R,c')} \geq \frac{1}{2r}$ and thus $\tau(x)$ is a $2^{z+1} \cdot 8$-approximation to $s(x)$, after clustering the remaining points in $X \setminus R$ with the other $k - 1$ centers. □

Finally, we show that the sensitivities of points are preserved under coresets. That is, the (k, z)-clustering sensitivity of x with respect to X is well-approximated by the sensitivity of x with respect to a coreset Z of X.

Lemma C.9. *Let $\gamma \geq 1$ be fixed and let Z be a γ-strong coreset for X. For $p \in X$, let $s_X(p) = \max_{C:|C|\leq k} \frac{\text{cost}(p,C)}{\text{cost}(X,C)}$ and let $s_Z(p) = \max_{C:|C|\leq k} \frac{\text{cost}(p,C)}{\text{cost}(Z,C)}$. Then*

$$\frac{1}{\gamma} \cdot s_Z(p) \leq s_X(p) \leq \gamma \cdot s_Z(p).$$

Proof. Let S be a set of k centers in C that achieves the sensitivity with respect to Z, so that $s_Z(p) = \frac{\text{cost}(p,S)}{\text{cost}(X,S)}$. Since Z is a γ-strong coreset for X, then we have $\frac{1}{\gamma} \cdot \text{cost}(Z, S) \leq \cdot \text{cost}(X, S) \leq \gamma \cdot (Z, S)$. Therefore, we have

$$\frac{1}{\gamma} \cdot s_Z(p) \leq s_X(p) \leq \gamma \cdot s_Z(p).$$

□

D (k, z)-Clustering II: Faster Sensitivity Estimation

In this section, we show that our streaming algorithm can be implemented using $dk \cdot \text{polylog}(\log(n\Delta))$ amortized update time, while also providing crucial structural properties that will be ultimately used for our algorithm that uses $\mathcal{O}(d \log k) + \log^2(k) \cdot \text{polylog}(\log(n\Delta))$ amortized update time in Appendix E. We remark that by comparison, existing algorithms (Henzinger & Kale, 2020; Bhattacharya et al., 2023; 2024a;b) use $\text{poly}(k) \cdot \text{polylog}(n\Delta)$ amortized update time, which is exponentially larger in terms of the dependency in the n and Δ factors. On the other hand, we emphasize that the main goal of (Henzinger & Kale, 2020; Bhattacharya et al., 2023; 2024a;b) is to handle the fully dynamic setting, where insertions and deletions of points are both permitted, which we cannot handle, though they also require the entire dataset to be stored.

We first use the following guarantee about quadratic runtime for the local search algorithm for (k, z)-clustering.

Theorem D.1. *(Gupta & Tangwongsan, 2008) For each constant $z \geq 1$, there exists a polynomial time algorithm that outputs a $\mathcal{O}(z)$-approximation to (k, z)-clustering in time $\mathcal{O}(dn^2)$.*

We now show how the approximately solve the constrained clustering problem $\min \text{cost}(X, C)$ across all sets $C \subset X$ of k centers containing a center at a fixed center x. The algorithm is quite simple. We compute a constant-factor approximation C to the optimal (k, z)-medoids clustering on X. We then swap one of the centers of C with $\{x\}$, choosing the swap with the best subsequent clustering cost.

Lemma D.2. *For a fixed constant $\gamma \geq 1$, let C be a γ-approximation to the optimal (k, z)-medoids clustering on $X \subset [\Delta]^d$. Let S be the optimal (k, z)-medoids clustering on $X \subset [\Delta]^d$ containing a center at a fixed $x \in X$. Let W be the set of $k-1$ centers $W \subseteq C$ such that $W \cup \{x\}$ has the minimum (k, z)-medoids clustering cost on X, and let $C' = W \cup \{x\}$. Then*

$$\mathrm{cost}(X, S) \leq \mathrm{cost}(X, C') \leq (2^z + 2^{2z} + 2^{2z} \cdot \gamma) \cdot \mathrm{cost}(X, S).$$

Proof. Since C' is a set of k centers containing x and S is the optimal clustering among such sets of k centers containing x, then $\mathrm{cost}(X, S) \leq \mathrm{cost}(X, C')$. It thus remains to prove the right-hand side of the inequality.

Let $S' = C \cup \{p\}$ so that $|S'| = k + 1$ and also $C' \subset S'$ with $|S' \setminus C'| = 1$. Let $\pi_{S'} : S \to C'$ be the mapping that assigns each center $s \in S$ to the closest center in S', breaking ties arbitrarily. Let $Q = \{\pi_{S'}(s) \mid s \in S\}$ and note that $x \in Q$ since $x \in S$ and $x \in S'$. Moreover, since $|S| = k$, then $|Q| = k$, and since $Q \subset S'$ and C' is the best subset of S', then $\mathrm{cost}(X, C') \leq \mathrm{cost}(X, Q)$.

Now for any $p \in X$, consider its contribution to $\mathrm{cost}(X, Q)$. Let $\pi_S : X \to S$ be the mapping that assigns each point $x \in X$ to the closest center in S, breaking ties arbitrarily. Then by generalized triangle inequality, c.f., Fact A.1,

$$\mathrm{cost}(p, Q) \leq 2^z \cdot \mathrm{cost}(p, \pi_S(p)) + 2^z \cdot \mathrm{cost}(\pi_S(p), Q).$$

Since $\pi_{S'}$ maps each center in S to its closest center in S', then we have

$$\mathrm{cost}(\pi_S(p), Q) = \mathrm{cost}(\pi_S(p), S').$$

Thus,

$$\mathrm{cost}(p, Q) \leq 2^z \cdot \mathrm{cost}(p, \pi_S(p)) + 2^z \cdot \mathrm{cost}(\pi_S(p), S').$$

We also have by generalized triangle inequality, c.f., Fact A.1,

$$\mathrm{cost}(\pi_S(p), S') \leq 2^z \cdot \mathrm{cost}(\pi_S(p), p) + 2^z \cdot \mathrm{cost}(p, S') = 2^z \cdot \mathrm{cost}(p, S) + 2^z \cdot \mathrm{cost}(p, S').$$

Therefore,

$$\begin{aligned} \mathrm{cost}(p, Q) &\leq 2^z \cdot \mathrm{cost}(p, \pi_S(p)) + 2^{2z} \cdot \mathrm{cost}(p, S) + 2^{2z} \cdot \mathrm{cost}(p, S') \\ &= 2^z \cdot \mathrm{cost}(p, S) + 2^{2z} \cdot \mathrm{cost}(p, S) + 2^{2z} \cdot \mathrm{cost}(p, S'). \end{aligned}$$

Summing across $p \in X$, we thus have

$$\mathrm{cost}(X, Q) \leq 2^z \cdot \mathrm{cost}(X, S) + 2^{2z} \cdot \mathrm{cost}(X, S) + 2^{2z} \cdot \mathrm{cost}(X, S').$$

Since C provides a γ-approximation to the optimal (k, z)-medoids clustering on $X \subset [\Delta]^d$ and S is the optimal (k, z)-medoids clustering on X containing a center at x, then $\mathrm{cost}(X, C) \leq \gamma \cdot \mathrm{cost}(X, S)$. Since $S' = C \cup \{p\}$, then $\mathrm{cost}(X, S') \leq \mathrm{cost}(X, C) \leq \gamma \cdot \mathrm{cost}(X, S)$ and thus we have

$$\mathrm{cost}(X, Q) \leq (2^z + 2^{2z} + 2^{2z} \cdot \gamma) \cdot \mathrm{cost}(X, S).$$

Finally, since $\mathrm{cost}(X, C') \leq \mathrm{cost}(X, Q)$, then the desired claim follows. □

Unfortunately, we cannot afford to exactly compute the cost of the best replacement of a center in C with $\{x\}$. Thus we show that by snapping the points served by each center $c \in C$ and moving these points all to the closest center if c is removed gives a good approximation to the cost of the best swap. In particular, if n_c is the number of points served by c and p is the closest center of $C \cup \{x\}$ to c, then $\mathrm{cost}(X, C) + n_c \cdot \|c - p\|_2^z$ is a good approximation to the cost of replacing c with x in C.

Lemma D.3. *Let $\Psi = \mathrm{cost}(X, C) + \psi$ be the output of Algorithm 1. Let S be the optimal (k, z)-medoids clustering on $X \subset [\Delta]^d$ containing a center at a fixed $x \in X$. Then there exists a constant $\gamma = 2^{\Theta(z)}$ such that*

$$\mathrm{cost}(X, S) \leq \Psi \leq \gamma \cdot \mathrm{cost}(X, S).$$

Proof. Let W be the set of $k-1$ centers $W \subseteq C$ such that $W \cup \{x\}$ has the minimum (k, z)-medoids clustering cost on X, and let $C' = W \cup \{x\}$. By Lemma D.2, there exists a constant $\gamma_1 \geq 1$ such that

$$\mathrm{cost}(X, S) \leq \mathrm{cost}(X, C') \leq (2^z + 2^{2z} + 2^{2z} \cdot \gamma_1) \cdot \mathrm{cost}(X, S).$$

Algorithm 5 Approximation algorithm for finding a (k, z)-medoids clustering containing a fixed center, restatement of Algorithm 1

Require: Constant-factor approximation $C \subset X \subset [\Delta]^d$ for (k, z)-medoids clustering on X; query point $x \in X$
Ensure: Constant-factor approximation for (k, z)-medoids clustering on X with a center at x
1: Initialize $\psi \leftarrow nd\Delta$ and $\psi_x \leftarrow \emptyset$
2: **for** $c \in C$ **do**
3: Let n_c be the number of points in X assigned to c under C
4: **if** $n_c \cdot \|c - x\|_2^z < \psi$ **then**
5: Update
$$\psi \leftarrow n_c \cdot \min_{p \in C \cup \{x\} \setminus \{c\}} \|c - p\|_2^z$$
and set $\psi_x \leftarrow c$
6: **end if**
7: **end for**
8: **return** $(\psi_x, \text{cost}(X, C) + \psi)$

We claim Ψ gives a constant-factor approximation to $\text{cost}(X, C')$.

By generalized triangle inequality,

$\text{cost}(X, C') \le 2^z \cdot \text{cost}(X, C) + 2^z \cdot \text{cost}(C, C') \le 2^z \cdot \text{cost}(X, C) + 2^{2z} \cdot \text{cost}(X, C) + 2^{2z} \cdot \text{cost}(X, C')$.

Let C be a γ_2-approximation to the optimal unconstrained (k, z)-medoids clustering. Then $\text{cost}(X, C) \le \gamma_2 \cdot \text{cost}(X, C')$, so that

$$\text{cost}(X, C') \le 2^z \cdot \text{cost}(X, C) + 2^z \cdot \text{cost}(C, C') \le (2^z \gamma_2 + 2^{2z} \gamma_2 + 1) \cdot \text{cost}(X, C').$$

Hence, it suffices to show that Ψ gives a constant-factor approximation to $\text{cost}(X, C) + \text{cost}(C, C')$.

Note that $|C \setminus C'| = 1$, then $\text{cost}(C, C') = \min_{c \in C} n_c \cdot \min_{p \in C'} \|c - p\|_2^z$, where n_c is the number of points assigned to c by X. Thus, we have $\psi = \text{cost}(C, C')$, so that

$$\Psi = \text{cost}(X, C) + \text{cost}(C, C').$$

Putting things together, we have that there exists a constant $\gamma = 2^{\Theta(z)}$ such that

$$\text{cost}(X, S) \le \Psi \le \gamma \cdot \text{cost}(X, S).$$

□

Unfortunately, we may not have time to compute the closest center p to each center c among $C \cup \{x\} \setminus \{c\}$. Thus we show that it suffices to just update the information for the center $u \in C$ that is closest to x.

Lemma D.4. *Let C be a constant-approximation to the optimal (k, z)-medoids clustering on $X \subset [\Delta]^d$. For a point $x \in X$, let u be the center of C closest to x, breaking ties arbitrarily. Then for any center $c \in C \setminus \{u\}$,*

$$\min_{p \in C \cup \{x\} \setminus \{c\}} \|c - p\|_2^z \le \min_{p \in C \setminus \{c\}} \|c - p\|_2^z \le 2^{z+1} \min_{p \in C \cup \{x\} \setminus \{c\}} \|c - p\|_2^z.$$

Proof. Note that the left-hand side of the inequality immediately holds because the minimization is taken over a larger superset on the left-most term. We now justify the right-hand side of the inequality.

Let p be the closest center of $S \setminus \{c\}$ to c. Observe that if $\|c - p\|_2 \le \|c - x\|_2$, then p remains the closest center of $S \cup \{x\} \setminus \{c\}$ to c and thus the claim holds.

Hence, it remains to consider the setting where $\|c - p\|_2 < \|c - x\|_2$. In this case, we have by the optimality of p in S, $\|c - p\|_2 \le \|c - u\|_2$. Thus by generalized triangle inequality,

$$\|c - p\|_2^z \le \|c - u\|_2^z \le 2^z \cdot \|u - x\|_2^z + 2^z \cdot \|c - x\|_2^z.$$

Since u is the closest center in S to x and $c \in S$, then it follows that $\|u - x\|_2 \le \|c - x\|_2$ and thus

$$\|c - p\|_2^z \le 2^{z+1} \cdot \|c - x\|_2^z.$$

□

To estimate the sensitivity of a point x, we enumerate over all possible centers p that serve x. To that end, we show that $\Phi + n_b \cdot r^z$ can only be an underestimate to the optimal (k, z)-medoids clustering constrained to the center p being the closest to x.

Lemma D.5. *Let $x \in X$ be a fixed point and $p \in X$ be a fixed center with $r = \|x - p\|_2$. Let Q be a constant-factor approximation to the optimal (k, z)-medoids clustering on X containing a center at p and let Ψ be a constant-factor approximation to $\mathrm{cost}(X, Q)$. Let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$. Let n_b be the number of points assigned to centers in $B_{r/2}(x)$. There exists a constant $\gamma \geq 1$ such that*

$$\Psi + n_b \cdot r^z \leq \gamma \cdot \mathrm{cost}(X, S).$$

Proof. Let Q be a ζ_1-approximation to the optimal (k, z)-medoids clustering on X containing a center at p. Since S is the optimal solution for a more constrained search space, i.e., sets of at most k centers that contain a center at p and no center in the interior of $B_r(x)$, then we have

$$\mathrm{cost}(X, Q) \leq \zeta_1 \cdot \mathrm{cost}(X, S).$$

Let Ψ be a ζ_2-approximation to $\mathrm{cost}(X, Q)$, so that

$$\Psi \leq \zeta_1 \zeta_2 \cdot \mathrm{cost}(X, S).$$

Now, consider a point y served by a center $q \in Q$ inside $B_{r/2}(x)$. If $\|y - q\|_2 \geq \frac{r}{4}$, then we have

$$\|y - q\|_2 + r \leq 5 \cdot \|y - q\|_2.$$

Otherwise, if $\|y - q\|_2 < \frac{r}{4}$, then by triangle inequality, $\mathrm{dist}(y, S) \geq \mathrm{dist}(q, S) - \|y - q\|_2 \geq \frac{3r}{4}$. Thus,

$$\|y - q\|_2 + r \leq \|y - q\|_2 + 2 \cdot \mathrm{dist}(y, S).$$

Taking both cases together, we have

$$\|y - q\|_2 + r \leq 5 \cdot \|y - q\|_2 + 2 \cdot \mathrm{dist}(y, S).$$

Summing across all y and q, we have

$$\mathrm{cost}(X, Q) + n_b \cdot r^z \leq 5\,\mathrm{cost}(X, Q) + 2\,\mathrm{cost}(X, S) \leq (5\gamma + 2) \cdot \mathrm{cost}(X, S).$$

Since Ψ is a ζ_2-approximation to $\mathrm{cost}(X, Q)$, then there exists a constant $\gamma_2 \geq 1$ such that

$$\Psi + n_b \cdot r^z \leq \gamma \cdot \mathrm{cost}(X, S).$$

□

Now we show that if there is a large number of points served by centers inside the ball $B_{r/2}(x)$ of radius $\frac{r}{2}$ around x, where $r := \|x - p\|_2$, then $\Phi + n_b \cdot r^z$ is a good estimate to the cost of the optimal (k, z)-medoids clustering constrained to the center p being the closest to x.

Lemma D.6. *Let $x \in X$ be a fixed point and $p \in X$ be a fixed center with $r = \|x - p\|_2$. Let Q be a constant-factor approximation to the optimal (k, z)-medoids clustering on X containing a center at p and let Ψ be a constant-factor approximation to $\mathrm{cost}(X, Q)$. Let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$. Let n_b be the number of points assigned to centers in $B_{r/2}(x)$ and let n_a be the number of points assigned to centers in the annulus $B_r(x) \setminus B_{r/2}(x)$. If $n_b \geq n_a$, then there exist constants $\gamma_1, \gamma_2 \geq 1$ such that*

$$\mathrm{cost}(X, S) \leq \gamma_1(\Psi + n_b \cdot r^z) \leq (\gamma_1 \gamma_2) \cdot \mathrm{cost}(X, S).$$

Proof. We show that there exists a constant $\gamma_1 \geq 1$ with $\mathrm{cost}(X, S) \leq \gamma_1(\Psi + n_b \cdot r^z)$. Observe that this would imply by Lemma D.5 that there also exists a constant $\gamma_2 \geq 1$ such that $\gamma_1(\Psi + n_b \cdot r^z) \leq (\gamma_1 \gamma_2) \cdot \mathrm{cost}(X, S)$.

Recall that S is the optimal solution across sets of at most k centers that contain a center at p and no center in the interior of $B_r(x)$.

Consider a point y served by a center $q \in Q$ inside $B_{r/2}(x)$. Let $\pi_S(y)$ be the closest center in S to y, breaking ties arbitrarily. By generalized triangle inequality,

$$\mathrm{cost}(y, S) \leq 2^z \cdot \|y - q\|_2^z + 2^z \cdot \mathrm{cost}(q, \pi_S(y)).$$

Summing across all y served by centers $q \in B_{r/2}(x)$ and noting that $\text{dist}(q, S) \leq 2r$, we have

$$\sum_{y:\pi_Q(y)\in B_{r/2}(x)} \text{cost}(y, S) \leq 8^z \cdot n_b \cdot \left(\frac{r}{2}\right)^z + 2^z \cdot \sum_{y:\pi_Q(y)\in B_{r/2}(x)} \text{cost}(y, Q).$$

Next, consider a point y served by a center $q \in Q$ inside $B_r(x)\setminus(B_{r/2}(x))$. Note that if $\|y-q\|_2 \geq \frac{r}{4}$, then $\|y - p\|_2 \leq \|y - q\|_2 + \|q - p\|_2 \leq 9 \cdot \|y - q\|_2$. Otherwise if $\|y - q\|_2 < \frac{r}{4}$, then $\|y - p\|_2 \leq \|y - q\|_2 + \|q - p\|_2 < 3r$. Summing across all $y \in X$ served by centers $q \in B_{r/2}(x)$ and noting that $n_b \geq n_a$, we have

$$\sum_{y:\pi_Q(y)\in B_r(x)\setminus(B_{r/2}(x))} \text{cost}(y, S) \leq 12^z \cdot n_b \cdot \left(\frac{r}{2}\right)^z + 9 \cdot 2^z \cdot \sum_{y:\pi_Q(y)\in B_r(x)\setminus(B_{r/2}(x))} \text{cost}(y, Q).$$

Finally, for the points y served by a center $q \in Q$ outside $B_r(x)$, note that S can only have more centers outside $B_r(x)$ than C because S cannot have centers inside $B_r(x)$ whereas C can. Thus by optimality of S,

$$\sum_{y:\pi_Q(y)\in X\setminus(B_r(x))} \text{cost}(y, S) \leq \sum_{y:\pi_Q(y)\in X\setminus(B_r(x))} \text{cost}(y, C).$$

Putting these together by summing across all y served by centers $q \in B_{r/2}(x)$, $q \in B_r(x) \setminus B_{r/2}(x)$, and $q \in X \setminus B_r(x)$, we have that there exists a constant $\zeta \geq 1$ such that

$$\text{cost}(X, S) \leq \zeta \cdot 2^{\mathcal{O}(z)} \cdot \left(\text{cost}(X, Q) + n_b \cdot \left(\frac{r}{2}\right)^z\right).$$

Since Φ is a ζ_2-factor approximation to $\text{cost}(X, Q)$, then there exists a constant γ_1 such that

$$\text{cost}(X, S) \leq \gamma_1(\Psi + n_b \cdot r^z).$$

Therefore, we have

$$\text{cost}(X, S) \leq \gamma_1(\Psi + n_b \cdot r^z) \leq (\gamma_1\gamma_2) \cdot \text{cost}(X, S),$$

as desired. □

On the other hand, if the number of points served by centers inside the ball $B_{r/2}(x)$ of radius $\frac{r}{2}$ around x is small, then $\Phi + n_b \cdot r^z$ may not be a good estimate to the cost of the optimal (k, z)-medoids clustering S constrained to the center p being the closest to x. Nevertheless, we show that it can provide an upper bound on $\frac{\text{cost}(x,S)}{\text{cost}(X,S)}$, while still being at most the sensitivity $s(x)$ of x.

Lemma D.7. *Let $x \in X$ be a fixed point and $p \in X$ be a fixed center with $r = \|x - p\|_2$. Let $s(x)$ be the sensitivity of (k, z)-medoids clustering with respect to X. Let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$. Let n_b be the number of points assigned to centers in $B_{r/2}(x)$ and let n_a be the number of points assigned to centers in the annulus $B_r(x) \setminus B_{r/2}(x)$. If $n_b < n_a$, then*

$$\frac{1}{\gamma_1}\frac{\text{cost}(x, S)}{\text{cost}(X, S)} \leq \frac{r^z}{\Psi + n_b \cdot r^z} \leq 2^z \cdot \gamma_2 \cdot s(x),$$

for fixed constants $\gamma_1, \gamma_2 \geq 1$.

Proof. Let Q be a ζ_1-factor approximation to the optimal (k, z)-medoids clustering on X containing a center at p and let Ψ be a ζ_2-factor approximation to $\text{cost}(X, Q)$. By Lemma D.5, we have that there exists a constant $\gamma_1 \geq 1$ such that

$$\Psi + n_b \cdot r^z \leq \gamma_1 \cdot \text{cost}(X, S).$$

Since S contains a center at p and no centers in the interior of $B_r(x)$, then $\text{cost}(x, S) = r^z$. Then we have

$$\frac{1}{\gamma_1} \cdot \frac{\text{cost}(x, S)}{\text{cost}(X, S)} \leq \frac{r^z}{\Psi + n_b \cdot r^z}.$$

On other hand, for a set W of k centers that contains no center in the interior of $B_{r/2}(x)$ and a center at p, so that we have $\frac{r}{2} \leq \text{dist}(x, W) \leq r$ and $\text{cost}(X, W) \leq \gamma_2(\Psi + n_b \cdot r^z)$ for some constant $\gamma_2 \geq 1$. Thus by the maximality of sensitivity, we have

$$\frac{r^z}{\Psi + n_b \cdot r^z} \leq 2^z \cdot \gamma_2 \cdot \frac{\text{cost}(x, W)}{\text{cost}(X, w)} \leq 2^z \cdot \gamma_2 \cdot s(x).$$

□

Algorithm 6 BATCHSENS: Fast batch approximation of sensitivities, restatement of Algorithm 2

Require: Input set $Z \subset [\Delta]^d$ for (k, z)-medoids clustering; batch B of k query points
Ensure: Constant-factor approximation of the sensitivity of x for (k, z)-medoids clustering on $Z \cup B$, for all $x \in B$

Compute S as a constant-factor solution on $Z \cup B$ for (k, z)-medoids clustering
for $x \in B$ **do**
  Initialize $\widehat{s(x)} \leftarrow 0$
  for $p \in Z \cup B$ **do**
    Let Q be a constant-factor approximation of the optimal (k, z)-medoids clustering on X that contains a center at p {Only needs to be preprocessed and computed once per p across all x}
    Use S to compute a constant-factor approximation Ψ of $\text{cost}(X, Q)$ {Algorithm 1}
    Set $r \leftarrow \|x - p\|_2$
    Let n_b be the number of points assigned by Q to centers of Q lying in the ball $B_{r/2}(x)$
    Update

$$\widehat{s(x)} \leftarrow \max\left(\widehat{s(x)}, \frac{r^z}{\Psi + n_b \cdot r^z}\right).$$

  end for
  return $\widehat{s(x)}$
end for

It follows that our algorithm in BATCHSENS provides constant-factor approximations to all sensitivities x in the batch B.

Lemma D.8. *For a fixed $x \in X$, let $s(x)$ be the sensitivity of (k, z)-medoids clustering with respect to X. Let $\widehat{s(x)}$ be the output of Algorithm 2, for each $x \in B$, where B is the input batch. Then there exist constants $\gamma_1, \gamma_2 \geq 1$ such that*

$$\frac{1}{\gamma_1} \cdot s(x) \leq \widehat{s(x)} \leq 2^z \cdot \gamma_2 \cdot s(x).$$

Proof. For each fixed $p \in X$, let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$. Let n_b be the number of points assigned to centers in $B_{r/2}(x)$ and let n_a be the number of points assigned to centers in the annulus $B_r(x) \setminus B_{r/2}(x)$. We perform casework on whether $n_b < n_a$ or $n_b \geq n_a$.

In the case where $n_b \geq n_a$, then by Lemma D.6, we have

$$\text{cost}(X, S) \leq \gamma_1(\Psi + n_b \cdot r^z) \leq (\gamma_1\gamma_2) \cdot \text{cost}(X, S).$$

Since $\text{cost}(x, S) = r^z$, then it holds that $\frac{r^z}{\Psi + n_b \cdot r^z}$ is a constant-factor approximation to $\frac{\text{cost}(x,S)}{\text{cost}(X,S)}$.

Otherwise if $n_b < n_a$, then by Lemma D.7,

$$\frac{1}{\gamma_1} \frac{\text{cost}(x, S)}{\text{cost}(X, S)} \leq \frac{r^z}{\Psi + n_b \cdot r^z} \leq 2^z \cdot \gamma_2 \cdot s(x),$$

for fixed constants $\gamma_1, \gamma_2 \geq 1$. In particular, for the center $p \in X$ that realizes the sensitivity $s(x)$, we have that $\frac{r^z}{\Psi + n_b \cdot r^z}$ is an $\mathcal{O}(2^z)$-approximation to $s(x)$. For centers $q \in X$ that do not realize the sensitivity, we have that $\frac{r^z}{\Psi + n_b \cdot r^z} \leq \gamma_2 \cdot s(x)$. Therefore,

$$\frac{1}{\gamma_1} \cdot s(x) \leq \widehat{s(x)} \leq 2^z \cdot \gamma_2 \cdot s(x).$$

□

It remains to analyze the amortized runtime of the algorithm BATCHSENS.

Lemma D.9. *Given a constant-factor coreset Z to $X \subset [\Delta]^d$, there exists an absolute constant $C > 1$ and an algorithm* BATCHSENS *that outputs C-approximations to the sensitivities to each of the points $x \in B$ in a batch B of k points of X, using amortized $\frac{(|Z|+k\log k)^2}{k} \cdot \mathrm{poly}(d)$ time.*

Proof. Consider BATCHSENS in Algorithm 2. By Lemma C.9, to compute the sensitivity of $x \in B$ with respect to X, it suffices to compute the sensitivity of $x \in Z \cup B$, since Z is a constant-factor coreset to Y and thus $Z \cup B$ is a constant-factor coreset to $X = Y \cup B$. By Theorem C.8, it suffices to compute a constant-factor approximation to the (k, z)-medoids clustering sensitivity of x with respect to $Z \cup B$. Hence, we want to compute $\max_{C \subset Z \cup B : |C| \le k} \frac{\mathrm{cost}(x,C)}{\mathrm{cost}(X,C)}$. Correctness thus follows by Lemma D.8.

It remains to analyze the amortized runtime of BATCHSENS. First, note that it takes time $\mathcal{O}\left(d(|Z|+k)^2\right)$ to compute a set S of k centers that serve as a constant-factor approximation to $Z \cup B$. We can then pre-process $(Z \cup B)$ in $\mathcal{O}\left(d(|Z|+k)^2\right)$ time so that for any $x \in Z \cup B$, we can find the set W of $k-1$ centers of S that after adjoining with $\{x\}$ achieves the best k-medoids clustering on $(Z \cup B)$ in $\mathcal{O}(k)$ time. Specifically, for each center $s \in S$, we can store the number η_s of points assigned to s, as well as the distance d_s to the closest center in $S \setminus \{s\}$.

We can then update the information in time $\mathcal{O}(d(k+|Z|))$ for each query $\{p\}$ to add to S, to ultimately form Q. In particular, we find $u \in S$ that is the nearest center of S to p and update d_u to be $\|u-p\|_2$ if $\|u-p\|_2 < d_u$. By Lemma D.4, the points served by the remaining centers all have the distances to their nearest centers preserved up to a 2^{z+1}-factor. Thus, we can approximate ψ in Algorithm 1 up to a 2^{z+1}-factor, along with the corresponding center ψ_p to be removed from S. By Lemma D.2, the resulting set Q of k centers is a $2^{\mathcal{O}(z)}$-approximation to the optimal set S of centers for (k, z)-medoids clustering that contains a center at x.

It remains to pre-process $(Z \cup B)$ and S to efficiently compute the number of centers n_b assigned to centers of Q in $B_{r/2}(x)$ for each $x \in B$ and each radius $r := \|x-p\|_2$ for all $p \in (Z \cup B)$. To that end, we can first sort the points $y \in (Z \cup B)$ by their distances from x. Now we can scan radially outward from x, finding the closest center in $Z \cup B$ and iterating outward.

Given two centers $u, v \in (Z \cup B)$, let Q_u (resp. Q_v) be the clustering Q induced by removing the output of ϕ_u (resp. ϕ_v) from $S \cup \{u\}$ (resp. $S \cup \{v\}$) by Algorithm 1 with query center u (resp. v). In particular, suppose that after scanning radially outward from x, we first see u, followed immediately by v. Since $B_{r/2}(x)$ is monotonically increasing as r increases, it suffices to simply consider the additional centers in $Q_v \setminus Q_u$. Since Q_u and Q_v are both formed by swapping a center of S with $\{u\}$ and $\{v\}$ respectively, then we have $|Q_v \setminus Q_u| \le 2$. Hence, to compute n_b for v, it suffices to use the previous computation of n_b for u and consider the number of points assigned to at most two centers of Q_u, which can be done in $\mathcal{O}(1)$ time.

Therefore, the total time to compute S and perform additional pre-processing is $\mathcal{O}\left(d(|Z|+k)^2\right)$. For each possible center $p \in (Z \cup B)$, we use $\mathcal{O}(k+|Z|)$ time to update the pre-processing information to obtain Ψ. For each point $x \in B$, we use $\mathcal{O}(dk\log k)$ time to sort the points by their distances from x. For each point $x \in B$ and possible center $p \in (Z \cup B)$, we use $\mathcal{O}(1)$ time to compute n_b. Hence, the amortized runtime is $\frac{d(|Z|+k\log k)^2}{k}$. □

Theorem D.10 (Wang et al. (2026))**.** *Given a set X of n points on $[\Delta]^d$ and an accuracy parameter $\varepsilon \in (0,1)$, there is a one-pass insertion-only streaming algorithm that uses $\mathcal{O}(dk\log(n\Delta)) + f\left(n, d, \Delta, k, \frac{\mathrm{polylog}(\log(nd\Delta))}{\varepsilon}, z\right) \cdot \mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\Delta)\right)$ bits of space and outputs a $(1+\varepsilon)$-coreset of X, where $f\left(n, d, \Delta, k, \frac{1}{\varepsilon}, z\right)$ is the number of points of a coreset construction with weights $[1, \mathrm{poly}(nd\Delta)]$ for (k, z)-clustering.*

Putting together Theorem D.10 and Lemma D.9, we have:

Theorem D.11. *Given a set X of n points on $[\Delta]^d$, let $f\left(n, d, \Delta, k, \frac{1}{\varepsilon}, z\right)$ be the number of points of a coreset construction with weights $[1, \mathrm{poly}(nd\Delta)]$ for (k, z)-clustering. There exists*

Algorithm 7 Fast (k, z)-clustering

Require: Set $X = \{x_1, \ldots, x_n\}$ of points in $[\Delta]^d$ arriving as a stream
Ensure: (k, z)-clustering coreset for X
1: Initialize $\mathcal{S} \leftarrow \emptyset$ and $\lambda \leftarrow \mathcal{O}\left(\frac{kd}{\varepsilon^2}\log(n\Delta)\right)$
2: Batch $x_1, \ldots, x_n$ into blocks $B_1, \ldots, B_{n/k}$ of k updates
3: **for** $b \in [n/k]$ **do**
4: Let Z be a coreset for block B_{b-1}
5: Call BATCHSENS to compute a batch approximation of sensitivities for x_t with $t \in B_b$
6: Sample points of B_{b-1} into a stream $\mathcal{S}'$ using sensitivity sampling
7: Update Z by running merge-and-reduce on $\mathcal{S}'$
8: **end for**

an algorithm that uses $dk \cdot \mathrm{polylog}(\log(nd\Delta))$ amortized update time and $\mathcal{O}(dk\log(n\Delta)) + f\left(n, d, \Delta, k, \frac{\mathrm{polylog}(\log(nd\Delta))}{\varepsilon}, z\right) \cdot \mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\Delta)\right)$ bits of space, and outputs a $(1+\varepsilon)$-coreset of X.

E (k, z)-CLUSTERING III: FILTERING OF LOW-SENSITIVITY POINTS

In this section, we show that our one-pass streaming algorithm can be implemented in $\mathcal{O}(d\log k) + \log^2(k) \cdot \mathrm{polylog}(\log(n\Delta))$ amortized update time. To achieve this amortized time, we introduce a crude sampling scheme with fast update time via crude quadtree embedding and then implement our fast framework in Section B.

We first define a quadtree embedding. Due to the desiderata of fast update time, our construction is somewhat non-standard. Given $s = (s_1, \ldots, s_d) \in \mathbb{Z}^d$, $t \in \{0, 1, \ldots, \ell\}$, and a parameter $z > 1$, we define the axis-aligned grid $\mathcal{G}_{s,t,\zeta}$ over $\mathbb{Z}^d$ with side length ζ^t, so that $s = (s_1, \ldots, s_d)$ lies on one of the corners of the grid. Then for $X \in \mathbb{R}^{[\Delta]^d}$, we define $G_{s,t,\zeta}(X)$ as the frequency vector over the hypercubes of the grid $\mathcal{G}_{s,t,\zeta}$ that counts the total number or weight of points in each hypercube. Specifically, each cell of a grid of length ζ^t has closed boundaries on one side and open boundaries on the other side, i.e., a cell containing $(u_1, \ldots, u_d)$ cannot also contain $(v_1, \ldots, v_d)$ for any $v_i \geq u_i + \zeta^t$. We define $\mathrm{TreeDist}_{s,\zeta}(x, y)$ to be $\sqrt{d} \cdot \zeta^\alpha$, where α is the smallest integer such that x and y are in two different cells of $\mathcal{G}_{s,\alpha,\zeta}$ or $\mathrm{TreeDist}_{s,\zeta}(x, y) = 0$ if no such grid exists. For our purposes, we will set $\zeta = n^{1-c}$ for a fixed constant $c \in (0, 1)$ so that $\ell = \mathcal{O}\left(\frac{1}{c}\right)$ for $\Delta = \mathrm{poly}(n)$.

We first show that the distance between a pair of points cannot be underestimated by the quadtree.

Lemma E.1. *For every $x, y \in [\Delta]^d$ and every $s = (s_1, \ldots, s_d) \in \mathbb{Z}^d$ and $\zeta > 1$, we have*

$$\mathit{dist}(x, y) \leq \operatorname*{TreeDist}_{s,\zeta}(x, y).$$

Proof. Since $x, y \in [\Delta]^d$, then by an averaging argument there exists some coordinate for which x and y are separated by distance $\frac{\|x-y\|_2}{\sqrt{d}}$. Then x and y must be separated in grid $\mathcal{G}_{s,\alpha,\zeta}$ for $\zeta^\alpha \leq \frac{\|x-y\|_2}{\sqrt{d}}$. Therefore, $\mathrm{TreeDist}_{s,\zeta}(x, y) = \sqrt{d} \cdot \zeta^\beta$ for $\zeta^\beta > \frac{\|x-y\|_2}{\sqrt{d}}$. Then it follows $\mathrm{dist}(x, y) \leq \mathrm{TreeDist}_{s,\zeta}(x, y)$. □

We next bound the probability the distance between a pair of points is overestimated by the quadtree for a fixed distortion rate.

Lemma E.2. *Given a set S and a parameter $t > 2$, let $\mathcal{E}$ be the event where no points of S are within $\frac{1}{t}$ fraction toward the boundary of any cell in the grid that induces* TreeDist. *Then conditioned on $\mathcal{E}$, we have that for all $x, y \in S$,*

$$\operatorname*{TreeDist}_{s,\zeta}(x, y) \leq t \cdot \sqrt{d} \cdot \zeta \cdot \mathit{dist}(x, y).$$

Moreover, we have

$$\mathbf{Pr}\left[\mathcal{E}\right] \geq 1 - \frac{d|S|}{t}.$$

Proof. Observe that if x and y are first in the same cell at level i, then $\text{TreeDist}_{s,\zeta}(x,y) = \sqrt{d} \cdot \zeta^i$. However, because the cells at level $i-1$ have length ζ^{i-1} and conditioned on $\mathcal{E}$, both of x and y are at least $\frac{1}{t}$ fraction away from the boundary, then we have $\text{dist}(x,y) \geq \frac{\zeta^{i-1}}{t}$, which implies that

$$\underset{s,\zeta}{\text{TreeDist}}(x,y) \leq t \cdot \sqrt{d} \cdot \zeta \cdot \text{dist}(x,y).$$

Next, we note that for each coordinate, the probability that x is contained a distance that is within $\frac{1}{t}$ fraction toward the boundary is $\frac{1}{t}$. The desired claim then follows from a union bound over all $x \in S$ and all d dimensions. □

We now require the following generalization of the fast Euclidean k-means approximation algorithm by (Cohen-Addad et al., 2020) to (k,z)-clustering. We remark that the statement in (Cohen-Addad et al., 2020) provides a $\mathcal{O}_\varepsilon(\log k)$-approximation to k-means in runtime $\tilde{\mathcal{O}}\left(nd + (n\log\Delta)^{1+\varepsilon}\right)$. We briefly describe the algorithm of (Cohen-Addad et al., 2020) and the necessary adjustments to provide Theorem E.3.

The algorithm of (Cohen-Addad et al., 2020) first applies the Johnson-Lindenstrauss transformation (Johnson & Lindenstrauss, 1984) to all the points to reduce the dimension to $\mathcal{O}(\log k)$, which preserves all clustering costs within a constant factor (Makarychev et al., 2019; Izzo et al., 2021). It then constructs a standard quadtree embedding, which it uses to adaptively sample a set C of k centers. Specifically, adaptive sampling randomly selects a point of the input set to be the first center of C. It then iterates, iteratively choosing $k-1$ additional points from the input set, with probability proportional to the squared distance of the point from the current set of centers. To do this quickly, (Cohen-Addad et al., 2020) uses multiple independent instances of the quadtree embedding to estimate the distance of each point from the current set of centers. Finally, to roughly estimate the clustering cost induced by C, (Cohen-Addad et al., 2020) applies a locality-sensitivity hashing procedure to assign each point to its approximate nearest neighbor in C.

For (k,z)-clustering, we instead perform adaptive sampling by iteratively choosing each of the input points with probability proportional to the z-th power of distances of the point from the centers already opened in C. The runtime dependency of $\mathcal{O}(\log\Delta)$ of the algorithm of (Cohen-Addad et al., 2020) is due to the $\mathcal{O}(\log\Delta)$ levels in the quadtree used to estimate these distances. To achieve runtime independent of $\mathcal{O}(\log\Delta)$, we instead use our crude quadtree with $\mathcal{O}(1)$ levels, to provide $n^{\Theta(1)}$-approximations to these distances. Once C is acquired from adaptive sampling, we then estimate the clustering cost by again using the crude quadtree to assign each point to its closest center in the quadtree, which uses constant time per point, since the crude quadtree has $\mathcal{O}(1)$ levels. Finally, we remark that unlike (Cohen-Addad et al., 2020), we can perform the quadtree embedding multiple times to avoid any cases where the distortion between pairs of points is too large. Thus we have:

Theorem E.3. *(Cohen-Addad et al., 2020) For any constant $\alpha \in (0,1)$ and $N \gg n$, there exists a N^α-approximation algorithm to (k,z)-clustering on X with n weighted points, that uses $\mathcal{O}(nd\log(nd))$ expected runtime.*

To estimate the sensitivity of a point x, we now enumerate over the distances induced by all levels of the quadtree for centers that serve x. To that end, we show that $\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}$ can only be an underestimate to the optimal (k,z)-medoids clustering constrained to β being the first level in which the center serving x is in the same cell as x in the quadtree. In particular, the following lemma should be considered the quadtree analog to Lemma D.5 for the radial search.

Lemma E.4. *Let $x \in X$ be a fixed point and $p \in X$. Suppose* TreeDist *preserves all pairwise distances in X up to a factor of κ and suppose $\text{TreeDist}(p,x) = \sqrt{d}\cdot\zeta^\beta$. Let $\kappa \geq 2$ be a parameter, so that Q is a κ-factor approximation to the optimal (k,z)-medoids clustering on X containing a center at p and let Ψ be is a κ-factor approximation to $\text{cost}(X,Q)$. Let S be the optimal (k,z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$, where $r := \|x-p\|_2$. Let n_β be the weight of the points assigned to centers $q \in Q$ such that $\text{TreeDist}(x,q) \leq \sqrt{d}\cdot\zeta^{\beta-1}$. There exists a constant $\gamma \geq 1$ such that*

$$\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z} \leq \kappa^{\mathcal{O}(z)} \cdot \text{cost}(X,S).$$

Proof. Since S is the optimal solution for a more constrained search space than Q, i.e., sets of at most k centers that contain a center at p and no center in the interior of $B_r(x)$, then we have

$$\kappa^2 \cdot \text{cost}(X, S) \geq \kappa \cdot \text{cost}(X, Q) \geq \Psi.$$

Now, consider a point y served by a center $q \in Q$ such that $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$. If $\|y - q\|_2 \geq \frac{r}{4}$, then

$$\|y - q\|_2 + r \leq 5 \cdot \|y - q\|_2.$$

Otherwise, for $\|y - q\|_2 < \frac{r}{4}$, then by triangle inequality, we have

$$\text{dist}(y, S) \geq \text{dist}(q, S) - \|y - q\|_2 \geq \frac{3r}{4}.$$

Therefore,

$$\|y - q\|_2 + r \leq \|y - q\|_2 + 2 \cdot \text{dist}(y, S).$$

Putting the two cases together, it follows that

$$\|y - q\|_2 + r \leq 5 \cdot \|y - q\|_2 + 2 \cdot \text{dist}(y, S).$$

Summing across all $y \in X$ served by q such that $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$,

$$\text{cost}(X, Q) + n_\beta \cdot r^z \leq 5\,\text{cost}(X, Q) + 2\,\text{cost}(X, S) \leq (5\kappa + 2) \cdot \text{cost}(X, S).$$

Since Ψ is a κ-approximation to $\text{cost}(X, Q)$ and conditioned on the TreeDist procedure preserving all pairwise distances in X up to a factor of κ and thus $\sqrt{d} \cdot \zeta^\beta$ is a κ-approximation to r, then

$$\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z} \leq \kappa^{\mathcal{O}(z)} \cdot \text{cost}(X, S),$$

for $\kappa \geq 2$, i.e., bounded away from 1. □

We next show that if there is a large number of points served by centers that are first in the same cell as x in a level of the quadtree before β, then $\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}$ is a good estimate to the cost of the optimal (k, z)-medoids clustering constrained to x being served at a center that is first in the same cell as x at level β in the quadtree. In particular, the following lemma should be considered the quadtree analog to Lemma D.6 for the radial search.

Lemma E.5. *Let $x \in X$ be a fixed point and $p \in X$. Suppose* TreeDist *preserves all pairwise distances in X up to a factor of κ and suppose* $\text{TreeDist}(p, x) = \sqrt{d} \cdot \zeta^\beta$. *Let $\kappa \geq 2$ be a parameter, so that Q is a κ-factor approximation to the optimal (k, z)-medoids clustering on X containing a center at p and let Ψ be is a κ-factor approximation to* $\text{cost}(X, Q)$. *Let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$, where $r := \|x - p\|_2$. Let n_β be the weight of the points assigned to centers $q \in Q$ such that* $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$ *and let n_α be the weight of the points assigned to centers $q \in Q$ such that* $\text{TreeDist}(x, q) \geq \sqrt{d} \cdot \zeta^\beta$. *If $n_\beta \geq n_\alpha$, then there exist constants $\gamma_1, \gamma_2 \geq 1$ such that*

$$\text{cost}(X, S) \leq \kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right) \leq \kappa^{\mathcal{O}(\gamma_1 \gamma_2 z)} \cdot \text{cost}(X, S).$$

Proof. We first show that there exists a constant $\gamma_1 \geq 1$ such that $\text{cost}(X, S) \leq \kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right)$. Observe that this would imply by Lemma E.4 that there also exists a constant $\gamma_2 \geq 1$ such that $\kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right) \leq \kappa^{\mathcal{O}(\gamma_1 \gamma_2 z)} \cdot \text{cost}(X, S)$.

Consider a point y served by a center $q \in Q$ with $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$. Let $\pi_S(y)$ be the closest center in S to y, breaking ties arbitrarily. By generalized triangle inequality,

$$\text{cost}(y, S) \leq 2^z \cdot \|y - q\|_2^z + 2^z \cdot \text{cost}(q, \pi_S(y)).$$

Summing across all y served by centers q with $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$ and noting that $\text{dist}(s, q) \leq 2\sqrt{d} \cdot \zeta^\beta$ conditioned on the correctness of TreeDist, as well as and noting that $n_\beta \geq n_\alpha$, we have

$$\sum_{y:\text{TreeDist}(x, \pi_Q(y)) \leq \sqrt{d} \cdot \zeta^{\beta-1}} \text{cost}(y, S) \leq 2^z \cdot n_\beta \cdot \left(2\sqrt{d} \cdot \zeta^\beta\right)^z + 2^z \cdot \sum_{y:\text{TreeDist}(x, \pi_Q(y)) \leq \sqrt{d} \cdot \zeta^{\beta-1}} \text{cost}(y, Q).$$

Next, consider a point y served by a center $q \in Q$ with $\text{TreeDist}(x, q) \geq \sqrt{d} \cdot \zeta^\beta$. We have that if $\|y - q\|_2 \geq \frac{r}{4}$, then $\|y - p\|_2 \leq \|y - q\|_2 + \|q - p\|_2 \leq 9 \cdot \|y - q\|_2$. Else if $\|y - q\|_2 < \frac{r}{4}$, then $\|y - p\|_2 \leq \|y - q\|_2 + \|q - p\|_2 < 3r$. Summing across all y served by centers $q \in Q$ with $\text{TreeDist}(x, q) \geq \sqrt{d} \cdot \zeta^\beta$,

$$\sum_{y:\text{TreeDist}(x,\pi_Q(y)) \geq \sqrt{d}\cdot\zeta^\beta} \text{cost}(y, S) \leq 12^z \cdot n_\beta \cdot \left(\frac{r}{2}\right)^z + 9 \cdot 2^z \cdot \sum_{y:\text{TreeDist}(x,\pi_Q(y)) \geq \sqrt{d}\cdot\zeta^\beta} \text{cost}(y, Q).$$

Conditioned on the correctness of TreeDist, we have that $\sqrt{d} \cdot \zeta^\beta$ is a κ-approximation to r. Thus summing the two cases across all $y \in X$, we have that there exists a constant $\zeta \geq 1$ such that

$$\text{cost}(X, S) \leq \kappa^{\zeta z} \cdot \left(\text{cost}(X, Q) + n_\beta \cdot \left(\sqrt{d} \cdot \zeta^\beta\right)^z\right).$$

Since Φ is a κ-factor approximation to $\text{cost}(X, Q)$, then there exists a constant γ_1 such that

$$\text{cost}(X, S) \leq \kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right).$$

Thus,

$$\text{cost}(X, S) \leq \kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right) \leq \kappa^{\mathcal{O}(\gamma_1 \gamma_2 z)} \cdot \text{cost}(X, S).$$

□

On the other hand, if the number of points served by centers that are first in the same cell as x in a level of the quadtree before β is small, then $\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}$ may not be a good estimate to the cost of the optimal (k, z)-medoids clustering constrained to x being served at a center that is first in the same cell as x at level β in the quadtree. However, we next show that it can nevertheless provide an upper bound on the $\frac{\text{cost}(x,S)}{\text{cost}(X,S)}$ for any S where x is served by a center that is firs tin the same cell as x at level β in the quadtree. Importantly, the quantity is at most the sensitivity $s(x)$ of x. In particular, the following lemma should be considered the quadtree analog to Lemma D.7 for the radial search.

Lemma E.6. *Let $x \in X$ be a fixed point and $p \in X$. Suppose* TreeDist *preserves all pairwise distances in X up to a factor of κ and suppose $\text{TreeDist}(p, x) = \sqrt{d} \cdot \zeta^\beta$. Let $s(x)$ be the sensitivity of (k, z)-medoids clustering with respect to X. Let $\kappa \geq 2$ be a parameter, so that Q is a κ-factor approximation to the optimal (k, z)-medoids clustering on X containing a center at p and let Ψ be is a κ-factor approximation to $\text{cost}(X, Q)$. Let S be the optimal (k, z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$, where $r := \|x - p\|_2$. Let n_β be the weight of the points assigned to centers $q \in Q$ such that $\text{TreeDist}(x, q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$ and let n_α be the weight of the points assigned to centers $q \in Q$ such that $\text{TreeDist}(x, q) \geq \sqrt{d} \cdot \zeta^\beta$. If $n_b < n_a$, then*

$$\frac{1}{\kappa^{\mathcal{O}(\gamma_1 z)}} \frac{\text{cost}(x, S)}{\text{cost}(X, S)} \leq \frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}} \leq \kappa^{\mathcal{O}(\gamma_2 z)} \cdot s(x),$$

for fixed constants $\gamma_1, \gamma_2 \geq 1$.

Proof. By Lemma E.4, there exists a constant $\gamma \geq 1$ such that

$$\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z} \leq \kappa^{\gamma z} \cdot \text{cost}(X, S).$$

Since S contains a center at p and no centers in the interior of $B_r(x)$, then $\text{cost}(x, S) = r^z$. Then conditioned on the correctness of TreeDist giving $d^{z/2} \cdot \zeta^{\beta z}$ as a κ^z-approximation to r^z, we have that

$$\frac{1}{\kappa^{\mathcal{O}(\gamma_1 z)}} \frac{\text{cost}(x, S)}{\text{cost}(X, S)} \leq \frac{r^z}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}}.$$

However, for a set W of k centers that contains a center at p but no center q with $\text{TreeDist}(x, q) < \sqrt{d} \cdot \zeta^{\beta-1}$, then we have $\frac{\sqrt{d}\cdot\zeta^{\beta-1}}{\kappa} \leq \text{dist}(x, W)$ and $\text{cost}(X, W) \leq \kappa^{\gamma_2 z}(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z})$ for some constant $\gamma_2 \geq 1$. Thus by the maximality of sensitivity, we have

$$\frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_b \cdot r^z} \leq \kappa^{\gamma_2 z} \cdot \frac{\text{cost}(x, W)}{\text{cost}(X, w)} \leq \kappa^{\gamma_2 z} \cdot s(x).$$

□

Hence if TreeDist provides a κ-approximation to the pairwise distance of all points $x \in X$, then the algorithm in RoughSens provides a $\kappa^{\mathcal{O}(1)}$-factor approximations to all sensitivities x in the batch B.

Lemma E.7. *For a fixed $x \in X$, let $s(x)$ be the sensitivity of (k,z)-medoids clustering with respect to X. Suppose* TreeDist *preserves all pairwise distances in X up to a factor of κ. Let $\widehat{s(x)}$ be the output of Algorithm 8, for each $x \in B$, where B is the input batch. Then there exist constants $\gamma_1, \gamma_2 \geq 1$ such that*

$$\frac{1}{\kappa^{\gamma_1 z}} \cdot s(x) \leq \widehat{s(x)} \leq \kappa^{\gamma_2 z} \cdot s(x).$$

Proof. For each fixed $p \in X$, let S be the optimal (k,z)-medoids clustering on X containing a center at p and no center in the interior of $B_r(x)$. Let n_β be the weight of the points assigned to centers $q \in Q$ such that $\text{TreeDist}(x,q) \leq \sqrt{d} \cdot \zeta^{\beta-1}$ and let n_α be the weight of the points assigned to centers $q \in Q$ such that $\text{TreeDist}(x,q) \geq \sqrt{d} \cdot \zeta^{\beta}$. We perform casework on whether $n_\beta < n_\alpha$ or $n_\beta \geq n_\alpha$.

In the case where $n_\beta \geq n_\beta$, then by Lemma E.5,

$$\text{cost}(X,S) \leq \kappa^{\mathcal{O}(\gamma_1 z)} \left(\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}\right) \leq \kappa^{\mathcal{O}(\gamma_1 \gamma_2 z)} \cdot \text{cost}(X,S).$$

Since $\text{cost}(x,S) = r^z$, which is a κ^z-approximation to $(d^{z/2} \cdot \zeta^{\beta z})$ conditioned on the correctness of TreeDist, then it holds that $\frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}}$ is a $\kappa^{\mathcal{O}(z)}$ approximation to $\frac{\text{cost}(x,S)}{\text{cost}(X,S)}$.

Otherwise if $n_\beta < n_\alpha$, then by Lemma E.6,

$$\frac{1}{\kappa^{\mathcal{O}(\gamma_1 z)}} \frac{\text{cost}(x,S)}{\text{cost}(X,S)} \leq \frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}} \leq \kappa^{\mathcal{O}(\gamma_2 z)} \cdot s(x),$$

for fixed constants $\gamma_1, \gamma_2 \geq 1$. In particular, for the center $p \in X$ that realizes the sensitivity $s(x)$, we have that $\frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}}$ is a $\kappa^{\mathcal{O}(z)}$ approximation to $\frac{\text{cost}(x,S)}{\text{cost}(X,S)}$. Furthermore, for centers $q \in X$ that do not realize the sensitivity, we have that $\frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot d^{z/2} \cdot \zeta^{\beta z}} \leq \kappa^{\mathcal{O}(\gamma_2 z)} \cdot s(x)$. Thus,

$$\frac{1}{\kappa^{\gamma_1 z}} \cdot s(x) \leq \widehat{s(x)} \leq \kappa^{\gamma_2 z} \cdot s(x).$$

□

We now analyze the properties of RoughSens. Since correctness is mostly given by Lemma E.7, the main consideration is the amortized update time.

Lemma E.8. *Given any constant $\alpha \in (0,1)$ and a constant-factor coreset Z of size $k \cdot \text{polylog}(\log n)$ to $X \subset [\Delta]^d$, there exists an algorithm* RoughSens *that outputs n^α-approximations to the sensitivities to each of the points $x \in B$ in a batch B of k points of X, using amortized $d \log(k) \cdot \text{polylog}(\log(n\Delta))$ time, with probability at least $1 - \frac{1}{\sqrt{n}}$.*

Proof. By Lemma C.9, we can compute the sensitivity of $x \in Z \cup B$ as a good approximation to computing the $x \in B$ with respect to X, since Z is a constant-factor coreset to Y so that $Z \cup B$ is a constant-factor coreset to $X = Y \cup B$. Moreover, it suffices to compute a constant-factor approximation to the (k,z)-medoids clustering sensitivity of x with respect to $Z \cup B$, by Theorem C.8. Therefore, the goal is to compute a crude approximation to $\max_{C \subset Z \cup B : |C| \leq k} \frac{\text{cost}(x,C)}{\text{cost}(X,C)}$. Let $\mathcal{E}$ be the event that TreeDist approximates all pairwise distances within a factor of $\frac{1}{\text{poly}(d)} \cdot n^{\frac{1}{100p}}$. Then conditioned on $\mathcal{E}$, correctness follows by Lemma E.7. We remark that the algorithm iterates until $\mathcal{E}$ occurs, so the dependency on $\mathcal{E}$ is not in the correctness, but rather than runtime.

Thus, we now analyze the amortized runtime of RoughSens. By Theorem E.3, we use $kd \log(kd) \cdot \text{polylog}(\log(n\Delta))$ time to find a κ-approximation S of k centers that serve as a constant-factor approximation to $Z \cup B$. For $\Delta = \text{poly}(n)$, the tree has height $\mathcal{O}\left(\frac{1}{\iota}\right)$ and thus by pre-processing the number of points in each cell, the total time form Q across all levels is $\mathcal{O}\left(\frac{1}{\iota}\right)$. Similarly, we can

Algorithm 8 ROUGHSENS: Rough batch approximation of sensitivities

Require: Input set $Z \subset [\Delta]^d$ for (k, z)-medoids clustering; batch B of k query points; approximation parameter $\alpha \in (0, 1)$

Ensure: n^α-approximation of the sensitivity of x for (k, z)-medoids clustering on $Z \cup B$, for all $x \in B$

Set $\kappa \leftarrow n^\alpha$ and $\zeta \leftarrow n^\iota$ for sufficiently small $\iota < \alpha$
Compute S as a constant-factor solution on $Z \cup B$ for (k, z)-medoids clustering
for $x \in B$ **do**
  Initialize $\widehat{s(x)} \leftarrow 0$
  Build a tree embedding on $Z \cup B$
  while there exists a point of $Z \cup B$ within a $\frac{1}{\kappa}$-fraction of the distance to a cell in the tree embedding **do**
    Create a tree embedding on $Z \cup B$
  end while
  for level ℓ_β in the tree **do**
    Let Q be a κ-factor approximation of the optimal (k, z)-medoids clustering on X that is initially separated from x at level ℓ_β
    Use TreeDist to compute a constant-factor approximation Ψ of $\mathrm{cost}(X, Q)$
    Let n_β be the number of points served by centers of Q in the same cell as x before level ℓ_β
    Update

$$\widehat{s(x)} \leftarrow \max\left(\widehat{s(x)}, \frac{d^{z/2} \cdot \zeta^{\beta z}}{\Psi + n_\beta \cdot (d^{z/2} \cdot \zeta^{\beta z})}\right).$$

  end for
  return $\widehat{s(x)}$
end for

approximate Ψ up to a κ-factor, along with the corresponding center ψ_β to be removed from S that is consistent with Ψ.

It remains to pre-process $(Z \cup B)$ and S to efficiently compute the number of centers n_β assigned to centers of $q \in Q$ in such that $\mathrm{TreeDist}(x, q) < \sqrt{d} \cdot \zeta^\beta$. Again, this is handled by pre-processing all the points $y \in (Z \cup B)$ in the quadtree by pre-computing the number of points in each cell at each level, which takes total time $k \cdot \mathrm{polylog}(\log n)$ due to the size of X. Then by starting at $\beta = 0$ and iterating to $\beta = \mathcal{O}\left(\frac{1}{\iota}\right)$, we proceed up the tree to compute each n_β, using $\mathcal{O}\left(\frac{1}{\iota}\right)$ time in total.

Therefore, the total time to compute S and perform additional pre-processing is $kd \log(kd) \cdot \mathrm{polylog}(\log(n\Delta))$. Across all possible levels β, we use $\mathcal{O}\left(\frac{1}{\iota}\right)$ time to update the pre-processing information to obtain Ψ and compute each value of n_β. Hence, the amortized runtime is $d \log(kd) \cdot \mathrm{polylog}(\log(n\Delta))$. Finally, we remark that if $d \geq \mathcal{O}(\log n)$, then we can apply the standard Johnson-Lindenstrauss dimensionality reduction technique (Makarychev et al., 2019) to achieve dimension $\mathcal{O}(\log n)$, and thus the resulting amortized runtime is $d \log(k) \cdot \mathrm{polylog}(\log(n\Delta))$. □

Applying dimensionality reduction as a pre-processing and then putting everything together, we have:

Theorem E.9. *Given a set X of n points on $[\Delta]^d$, let $f\left(n, d, \Delta, k, \frac{1}{\varepsilon}, z\right)$ be the number of points of a coreset construction with weights $[1, \mathrm{poly}(nd\Delta)]$ for (k, z)-clustering. There exists a one-pass streaming algorithm that uses $\mathcal{O}(d \log k) + \log^2(k) \cdot \mathrm{polylog}(\log(n\Delta))$ amortized update time and $\mathcal{O}(dk \log(n\Delta)) + f\left(n, d, \Delta, k, \frac{\mathrm{polylog}(\log(nd\Delta))}{\varepsilon}, z\right) \cdot \mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\Delta)\right)$ bits of space, and outputs a $(1 + \varepsilon)$-coreset of X.*

Proof. We perform induction on the time t throughout the stream. Correctness at the first step is immediate, since the first point is inserted into the batch. Now we suppose for our inductive hypothesis that at time $t - 1$, we have both a constant-factor coreset and a $(1 + \varepsilon)$-coreset at time $t - 1$.

Consider time t for our inductive step. Either the new point is added to the batch of size k, in which case both coreset invariants are maintained, or we perform a sampling step on the batching using crude approximations to the sensitivities as an initial filtering step, because the batch has gotten too large. In the latter case, we first use the constant-factor coreset through Lemma E.8 to compute rough approximations to the (k, z)-sensitivities in a batch of k points. This uses $d \log(k) \cdot \operatorname{polylog}(\log n)$ amortized runtime and produces an insertion-only stream $\mathcal{S}$ of length $\mathcal{O}\left(n^{1-\Omega(1)}\right)$. We feed this insertion-only stream $\mathcal{S}$ as the input to Algorithm 7. By Theorem D.11, the algorithm uses $dk \operatorname{polylog}(\log(n\Delta))$ update time on the stream $\mathcal{S}$ of length $o(n)$ to generate both a constant-factor coreset and a $(1+\varepsilon)$-factor coreset at time t, which completes our induction. Since this is a lower-order term, then the total amortized runtime is $d \log(k) \cdot \operatorname{polylog}(\log n)$.

As a pre-processing step, we can use $\mathcal{O}(d \log k)$ time to apply Johnson-Lindenstrauss (Makarychev et al., 2019) to each input point, to reduce the dimension down to $\mathcal{O}(\log k)$. Then it follows that the subsequent amortized runtime is $\log^2(k) \cdot \operatorname{polylog}(\log n)$, so that the total amortized runtime is $\mathcal{O}(d \log k) + \log^2(k) \cdot \operatorname{polylog}(\log(n\Delta))$. □

Combining Theorem E.9 and the offline coreset size for (k, z)-clustering in Theorem C.2, we have the following corollary of our main result. Recall that we allow each word of space to use $\Theta(\log(nd\Delta))$ bits of storage, and thus, we achieve a word of space independent of n.

Corollary E.10. *Given a set X of n points on $[\Delta]^d$, there exists a one-pass streaming algorithm that uses $\mathcal{O}(d \log k) + \log^2(k) \cdot \operatorname{polylog}(\log(n\Delta))$ amortized update time and $\tilde{\mathcal{O}}\left(\frac{dk}{\min(\varepsilon^4, \varepsilon^{2+z})}\right)$ words of space, and outputs a $(1+\varepsilon)$-coreset of X.*

Finally, we note the implications of Theorem E.9 to dynamic (k, z)-clustering in the incremental setting.

Theorem E.11. *Given an insertion-only stream of n points that defines a dataset X on $[\Delta]^d$, there exists a one-pass streaming algorithm that uses $\mathcal{O}(d \log k) + \log^2(k) \cdot \operatorname{polylog}(\log(n\Delta))$ amortized update time and $\tilde{\mathcal{O}}(dk \log(n\Delta))$ bits of space, and outputs a $\mathcal{O}(z)$-approximation to (k, z)-clustering at all times in the stream.*

Proof. By Theorem C.2, there exists a coreset construction for Euclidean (k, z)-clustering that samples $\tilde{\mathcal{O}}(k)$ points of the input dataset. Thus, Theorem E.9 guarantees a constant-approximation coreset for X using $\mathcal{O}(d \log k) + \log^2(k) \cdot \operatorname{polylog}(\log(n\Delta))$ amortized update time. In fact, because the proof of Theorem E.9 uses induction over the course of the stream, it actually achieves a constant-approximation coreset to every prefix of the stream. We can thus achieve a $\mathcal{O}(z)$-approximation at all times in the stream by applying a standard polynomial-time clustering approximation algorithm such as local search, c.f., Theorem D.1, each time the merge-and-reduce data structure is updated. Moreover, note that by Theorem C.6, the final input stream induced by online sensitivity sampling to the merge-and-reduce data structure has length $\mathcal{O}\left(dk^2 \log^3(n\Delta)\right)$. By applying a standard Johnson-Lindenstrauss transformation, c.f., Theorem A.2, we can view $d = \mathcal{O}(\log k)$. Therefore, the total runtime of all iterations of local search is $\mathcal{O}\left(k^2 \log^4(n\Delta)\right) \cdot \tilde{\mathcal{O}}\left(k^2 \log n\right)$. Thus for $n \gg k^4$, the amortized runtime remains $\mathcal{O}(d \log k) + \log^2(k) \cdot \operatorname{polylog}(\log(n\Delta))$, due to the computation of the constant-approximation coreset. □

F Subspace Embeddings

In this section, we describe our one-pass streaming algorithm on insertion-only streams for L_p subspace embeddings that uses total runtime $\mathcal{O}(\operatorname{nnz}(\mathbf{A}))$. Recall that in problem of L_p subspace embeddings, the rows of a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ arrive one-by-one, and the goal is to produce a matrix $\mathbf{M} \in \mathbb{R}^{m \times d}$ such that for a fixed $\varepsilon \in (0, 1)$ and for all $\mathbf{x} \in \mathbb{R}^d$, we have

$$(1-\varepsilon)\|\mathbf{A}\mathbf{x}\|_p \leq \|\mathbf{M}\mathbf{x}\|_p \leq (1+\varepsilon)\|\mathbf{A}\mathbf{x}\|_p.$$

L_p subspace embeddings are widely used in data compression applications where preserving the geometry of data is critical, e.g., dimensionality reduction, compressed sensing, and linear/robust regression. Hence, there is a long line of active work studying L_p subspace embeddings (Drineas

et al., 2006a;b; Sarlós, 2006; Dasgupta et al., 2009; Clarkson & Woodruff, 2009; Cohen & Peng, 2015; Cohen et al., 2020; Braverman et al., 2020; Parulekar et al., 2021; Mahabadi et al., 2022; Meyer et al., 2022; Musco et al., 2022; Cherapanamjeri et al., 2023; Meyer et al., 2023; Woodruff & Yasuda, 2023).

We first recall the following definition of leverage scores, which intuitively quantifies the importance of a row for L_2 subspace embeddings.

Definition F.1 (Leverage scores)**.** *The* leverage score *of a row* $\mathbf{a}_i \in \mathbb{R}^d$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *is*

$$\max_{\mathbf{x}:\|\mathbf{A}\mathbf{x}\|_2=1} \langle \mathbf{a}, \mathbf{x}\rangle^2.$$

It is known that the leverage score for $\mathbf{a}_i$ *also admits the closed form* $\mathbf{a}_i^\top (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{a}_i$.

We next recall one of the possible generalizations of leverage scores to L_p subspace embeddings.

Definition F.2 (L_p Lewis weights)**.** *The* Lewis weight *of a row* $\mathbf{a}_i \in \mathbb{R}^d$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *for* L_p *subspace embedding is the* i*-th diagonal entry of the unique diagonal matrix* $\mathbf{W}$ *such that for all* $i \in [n]$

$$W_{i,i} = \ell_i(\mathbf{W}^{\frac{1}{2}-\frac{1}{p}}\mathbf{A}),$$

where ℓ_i *denotes the* i*-th leverage score, i.e., the leverage score of the* i*-th row of the matrix* $\mathbf{W}^{\frac{1}{2}-\frac{1}{p}}\mathbf{A}$.

The intuition for L_p Lewis weights is not obvious; they are the leverage scores of the matrix after applying a proper change-of-density, where each row is weighted the number of times necessary to preserve $\|\mathbf{A}\mathbf{x}\|_p$ when consider $\|\mathbf{A}'\mathbf{x}\|_2$, where $\mathbf{A}'$ is the reweighted matrix (Cohen & Peng, 2015). An advantage of using Lewis weights to sample rows over other quantities such as the L_p leverage scores (Dasgupta et al., 2009) or the L_p sensitivities (Braverman et al., 2020; Chen & Derezinski, 2021) is that the L_p Lewis weights are known to admit the optimal sampling complexity. On the other hand, for the special case of $p = 2$, (Sarlós, 2006) showed the existence of an oblivious subspace embedding with $\mathcal{O}\left(\frac{d}{\varepsilon^2}\right)$ rows. Combining these methods, we have:

Theorem F.3. *(Sarlós, 2006; Cohen & Peng, 2015; Woodruff & Yasuda, 2023) Given a matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, accuracy* $\varepsilon \in (0, 1)$*, and* $p \geq 1$*, let*

$$f\left(d, \frac{1}{\varepsilon}, p\right) = \begin{cases} \frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}\,\mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p \in [1, 2) \\ \mathcal{O}\left(\frac{d}{\varepsilon^2}\right), & p = 2 \\ \frac{d^{p/2}}{\varepsilon^2}\cdot\mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p > 2 \end{cases}.$$

Then there exists a coreset construction for L_p *subspace embeddings that consists of* $f\left(d, \frac{1}{\varepsilon}, p\right)$ *rows, each with entry at most* $\|\mathbf{A}\|_\infty \cdot \mathrm{poly}(n)$*, which can be constructed in time polynomial in* n *and* d.

We next define the generalization of L_p Lewis weights to the online setting.

Definition F.4 (Online L_p Lewis weights)**.** *The online* L_p Lewis weight *of a row* $\mathbf{a}_i \in \mathbb{R}^d$ *of a matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$ *is the Lewis weight of* $\mathbf{a}_i$ *with respect to the matrix* $\mathbf{A}_i = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_i$*, i.e., the submatrix of* $\mathbf{A}$ *consisting of the first* i *rows of* $\mathbf{A}$.

We recall the following upper bounds on the sum of the online Lewis weights.

Theorem F.5. *(Braverman et al., 2020; Woodruff & Yasuda, 2023) Let* $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \in \mathbb{R}^{n\times d}$ *be a matrix of* n *rows and let* $\sigma(\mathbf{a}_t)$ *denote the online Lewis weight of* $\mathbf{a}_t$ *for* $t \in [n]$ *for* L_p*-subspace embedding, where* $p \geq 1$*. Then for* $p \in [1, 2]$*,*

$$\sum_{t=1}^{n} \sigma(\mathbf{a}_t) = d \cdot \mathrm{polylog}(n\kappa),$$

and for $p > 2$*,*

$$\sum_{t=1}^{n} \sigma(\mathbf{a}_t) = d^{p/2} \cdot \mathrm{polylog}(n\kappa),$$

where κ *is the online condition number.*

We now recall the full guarantees of online Lewis weight sampling.

Theorem F.6 (Online Lewis weight sampling). *(Woodruff & Yasuda, 2023) Given a sequence $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^{n\times d}$ of rows, suppose each point $\mathbf{a}_t$ is sampled with probability $p_t \geq \min(1, \gamma \cdot \sigma(\mathbf{a}_t))$, where $\sigma(\mathbf{a}_t)$ is the online Lewis weight of $\mathbf{a}_t$ and $\gamma = \mathcal{O}\left(\frac{\log n}{\varepsilon^2}\right)$ and multiplied by $\frac{1}{p_t}$ if $\mathbf{a}_t$ is sampled. Then with high probability, the rescaled sample is a $(1+\varepsilon)$-coreset for L_p-subspace embedding that has a number of rows that is at most:*

$$g\left(n, d, \frac{1}{\varepsilon}, \kappa, p\right) = \begin{cases} \frac{d}{\varepsilon^2} \cdot \text{polylog}(n\kappa), & p \in [1,2] \\ \frac{d^{p/2}}{\varepsilon^2} \cdot \text{poly}(\log^{p/2}(n\kappa)), & p > 2 \end{cases}.$$

We use standard approaches, e.g., Lemma 4.1 in (Clarkson & Woodruff, 2009) to upper bound the magnitude of the logarithm of any nonzero singular value of a matrix with bounded integer entries. This in turn upper bounds the logarithm of the online condition number.

Lemma F.7. *Suppose $\mathbf{A} \in \mathbb{R}^{n\times d}$ has integer entries bounded in magnitude by M. Then for any nonzero singular value σ_i of $\mathbf{A}$, we have $|\log \sigma_i| = \mathcal{O}(d \log(nM))$.*

Proof. We have that the characteristic polynomial of $\mathbf{A}^\top\mathbf{A}$ is $p(x) = \det(x \cdot \mathbb{I}_d - \mathbf{A}^\top\mathbf{A})$. Note that if $\mathbf{A}$ has rank r, then $p(x) = x^{d-r}\prod_{i=1}^{r}(x - \lambda_i)$, where λ_i is the i-th eigenvalue of $\mathbf{A}^\top\mathbf{A}$, writing $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r$. Since all entries of $\mathbf{A}^\top\mathbf{A}$ are integers, then the coefficients of $p(x)$ are also integers. Moreover, since the eigenvalues of $\mathbf{A}^\top\mathbf{A}$ are non-negative, then $\prod_{i=1}^{r}\lambda_i \geq 1$. Because all entries of $\mathbf{A}^\top\mathbf{A}$ are at most $\text{poly}(ndM)$ in magnitude, then

$$\lambda_i \leq \lambda_1 \leq \|\mathbf{A}^\top\mathbf{A}\|_F \leq \text{poly}(ndM).$$

Since $\prod_{i=1}^{r}\lambda_i \geq 1$, then we have that $\lambda_i \geq \frac{1}{\text{poly}(ndM)^d}$. Since $\sigma_i^2 = \lambda_i$, then $|\log \sigma_i| = \mathcal{O}(d\log(nM))$. □

F.1 Crude Lewis Weight Approximation

In this section, we introduce our method to compute a crude n^α-approximation to the online Lewis weight. To begin with, we define the L_p sensitivity of a row of a matrix, describing the importance of a row for the purposes of L_p subspace embeddings.

Definition F.8 (L_p sensitivity). *For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, the L_p sensitivity of row $\mathbf{a}_t$ is the quantity*

$$\max_{\mathbf{y}\in\mathbb{R}^d:\|\mathbf{A}\mathbf{y}\|_p=1} |\langle \mathbf{a}_t, \mathbf{y}\rangle|^p.$$

We recall that the L_p sensitivity is a $\text{poly}(d)$ approximation to the L_p Lewis weights.

Lemma F.9. *(Cohen & Peng, 2015; Chen & Derezinski, 2021; Woodruff & Yasuda, 2023) For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $s(\mathbf{a}_t) := \max_{\mathbf{y}\in\mathbb{R}^d:\|\mathbf{A}\mathbf{y}\|_p=1} |\langle \mathbf{a}_t, \mathbf{y}\rangle|^p$ denote the L_p sensitivity of the t-th row of $\mathbf{A}$ and let $w(\mathbf{a}_t)$ denote the L_p Lewis weight of $\mathbf{a}_t$. Then*

$$\frac{w(\mathbf{a}_t)}{d^{|1-p/2|}} \leq s(\mathbf{a}_t) \leq w(\mathbf{a}_t).$$

We next relate the square root of the leverage score of a row $\mathbf{a}_t$ in a matrix $\mathbf{A}$ with the p-th root of the L_p sensitivity of $\mathbf{a}_t$.

Lemma F.10. *For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\xi(\mathbf{a}_t)$ denote the square root of the leverage score of $\mathbf{a}_t$ and let $\chi(\mathbf{a}_t)$ denote the L_p sensitivity of $\mathbf{a}_t$ raised to the $\frac{1}{p}$ power, i.e., $(s(\mathbf{a}_t))^{1/p}$, where $s(\mathbf{a}_t)$ is the L_p sensitivity of $\mathbf{a}_t$. Then for $p \in [1,2]$, we have*

$$s(\mathbf{a}_t) \leq \xi(\mathbf{a}_t) \leq n^{\frac{1}{p}-\frac{1}{2}} \cdot s(\mathbf{a}_t),$$

and for $p > 2$, we have

$$\xi(\mathbf{a}_t) \leq s(\mathbf{a}_t) \leq n^{\frac{1}{2}-\frac{1}{p}} \cdot \xi(\mathbf{a}_t).$$

Proof. Note that for a fixed vector $\mathbf{x} \in \mathbb{R}^d$ that is not in the kernel of $\mathbf{A}$, we have that $\frac{\langle \mathbf{a}_t, \mathbf{x} \rangle}{\|\mathbf{A}\mathbf{x}\|_2}$ is within a $\sqrt{n}$ factor of $\frac{\langle \mathbf{a}_t, \mathbf{x} \rangle}{\|\mathbf{A}\mathbf{x}\|_p}$, for $p \in [1, \infty)$. The claim then follows from the definition that the square root of the leverage score is the maximum of $\frac{\langle \mathbf{a}_t, \mathbf{x} \rangle}{\|\mathbf{A}\mathbf{x}\|_2}$ across all possible such $\mathbf{x}$, while the $\frac{1}{p}$-th root of the L_p sensitivity is the maximum of $\frac{\langle \mathbf{a}_t, \mathbf{x} \rangle}{\|\mathbf{A}\mathbf{x}\|_p}$ over all such possible $\mathbf{x}$. □

Given Lemma F.9 and Lemma F.10, We observe that the square root of the leverage score gives a n^α-approximation to the L_p Lewis weight of $\mathbf{a}_t$ as long as $p \in [1, 4)$, and so the problem is reduced to estimating square root of the leverage scores of the rows of the matrix. The key intuition here is that random Gaussian projection suffices to estimate the 2-norm with multiplicative error polynomial in n. Our algorithm then takes the median of the 2-norms of row $\mathbf{J}\mathbf{a}_t$ projected by a number of independent Gaussian vectors, where $\mathbf{J}$ is the matrix in the computation of the leverage score. We next state the formal guarantees of our crude approximation method.

Lemma F.11. *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$, let $\mathbf{Z} \in \mathbb{R}^{m \times d}$ be a matrix such that $\frac{1}{C} \cdot \mathbf{A}^\top \mathbf{A} \preceq \mathbf{Z}^\top \mathbf{Z} \preceq C \cdot \mathbf{A}^\top \mathbf{A}$, for some constant C, there exists an algorithm that computes an n^α approximation to the L_p Lewis weight using $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ runtime.*

Proof. We show the correctness of Algorithm 3. Recall that the leverage score of $\mathbf{a}_t$ with respect to $\mathbf{A}$ is $\mathbf{a}_t^\top (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{a}_t$. Since $\frac{1}{C} \cdot \mathbf{A}^\top \mathbf{A} \preceq \mathbf{Z}^\top \mathbf{Z} \preceq C \cdot \mathbf{A}^\top \mathbf{A}$, then $\mathbf{a}_t^\top (\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{a}_t$ is a C-approximation to the leverage score. Let $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z})^{-1/2}$, so that $\|\mathbf{J}\mathbf{a}_t\|_2^2$ is a C-approximation to the leverage score of $\mathbf{a}_t$. Thus it suffices to find an $n^{\alpha/2}$-approximation to $\|\mathbf{J}\mathbf{a}_t\|_2$. To that end, observe that for a unit vector $\mathbf{v} \in \mathbb{R}^d$ and a random Gaussian vector $\mathbf{g} \in \mathbb{R}^d$ whose entries are independently drawn from $\mathcal{N}(0, 1)$, we have that

$$\mathbf{Pr}\left[\langle \mathbf{v}, \mathbf{g} \rangle \geq \log^2 n\right] \leq 1 - \frac{1}{\mathrm{poly}(n)}$$
$$\mathbf{Pr}\left[\langle \mathbf{v}, \mathbf{g} \rangle \leq \frac{1}{n^{1/100p}}\right] \leq \mathcal{O}\left(\frac{1}{n^{1/100p}}\right).$$

Thus by the median over $\mathcal{O}\left(\frac{1}{100p}\right)$ values of $\|\mathbf{g}^{(i)}\mathbf{J}\mathbf{a}_t\|_2$, we can approximate the leverage score of $\mathbf{a}_t$ within $n^{1/100p}$. Hence, we can also approximate the square root of the leverage score of $\mathbf{a}_t$ within $n^{1/100p}$. By Lemma F.9 and Lemma F.10, this results in a n^α-approximation to the L_p Lewis weight of $\mathbf{a}_t$.

For the runtime, observe that $\mathbf{A}^\top \mathbf{A}$ over changes $\mathrm{poly}(d)$ times throughout the stream, so the computation of $\mathbf{g}^{(i)}\mathbf{J}$ is a lower order term than the stream length, i.e., total runtime over all values of $\mathbf{J}$ is $o(n)$. To compute $\mathbf{g}^{(i)}\mathbf{J}\mathbf{a}_t$ from $\mathbf{a}_t$, the algorithm uses $\mathcal{O}(d)$ time per row and in fact, input-sparsity time over the entire matrix. □

F.2 Fast Algorithm for Subspace Embeddings

Next, we apply Algorithm 3 in our framework in Algorithm 4 to give a fast streaming algorithm for L_p subspace embedding. We recall a refined sampling scheme for L_p subspace embedding that achieves optimal space but slow update time.

Theorem F.12 (Wang et al. (2026)). *Given a matrix $\mathbf{A}$ of n rows on $[-M, \ldots, -1, 0, 1, \ldots, M]^d$, with $M = \mathrm{poly}(n)$ and online condition number κ, there exists a one-pass streaming algorithm in the row arrival model that outputs a $(1+\varepsilon)$-coreset of $\mathbf{A}$ and uses $\mathcal{O}(d) \cdot f\left(d, \frac{1}{\varepsilon}, p\right)$ words of space, where*

$$f\left(d, \frac{1}{\varepsilon}, p\right) = \begin{cases} \frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon} \, \mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p \in [1, 2) \\ \mathcal{O}\left(\frac{d}{\varepsilon^2}\right), & p = 2 \\ \frac{d^{p/2}}{\varepsilon^2} \cdot \mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p > 2 \end{cases}.$$

Moreover, the update time is $\mathrm{poly}(d)$.

We first down-sample the stream using the crude approximation from Algorithm 3 to obtain a substream $\mathcal{S}'$ with size n^α, and then we pass it on to the refined sampling scheme to achieve both fast update time and optimal space. We present our algorithm in Algorithm 9.

Algorithm 9 Fast L_p-embedding on insertion-only streams based on Algorithm 4

Require: Matrix $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \in \mathbb{R}^{n \times d}$ arriving as a data stream; $\varepsilon \in (0, 1)$; parameter $p \geq 1$
Ensure: $(1 + \varepsilon)$-coreset for L_p-subspace embedding

1: Initialize $\mathbf{Z} \leftarrow \emptyset$, $\lambda \leftarrow \mathcal{O}\left(\frac{d}{\varepsilon^2} \log d \log n\right)$, and $Q \leftarrow \mathcal{O}\left(\frac{1}{100p}\right)$
2: Generate independent random Gaussian vectors $\mathbf{g}^{(1)}, \ldots, \mathbf{g}^{(Q)} \in \mathbb{R}^d$
3: **for** $t \in [n]$ **do**
4: Use $\mathbf{g}^{(1)}\mathbf{J}, \ldots, \mathbf{g}^{(Q)}\mathbf{J}$ to compute an n^α-approximation $\widetilde{\sigma(\mathbf{a}_t)}$ to the online Lewis weight of $\mathbf{a}_t$ {See Algorithm 3}
5: Set $\bar{p}(\mathbf{a}_t) \leftarrow \min\left(1, n^\alpha \lambda \cdot \widetilde{\sigma(\mathbf{a}_t)}\right)$
6: With probability $\bar{p}(\mathbf{a}_t)$, sample $\frac{1}{\bar{p}(\mathbf{a}_t)} \cdot \mathbf{a}_t$ into a stream $\mathcal{S}'$
7: **if** $\mathbf{a}_t$ is sampled into $\mathcal{S}'$ **then**
8: Sample x_t using the refined sampling scheme to obtain a $(1 + \varepsilon)$-coreset $\mathbf{Z}$
9: **end if**
10: **end for**
11: **return** $\mathbf{Z}$

The following statement provides the full guarantee of Algorithm 9.

Theorem F.13. *Given a matrix $\mathbf{A}$ of n rows on $[-M, \ldots, -1, 0, 1, \ldots, M]^d$, with $M = \mathrm{poly}(n)$ and online condition number κ, there exists a one-pass streaming algorithm in the row arrival model that outputs a $(1 + \varepsilon)$-coreset of $\mathbf{A}$ and uses $\mathcal{O}(d) \cdot f\left(d, \frac{1}{\varepsilon}, p\right)$ words of space, where*

$$f\left(d, \frac{1}{\varepsilon}, p\right) = \begin{cases} \frac{d}{\varepsilon^2} \log \frac{d}{\varepsilon} \, \mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p \in [1, 2) \\ \mathcal{O}\left(\frac{d}{\varepsilon^2}\right), & p = 2 \\ \frac{d^{p/2}}{\varepsilon^2} \cdot \mathrm{polylog}\left(\frac{d}{\varepsilon}\right), & p \in (2, 4) \end{cases}.$$

Moreover, the total runtime is $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$.

Proof. We proceed by induction on the time t. The correctness at the first time is clear, since the initial nonzero row must be sampled. Now assuming that the correctness of the both a constant-factor L_2 subspace embedding coreset, which is used to estimate the online Lewis weights, and a $(1 + \varepsilon)$-coreset for L_p subspace embedding at time $t - 1$. Then at time t, if the row is sampled by the refined sampling scheme, either the row is added to a coreset in the merge-and-reduce tree, in which case both coreset invariants are maintained, or the sampling procedure is performed because the batch has gotten too large. In the latter case, we use Lemma F.11 to compute a crude approximation to the Lewis scores of each row in a batch of k rows. This uses $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ total runtime and produces an insertion-only stream of length $\mathcal{O}\left(n^{1-\Omega(1/p)}\right)$. We then use this as the input to the refined sampling scheme. By Theorem F.12, there is $\mathrm{poly}(d)$ update time on the stream of length $o(n)$ to generate both a constant-factor coreset and a $(1 + \varepsilon)$-factor coreset at time t, which completes our induction. Thus, the total runtime of the refined sampling scheme is $\mathcal{O}\left(n^{1-\Omega(1/p)} \, \mathrm{poly}(d)\right)$, which is a lower order term for $\mathrm{poly}(d) \ll n^{1/p}$, and the overall runtime is dominated by $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$, which is at most amortized runtime $\mathcal{O}(d)$. In addition, the space complexity follows directly from Theorem F.12. Note that in the crude sampling scheme, we compute $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z})^{-1/2} \in \mathbb{R}^{d \times d}$ and store $\mathbf{g}^{(1)}\mathbf{J}, \ldots, \mathbf{g}^{(Q)}\mathbf{J} \in \mathbb{R}^d$ using $Q \cdot d$ words of space, where Q is a constant, and so it is dominated by the other terms in the space complexity. □

G Projection-Cost Preservation

In this section, we describe our one-pass streaming algorithm on insertion-only streams for projection-cost preservation (PCP) that uses total runtime $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$. Recall that in the problem of projection-cost preservation, the rows of matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ arrive one-by-one and the goal is to obtain a matrix $\mathbf{B} \in \mathbb{R}^{m \times d}$ such that for all rank-k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d \times d}$,

$$(1 - \varepsilon)\|\mathbf{A} - \mathbf{AP}\|_F^2 \leq \|\mathbf{B} - \mathbf{BP}\|_F^2 \leq (1 + \varepsilon)\|\mathbf{A} - \mathbf{AP}\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. PCP sketches preserve the geometry properties of the dataset by preserving the sum of squared distances of the columns from any k-dimensional subspace up to multiplicative factors, and thus has wide applications in dimensionality reduction, random matrix algorithms, low-rank approximation (Cohen et al., 2015; 2017; Braverman et al., 2020; Musco & Musco, 2020).

We now recall the λ-ridge leverage score of a row in $\mathbf{A}$, which is generalized from the standard leverage score.

Definition G.1 (λ-ridge leverage score)**.** *For a given parameter λ, the* λ-ridge leverage score *of a row $\mathbf{a}_i \in \mathbb{R}^d$ of a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ is*

$$\mathbf{a}_i^\top (\mathbf{A}^\top \mathbf{A} + \lambda \mathbb{I}_d)^{-1} \mathbf{a}_i.$$

In this work, we set $\lambda_{\mathbf{A}} := \frac{\|\mathbf{A}-\mathbf{A}_k\|_F^2}{k}$, where $\mathbf{A}_k$ is the optimal rank-k approximation of matrix $\mathbf{A}$. It is known that sampling each row in $\mathbf{A}$ according to the $\lambda_{\mathbf{A}}$-ridge leverage score would give a $(1+\varepsilon)$ PCP. We now recall the full guarantees of the offline algorithm for PCP.

Theorem G.2 (Offline rank-k PCP sketch)**.** *(Cohen et al., 2017) Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, there exists an algorithm that with probability $1-\delta$, outputs a matrix $\mathbf{M}$ that has $f\left(k, \frac{1}{\varepsilon}, \frac{1}{\delta}\right) = \mathcal{O}\left(\frac{k}{\varepsilon^2}\log\frac{k}{\delta}\right)$ (rescaled) rows of $\mathbf{A}$ and for all rank k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d\times d}$,*

$$(1-\varepsilon)\|\mathbf{A}-\mathbf{A}\mathbf{P}\|_F^2 \le \|\mathbf{M}-\mathbf{M}\mathbf{P}\|_F^2 \le (1+\varepsilon)\|\mathbf{A}-\mathbf{A}\mathbf{P}\|_F^2.$$

We next recall the online λ-ridge leverage score, which is used to obtain an online rank-k PCP.

Definition G.3 (Online λ-ridge leverage score)**.** *For a given parameter λ, the* online λ-ridge leverage score *of a row $\mathbf{a}_i \in \mathbb{R}^d$ of a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ is*

$$\mathbf{a}_i^\top (\mathbf{A}_{i-1}^\top \mathbf{A}_{i-1} + \lambda \mathbb{I}_d)^{-1} \mathbf{a}_i,$$

where $\mathbf{A}_{i-1}$ is the matrix containing the first $i-1$ rows of $\mathbf{A}$.

Unfortunately, in the online setting, we do not have prior knowledge about $\lambda_{\mathbf{A}} := \frac{\|\mathbf{A}-\mathbf{A}_k\|_F^2}{k}$. Instead, we choose $\lambda_i = \Theta\left(\frac{\|\mathbf{M}_{i-1}-(\mathbf{M}_{i-1})_k\|_F^2}{k}\right)$ for each row $\mathbf{a}_i$, where $\mathbf{M}_{i-1}$ consists of the (rescaled) sampled rows from previous steps. Since this quantity is always smaller than $\lambda_{\mathbf{A}}$, it provides a valid PCP sketch by oversampling. We now recall the full guarantees of the online algorithm for PCP.

Theorem G.4 (Online rank-k PCP sketch)**.** *(Braverman et al., 2020) Given parameters $\varepsilon > 0, k > 0$, and a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, whose rows $\mathbf{A}_1, \ldots, \mathbf{A}_n$ arrive sequentially in a stream with condition number κ, there exists an online algorithm that with high probability, outputs a matrix $\mathbf{M}$ that has $g\left(n, k, \frac{1}{\varepsilon}, \kappa\right) = \mathcal{O}\left(\frac{k}{\varepsilon^2}\log n \log^2\kappa\right)$ (rescaled) rows of $\mathbf{A}$ and for all rank k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d\times d}$,*

$$(1-\varepsilon)\|\mathbf{A}-\mathbf{A}\mathbf{P}\|_F^2 \le \|\mathbf{M}-\mathbf{M}\mathbf{P}\|_F^2 \le (1+\varepsilon)\|\mathbf{A}-\mathbf{A}\mathbf{P}\|_F^2.$$

In addition, the algorithm needs $\mathrm{nnz}(\mathbf{A}) \cdot \mathrm{polylog}\,\frac{n}{\varepsilon}$ total runtime.

We next state a useful result in (Braverman et al., 2020) that bounds the sum of online ridge leverage scores, which is used to prove the number of sampled rows in our algorithm.

Lemma G.5 (Upper bound on the sum of online ridge leverage scores)**.** *(Braverman et al., 2020) For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ with condition number κ and each index $i \in [n]$, let $\lambda_{\mathbf{a}_i} := \frac{\|\mathbf{A}_{i-1}-(\mathbf{A}_{i-1})_k\|_F^2}{k}$ be the ridge defined by the prefix-matrix $\mathbf{A}_{i-1}$ and let $\sigma(\mathbf{a}_i) := \mathbf{a}_i^\top (\mathbf{A}_{i-1}^\top \mathbf{A}_{i-1} + \lambda_{\mathbf{a}_i}\mathbb{I}_d)^{-1}\mathbf{a}_i$ be the online ridge leverage score of $\mathbf{a}_i$. Then we have $\sum_{i=1}^n \sigma(\mathbf{a}_i) = \mathcal{O}\left(k\log^2\kappa\right)$.*

G.1 Crude Lewis Weight Approximation

Similar as L_p subspace embedding, we first require a crude n^α-approximation to the online ridge leverage score for each row that arrives in the stream. Again, we utilize the random Gaussian projection to approximate the computation of the leverage scores.

Algorithm 10 Crude ridge leverage score approximation

Require: Matrix $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \in \mathbb{R}^{n \times d}$ that arrives as a data stream, matrix $\mathbf{Z} \in \mathbb{R}^{m \times d}$ satisfying $\frac{1}{C} \cdot \mathbf{A}^\top \mathbf{A} \preceq \mathbf{Z}^\top \mathbf{Z} \preceq C \cdot \mathbf{A}^\top \mathbf{A}$ for some constant C, vector $\mathbf{a} \in \mathbb{R}^d$, parameter $p \geq 1$
Ensure: n^α approximation to the L_p Lewis weight of $\mathbf{a}$

1: Generate $Q = \mathcal{O}\left(\frac{1}{100p}\right)$ independent random Gaussian vectors $\mathbf{g}^{(1)}, \ldots, \mathbf{g}^{(Q)} \in \mathbb{R}^d$
2: Given $\widehat{\lambda}$ a 2-approximation to $\lambda = \frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{k}$, let $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda})^{-1/2}$
3:
4: **return** $\widehat{\sigma(\mathbf{a})} \leftarrow \mathrm{median}_{q \in [Q]} \left|\mathbf{g}^{(q)} \mathbf{J} \mathbf{a}\right|$

Lemma G.6. *Given $\mathbf{A} \in \mathbb{R}^{n \times d}$ and $\widehat{\lambda}$ being a 2-approximation of $\lambda_{\mathbf{A}} := \frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{k}$, let $\mathbf{Z} \in \mathbb{R}^{m \times d}$ be a matrix such that*

$$\frac{1}{C} \cdot (\mathbf{A}^\top \mathbf{A} - \lambda_{\mathbf{A}} \cdot \mathbb{I}_d) \preceq \mathbf{Z}^\top \mathbf{Z} \preceq C \cdot (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d),$$

for some constant C, there exists an algorithm that computes an n^α approximation to the $\lambda_{\mathbf{A}}$-ridge leverage score using $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ runtime.

Proof. We prove the guarantees of Algorithm 10 We use the same construction as Lemma F.11. Since

$$\frac{1}{C} \cdot (\mathbf{A}^\top \mathbf{A} - \lambda_{\mathbf{A}} \cdot \mathbb{I}_d) \preceq \mathbf{Z}^\top \mathbf{Z} \preceq C \cdot (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d),$$

we have

$$\frac{1}{10C} \cdot (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d) \preceq \mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda} \cdot \mathbb{I}_d \preceq 10C \cdot (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d).$$

Therefore, we have

$$\frac{1}{10C} \cdot \mathbf{a}_t^\top (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d)^{-1} \mathbf{a}_t \leq \mathbf{a}_t^\top (\mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda} \cdot \mathbb{I}_d)^{-1} \mathbf{a}_t \leq 10C \cdot \mathbf{a}_t^\top (\mathbf{A}^\top \mathbf{A} + \lambda_{\mathbf{A}} \cdot \mathbb{I}_d)^{-1} \mathbf{a}_t.$$

That is, $\mathbf{a}_t^\top (\mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda})^{-1} \mathbf{a}_t$ is a $10C$-approximation to the $\lambda_{\mathbf{A}}$-ridge leverage score of $\mathbf{a}_t$. Let $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda})^{-1/2}$, so that $\|\mathbf{J}\mathbf{a}_t\|_2^2$ is a $10C$-approximation to the $\lambda_{\mathbf{A}}$-ridge leverage score of $\mathbf{a}_t$. Thus, it suffices to find an $n^{\alpha/2}$-approximation to $\|\mathbf{J}\mathbf{a}_t\|_2$. Again, observe that for a unit vector $\mathbf{v} \in \mathbb{R}^d$ and a random Gaussian vector $\mathbf{g} \in \mathbb{R}^d$ whose entries are independently drawn from $\mathcal{N}(0, 1)$, we have that

$$\mathbf{Pr}\left[\langle \mathbf{v}, \mathbf{g} \rangle \geq \log^2 n\right] \leq 1 - \frac{1}{\mathrm{poly}(n)}$$
$$\mathbf{Pr}\left[\langle \mathbf{v}, \mathbf{g} \rangle \leq \frac{1}{n^{1/100p}}\right] \leq \mathcal{O}\left(\frac{1}{n^{1/100p}}\right).$$

Thus by the median over $\mathcal{O}\left(\frac{1}{100}\right)$ values of $\|\mathbf{g}^{(i)}\mathbf{J}\mathbf{a}_t\|_2$, we can approximate the $\lambda_{\mathbf{A}}$-ridge leverage score of $\mathbf{a}_t$ within $n^{1/100}$, and we success with high probability via a union bound over all arrivals in the stream.

For the runtime, observe that $\mathbf{A}^\top \mathbf{A}$ over changes $\mathrm{poly}(d)$ times throughout the stream, so the computation of $\mathbf{g}^{(i)}\mathbf{J}$ is a lower order term than the stream length, i.e., total runtime over all values of $\mathbf{J}$ is $o(n)$. To compute $\mathbf{g}^{(i)}\mathbf{J}\mathbf{a}_t$ from $\mathbf{a}_t$, the algorithm uses $\mathcal{O}(d)$ time per row and in fact, input-sparsity time over the entire matrix. □

G.2 Fast Algorithm for Projection-Cost Preservation

In this section, we show that our one-pass streaming algorithm can be implemented in $\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$ total time by applying our framework in Section B. We first recall a refined sampling scheme for PCP that achieves optimal space but slow update time.

Theorem G.7 (Wang et al. (2026)). *Given a matrix* $\mathbf{A}$ *of* n *rows on* $[-M, \ldots, -1, 0, 1, \ldots, M]^d$ *with* $M = \mathrm{poly}(n)$*, there exists a one-pass streaming algorithm in the row arrival model that outputs a* $(1+\varepsilon)$*-PCP of* $\mathbf{A}$ *with probability* 0.99*. The algorithm uses* $\mathcal{O}\left(k \,\mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\kappa)\right) + d\right) \cdot f\left(d, \frac{1}{\varepsilon}, \frac{1}{0.01}\right)$ *words of space, where* $f\left(k, \frac{1}{\varepsilon}, \frac{1}{\delta}\right) = \mathcal{O}\left(\frac{k}{\varepsilon^2} \log \frac{k}{\delta}\right)$*. Moreover, the update time is* $\mathrm{poly}(d)$.

Next, we apply Algorithm 10 in our framework in Algorithm 4 to give a fast streaming algorithm. Note that Algorithm 10 gives a crude approximation to the online ridge leverage scores, we then sample with the crude approximation to obtain a stream $\mathcal{S}'$, and next, we run the algorithm in Theorem G.7 on $\mathcal{S}'$ such that the slower steps can be amortized into lower order terms. We present our algorithm in Algorithm 11.

Algorithm 11 Fast Projection-Cost Preservation on Insertion-Only Streams

Require: Matrix $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\} \in \mathbb{R}^{n \times d}$ arriving as a data stream; $\varepsilon \in (0, 1)$; parameter $p \geq 1$
Ensure: $(1+\varepsilon)$-coreset for L_p-subspace embedding

1: Initialize $\mathbf{Z} \leftarrow \emptyset$, $\alpha \leftarrow \mathcal{O}\left(\frac{\log n}{\varepsilon^2}\right)$, and $Q \leftarrow \mathcal{O}\left(\frac{1}{100p}\right)$
2: Generate independent random Gaussian vectors $\mathbf{g}^{(1)}, \ldots, \mathbf{g}^{(Q)} \in \mathbb{R}^d$
3: **for** $t \in [n]$ **do**
4: Use $\mathbf{g}^{(1)}\mathbf{J}, \ldots, \mathbf{g}^{(Q)}\mathbf{J}, \widehat{\lambda}$ to compute an n^α-approximation $\widetilde{\sigma(\mathbf{a}_t)}$ to the online ridge leverage score of $\mathbf{a}_t$ {See Algorithm 10}
5: Set $\bar{p}(\mathbf{a}_t) \leftarrow \min\left(1, n^\alpha \lambda \cdot \widetilde{\sigma(\mathbf{a}_t)}\right)$
6: With probability $\bar{p}(\mathbf{a}_t)$, sample $\frac{1}{\bar{p}(\mathbf{a}_t)} \cdot \mathbf{a}_t$ into a substream $\mathcal{S}'$
7: **if** $\mathbf{a}_t$ is sampled into $\mathcal{S}'$ **then**
8: Sample x_t using the refined sampling scheme to obtain a $(1+\varepsilon)$-coreset $\mathbf{Z}$
9: **end if**
10: **end for**
11: **return** $\mathbf{Z}$

The next statement provides the full guarantee of Algorithm 11.

Theorem G.8. *Given a matrix* $\mathbf{A}$ *of* n *rows on* $[-M, \ldots, -1, 0, 1, \ldots, M]^d$ *with* $M = \mathrm{poly}(n)$*, there exists a one-pass streaming algorithm in the row arrival model that outputs a* $(1+\varepsilon)$*-PCP of* $\mathbf{A}$ *with probability* 0.99*. The algorithm uses* $\mathcal{O}\left(k \,\mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\kappa)\right) + d\right) \cdot f\left(d, \frac{1}{\varepsilon}, \frac{1}{0.01}\right)$ *words of space, where* $f\left(k, \frac{1}{\varepsilon}, \frac{1}{\delta}\right) = \mathcal{O}\left(\frac{k}{\varepsilon^2} \log \frac{k}{\delta}\right)$*. Moreover, the total runtime is* $\mathcal{O}\left(\mathrm{nnz}(\mathbf{A})\right)$.

Proof. We proceed by induction on the time t. The correctness at the first time is clear, since the initial nonzero row must be sampled. Now assuming that the $\mathbf{Z}$ is both a $(1+\varepsilon)$-PCP of and a constant-factor subspace embedding of $\mathbf{A}$ at time $t-1$, then at time t, we use Lemma G.6 to compute a crude approximation to the online ridge leverage score. This uses $\mathcal{O}\left(\mathrm{nnz}(\mathbf{A})\right)$ total runtime and produces an insertion-only stream of length $\mathcal{O}\left(n^{1-\Omega(1/p)}\right)$. We then use this as the input to the refined sampling scheme. By Theorem G.7, there is $\mathrm{poly}(d)$ update time on the stream of length $o(n)$ to generate a $(1+\varepsilon)$-factor coreset at time t, which completes our induction. Thus, the total runtime for the refined sampling scheme is $\mathcal{O}\left(n^{1-\Omega(1/p)} \,\mathrm{poly}(d)\right)$, which is a lower order term for $\mathrm{poly}(d) \ll n^{1/p}$, and so the overall runtime is dominated by $\mathcal{O}\left(\mathrm{nnz}(\mathbf{A})\right)$, which is amortized runtime $\mathcal{O}\left(d\right)$. In addition, the space complexity follows directly from Theorem G.7. Note that in the crude sampling scheme, we compute $\mathbf{J} = (\mathbf{Z}^\top \mathbf{Z} + 4C\widehat{\lambda})^{-1/2} \in \mathbb{R}^{d \times d}$ and store $\mathbf{g}^{(1)}\mathbf{J}, \ldots, \mathbf{g}^{(Q)}\mathbf{J} \in \mathbb{R}^d$ using $Q \cdot d$ words of space, where Q is a constant, and so it is dominated by the other terms in the space complexity. □

H Additional Experimental Results and Implementation Details

In this section, we further discuss the parameter choices in our experiments, and we present the results of L_2-subspace embedding.

H.1 Implementation of k-means Coreset Construction

Recall that our experiments in Section 6 implement the following three baselins:

- Pure (batched) online sensitivity sampling: We implement BatchSens (Algorithm 2) with batch size $B = 100$ to estimate the online sensitivities and perform online sampling.
- CWZ framework (Cohen-Addad et al., 2023): We pass the online sampled sub-stream by Algorithm 2 to a merge-and-reduce data structure.
- Crude sampling + CWZ framework: We use RoughSens (Algorithm 8) with batch size $B = 100$ to estimate the online sensitivities and perform online sampling. Then we feed the resulted sub-stream to merge-and-reduce.

We select the sampling constants to compare running times and qualities at similar coreset sizes. We use target budgets of $64, 128, 256, 512$ for k-means. For each target size m, we first tune the online sampling constant $C_{\text{ol}}^{\text{OL}}$ such that the Online baseline roughly reports m points. For CWZ, we set the constant $C_{\text{ol}}^{\text{CWZ}}$ for sampling the online sub-stream before post-filtering to m, and then we tune its merge-and-reduce constant, $C_{\text{mr}}^{\text{CWZ}}$, so that it has mean output size close to m. Finally, our crude sampling method has two constants to calibrate: C_{crude}, which controls crude sampling before merge-and-reduce, and $C_{\text{mr}}^{\text{ours}}$, which controls compression within merge-and-reduce. We jointly calibrate C_{crude} and $C_{\text{mr}}^{\text{ours}}$, evaluating each candidate pair on pilot runs. We select settings whose mean coreset size is close to the target and whose clustering cost is within a prescribed tolerance of CWZ's, while keeping the two methods' merge-and-reduce constants close. We note that the empirical coreset sizes in the experiments might not perfectly match the budget, but it is within a 0.15-fraction. We report the empirical mean coreset sizes. The experiments in Appendix H.2 use the same way to choose scaling factors under budgets $2500, 5000, 7500, 10000$ for subspace embedding.

H.2 Experimental Results of L_2-Subspace Embedding

In this section, we further demonstrate the practical advantage of our fast streaming framework on L_2-subspace embedding.

Real-world dataset. We evaluate our algorithms on the Wave Energy Farm dataset Neshat et al. (2020) from the UCI Machine Learning Repository, which contains $n = 36043$ instances with $d = 149$ attributes, representing positions of wave energy converters and their power outputs. This dataset is widely used for large-scale regression tasks; its high dimensionality and substantial number of observations make it an ideal candidate for testing the performance of streaming algorithms. We test our L_2 subspace embedding algorithm on the $n \times d$ matrix loaded by the Wave Energy Farm dataset, since it is an essential approach to accelerate regression tasks.

Baseline. We again choose CWZ as our primary baseline since it significantly out-performs other baselines in approximation quality under strict space constraints, both theoretically and empirically (see Cohen-Addad et al. (2025) and Section 6). In particular, it is more preferable than pure online sensitivity sampling under limited memory, which is essential for streaming applications. We thus show our crude sampling approach improve its update time. We use the exact same construction as described in Section 6 and Appendix H.1. The only difference is the online sensitivity approximation: the sensitivity (i.e., leverage score) of row $\mathbf{a}$ can be approximated by directly computing $\mathbf{a}^\top(\widetilde{\mathbf{A}}^\top\widetilde{\mathbf{A}})^{-1}\mathbf{a}$, where $\widetilde{\mathbf{A}}$ is the currently stored L_2-subspace embedding; we do not need the complicated approximation algorithm in (k, z)-clustering.

Evaluation Metrics. We evaluate subspace distortion using the weighted Gram matrix of the coreset. Let A' contain the sampled rows and W be the diagonal matrix of their weights. We solve the generalized eigenvalue problem $(A'^\top W A')x = \lambda(A^\top A)x$, restricting both matrices to the eigenspace of $A^\top A$ with eigenvalues greater than 10^{-7}. We report $\max_i |\lambda_i - 1|$, which measures the worst relative distortion of squared Euclidean norms over this subspace. We vary the sampling coefficients to obtain different coreset sizes and evaluate how the runtime scales with the budget.

Results and Discussion. We summarize our empirical results on the Wave Energy Farm dataset in Figure 4, which plots the curves of the average of 10 trials.

The results are similar to k-means clustering. As shown in Figure 4, our fast streaming framework consistently outperforms the baseline, providing a substantial speedup across all coreset sizes.

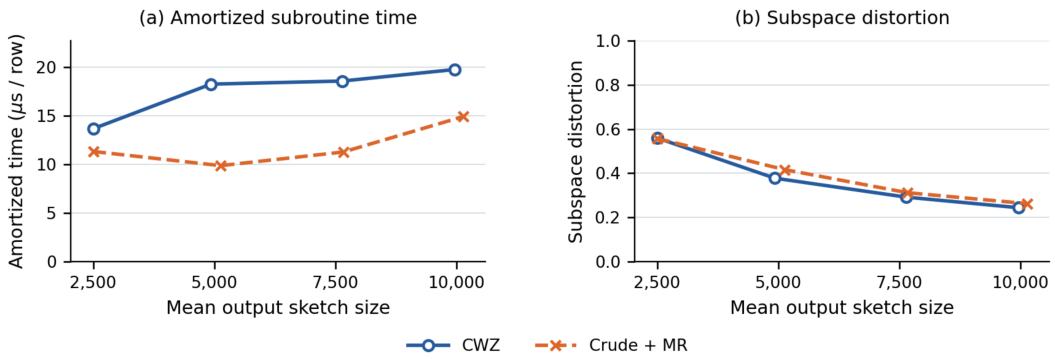


Figure 4: L_2-subspace embedding on WAVE ENERGY FARM.

Meanwhile, the distortion nearly matches the result of CWZ. Therefore, this provide additional evidence that our crude sampling framework can accelerate coreset construction in practice, while maintaining the coreset quality under small space budgets.