

Coresets for Robust Clustering via Black-box Reductions to Vanilla Case

Shaofeng H.-C. Jiang*
Peking University

Jianing Lou†
Peking University

Abstract

We devise ε -coresets for robust (k, z) -CLUSTERING with m outliers through black-box reductions to vanilla clustering. Given an ε -coreset construction for vanilla clustering with size N , we construct coresets of size $N \cdot \text{poly} \log(km\varepsilon^{-1}) + O_z(\min\{km\varepsilon^{-1}, m\varepsilon^{-2z} \log^z(km\varepsilon^{-1})\})$ for various metric spaces, where O_z hides $2^{O(z \log z)}$ factors. This increases the size of the vanilla coreset by a small multiplicative factor of $\text{poly} \log(km\varepsilon^{-1})$, and the additive term is up to a $(\varepsilon^{-1} \log(km))^{O(z)}$ factor to the size of the optimal robust coreset. Plugging in recent vanilla coreset results of [Cohen-Addad, Saulpic and Schwiegelshohn, STOC’21; Cohen-Addad, Draganov, Russo, Saulpic and Schwiegelshohn, SODA’25], we obtain the first coresets for (k, z) -CLUSTERING with m outliers with size near-linear in k while previous results have size at least $\Omega(k^2)$ [Huang, Jiang, Lou and Wu, ICLR’23; Huang, Li, Lu and Wu, SODA’25].

Technically, we establish two conditions under which a vanilla coreset is as well a robust coreset. The first condition requires the dataset to satisfy special structures – it can be broken into “dense” parts with bounded diameter. We combine this with a new bounded-diameter decomposition that has only $O_z(km\varepsilon^{-1})$ non-dense points to obtain the $O_z(km\varepsilon^{-1})$ additive bound. Another sufficient condition requires the vanilla coreset to possess an extra size-preserving property. To utilize this condition, we further give a black-box reduction that turns a vanilla coreset to the one that satisfies the said size-preserving property, and this leads to the alternative $O_z(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1}))$ additive size bound.

We also give low-space implementations of our reductions in the *dynamic* streaming setting. Combined with known streaming constructions for vanilla coresets [Braverman, Frahling, Lang, Sohler and Yang, ICML’17; Hu, Song, Yang and Zhong, arXiv’1802.00459], we obtain the first dynamic streaming algorithms for coresets for k -MEDIAN (and k -MEANS) with m outliers, using space $\tilde{O}(k + m) \cdot \text{poly}(d\varepsilon^{-1} \log \Delta)$ for inputs on a discrete grid $[\Delta]^d$.

1 Introduction

(k, z) -CLUSTERING is a fundamental problem that is well studied in both computer science and related areas such as operations research. Given a metric space (V, dist) and a dataset $X \subseteq V$, (k, z) -CLUSTERING aims to find a center set $C \subseteq V$ of k points, such that the clustering objective $\text{cost}_z(X, C)$ is minimized, i.e.,

$$\text{cost}_z(X, C) := \sum_{x \in X} (\text{dist}(x, C))^z,$$

where $\text{dist}(x, C) := \min_{c \in C} \text{dist}(x, c)$. This formulation generally captures several well known variants of k -clustering, including k -MEDIAN (when $z = 1$) and k -MEANS (when $z = 2$).

*Email: shaofeng.jiang@pku.edu.cn

†Email: loujn@pku.edu.cn

Unfortunately, datasets can often be noisy, and the objective of (k, z) -CLUSTERING is very sensitive to the noise. In fact, even adding a single *outlier* point that is distant to every other point could significantly bias the clustering centers towards the outlier point. Hence, robust variants of clustering are much desired in order to combat this issue. We consider a natural formulation suggested by [CKMN01], called (k, z) -CLUSTERING clustering with m outliers, (k, z, m) -CLUSTERING for short, where a parameter m is introduced to denote the number of outliers. The new cost function is denoted as $\text{cost}_z^{(m)}(X, C)$, and it is evaluated by first removing the m points that are furthest to C from X , denoting the resultant points as X' , and compute $\text{cost}_z(X', C)$ (i.e., using the vanilla (k, z) -CLUSTERING objective). Equivalently,

$$\text{cost}_z^{(m)}(X, C) := \min_{Y \in \binom{X}{m}} \text{cost}_z(X \setminus Y, C).$$

Coreset [HM04] is a powerful technique for obtaining efficient algorithms for clustering. Roughly speaking, an ε -coreset is a tiny proxy of the dataset that approximates the clustering objective within $(1 \pm \varepsilon)$ factor for every potential center set C . Coresets are not only useful for obtaining fast algorithms, but can also be converted into streaming [HM04], distributed [BEL13] and fully-dynamic algorithms [HK20] via merge-and-reduce framework [HM04]. Tremendous progress has been made on finding small coresets for (vanilla) (k, z) -CLUSTERING. Take k -MEDIAN ($z = 1$) in Euclidean \mathbb{R}^d (i.e., $V = \mathbb{R}^d$, $\text{dist} = \ell_2$) for example, the initial coreset size $O(k\varepsilon^{-d} \log n)$ [HM04] has been improved in a series of works [HK07, FL11, SW18, FSS20, HV20, BJKW21a, CSS21, CLSS22], all the way to $\tilde{O}(k\varepsilon^{-3})$ via a novel sampling framework [CSS21, CLSS22], and this nearly matches a lower bound of $\Omega(k\varepsilon^{-2})$ [CLSS22]. (Alternatively, in the regime of $\varepsilon^{-1} \gg k$, a better bound of $\tilde{O}(k^{4/3}\varepsilon^{-2})$ may be obtained [CLS⁺22, HLW24].)

The sampling framework proposed by [CSS21] is modified to handle several variants of clustering [BCJ⁺22], and this modified framework is recently adapted to obtain an $O(m) + \tilde{O}(k^3\varepsilon^{-5})$ size coreset for k -MEDIAN with m outliers [HJLW23], which exponentially improves a decade-old $(k + m)^{O(k+m)} \text{poly}(d\varepsilon^{-1} \log n)$ bound [FS12]. This bound is further improved to $O(m) + \tilde{O}(k^2\varepsilon^{-4})$ in a recent work [HLLW25]. However, even this improved bound is unlikely to be tight; Specifically, there is a sharp transition from $m = 0$ (i.e., the vanilla case) to $m = 1$ (i.e., only one outlier is considered) that increases the size by at least a $k\varepsilon^{-1}$ factor (compared with [CLSS22]). It is unclear if this transition is fundamental and whether or not it can be avoided. Technically, existing approaches for robust coresets are built on frameworks designed for the vanilla case and is adapted in an ad-hoc way. In case improved bounds are obtained for the vanilla case, it is still technically nontrivial to adapt it to the robust case.

In this paper, we systematically address these issues by proposing a new framework via black-box reductions to the vanilla case. Specifically, we wish to start from any vanilla coreset algorithm (which may be the optimal one), and figure out how to add only a few points to convert it into a robust coreset, ideally to obtain a size bound that is very close to the vanilla case. Indeed, this study helps to fundamentally understand “the price of robustness” for coresets, i.e., the exact gap in size bounds between vanilla clustering and robust clustering. Since the coreset construction is via reduction, it also opens the door to more applications, such as coreset algorithms for robust clustering in sublinear models (as long as the vanilla version can be constructed in the corresponding model).

1.1 Our Results

Our main result, stated in Theorem 1.1, is a novel coreset construction for (k, z, m) -CLUSTERING via a reduction to the vanilla case. Crucially, this reduction is *black-box* style which works with any

vanilla coresets construction without knowing its implementation detail. As mentioned, this type of bound helps to fundamentally understand the “price of robustness” for coresets. Our result works for general metric spaces, and we choose to state the result for the Euclidean case which is arguably the most natural setting for (k, z) -CLUSTERING. Overall, this size bound has linear dependence in N (albeit N is evaluated on slightly larger parameters), and the main increase in size is additive.

Theorem 1.1 (Euclidean case). *Assume there is an algorithm that constructs an ε -coreset for (k, z) -CLUSTERING of size $N(d, k, \varepsilon^{-1})$ for any dataset from \mathbb{R}^d . Then, there is an algorithm that constructs an ε -coreset for (k, z, m) -CLUSTERING of size*

$$\min \{N(d, k, O(\varepsilon^{-1})) + A_1, N(O(d), O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1})) + A_2\} \quad (1)$$

for any dataset from \mathbb{R}^d , where $A_1 = O_z(km\varepsilon^{-1})$ and $A_2 = O_z(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1}))$. The two bounds in the min follow from Theorem 3.1 and Theorem 4.2, respectively.

As mentioned, our result in Theorem 1.1 is applicable to various other metric spaces besides Euclidean, including doubling metrics, general finite metrics and shortest-path metrics for graphs that exclude a fixed minor. The size bounds only needs to change the parameters of N according to the metric and does not change the additive terms A_1 and A_2 . For instance, for doubling metrics, our bound simply replaces the d in (1) by the doubling dimension of the metric space (up to constant factor). Detailed size bounds can be found in Section B.

We start with justifying the tightness of our size bounds without plugging in any concrete vanilla coreset bound N . Observe that Theorem 1.1 actually provides two size bounds, each corresponding to one of the two terms in the min of (1). The first bound of $A_1 + N(d, k, O(\varepsilon^{-1}))$ is more useful when m is small. In particular, it makes our result the first to achieve a smooth transition from $m = 0$ (vanilla case) to $m > 0$ (robust case) in an asymptotic sense, even when plugging in the optimal vanilla coreset. Indeed, the $O_z(km\varepsilon^{-1})$ bound becomes $O_z(k\varepsilon^{-1})$ when $m = O(1)$, and this is asymptotically dominated by a lower bound of $\Omega(k\varepsilon^{-2})$ for vanilla coreset as shown in [CLSS22]. For the second bound, it needs to use k that is $\text{poly} \log(km\varepsilon^{-1})$ larger in N , but under the typical case of $N(d, k, \varepsilon^{-1}) = \text{poly}(dk\varepsilon^{-1})$, this only increases N by a factor of $\text{poly} \log(km\varepsilon^{-1})$ which is minor. Regarding its additive term A_2 , it is actually up to only a $(\varepsilon^{-1} \log(km))^{O(z)}$ factor to the optimal robust coresets, which is nearly tight with respect to m due to a lower bound of $\Omega(m)$ for robust coreset [HJLW23].

Specific Coreset Size Bounds Since our reduction is black box, the large body of results on coresets for vanilla clustering can be readily applied. We start with listing in Table 1 the concrete coreset bounds obtained by plugging in recent results for (k, z) -CLUSTERING [CLSS22, CLS⁺22, HLW24, CDR⁺25]. These lead to the first coresets of near-linear size in k for (k, z, m) -CLUSTERING in all the metric spaces listed, thereby improving the previous k^2 dependency [FS12, HJLW23, HLLW25]. We can also obtain improved coresets when plugging in results designed for specific parameters. For instance, in low-dimensional Euclidean space, a coreset for $(1, z)$ -CLUSTERING of size $2^{O(z \log z)} \cdot \tilde{O}(\sqrt{d}\varepsilon^{-1})$ was given by [HHHW23]. Applying this result to Theorem 1.1, we obtain a coreset for $(1, m, z)$ -CLUSTERING of size $2^{O(z \log z)} \cdot \tilde{O}((m + \sqrt{d})\varepsilon^{-1})$.

It is worth mentioning that our entire algorithm is deterministic, provided that the given vanilla coreset algorithm is deterministic. As highlighted in recent surveys [MS18, Fel20], deterministically constructing coresets is an important and less understood open question, even for vanilla clustering without outliers. The best-known deterministic construction works only in low-dimensional Euclidean space [HK07], yielding a coreset of size $\text{poly}(k) \cdot \varepsilon^{-O(d)}$. For Euclidean k -MEANS, a coreset of size $k^{\text{poly}(\varepsilon^{-1})}$ can be constructed deterministically [FSS20], and the size can be improved to

Table 1: List of relevant cores et bounds under various metric spaces. In all the bounds reported in the table we omit a minor multiplicative factor of $(z \log(km\varepsilon^{-1}))^{O(z)}$. For our result listed in the last row, the exact form (without omitting the abovementioned minor factor) is $\min\{\mathbf{N} + O_z(km\varepsilon^{-1}), \mathbf{N} \cdot \text{polylog}(km\varepsilon^{-1}) + O_z(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1}))\}$, noting that the \mathbf{N} in the second term of $\min\{\cdot, \cdot\}$ has a $\text{polylog}(km\varepsilon^{-1})$ factor.

metric space M	problem	size	reference
Euclidean \mathbb{R}^d	vanilla (\mathbf{N})	$kd\varepsilon^{-\max\{2,z\}}$	[CSS21]
	vanilla (\mathbf{N})	$k\varepsilon^{-2-z}$	[CLSS22]
	vanilla (\mathbf{N})	$k^{\frac{2z+2}{z+2}} \varepsilon^{-2}$	[HLW24]
	robust, $z = 1$	$(k + m)^{k+m} (\varepsilon^{-1} d \log n)^2$	[FS12]
	robust	$m + k^3 \varepsilon^{-3z-2}$	[HJLW23]
	robust	$m + k^2 \varepsilon^{-2z-2}$	[HLLW25]
doubling metrics	vanilla (\mathbf{N})	$k\varepsilon^{-\max\{2,z\}} \cdot \text{ddim}(M)$	[CSS21]
	robust	$m + k^2 \varepsilon^{-2z} \cdot (\text{ddim}(M) + \varepsilon^{-1})$	[HLLW25]
n -point metric space	vanilla (\mathbf{N})	$k\varepsilon^{-\max\{2,z\}} \cdot \log n$	[CSS21]
	robust, $z = 1$	$(k + m)^{k+m} \varepsilon^{-2} \log^4 n$	[FS12]
	robust	$m + k^2 \varepsilon^{-2z} (\log n + \varepsilon^{-1})$	[HLLW25]
bounded treewidth graphs	vanilla (\mathbf{N})	$k\varepsilon^{-\max\{2,z\}} \cdot \text{tw}$	[CSS21]
	robust	$m + k^2 \varepsilon^{-2z-2} \cdot \text{tw}$	[HLLW25]
excluded-minor graphs	vanilla (\mathbf{N}), $z = 1$	$k\varepsilon^{-2}$	[CDR ⁺ 25]
	vanilla (\mathbf{N})	$k\varepsilon^{-2} \cdot \min\{\varepsilon^{-z-1}, k\}$	[CDR ⁺ 25]
	robust	$m + k^2 \varepsilon^{-2z-2}$	[HLLW25]
all the above	robust	$\mathbf{N} + \min\{km\varepsilon^{-1}, m\varepsilon^{-2z}\}$	ours

$\text{poly}(k\varepsilon^{-1})$ if we consider a slightly more general definition of cores ets, known as cores ets with offset¹ [CSS23]. All these results can be plugged into our reduction to directly achieve a deterministic cores et construction for (k, z, m) -CLUSTERING, with the cores et size increasing as shown in (1).²

Streaming Implementation of Theorem 1.1 We also obtain reduction style cores et construction algorithms for (k, z, m) -CLUSTERING over a *dynamic* stream of points in \mathbb{R}^d using small space, which follows from a streaming implementation of Theorem 1.1. We consider a standard streaming model proposed by [Ind04] which has been widely employed in the literature. In this model, the input points come from a *discrete* grid $[\Delta]^d$ for some integer parameter Δ , and they arrive as a dynamic stream with point insertions and deletions. At the end of the stream, the algorithm should return a weighted set as the cores et. We present in Theorem 1.2 the result for the case of $z = 1$ which is k -MEDIAN with m outliers.

Theorem 1.2 (Informal version of Theorem 5.1). *Assume there is a streaming algorithm that constructs an ε -cores et for k -MEDIAN for every dataset from $[\Delta]^d$ presented as a dynamic stream, using space $W(d, \Delta, k, \varepsilon^{-1})$ and with a failure probability of at most $1/\text{poly}(d \log \Delta)$. Then, there is a streaming algorithm that constructs an ε -cores et for (k, m) -MEDIAN with constant probability for every dataset from $[\Delta]^d$ presented as a dynamic stream, using space $\min\{W_1, W_2\} \cdot \text{poly}(d \log \Delta)$,*

¹In the definition of cores ets with offset, the original clustering objective is preserved by adding a universal constant (the offset) to the clustering objective on the cores et.

²For cores ets with offset, our reductions and analysis can be easily adapted and still achieve asymptotically the same cores et size.

where

$$\begin{aligned} W_1 &= \tilde{O}(km\varepsilon^{-1}) + W(d, \Delta, k, O(\varepsilon^{-1})), \\ W_2 &= \tilde{O}(k + m\varepsilon^{-2}) + W(d + 1, \varepsilon^{-1}\Delta^{\text{poly}(d)}, k \text{poly}(d), O(\varepsilon^{-1})). \end{aligned}$$

Observe that we did not explicitly state the size of the coreset, and only focus on the space complexity, since the coreset size is upper bounded by the space complexity of the streaming algorithm. Moreover, once one obtains an ε -coreset at the end of the stream, one can always run another existing coreset construction again on top of it to obtain a possibly smaller coreset. Hence, it is only the space complexity that matters in streaming coreset construction. We also note that once the coreset is obtained from our streaming algorithm, one can continue to find a $(1 + \varepsilon)$ -approximation to (k, m) -MEDIAN without using higher order of space.

By combining Theorem 1.2 with an existing streaming coreset construction for k -MEDIAN [BFL⁺17], we obtain a space complexity of $\tilde{O}(\min\{km\varepsilon^{-1}, m\varepsilon^{-2}\} + k\varepsilon^{-2}) \cdot \text{poly}(d \log \Delta)$. Remarkably, the dependence on k and ε matches that of [BFL⁺17], which is $k\varepsilon^{-2}$. A similar bound for $z = 2$, i.e., k -MEANS with m outliers, may be obtained by combining with a dynamic streaming algorithm for k -MEANS (e.g., [HSYZ18]). These results are the first nontrivial dynamic streaming algorithms for robust k -clustering, where the dependence on $\varepsilon, k, d, \log \Delta$ is comparable to existing bounds for vanilla clustering (up to degree of polynomial), and the linear dependence in m is also shown to be necessary (see Claim D.1).

In fact, our result is also the first dynamic streaming coreset construction for *any* variant of (k, z) -CLUSTERING. Indeed, recent works manage to devise (offline) coresets for variants of clustering, and they can even immediately imply insertion-only streaming algorithm via standard techniques such as merge-and-reduce [HM04], but they are not readily applicable to dynamic point streams. Our new reduction style framework is a new approach that opens a door to obtaining dynamic streaming algorithms for other variants of clustering, which may be of independent interest.

1.2 Technical Overview

Our main technical contributions are the establishment of two conditions for a vanilla coreset to become a robust coreset, and black-box reductions that turn vanilla coresets to satisfy these conditions, which yields the bounds mentioned in Theorem 1.1. The two conditions reveal new fundamental structures of the robust clustering problem, especially the structural relation to vanilla clustering, which may be of independent interest. Moreover, to make vanilla coresets satisfy the two conditions in a black-box way, it requires us to devise a bounded-diameter decomposition called almost-dense decomposition, a new separated duplication transform for metric spaces, as well as adapting and refining a recently developed sparse partition [JLN⁺05, Fil20] (which was also studied under the notion of consistent hashing [CFJ⁺22]). The sparse partition/consistent hashing technique is also crucially used in our streaming implementation, which suggests a framework fundamentally different from those based on quadrees as in previous dynamic streaming algorithms for (vanilla) clustering [BFL⁺17, HSYZ18].

We focus on presenting these ideas for the (k, m) -MEDIAN problem (which is the $z = 1$ case) in Euclidean \mathbb{R}^d (although most discussion already works for general metrics). We also discuss how to implement the reductions in streaming.

1.2.1 First Reduction: $(O(km\varepsilon^{-1}) + N)$ Size Bound

Condition I: Vanilla Coresets on Dense Datasets Are Robust Our first condition comes from a simple intuition: we wish $\text{cost}(X, C) \approx \text{cost}^{(m)}(X, C)$ for every $C \subseteq \mathbb{R}^d$, which is perhaps

the most natural way to ensure a vanilla coreset is automatically a robust coreset. Now, a sufficient condition is to ensure the dataset is somewhat uniform, in the sense that for every center set $C \subseteq \mathbb{R}^d$, the contribution from every m data points is at most εOPT (or equivalently, each data point contributes $\varepsilon \text{OPT}/m$). This way, whatever the m outliers are, the change of objective from cost to $\text{cost}^{(m)}$ is always within ε factor.

We thus consider datasets that can be broken into parts such that each part has diameter $\lambda := O(\varepsilon \text{OPT}/m)$ and contains at least $\Omega(\varepsilon^{-1}m)$ points. In such datasets, every data point can find $\Omega(\varepsilon^{-1}m)$ points within distance $\lambda = O(\varepsilon \text{OPT}/m)$, hence its contribution is naturally charged to these nearby points, which is averaged to $O(\varepsilon \text{OPT}/m)$. We call such a dataset *dense*, and the above intuition leads to a formal reduction that for dense datasets, a vanilla coreset is as well a robust coreset (Lemma 3.3).

Reduction I: Almost-dense Decomposition Of course, a general dataset may not be dense. Nevertheless, we manage to show a weaker almost-dense decomposition exists for every dataset (Lemma 3.4). This decomposition breaks the dataset into two parts: a subset A that is guaranteed to be dense and a remaining subset B that only consists of $O(km\varepsilon^{-1})$ points. The subset B actually has more refined property: it can be broken into parts such that each part has diameter at most λ (which is similar to the dense part) and each contains at most $O(\varepsilon^{-1}m)$ points (which is the “complement” property of the dense part). We call subset B the *sparse* subset.

To construct such a decomposition, we start with an optimal solution C^* to the robust clustering (in the algorithm it suffices to use a tri-criteria solution, see Definition 2.4). Let $\lambda = O(\varepsilon \text{OPT}/m)$ be the target diameter bound of the dense and sparse parts. We first identify those isolated points F consisting of both the m outliers and those “far away” points with distance more than λ from C^* . The number of far away points is bounded by $\text{OPT}/\lambda = O(\varepsilon^{-1}m)$ using an averaging argument. Now since every remaining point is within λ to C^* , we cluster all these remaining points, namely $X \setminus F$, with respect to C^* . Each cluster automatically has a diameter bound by $O(\lambda)$. Finally, we take clusters with more than $\Omega(\varepsilon^{-1}m)$ points as the dense subset, and the remainder, along with F , forms the sparse subset, which contains at most $O(|C^*|\varepsilon^{-1}m) + O(\varepsilon^{-1}m) + m = O(km\varepsilon^{-1})$ points in total.

With such a decomposition, one can simply put the sparse subset who has $O(km\varepsilon^{-1})$ points into the coreset, and use the vanilla coreset on the dense subset. This yields our first $O(km\varepsilon^{-1}) + N$ size bound (recall that N is the size of the vanilla coreset) as in Theorem 1.1.

1.2.2 Second Reduction: $(m\varepsilon^{-2} + N) \text{poly log}(km\varepsilon^{-1})$ Size Bound

Overview of Reduction II Indeed, in Reduction I, the step of simply adding sparse subset into the coreset may sound suboptimal. Hence, our second algorithm is built on top of Reduction I, such that we first follow the steps in Reduction I to build a vanilla coreset on the dense subset, and then run more refined new steps to the sparse subset. In fact, the new steps can be viewed as a separate reduction that takes the sparse subset as input and generates a robust coreset. We focus on these new steps and call it Reduction II (without the steps already in Reduction I). Crucially, Reduction II itself is standalone and can run on any dataset (and thus can be independent of Reduction I), albeit it would result in a coreset of size $O(\log n)$ (where n is the size of its input dataset). Luckily, we would eventually run it on the sparse subset which has $O(km\varepsilon^{-1})$ points, so the $\log n$ factor becomes $\log(km\varepsilon^{-1})$, and the overall size bound is competitive (which yields the other size bound in Theorem 1.1).

Next, we start with describing our Condition II (which is also standalone), and discuss how our Reduction II make a vanilla coreset to satisfy this condition.

Condition II: Size-preserving Vanilla Coresets Are Robust The second condition requires a stronger property in the definition of the coreset (instead of the dense property in the dataset as in Condition I). We consider a *size-preserving* property, which is defined with respect to some λ -bounded diameter decomposition – a partition \mathcal{P} of point set X such that each part $P \in \mathcal{P}$ has diameter at most λ . For a given λ , a (vanilla) coreset S is size-preserving, if there *exists* a λ -bounded diameter decomposition \mathcal{P} , such that for every $P \in \mathcal{P}$, $|P| = |S \cap P|$ (here we interpret the coreset, which is a weighted set, as a multiset). This “size-preserving property” of coresets intuitively means the coreset could be interpreted as moving each data point by at most λ distance. More formally, there is a one-to-one correspondence g between dataset X and coreset S such that $\text{dist}(x, g(x)) \leq \lambda$.

Our Condition II states that if a vanilla coreset S satisfies the *size-preserving property* then it is automatically a robust coreset (Lemma 4.3). Next, we provide an overview of the difficulties we encountered in the proof of Condition II and the high-level ideas for resolving them.

Naive Approach: Bounding the Gap between cost and $\text{cost}^{(m)}$ We explore the condition to make the gap between $\text{cost}(X, C)$ and $\text{cost}^{(m)}(X, C)$ is small. Unlike Condition I, here we are now making no assumptions about the structure of X . Observe that the precise difference between the two cost functions is the outliers: Let X_{out} be the set of outliers (which is defined with respect to C), then the vanilla $\text{cost}(X, C)$ is larger and has an extra term $\sum_{x \in X_{\text{out}}} \text{dist}(x, C)$ compared with the robust $\text{cost}^{(m)}(X, C)$. An immediate idea to “avoid” this term in vanilla $\text{cost}(X, C)$, is to additionally put a center at each point in X_{out} so that points from X_{out} contribute 0. This works perfectly when each outlier is separated from every other point since each point in X_{out} forms a singleton cluster, and this suggests a necessary condition: the vanilla coreset S needs to hold for $k' = k + m$ centers, i.e., it is a coreset for k' -MEDIAN.

Unfortunately, centers on outliers may not form singleton clusters, in which case $\text{cost}(X, C \cup X_{\text{out}})$ can underestimate $\text{cost}^{(m)}(X, C)$. Even if it is possible to pick another C' to make this said gap small, it is still challenging to also prove that $\text{cost}(S, C \cup C')$ is always close to $\text{cost}^{(m)}(S, C)$ for an arbitrary S and C (which is needed to imply $\text{cost}^{(m)}(X, C) \approx \text{cost}^{(m)}(S, C)$).

Key Observation: Vanilla Coresets Also Preserve the Gap Hence, instead of seeking for a specific center set C' that makes the gap $\text{cost}^{(m)}(X, C) - \text{cost}(X, C \cup C')$ small for every C , we show a weaker guarantee, such that the *gap is preserved* between X and the coreset S : we allow a large gap, just that if the gap is large in X , then it is also large in S , and the quantity is comparable. Specifically, we show that a coreset S for k' -MEDIAN with size-preserving property can preserve this gap. Namely,

$$\text{cost}^{(m)}(S, C) - \text{cost}(S, C \cup C') = \text{cost}^{(m)}(X, C) - \text{cost}(X, C \cup C') \pm \varepsilon \text{cost}^{(m)}(X, C) \quad (2)$$

for some $C' \subseteq X_{\text{out}}$. Indeed, this combining with the guarantee $\text{cost}(S, C \cup C') \in (1 \pm \varepsilon) \text{cost}(X, C \cup C')$ (which follows from the k' -MEDIAN coreset guarantee of S) implies $\text{cost}^{(m)}(S, C) \approx \text{cost}^{(m)}(X, C)$ (which is the robust coreset guarantee).

To see why the gap can be preserved, the intuition is that there are only a few points that contribute non-zero values to the gap $\text{cost}^{(m)}(X, C) - \text{cost}(X, C \cup C')$. To put it simply, let us first consider $C' = X_{\text{out}}$. Then only those inliers x of X that are closer to outliers X_{out} than to C can make $\text{dist}(x, C) \neq \text{dist}(x, C \cup C')$ hold. Let D denote such inliers, and let us loosely assume that only $g(D)$ contributes non-zero values to the gap $\text{cost}^{(m)}(S, C) - \text{cost}(S, C \cup C')$ (recalling that g is a one-to-one correspondence between the dataset X and the coreset S , such that $\text{dist}(x, g(x)) \leq \lambda$ for every $x \in X$). Then the difference of these two gaps can be upper bounded by $|\text{cost}(D, C) - \text{cost}(g(D), C)| + |\text{cost}(D, C \cup C') - \text{cost}(g(D), C \cup C')| \leq O(|D| \cdot \lambda)$ using the triangle inequality,

which is sufficient if $|D| \leq O(m)$ and we pick $\lambda = \varepsilon \text{OPT} / m$. Unfortunately, the naive setup of $C' = X_{\text{out}}$, as we do in the analysis above, may *not* lead to a small D (noticing that D is defined with respect to C'). Moreover, C' cannot be made small as well, as this could result in a large error of $\varepsilon \cdot \text{cost}(X, C \cup C')$ (recalling that (2) combining the coreset guarantee $\text{cost}(S, C \cup C') \in (1 \pm \varepsilon) \cdot \text{cost}(X, C \cup C')$ forms the complete cost preservation). Hence, it remains to find an appropriate C' that satisfies both requirements, and we provide a simple algorithm (Algorithm 3) to precisely achieve this.

Reduction II: Making Vanilla Coresets Size-preserving To utilize Condition II, we need to show that, for a given $\lambda > 0$, we can build a vanilla coreset S that is size-preserving using a generic vanilla coreset algorithm in a black-box way. Specifically the coreset should satisfy that there exists a λ -bounded partition $\mathcal{P} = (P_1, \dots, P_t)$ of X such that $|P_i| = |S \cap P_i|$ for $i \in [t]$ (recalling that we treat a coreset as a multiset). We first assume a generic partition \mathcal{P} is given, and assume that $t = |\mathcal{P}| = k$ for simplicity.

Key Observation: Separated Instance is Easy We make use of a simple observation: if the k parts are well-separated, meaning they are sufficiently distant from each other, then a vanilla coreset for k -MEDIAN is already (nearly) size-preserving. To see this, if we put one center at each of the P_j 's for $j \neq i$ but without any center at P_i , the clustering cost is roughly $|P_i| \cdot \text{dist}(P_i, X \setminus P_i)$ and this is up to $(1 \pm \varepsilon)$ factor to $|S \cap P_i| \cdot \text{dist}(P_i, X \setminus P_i)$ (by the coreset guarantee of S). This implies a slightly relaxed $|S \cap P_i| \in (1 \pm \varepsilon)|P_i|$, and it can be further adjusted to become strictly size-preserving.

Key Idea: Ensuring Separation Artificially A straightforward idea to deal with a generic \mathcal{P} is to artificially separate the parts. More precisely, for Euclidean \mathbb{R}^d , we define a mapping $h : P \rightarrow \mathbb{R}^{d+1}$, such that for $x \in P_i$, $h(x) := (x, i \cdot w)$ for some sufficiently large w (which can be loosely considered as infinite). We define $P'_i := h(P_i)$. Clearly, $\mathcal{P}' := (P'_1, \dots, P'_k)$ is a λ -bounded partition for $X' := h(X)$, and now the partition \mathcal{P}' is well-separated. Thus, one can apply the above argument for \mathcal{P}' , obtain a size-preserving coreset S' for X' , and find the corresponding S for X .

New Issue: Ensuring Coreset Guarantee However, a crucial issue arises: how can we ensure that S is a coreset for X ? In other words, we need to translate the coreset guarantee of S' (on X') to that of S (on X). It seems that the only way is to find a “bridge” center set C' for any fixed C such that $\text{cost}(X, C) \approx \text{cost}(X', C')$ and $\text{cost}(S, C) \approx \text{cost}(S', C')$. If such C' exists and S' can handle at least $|C'|$ centers, then the coreset guarantee of S follows. Now, let us focus on the requirement $\text{cost}(X, C) \approx \text{cost}(X', C')$. The tricky situation arises when some points in X may have the same nearest center in C but belong to different parts; for instance, assume that $x_i \in P_i$ and $x_j \in P_j$ both have the nearest center $c \in C$. After mapping through h , the images $x'_i = h(x_i)$ and $x'_j = h(x_j)$ can be “infinitely” far apart. Therefore, to maintain the clustering cost, we need to include two copies, $(c, i \cdot w)$ and $(c, j \cdot w)$, in C' . In the worst case, we would need to include $|\mathcal{P}| = k$ copies of every center in C , resulting in a size of k^2 for C' , and thus S' should be a coreset for k^2 -MEDIAN. Unfortunately, this worst-case scenario is unavoidable if \mathcal{P} is generic, and the reduction to k^2 -MEDIAN falls far from satisfaction, as our ultimate goal is near-linear size in k .

Solution: Grouping Parts using Sparse Partition [JLN⁺05] To address the above problem, we start by using a more careful way to “prune” centers in C' . For a part $P_i \in \mathcal{P}$, if it is far from C , specifically $\text{dist}(P_i, C) > \varepsilon^{-1}\lambda \geq \varepsilon^{-1}\text{diam}(P_i)$, then it suffices to include only one center in C' for

P_i , which is $c_i = \arg \min_{c \in C} \text{dist}(c, P_i)$, and we can verify that $\text{cost}(P_i, \{c_i\}) \in (1 \pm \varepsilon) \cdot \text{cost}(P_i, C)$. Through this pruning, these far parts can increase C' by at most k , which is acceptable.

The challenging part is then handling the close parts. For a close part P_i , we denote by C_i the centers that is relevant to it, i.e., C_i contains the nearest centers to points in P_i . This implies that $\text{cost}(P_i, C) = \text{cost}(P_i, C_i)$, meaning we only need to include $\{(c, i \cdot w) : c \in C_i\}$ in C' for every i . As discussed earlier, $\sum_i |C_i|$ could be $\Omega(k^2)$ in the worst case.

To further reduce the total number of relevant centers, we propose a new idea: group these parts together to create a new partition (G_1, \dots, G_l) of X . The relevant centers of G_j are $\bigcup_{i: P_i \subseteq G_j} C_i$. Our goal then becomes to find a grouping strategy such that $\sum_j |\bigcup_{i: P_i \subseteq G_j} C_i|$ is small.

We achieve this using a more structured partition called *sparse partition* [JLN⁺05]. Specifically, we can compute a partition $\mathcal{Q} = (G_1, \dots, G_l)$ of X such that each part G_j has a diameter at most $O(\varepsilon^{-1} \lambda \log n)$, and any subset with a diameter $O(\varepsilon^{-1} \lambda)$ intersects at most $O(\log n)$ parts. For simplicity, assume each part in \mathcal{P} is entirely covered by exactly one part in \mathcal{Q} (though this is not generally true, the proof follows similarly). Then, $\mathcal{Q} = (G_1, \dots, G_l)$ fits our needs perfectly. To see this, consider any center c . Notice that c can appear in at most $O(\log n)$ of the sets $\bigcup_{i: P_i \subseteq G_j} C_i$, since otherwise the subset $\bigcup_{i: c \in C_i} P_i$, whose diameter is $O(\varepsilon^{-1} \lambda)$, would intersect more than $O(\log n)$ parts, contradicting the guarantee provided by the sparse partition. As a result, $\sum_j |\bigcup_{i: P_i \subseteq G_j} C_i| \leq O(k \log n)$. This ensures that it suffices to construct a coresset for $O(k \log n)$ -MEDIAN that is size-preserving with respect to \mathcal{Q} .

An issue is that \mathcal{Q} is $O(\varepsilon^{-1} \lambda \log n)$ -bounded, but this can be adjusted through rescaling, resulting in a size increase by a factor of $\text{poly}(\varepsilon^{-1} \log n)$. Given that this coresset applies to a subset of size $n = \text{poly}(km\varepsilon^{-1})$ in the full algorithm, the $\log n$ term is acceptable.

1.2.3 Streaming Implementations

We provide an overview of how these steps can be adapted to the streaming setting using foundational streaming algorithmic tools.

λ -Bounded Partition Recall that both of our reductions rely on a λ -bounded partition \mathcal{P} (where $\lambda = \varepsilon \text{OPT} / m$, and we can guess this OPT). Therefore, the key is to design a streaming algorithm that finds a λ -bounded partition. However, in the streaming setting, it is not possible to maintain the λ -bounded partition explicitly, as it requires $\Omega(n)$ space, which is not affordable.

Our strategy is to employ a hashing version of sparse partition defined in [CFJ⁺22] that is space efficient and data oblivious and thus suitable for streaming. This hashing partitions the entire \mathbb{R}^d into buckets of diameter of $O(\lambda)$. Hence, after the stream ends, those non-empty buckets form a λ -bounded partition of X . Moreover, this hashing also guarantees that any subset of points with a diameter of $\lambda / \text{poly}(d)$ intersects at most $\text{poly}(d)$ buckets (similar to the sparse partition guarantee). By combining with the existence of the small-sized partition in Lemma 3.4, we can conclude there are very few, i.e., $\text{poly}(d\varepsilon^{-1}) \cdot (k + m)$, distinct buckets after this hashing.

In the following, we fix such a hashing, and we outline how our reductions can be adapted to the streaming setting using foundational streaming algorithmic tools.

The First $O(km) \text{poly}(\varepsilon^{-1} d \log \Delta)$ Space Bound We first show how to implement our Reduction I in the streaming setting. Our main task is to identify all points that lie in small buckets containing fewer than $O(\varepsilon^{-1} m)$ points, which correspond to the sparse subset B , and construct a vanilla coresset for the points lying in other buckets, which correspond to the dense subset. The latter is easy; once

the former is achieved, we can simply run a vanilla dynamic streaming algorithm on a stream representing $X \setminus B$ (which can be achieved using some tricks).

The former task can be translated into a streaming algorithm: identify at most $a := \text{poly}(d\varepsilon^{-1}) \cdot (k+m)$ buckets, such that each contain at most $b := O(\varepsilon^{-1}m)$ points. This task is similar to sparse recover in streaming, but it is two-level which is different from the standard one, i.e., it should recover only from those buckets that contain small number of points. Hence, we introduce a modified sparse recovery structure (Lemma C.3) for this specific task, and it takes about $O(ab) \text{poly}(d \log \Delta) = O(km + m^2) \text{poly}(\varepsilon^{-1}d \log \Delta)$ space.

However, this m^2 dependence is still suboptimal. To achieve a near-linear space bound in m , we show that there exists a subset $F \subseteq X$ of size $m \cdot \text{poly}(d\varepsilon^{-1})$ such that, after removing F , the remaining points span very few buckets, allowing us to set $a := k \text{poly}(d)$ (see Lemma 5.3). This subset F can be efficiently identified using a two-level ℓ_0 -sampler [CFJ⁺22, Lemma 3.3] in the dynamic stream (see Lemma 5.4). We then use (modified) sparse recovery to identify the sparse subset after removing F , reducing the total space to $O(km) \text{poly}(\varepsilon^{-1}d \log \Delta)$.

The Second $\tilde{O}(k+m) \cdot \text{poly}(\varepsilon^{-1}d \log \Delta)$ Space Bound Our second algorithm first uses the same streaming algorithm to find F as in the first bound, and recall that the non-empty buckets after removing F is small, i.e., $|\varphi(X \setminus F)| \leq O(k \text{poly}(d))$. This implies that $\varphi(X \setminus F)$ gives rise to a small-sized λ -bounded partition, denoted by \mathcal{P} . Our Condition II states that if we construct a vanilla coreset S that is size-preserving, i.e., $|S \cap P| = |P|$ for every P , it is directly a robust coreset. Hence, in summary, our next step is to implement the construction of a size-preserving vanilla coreset in the dynamic streaming setting (see Reduction II in Section 1.2). Specifically, we map every point x in the stream into $(x, \varphi(x) \cdot w)$, where w is sufficiently large, and that $\varphi(x)$ can be encoded in $\text{poly}(d \log \Delta)$ bits and thus can be interpreted as an integer. This transformed data is then fed to an instance of a streaming algorithm designed for vanilla coresets. At the end of the stream, we obtain a coreset from this instance and consider its pre-image under φ (which deletes the last coordinate) as the coreset for X , which is approximately size-preserving. We run a sparse recovery algorithm to recover the sizes of each bucket to calibrate the weights (to make it strictly size-preserving).

1.3 Related Work

Besides clustering with outliers, researchers have extensively explored coresets for various other clustering variants. Two particularly interesting variants are clustering with capacity constraints and clustering with fairness constraints, both of which have been investigated in [CKLV17, SSS19, HJV19, CL19, BFS21, BCJ⁺22] for coreset constructions. Notably, the coresets for both these variants can be reduced to an assignment-preserving coreset [SSS19, HJV19], for which several studies apply a hierarchical sampling framework [Che09] to construct small-sized coresets with assignment-preserving properties [BFS21, BCJ⁺22]. Recently, Huang et al. [HLLW25] introduced a general unified model to capture a wide range of clustering problems with general assignment constraints, including clustering with fairness/capacity constraints, clustering with outliers (which we consider), and a fault-tolerant variant of clustering [KPS00].

Coresets for other variants of clustering have also been considered, including projective clustering [FL11, FSS20, TWZ⁺22], clustering with missing values [BJKW21b], ordered-weighted clustering [BJKW19], and line clustering [MF19, LSF22].

2 Preliminaries

For integer $n \geq 1$, let $[n] := \{1, 2, \dots, n\}$. Given a mapping $\varphi : X \rightarrow Y$, for a subset $Z \subseteq X$, denote $\varphi(Z) := \{\varphi(x) : x \in Z\}$, and for an image $y \in Y$, denote its inverse by $\varphi^{-1}(y) := \{x \in X : \varphi(x) = y\}$. Let $M = (V, \text{dist})$ be an underlying metric space, and we assume oracle access to dist . For a point set $X \subseteq V$, let $\text{diam}(X) := \max_{x, y \in X} \text{dist}(x, y)$ be the diameter of X . For a set $C = \{c_1, \dots, c_t\} \subseteq V$ ($t > 0$), a clustering of X with respect to C is a partition $\{P_1, \dots, P_t\}$ of X such that for every $i \in [t]$ and every point $x \in P_i$, $\text{dist}(x, c_i) \leq \text{dist}(x, C)$.

Weighted Set A set S with an associated weight function $w_S : S \rightarrow \mathbb{R}_{\geq 0}$ a *weighted set*. When we talk about a weighted set A , we denote the weight function of A by w_A unless otherwise specified. For an unweighted set B , we interpret it as weighted set with unit weight function $w_B \equiv 1$. We say a weighted set Y is a weighted subset of X , denoted by $(Y, w_Y) \subseteq (X, w_X)$, if $Y \subseteq X$ and for every $x \in Y$, $w_Y(x) \leq w_X(x)$. For two weighted sets X and Y , the weight of their union $Z := X \cup Y$ is defined as $w_Z := w_X + w_Y$. We denote by $X - Y$ the weighted set Z with the weight function w_Z , where $w_Z = w_X - w_Y$, and Z is the support of w_Z . Here, for two weight functions $f : X \rightarrow \mathbb{R}$ and $g : Y \rightarrow \mathbb{R}$, we use $f + g$ to denote the weight function $h : X \cup Y \rightarrow \mathbb{R}$ such that for every $x \in X \cup Y$, $h(x) := f(x) + g(x)$. Likewise, we denote by $f - g$ the subtraction of weight functions.³

Clustering with Outliers (Weighted) In our proof, we need to consider the weighted version of (k, z, m) -CLUSTERING, and we extend the definition as follows. For a weighted set $X \subseteq V$, and a center set $C \subseteq V$ with $|C| = k$, we define the clustering objective for (k, z) -CLUSTERING as $\text{cost}_z(X, C) := \sum_{x \in X} w_X(x) \cdot \text{dist}^z(x, C)$. Moreover, given real number $h \geq 0$, we denote by $\mathcal{L}_X^{(h)} := \{(Y, w_Y) \subseteq (X, w_X) : w_Y(Y) = h\}$ the collection of all possible weighted subset of X with a total weight equal to h . We define the objective for (k, z, m) -CLUSTERING as

$$\text{cost}_z^{(m)}(X, C) := \min_{(Y, w_Y) \in \mathcal{L}_X^{(m)}} \text{cost}_z(X - Y, C). \quad (3)$$

One can verify that when $m = 0$, the definition of $\text{cost}_z^{(m)}$ is equivalent to cost_z . We call a weighted set Y an m -outlier set of X with respect to C , if $(Y, w_Y) \in \mathcal{L}_X^{(m)}$ and $\text{cost}_z^{(m)}(X, C) = \text{cost}_z(X - Y, C)$. We use $\text{OPT}_z^{(m)}(X)$ to denote the optimal objective value for (k, z, m) -CLUSTERING on X , i.e., $\text{OPT}_z^{(m)}(X) := \min_{C \in V^k} \text{cost}_z^{(m)}(X, C)$.

Coresets Our definition of coreset is almost identical to the one in [HJLW23, Definition 2.1], except that the coreset preserves the objective for all potential number of outliers $0 \leq h \leq m$ up to m simultaneously.⁴

Definition 2.1 (Coresets). Given $0 < \varepsilon < 1$ and a dataset $X \subseteq V$, an ε -coreset of X for (k, z, m) -CLUSTERING is a weighted set $S \subseteq X$ such that

$$\forall C \in V^k, 0 \leq h \leq m, \quad \text{cost}_z^{(h)}(S, C) \in (1 \pm \varepsilon) \cdot \text{cost}_z^{(h)}(X, C).$$

A weighted set is called an ε -coreset for the special case of (k, z) -CLUSTERING if it is an ε -coreset for (k, z, m) -CLUSTERING with $m = 0$.

³If $x \notin X$, we define $f(x) := 0$, and similarly, if $x \notin Y$, we define $g(x) := 0$.

⁴Although the definition of coresets in [HJLW23] does not require preserving objective for all $0 \leq h \leq m$ simultaneously, their algorithm actually constructs such a coreset (see [HJLW23, Remark 3.2]).

For analysis purposes, we also need a weaker notion of coreset, in Definition 2.2, that allows for additive errors. Such notions were also widely used in previous coreset constructions (see e.g., [CSS21, CLSS22, BCJ⁺22, HJLW23]).

Definition 2.2 (Additive-error coresets). Given $0 < \varepsilon < 1, \eta \geq 0$ and a dataset $X \subseteq V$, an (ε, η) -coreset of X for (k, z, m) -CLUSTERING is a weighted set $S \subseteq X$ such that

$$\forall C \in V^k, 0 \leq h \leq m, \quad \left| \text{cost}_z^{(h)}(S, C) - \text{cost}_z^{(h)}(X, C) \right| \leq \varepsilon \cdot \text{cost}_z^{(h)}(X, C) + \eta.$$

An important property of coresets is that it is composable. Roughly speaking, if S_A is a coreset of A and S_B is a coreset of B , then $S_A \cup S_B$ is a coreset of $A \cup B$. It is well known that composability holds directly from the definition of vanilla coreset, and here we verify that it also holds for our definition of coresets for (k, z, m) -CLUSTERING. We present this fact with respect to the more general additive-error coreset, and we provide a proof in the Appendix A for completeness.

Fact 2.3 (Composability of coresets). For $0 < \varepsilon < 1, \eta_1, \eta_2 \geq 0$, and two datasets $X, Y \subseteq V$, if S_X is an (ε, η_1) -coreset of X for (k, z, m) -CLUSTERING, and S_Y is an (ε, η_2) -coreset of Y for (k, z, m) -CLUSTERING, then $S_X \cup S_Y$ is an $(\varepsilon, \eta_1 + \eta_2)$ -coreset of $X \cup Y$ for (k, z, m) -CLUSTERING.

Tri-criteria Approximation Similar to many previous works [BCJ⁺22, HJLW23, HLLW25], we need a tri-criteria approximation algorithm for the (k, z, m) -CLUSTERING problem. Here, an (α, β, γ) -approximation is a set of βk centers, whose cost is at most α times the optimal, allowing a violation γ to the number of outliers.

Definition 2.4 ((α, β, γ) -Approximation). Given $\alpha, \beta, \gamma \geq 1$ and a dataset $X \subseteq V$, we say a center set $C \subseteq V$ is an (α, β, γ) -approximation solution to (k, z, m) -CLUSTERING on X if it holds that $\text{cost}_z^{(\gamma m)}(X, C) \leq \alpha \cdot \text{OPT}_z^{(m)}(X)$ and $|C| \leq \beta k$.

There exists a randomized algorithm that computes a $(2^{O(z)}, O(1), O(1))$ -approximation in near-linear time with high probability [BVX19] (which is also applicable to general metrics). For a deterministic approach, a $(2^{O(z)}, O(1), 1)$ -approximation solution can be obtained in polynomial time (e.g., [FKRS19]). A more detailed discussion on other possible algorithms for such an approximation can be found in [HJLW23, Appendix A].

Lemma 2.5 (Generalized triangle inequalities). Let $a, b \geq 0$ and $\varepsilon \in (0, 1)$, then for $z \geq 1$, the following holds.

1. (Lemma A.1 of [MMR19]) $(a + b)^z \leq (1 + \varepsilon)^{z-1} \cdot a^z + (1 + \frac{1}{\varepsilon})^{z-1} \cdot b^z$;
2. (Claim 5 of [SW18]) $(a + b)^z \leq (1 + \varepsilon) \cdot a^z + (\frac{3z}{\varepsilon})^{z-1} \cdot b^z$.

3 Reduction I: Density of Datasets

Theorem 3.1. Suppose an underlying metric space $M = (V, \text{dist})$ is given. Assume there exists an algorithm \mathcal{A} that, given $0 < \varepsilon < 1$, integers $k, z \geq 1$ and an n -point dataset from M as input, constructs an ε -coreset for (k, z) -CLUSTERING of size $N(n, k, \varepsilon^{-1})$ in time $T(n, k, \varepsilon^{-1})$. Then there is an algorithm that, given $0 < \varepsilon < 1$, integers $k, z, m \geq 1$, an n -point dataset X from M and a $(2^{O(z)}, O(1), O(1))$ -approximation solution $C^* \subseteq V$ to (k, z, m) -CLUSTERING on X as input, constructs an ε -coreset S of X for (k, z, m) -CLUSTERING. This algorithm invokes \mathcal{A} a constant number of times and runs in time $\tilde{O}(nk) + T(n, k, O(\varepsilon^{-1}))$. The constructed coreset S has size $2^{O(z \log z)} \cdot O(km\varepsilon^{-1}) + N(n, k, O(\varepsilon^{-1}))$.

Construction Based on Bounded Partition As we mention, the reduction relies on a space partition that breaks the input into bounded partition. Here, by “bounded”, we mean the diameter of each part in the partition is upper bounded by some parameter $\lambda > 0$. Formally, we introduce the notion of λ -bounded partitions in Definition 3.2.

Definition 3.2 (λ -Bounded partition). For a dataset $Y \subseteq V$ and $\lambda \geq 0$, we say a partition $\mathcal{P} = \{P_i\}_i$ of Y is λ -bounded if for every $P_i \in \mathcal{P}$, $\text{diam}(P_i) \leq \lambda$.

We describe our algorithm in Algorithm 1, which additionally takes a λ -bounded partition of X as input for some $\lambda > 0$. We note that the value of λ would affect the accuracy of the final coreset, as shown in Lemma 3.3. Later on, we select an appropriate λ to ensure that the output of Algorithm 1 is an ε -coreset. Essentially, our algorithm computes a decomposition of the dataset, based on the λ -bounded partition \mathcal{P} , into $X_{\mathcal{D}}$ and $X_{\mathcal{S}}$, where $X_{\mathcal{D}}$ contains the points within dense parts, while $X_{\mathcal{S}}$ contains those within sparse parts. Here, “dense” and “sparse” refer to whether a part contains many or few points, respectively. Then, it simply constructs a coreset $S_{\mathcal{D}}$ of $X_{\mathcal{D}}$ for (k, z) -CLUSTERING and returns $S_{\mathcal{D}} \cup X_{\mathcal{S}}$ as the coreset for (k, z, m) -CLUSTERING of the dataset.

Algorithm 1 Coreset construction for (k, z, m) -CLUSTERING of X based on bounded partition

Require: a λ -bounded partition \mathcal{P} of X

- 1: let $\mathcal{D} \leftarrow \{P \in \mathcal{P} : |P| \geq (1 + \varepsilon^{-1})m\}$ and $\mathcal{S} \leftarrow \mathcal{P} \setminus \mathcal{D}$
 - 2: construct an ε -coreset $S_{\mathcal{D}}$ for (k, z) -CLUSTERING of $X_{\mathcal{D}} := \bigcup_{P \in \mathcal{D}} P$
 - 3: **return** $S_{\mathcal{D}} \cup X_{\mathcal{S}}$, where $X_{\mathcal{S}} := \bigcup_{P \in \mathcal{S}} P$
-

Lemma 3.3 (Vanilla coresets on dense datasets are robust). *For a dataset $X \subseteq V$, $0 < \varepsilon < 1$, and integers $k, z, m \geq 1$, if there exists a λ -bounded partition \mathcal{P} of X for some $\lambda \geq 0$ such that for every $P \in \mathcal{P}$, $|P| \geq (1 + \varepsilon^{-1})m$, then an ε -coreset of X for (k, z) -CLUSTERING is also an $(O(\varepsilon), 2^{O(z \log(z+1))} \cdot m\lambda^z)$ -coreset for (k, z, m) -CLUSTERING.*

Proof. **TOPROVE 0** □

The intuition behind Lemma 3.3 stems from the observation that, for every point x , there are at least $\Omega(\varepsilon^{-1}m)$ points lie in the same part with x and that these points are “close” to x (because of the λ -bounded property). This implies that the contribution for every x (for any center set C) is only roughly ε/m times the total cost of all points. Hence, removing up to m points as outliers only change the objective cost by $O(\varepsilon)$ times. This roughly means that the cost of outlier version and the vanilla version are very close. Since the vanilla coreset preserves the cost of the vanilla version, it naturally preserves the outlier version according to this observation.

Hence, if we are given a λ -bounded partition \mathcal{P} with sufficiently small λ , then $S_{\mathcal{D}}$ becomes an $O(\varepsilon)$ -coreset of $X_{\mathcal{D}}$ for (k, z, m) -CLUSTERING (with negligible additive error). By the composability of coresets and treating $X_{\mathcal{S}}$ as a naive coreset of itself, we conclude that $S = S_{\mathcal{D}} \cup X_{\mathcal{S}}$ is an adequately accurate coreset of $X = X_{\mathcal{D}} \cup X_{\mathcal{S}}$ for (k, z, m) -CLUSTERING.

Size Bound: Construction of λ -Bounded Partition As shown in Algorithm 1, the resulting coreset S contains $|S_{\mathcal{D}}| + |X_{\mathcal{S}}| = N(n, k, O(\varepsilon^{-1})) + |X_{\mathcal{S}}|$ points, where $|X_{\mathcal{S}}|$ is the number of points contained in sparse parts. To obtain an upper bound for the size of the coreset, we show in Lemma 3.4 an algorithm that finds an *almost-dense* λ -bounded partition (by “almost-dense” we mean the number of points in the sparse parts is small).

Lemma 3.4 (Almost-dense bounded partition). *Given $\lambda > 0$, a dataset $X \subseteq V$ and an (α, β, γ) -approximation solution C^* to (k, z, m) -CLUSTERING on X , one can construct in $\tilde{O}(\beta nk)$ time a 2λ -bounded partition \mathcal{P} of X . This partition can be decomposed into $\mathcal{P} = \mathcal{D} \cup \mathcal{S}$ with the following properties:*

1. (Dense parts) Every part in \mathcal{D} is dense, i.e., for every $P \in \mathcal{D}$, $|P| \geq (1 + \varepsilon^{-1})m$.
2. (Sparse parts) The number of sparse points is small, i.e., $|\bigcup_{P \in \mathcal{S}} P| \leq O(\beta km \varepsilon^{-1}) + \gamma m + \text{cost}_z^{(\gamma m)}(X, C^*) \cdot \lambda^{-z}$.

One may find the bound of the number of sparse points somewhat unnatural since it relates the number of points to the seemingly incomparable clustering objective. As mentioned earlier, in our applications we would set λ sufficiently small, say, $\lambda \leq \left((m^{-1} \text{cost}_z^{(\gamma m)}(X, C^*))^{1/z}\right)$, so that the $\text{cost}_z^{(\gamma m)}$ term is “canceled out”.

Proof. **TOPROVE 1** □

Proof of Theorem 3.1 Recall that C^* is a given (α, β, γ) -approximation solution to (k, z, m) -CLUSTERING on X , where $\alpha = 2^{O(z)}$ and $\beta = \gamma = O(1)$. Let $\lambda := (z + 1)^{-\xi} \cdot \left(\frac{\varepsilon \cdot \text{cost}_z^{(\gamma m)}(X, C^*)}{\alpha m}\right)^{1/z}$ for sufficiently large constant $\xi > 0$. We first apply Lemma 3.4 to compute a λ -bounded partition $\mathcal{P} = \mathcal{D} \cup \mathcal{S}$ of the dataset X , and then apply Algorithm 1 to this partition \mathcal{P} , which outputs a weighted set $S = S_{\mathcal{D}} \cup X_{\mathcal{S}}$. Guaranteed by Lemma 3.3, $S_{\mathcal{D}}$ is an $(O(\varepsilon), \eta)$ -coreset of $X_{\mathcal{D}}$ for (k, z, m) -CLUSTERING with additive error

$$\eta \leq 2^{O(z \log(z+1))} \cdot m \lambda^z \leq \varepsilon \cdot \alpha^{-1} \cdot \text{cost}_z^{(\gamma m)}(X, C^*) \leq \varepsilon \text{OPT}_z^{(m)}(X).$$

Since $X_{\mathcal{S}}$ is an ε -coreset of itself for (k, z, m) -CLUSTERING, by composability (Fact 2.3), we have that $S = S_{\mathcal{D}} \cup X_{\mathcal{S}}$ is an $(O(\varepsilon), \eta)$ -coreset of X for (k, z, m) -CLUSTERING. Then, consider any k -point center set $C \subseteq V$ and any $0 \leq h \leq m$, we have

$$\left| \text{cost}_z^{(h)}(X, C) - \text{cost}_z^{(h)}(S, C) \right| \leq O(\varepsilon) \cdot \text{cost}_z^{(h)}(X, C) + \varepsilon \cdot \text{OPT}_z^{(m)}(X) \leq O(\varepsilon) \cdot \text{cost}_z^{(h)}(X, C),$$

hence certifying that S is an $O(\varepsilon)$ -coreset of X for (k, z, m) -CLUSTERING. According to Lemma 3.4, the coreset size is $N(n, k, \varepsilon^{-1}) + |X_{\mathcal{S}}| \leq N(n, k, \varepsilon^{-1}) + O(\gamma + \beta k \varepsilon^{-1} + 2^{O(z \log z)} \alpha \varepsilon^{-1}) \cdot m = N(n, k, \varepsilon^{-1}) + 2^{O(z \log z)} \cdot O(k m \varepsilon^{-1})$. Regarding the runtime, our construction is merely the combination of the computation of a bounded partition (e.g., algorithm of Lemma 3.4) and Algorithm 1 (with the bottleneck of runtime being the vanilla coreset construction). Hence, the total running time is $\tilde{O}(nk) + T(n, k, \varepsilon^{-1})$. To finish the proof, it remains to scale ε . □

3.1 Vanilla Coresets on Dense Datasets Are Robust: Proof of Lemma 3.3

In this section, we prove Lemma 3.3, which demonstrates that a vanilla coreset on a dense dataset is also a robust coreset. As mentioned earlier, the proof is based on the observation that the contribution of every point is negligible. We formulate this in the following technical lemma.

Lemma 3.5. *For a dataset $X \subseteq V$, if there exists a λ -partition \mathcal{P} of X such that for every $P \in \mathcal{P}$, $|P| \geq (1 + \varepsilon^{-1})m$, then for any subset $C \subset V$ and for every $x \in X$, it holds that*

$$\text{dist}^z(x, C) \leq \frac{2\varepsilon}{m} \cdot \text{cost}_z^{(m)}(X, C) + (3z\lambda)^z.$$

Proof. TOPROVE 2 □

Proof. TOPROVE 3 □

4 Reduction II: Size-preserving Property

We present an alternative reduction in Theorem 4.2. The main difference to Theorem 3.1 is to avoid the multiplication of k and m in the coreset size. This reduction requires a slightly stronger technical guarantee from the vanilla coreset algorithm, such that the algorithm not only needs to construct a small coreset for metric M that contains the dataset, but also for a family of *separated duplications* of M . Roughly speaking, a separated duplication of M “copies” M into several disjoint sub-metrics, and makes the distance between two copies large. We provide the formal definition below.

Definition 4.1 (Seperated duplication of a metric space). Given a metric space $M = (V, \text{dist})$, for real number $w \geq 0$ and integer $h \geq 1$, a metric space $M' = (V \times [h], \text{dist}')$ is called an w -separated h -duplication of M , if

1. $\forall x, y \in X, i \in [h]$, it holds that $\text{dist}'((x, i), (y, i)) = \text{dist}(x, y)$; and
2. $\forall x, y \in X, i, j \in [h]$ such that $i \neq j$, it holds that $\text{dist}'((x, i), (y, j)) \geq \max\{w, \text{dist}(x, y)\}$.

For the special case of $h = 1$ and $w = 0$, we say M' is a w -separated h -duplication of M if and only if $M' = M$.

Theorem 4.2. Suppose an underlying metric space $M = (V, \text{dist})$ and a family $\mathcal{M}^{\text{dup}} = \{M_{h,w}^{\text{dup}}\}_{h \geq 1, w \geq 0}$ of w -separated h -duplication of M are given. Assume there exists an algorithm \mathcal{A} such that, for every $0 < \varepsilon < 1$, integers $k, z \geq 1$, metric $M^{\text{dup}} \in \mathcal{M}^{\text{dup}}$ and n -point dataset from M^{dup} , it runs in time $T(M^{\text{dup}}, n, k, \varepsilon^{-1})$ to construct an ε -coreset of size $N(M^{\text{dup}}, n, k, \varepsilon^{-1})$ for (k, z) -CLUSTERING on M^{dup} . Then, there is an algorithm that, given $0 < \varepsilon < 1$, integers $k, z, m \geq 1$, an n -point dataset $X \subseteq V$ and a $(2^{O(z)}, O(1), O(1))$ -approximation $C^* \subseteq V$ to (k, z, m) -CLUSTERING on X as input, constructs ε -coreset for (k, z, m) -CLUSTERING of size

$$A + N(M, n, k, O(\varepsilon^{-1})) + N\left(M_{h',w'}^{\text{dup}}, O(km\varepsilon^{-1}), O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1})\right),$$

where $A = 2^{O(z \log z)} \cdot O(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1}))$, $h' = \min\{O(k \log(km\varepsilon^{-1})), n\}$ and $w' = O(z\varepsilon^{-1} \cdot \text{diam}(X) \cdot n^{1/z})$. This algorithm invokes \mathcal{A} a constant number of times and runs in time

$$\tilde{O}(nk) + \text{poly}(km\varepsilon^{-1}) + T(M, n, k, \varepsilon^{-1}) + T\left(M_{h',w'}^{\text{dup}}, h', O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1})\right).$$

Notice that we assume a family \mathcal{M}^{dup} is *given* alongside M in Theorem 4.2, since this family \mathcal{M}^{dup} may depend on the specific type of M (e.g., M is Euclidean or shortest-path metric of a graph) and may require some careful design. Luckily, in most cases the separated duplication family \mathcal{M}^{dup} needs not be much more “complex” than M , and this particularly means as long as a vanilla coreset works on M , it can also deal with metrics in \mathcal{M}^{dup} .

Specifically, our main claim is that for metrics including Euclidean spaces, general finite metrics, doubling metrics and shortest-path metrics of minor-excluded graphs, the complexity/dimension parameter (such as Euclidean dimension) of the separated duplication only increases by a constant (factor). For instance, for the d -dimension Euclidean metric $M = (\mathbb{R}^d, \ell_2)$, we show that for every

w and h there exists a w -separated h -duplication of M that can be realized by a $(d+1)$ -dimensional Euclidean space $(\mathbb{R}^{d+1}, \ell_2)$ (Lemma B.1). Hence, a vanilla coreset construction for those metrics automatically works for the separated duplication (with only a constant factor increase in coreset size). We summarize the results for the separated duplication in Table 2.

Table 2: Summary of the results for the separated duplication. For all cases of the original metric space M that we list, we demonstrate the existence of a family \mathcal{M}^{dup} , such that each $M^{\text{dup}} \in \mathcal{M}^{\text{dup}}$ possesses the property as shown in the table.

	space M	property of \mathcal{M}^{dup}	reference
	(\mathbb{R}^d, ℓ_p) for $p \geq 1$	can be realized by $(\mathbb{R}^{d+1}, \ell_p)$	Lemma B.1
general metric	doubling dimension $\text{ddim}(M)$	$\text{ddim}(M^{\text{dup}}) \leq 2 \text{ddim}(M) + 2$	Lemma B.3
	ambient size n	ambient size $\leq n^2$	Remark B.5
graph metric	treewidth tw	treewidth tw	Lemma B.6
	excluding a fixed minor H	excluding the same minor H	Lemma B.6

4.1 Proof of Theorem 4.2

Recall that in Theorem 3.1, the additive term $O_z(km\varepsilon^{-1})$ in the coreset size corresponds to the number of sparse points, i.e. X_S , since we directly add these points into the coreset. In this proof, we still follow the steps of Theorem 3.1, except that we use a more refined method to construct a coreset for X_S .

We do not attempt to leverage any structural property of X_S since it may be quite arbitrary. Instead, we suggest a new framework: we show in Lemma 4.3 that as long as a vanilla coreset construction algorithm additionally satisfies the “size-preserving” property, then it is as well a coreset for clustering with outliers. Of course, this size-preserving property is nontrivial, and we cannot simply assume a vanilla coreset algorithm to satisfy this property. Hence, another important step (described in Algorithm 2) is a general black-box reduction (albeit it requires the vanilla coreset algorithm also works for separated duplication of the underlying metric), that turns a generic vanilla coreset algorithm into the one that is size-preserving.

Actually, the abovementioned reduction steps work for a general dataset (not only for X_S). However, the caveat is that this reduction may lead to a $\text{poly log}(n)$ factor in the final coreset size where n is the size of the dataset. This is in general not acceptable, but luckily, since we only need to apply this reduction on X_S which has only $O(km\varepsilon^{-1})$ points, this poly log factor becomes negligible. Indeed, the only special property we use from X_S is that it has a small number of points.

Size-preserving (Vanilla) Coresets Are Robust We define a coreset (or more generally, a weighted set) $S \subseteq X$ to be size-preserving with a diameter bound λ if there exists a λ -bounded partition \mathcal{P} of X such that, for every $P \in \mathcal{P}$, $w_S(P \cap S) = |P|$. Similar to Lemma 3.3, we establish a relationship between the error of S for being a robust coreset and the diameter bound of the partition \mathcal{P} .

Lemma 4.3 (Size-preserving vanilla coresets are robust). *Suppose a metric $M = (V, \text{dist})$ is given. For $0 < \varepsilon < 1$, $\lambda > 0$, integers $k, z, m \geq 1$ and a dataset $X \subseteq V$, if a weighted set $S \subseteq X$ satisfies that there exists a λ -bounded partition \mathcal{P} of X such that*

1. S is an ε -coreset of X for $(k + |\mathcal{P}|, z)$ -CLUSTERING, and

$$2. \forall P \in \mathcal{P}, |P| = w_S(S \cap P),$$

then S is an $(O(\varepsilon), 2^{O(z \log(z+1))} \cdot m\lambda^z \varepsilon^{1-z})$ -coreset of X for (k, z, m) -CLUSTERING.

Proof. **TOPROVE 4** □

Reduction from Vanilla Coresets to Size-preserving Coresets We next show how to turn a vanilla coreset to a stronger vanilla coreset that also satisfies the size-preserving property. Our algorithm needs a metric decomposition called *sparse partition*, which is introduced by [JLN⁺05]. A sparse partition is a partition of the metric/dataset, such that each part has bounded diameter and the partition additionally satisfies a sparsity property. This sparsity property requires that any metric ball of small radius intersects only a few parts. We notice that this sparsity property is different from the previously mentioned sparsity as in the almost-dense decomposition, which instead requires that each part contains only a few points and is not about intersection.

We provide a formal definition and a specific version that we would use [JLN⁺05], restated using our language as follows.

Definition 4.4 (Sparse partition [JLN⁺05]). Given a metric space $M = (V, \text{dist})$ and a subset $X \subseteq V$, we say a partition \mathcal{P} of X is a (μ, Γ, Λ) -sparse partition if it holds that

- a) (Bounded diameter) for every part $P \in \mathcal{P}$, $\text{diam}(P) \leq \mu$; and
- b) (Sparsity) for every $x \in X$, the ball $\text{Ball}_X(x, \mu/\Gamma) = \{y \in X : \text{dist}(x, y) \leq \mu/\Gamma\}$ intersects at most Λ parts in \mathcal{P} , i.e., $|\{P \in \mathcal{P} : P \cap \text{Ball}_X(x, \mu/\Gamma) \neq \emptyset\}| \leq \Lambda$.

Theorem 4.5 ([JLN⁺05]). Given a metric space $M = (V, \text{dist})$, an n -point dataset $X \subseteq V$ and $\mu > 0$, there is a (μ, Γ, Λ) -sparse partition of X with $\Gamma = O(\log n)$, $\Lambda = O(\log n)$ that can be computed in time $\text{poly}(n)$.

We describe our algorithm in Algorithm 2, which takes as input an n -point dataset $X \subseteq V$, a real number $\mu > 0$ representing the diameter bound, and an integer $k' \geq 1$ specifying the number of centers that a coreset can handle. The algorithm first computes a sparse partition $\mathcal{Q} = (X_1, \dots, X_l)$ of X using Theorem 4.5. Then, it maps the dataset X into some $M_{l,w}^{\text{dup}} \in \mathcal{M}^{\text{dup}}$ with sufficiently large w to ensure that the parts of \mathcal{Q} are well-separated. This can be achieved by mapping each point $x \in X_i$ to (x, i) . Guaranteed by the separation property of $M_{l,w}^{\text{dup}}$, the distance between any pair of parts is at most w . We note that this well-separation property is crucial for ensuring the size-preserving property. Finally, the algorithm constructs a coreset for (k', z) -CLUSTERING on $M_{l,w}^{\text{dup}}$, and returns the pre-image of this coreset. We have the following lemma that demonstrates the correctness of Algorithm 2.

Lemma 4.6 (Correctness of Algorithm 2). If there exists a $\frac{\varepsilon\mu}{1000z\Gamma}$ -bounded partition of X of size t ($t \geq 1$) and k' satisfies that $k' \geq (k + t)\Lambda$, then Algorithm 2 returns a weighted set $S \subseteq X$ and a μ -bounded partition $\mathcal{Q} = (X_1, \dots, X_l)$ of X with $l \leq t\Lambda$ such that

- 1. S is an ε -coreset of X for (k, z) -CLUSTERING, and
- 2. $\forall i \in [l], w_S(S \cap X_i) \in (1 + \varepsilon) \cdot |X_i|$.

Proof. **TOPROVE 5** □

Algorithm 2 Reduction from size-preserving vanilla coresets to vanilla coresets

Require: an n -point dataset $X \subseteq V$, real number $\mu > 0$ and an integer $k' \geq 1$

- 1: compute a (μ, Γ, Λ) -sparse partition $\mathcal{Q} = (X_1, \dots, X_l)$ of X \triangleright use algorithm of Theorem 4.5
 - 2: for every $i \in [l]$, let $X'_i := \{(x, i) \in V \times [l] : x \in X_i\}$, and let $X' := \bigcup_{i=1}^l X'_i$
 - 3: construct an ε -coreset S' of X' for (k', z) -CLUSTERING on metric $M_{l, 200z\varepsilon^{-1} \cdot \text{diam}(X) \cdot n^{1/z}}^{\text{dup}} \in \mathcal{M}^{\text{dup}}$
 \triangleright use the assumed algorithm as a black-box
 - 4: let $S := \{x \in X : \exists i \in [l], (x, i) \in S'\}$, and define $w_S : S \rightarrow \mathbb{R}_{\geq 0}$ such that $\forall (x, i) \in S, w_S(x) = w_{S'}((x, i))$
 - 5: **return** (S, w_S) and the partition \mathcal{Q}
-

We note that the size-preserving property guaranteed by Lemma 4.6 is actually weaker than what we need, as it only approximately preserves the sizes. However, we argue that this is still sufficient because we can calibrate the weights of this coreset, which only affects the accuracy by a factor of $1 + \varepsilon$.

As shown in Lemma 4.6, the lower bound of k' relies on two factors: the minimum size of the bounded partition of X and the parameter Λ of the sparse partition \mathcal{Q} . Therefore, it is essential for both of these factors to be as small as possible. According to Lemma 3.4, an $O(k)$ -sized bounded partition exists for the majority of points. This suggests applying Algorithm 2 only to this majority and directly incorporating the remaining points into the coreset. Regarding the parameter Λ , we assert that the result of Theorem 4.5, which provides a sparse partition with $\Lambda = O(\log n)$, is sufficient for our needs. This is because we will apply the algorithm to the small subset, say, of size $\text{poly}(km\varepsilon^{-1})$, rendering such $\log n$ factor negligible.

Concluding Theorem 4.2 Recall that we are given an (α, β, γ) -approximation C^* to (k, z, m) -CLUSTERING on X , where $\alpha = 2^{O(z)}, \beta = \gamma = O(1)$. We first apply Lemma 3.4 to X with the diameter bound

$$\lambda := (z + 1)^{-\xi} \cdot \varepsilon^2 \cdot \left(\frac{\text{cost}_z^{(\gamma m)}(X, C^*)}{\alpha m} \right)^{1/z} \cdot (\log(km\varepsilon^{-1}))^{-1}$$

for sufficiently large constant $\xi > 0$. Let $\mathcal{P} = \mathcal{D} \cup \mathcal{S}$ denote the λ -bounded partition computed by the algorithm of Lemma 3.4, where for every $P \in \mathcal{D}$, $|P| \geq (1 + \varepsilon^{-1})m$. Let $X_{\mathcal{D}} := \bigcup_{P \in \mathcal{D}} P$. We construct an ε -coreset $S_{\mathcal{D}}$ of $X_{\mathcal{D}}$ for (k, z) -CLUSTERING on the original metric space M using the assumed algorithm as a black-box. Guaranteed by Lemma 3.3, $S_{\mathcal{D}}$ is also an $(O(\varepsilon), O(\varepsilon) \cdot \text{OPT}_z^{(m)}(X))$ -coreset of $X_{\mathcal{D}}$ for (k, z, m) -CLUSTERING.

As for the sparse subsets \mathcal{S} , we further decompose it into \mathcal{S}_1 and \mathcal{S}_2 , where $|\mathcal{S}_1| \leq \beta k$ and \mathcal{S}_2 covers only a few points. Such a decomposition always exists. Moreover, we claim that the partition computed by Lemma 3.4 already provides this, as stated in the following claim.

Claim 4.7. *Given a metric $M = (V, \text{dist})$, a integer $\lambda > 0$, a dataset $X \subseteq V$ and an (α, β, γ) -approximation solution C^* to (k, z, m) -CLUSTERING of X , let $\mathcal{P} = \mathcal{D} \cup \mathcal{S}$ be a λ -bounded partition computed using Lemma 3.4, the sparse subsets \mathcal{S} can be further decomposed into \mathcal{S}_1 and \mathcal{S}_2 such that $|\mathcal{D}| + |\mathcal{S}_1| \leq \beta k$ and $|\bigcup_{P \in \mathcal{S}_2} P| \leq \gamma m + \text{cost}_z^{(\gamma m)}(X, C^*) \cdot \lambda^{-z}$.*

The claim follows directly from the construction and the size analysis of \mathcal{S} in the proof of Lemma 3.4. Provided by this decomposition, let $X_{\mathcal{S},1} := \bigcup_{P \in \mathcal{S}_1} P$ and $X_{\mathcal{S},2} := \bigcup_{P \in \mathcal{S}_2} P$, we have $|X_{\mathcal{S},1}| \leq O(km\varepsilon^{-1})$ and $X_{\mathcal{S},1}$ admits a λ -bounded partition of size at most βk .

Hence, we apply Algorithm 2 to $X_{S,1}$ with $\mu = 1000\varepsilon^{-1}z\Gamma\lambda$ and $k' = (k + \beta k\Lambda + \beta k)\Lambda$, where $\Gamma = \Lambda = O(\log |X_{S,1}|) = O(\log(km\varepsilon^{-1}))$ as stated in Theorem 4.5. By Lemma 4.6, Algorithm 2 returns a weighted set $S \subseteq X_{S,1}$ and a μ -bounded partition \mathcal{Q} of X with $|\mathcal{Q}| \leq \beta k\Lambda$ such that S is an ε -coreset of $X_{S,1}$ for $(k + \beta k\Lambda, z)$ -CLUSTERING, and for all $Q \in \mathcal{Q}$, $w_S(S \cap Q) \in (1 + \varepsilon) \cdot |Q|$.

We calibrate the weight of S to meet the exact size-preserving requirement of Lemma 4.3 as follows: we define a new weight function $w'_S : S \rightarrow \mathbb{R}_{\geq 0}$ such that, for every $Q \in \mathcal{Q}$ and every $x \in S \cap Q$, $w'_S(x) = w_S(x) \cdot \frac{|Q|}{w_S(S \cap Q)}$. It is easy to verify that the weighted set (S, w'_S) is size-preserving with respect to \mathcal{Q} . Furthermore, $w_S(S \cap Q) \in (1 \pm \varepsilon) \cdot |Q|$ implies that for every $x \in Q$, $w'_S(x) \in (1 \pm O(\varepsilon)) \cdot w_S(x)$. Therefore, for every $C \in V^k$, we have

$$\sum_{x \in S} w'_S(x) \cdot \text{dist}^z(x, C) \in (1 \pm O(\varepsilon)) \cdot \sum_{x \in S} w_S(x) \cdot \text{dist}^z(x, C).$$

Since $\sum_{x \in S} w_S(x) \cdot \text{dist}^z(x, C) \in (1 \pm \varepsilon) \cdot \text{cost}_z(X_{S,1}, C)$ due to the fact that (S, w_S) is an ε -coreset, we conclude that (S, w'_S) is an $O(\varepsilon)$ -coreset of $X_{S,1}$ for (k, z) -CLUSTERING.

We then use the function w'_S to weight S instead of using w_S . Therefore, S is an ε -coreset for $(k + |\mathcal{Q}|, z)$ -CLUSTERING that is size-preserving with respect to \mathcal{Q} . By Lemma 4.3, S is an $(O(\varepsilon), 2^{O(z \log(z+1))} \cdot m\mu^z \varepsilon^{1-z})$ -coreset of $X_{S,1}$ for (k, z, m) -CLUSTERING, where the additive error is bounded by

$$2^{O(z \log(z+1))} \cdot m\mu^z \varepsilon^{1-z} = 2^{O(z \log(z+1))} \cdot m\varepsilon^{1-2z} \cdot \lambda^z \cdot \log^z(km\varepsilon^{-1}) \leq \varepsilon \cdot \text{OPT}_z^{(m)}(X),$$

since $\lambda \leq (z+1)^{-\xi} \cdot \varepsilon^2 \cdot \left(\frac{\text{OPT}_z^{(m)}(X)}{m} \right)^{1/z} \cdot (\log(km\varepsilon^{-1}))^{-1}$ with $\xi > 0$ being a sufficiently large constant.

Finally, by the composability of coresets (see Fact 2.3), $S_{\mathcal{D}} \cup S \cup X_{S,2}$ is an $O(\varepsilon)$ -coreset of X for (k, z, m) -CLUSTERING, and the coreset size is

$$N(M, n, k, O(\varepsilon^{-1})) + N\left(M_{l, 200z\varepsilon^{-1} \cdot \text{diam}(X) \cdot n^{1/z}}^{\text{dup}}, |X_{S,1}|, k', O(\varepsilon^{-1})\right) + |X_{S,2}|,$$

where $|X_{S,1}| = O(km\varepsilon^{-1})$ and $k' = (k + \beta k\Lambda + \beta k)\Lambda = O(k \log^2(km\varepsilon^{-1}))$. By Claim 4.7, the set $X_{S,2}$ has a size of

$$\gamma m + \text{cost}_z^{(\gamma m)}(X, C^*) \cdot \lambda^z \leq 2^{O(z \log z)} \cdot O(m\varepsilon^{-2z} \cdot \log^z(km\varepsilon^{-1})).$$

Regarding the running time, the construction is a continuation of that of Theorem 3.1, and the additional runtime is primarily due to the invocation of the vanilla coreset construction on $M_{l, 200z\varepsilon^{-1} \cdot \text{diam}(X) \cdot n^{1/z}}^{\text{dup}}$. Hence, the time complexity shown in Theorem 4.2 follows.

It remains to scale ε by a constant. □

4.2 Size-preserving Vanilla Coresets Are Robust: Proof of Lemma 4.3

Lemma 4.3 (Size-preserving vanilla coresets are robust). *Suppose a metric $M = (V, \text{dist})$ is given. For $0 < \varepsilon < 1$, $\lambda > 0$, integers $k, z, m \geq 1$ and a dataset $X \subseteq V$, if a weighted set $S \subseteq X$ satisfies that there exists a λ -bounded partition \mathcal{P} of X such that*

1. S is an ε -coreset of X for $(k + |\mathcal{P}|, z)$ -CLUSTERING, and
2. $\forall P \in \mathcal{P}, |P| = w_S(S \cap P)$,

then S is an $(O(\varepsilon), 2^{O(z \log(z+1))} \cdot m\lambda^z \varepsilon^{1-z})$ -coreset of X for (k, z, m) -CLUSTERING.

Let $\mathcal{P} = (X_1, \dots, X_t)$ for some $1 \leq t \leq n$ be a λ -bounded satisfying the premise of Lemma 4.3. For every $i \in [t]$, let $S_i := S \cap X_i$ be a weighted set with weight function w_{S_i} such that for every $x \in S_i$, $w_{S_i}(x) = w_S(x)$. Recall that the weighted set S preserves the size of each part, so we have $w_{S_i}(S_i) = |X_i|$ for every $i \in [t]$.

The proof begins with a sufficient condition for S to be a coreset for (k, z, m) -CLUSTERING, as stated in the following claim. This claim is standard, and the proof can be found in the literature of coresets for robust clustering (e.g., [HJLW23, HLLW25]). For completeness, we provide a proof in Section A.

Lemma 4.8. *For some $0 < \varepsilon < 1$ and $\eta > 0$, if for all $C \in V^k$ and real numbers $h_1, \dots, h_t \geq 0$ with $\sum_{i=1}^t h_i \leq m$, it holds that*

$$\left| \sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) - \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) \right| \leq \varepsilon \cdot \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) + \eta, \quad (4)$$

then S is an (ε, η) -coreset (see Definition 2.2) of X for (k, z, m) -CLUSTERING.

Therefore, we consider a fixed center set $C \in V^k$ and fixed $h_1, \dots, h_t \geq 0$ with $\sum_{i=1}^t h_i \leq m$ in the following, and our goal is to show that

$$\begin{aligned} & \left| \sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) - \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) \right| \\ & \leq O(\varepsilon) \cdot \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) + 2^{O(z \log(z+1))} \cdot m \lambda^z \varepsilon^{1-z}. \end{aligned} \quad (5)$$

If this is true, then the proof is finished by applying Lemma 4.8. To prove (5), the key idea is to find an auxiliary center set C^{aux} such that

$$|\text{cost}_z(X, C \cup C^{\text{aux}}) - \text{cost}_z(S, C \cup C^{\text{aux}})| \leq \eta_1 \quad (6)$$

and

$$\left| \left(\sum_{i=1}^t \text{cost}_z^{(h_i)}(X_i, C) - \text{cost}_z(X, C \cup C^{\text{aux}}) \right) - \left(\sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) - \text{cost}_z(S, C \cup C^{\text{aux}}) \right) \right| \leq \eta_2 \quad (7)$$

with $\eta_1 + \eta_2 \leq O(\varepsilon) \cdot \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) + 2^{O(z \log(z+1))} \cdot m \lambda^z \varepsilon^{1-z}$, then (5) follows from triangle inequality. Hence, in the following, we focus on showing the existence of such C^{aux} .

Construction of Auxiliary Center Set C^{aux} We explicitly provide the construction of C^{aux} . The construction consists of two steps: we first find a (weighted) set of “significant” outliers, denoted by Z , and then we define the auxiliary center set C^{aux} as a λ -covering of Z , which satisfies that $\forall x \in Z$, $\text{dist}(x, C^{\text{aux}}) \leq \lambda$.

To find Z , we first identify the outliers and inliers of each X_i with respect to C and h_i . Specifically, let $X_{\text{out}} \subseteq X$ with weight function $w_X^{\text{out}} : X_{\text{out}} \rightarrow \mathbb{R}_{\geq 0}$ be an outlier set such that for every $i \in [t]$, $X_{\text{out}} \cap X_i$ is the h_i -outlier set of X_i , i.e., $w_X^{\text{out}}(X_{\text{out}} \cap X_i) = h_i$ and $\text{cost}_z(X_i, C) - \text{cost}_z(X_{\text{out}} \cap X_i, C) = \text{cost}_z^{(h_i)}(X_i, C)$. Moreover, let weighted set $X_{\text{in}} := X - X_{\text{out}}$ with weight function $w_X^{\text{in}} = w_X - w_X^{\text{out}}$ (recalling that $w_X \equiv 1$ for unweighted X) be the inlier set. We find the significant outliers Z among X_{out} via the process described in Algorithm 3.

According to the process of Algorithm 3, we first have the following fact regarding the weighted sets U , Z , and the values $a_{p,q}$.

Algorithm 3 Finding significant outliers Z

```
1: initialize  $(U, w_U) \leftarrow (X_{\text{in}}, w_X^{\text{in}})$ ,  $(Z, w_Z) \leftarrow (X_{\text{out}}, w_X^{\text{out}})$ 
2: initialize  $a_{q,p} \leftarrow 0$  for every  $q \in X_{\text{out}}, p \in X_{\text{in}}$ 
3: while  $\exists q \in Z, p \in U$  s.t.  $\text{dist}(p, q) \leq \text{dist}(p, C) + 4\lambda$  do
4:   let  $a_{q,p} \leftarrow \min\{w_Z(q), w_U(p)\}$   $\triangleright p$  eliminates a fraction  $a_{q,p}$  of  $q$ 
5:   let  $w_U(p) \leftarrow w_U(p) - a_{q,p}$  and  $w_Z(q) \leftarrow w_Z(q) - a_{q,p}$ 
6:   if  $w_U(p) = 0$  then
7:     let  $U \leftarrow U \setminus \{p\}$ 
8:   end if
9:   if  $w_Z(q) = 0$  then
10:    let  $Z \leftarrow Z \setminus \{q\}$ 
11:  end if
12: end while
13: return  $Z$ 
```

Fact 4.9. *The following holds after Algorithm 3 terminates:*

1. For every $q \in X_{\text{out}}$, $\sum_{p \in X_{\text{in}}} a_{q,p} = w_X^{\text{out}}(q) - w_Z(q)$, and therefore $\sum_{q \in X_{\text{out}}} \sum_{p \in X_{\text{in}}} a_{q,p} \leq w_X^{\text{out}}(X_{\text{out}}) \leq m$.
2. For every $p \in X_{\text{in}}$, we have $\sum_{q \in X_{\text{out}}} a_{q,p} = w_X^{\text{in}}(p) - w_U(p) \leq w_X^{\text{in}}(p)$.

Then, we define the auxiliary center set $C^{\text{aux}} \subseteq Z$ to be a λ -covering of Z of size at most t . The existence of C^{aux} is presented in the following lemma.

Lemma 4.10. *There exists a set $C^{\text{aux}} \subseteq Z$ with $|C^{\text{aux}}| \leq t$ such that C^{aux} is a λ -covering of Z , i.e., for every $x \in Z$, $\text{dist}(x, C^{\text{aux}}) \leq \lambda$.*

Proof. **TOPROVE 6** □

Before we proceed to prove (6) and (7), we establish some lemmas regarding significant outliers Z and the remaining outliers $X_{\text{out}} - Z$, which we call insignificant outliers. Notice that $X_{\text{out}} - Z$ are the outliers eliminated by Algorithm 3. For each outlier $q \in X_{\text{out}}$, we have recorded a value $a_{q,p}$, which represents the fraction of q that is eliminated by p . We demonstrate that the contribution to $\text{cost}_z(X_{\text{out}}, C)$ from the eliminated fraction of an outlier q is comparable to the contribution from the inliers eliminating it, leading to a small $\text{cost}_z(X_{\text{out}} - Z, C)$.

Lemma 4.11 (Cost of non-significant outliers). *It holds that*

$$\text{cost}_z(X_{\text{out}} - Z, C) \leq O(1) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \lambda^z.$$

Proof. **TOPROVE 7** □

For significant outliers, it is challenging to establish an upper bound on their cost, as they might be significantly far away from inliers. However, from another perspective, this implies that if we place a center on a significant outlier (or its λ -covering), the center will “absorb” only a few inliers from the original center set C . We formalize this observation in the following lemma, where we actually upper-bound the size of a larger set that contains the inliers absorbed by C^{aux} .

Lemma 4.12 (C^{aux} absorbs only a few inliers). *Let $A := \{x \in X_{\text{in}} : \text{dist}(x, C^{\text{aux}}) \leq \text{dist}(x, C) + 4\lambda\}$, then it holds that $w_X^{\text{in}}(A) \leq m$.*

Proof. **TOPROVE 8** □

Upper Bound for (6) Recall that S is an ε -coreset of X for $(k + t, z)$ -CLUSTERING, and Lemma 4.10 implies that $|C \cup C^{\text{aux}}| \leq k + t$. We have

$$\begin{aligned}
& |\text{cost}_z(S, C \cup C^{\text{aux}}) - \text{cost}_z(X, C \cup C^{\text{aux}})| \\
& \leq \varepsilon \cdot \text{cost}_z(X, C \cup C^{\text{aux}}) \\
& = \varepsilon \cdot (\text{cost}_z(X_{\text{in}}, C \cup C^{\text{aux}}) + \text{cost}_z(X_{\text{out}}, C \cup C^{\text{aux}})) \\
& \leq \varepsilon \cdot (\text{cost}_z(X_{\text{in}}, C) + \text{cost}_z(Z, C \cup C^{\text{aux}}) + \text{cost}_z(X_{\text{out}} - Z, C \cup C^{\text{aux}}))
\end{aligned} \tag{8}$$

where the last inequality is due to $\text{cost}_z(X_{\text{in}}, C \cup C^{\text{aux}}) \leq \text{cost}_z(X_{\text{in}}, C)$ and $\text{cost}_z(X_{\text{out}}, C \cup C^{\text{aux}}) = \text{cost}_z(Z, C \cup C^{\text{aux}}) + \text{cost}_z(X_{\text{out}} - Z, C \cup C^{\text{aux}})$ since $(Z, w_Z) \subseteq (X_{\text{out}}, w_X^{\text{out}})$.

For $\text{cost}_z(Z, C \cup C^{\text{aux}})$, we use the fact that C^{aux} is a λ -covering of Z (see Lemma 4.10), and thus we have

$$\text{cost}_z(Z, C \cup C^{\text{aux}}) \leq \text{cost}_z(Z, C^{\text{aux}}) \leq w_Z(Z) \cdot \lambda^z \leq m \cdot \lambda^z. \tag{9}$$

For $\text{cost}_z(X_{\text{out}} - Z, C \cup C^{\text{aux}})$, which is the cost of insignificant outliers, we have established an upper bound for it in Lemma 4.11, specifically,

$$\text{cost}_z(X_{\text{out}} - Z, C \cup C^{\text{aux}}) \leq \text{cost}_z(X_{\text{out}} - Z, C) \leq O(1) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \lambda^z. \tag{10}$$

By plugging (9) and (10) into (8), we obtain the following result for (6).

$$|\text{cost}_z(X, C \cup C^{\text{aux}}) - \text{cost}_z(S, C \cup C^{\text{aux}})| \leq O(\varepsilon) \cdot (\text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \lambda^z). \tag{11}$$

Upper Bound for (7) Let $(S_{\text{out}}, w_S^{\text{out}}) \subseteq (S, w_S)$ be an outlier set such that for every $i \in [t]$, $w_S^{\text{out}}(S_{\text{out}} \cap X_i) = h_i$, and $\text{cost}_z(S_i, C) - \text{cost}_z(S_{\text{out}}, C) = \text{cost}_z^{(h_i)}(S_i, C)$. Let weighted set $S_{\text{in}} := S - S_{\text{out}}$ with weight function $w_S^{\text{in}} = w_S - w_S^{\text{out}}$ be the inlier set. By this definition and the definition of X_{in} , we have

$$\sum_{i=1}^t \text{cost}_z^{(h_i)}(X_i, C) = \text{cost}_z(X_{\text{in}}, C), \quad \sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) = \text{cost}_z(S_{\text{in}}, C).$$

Therefore, (7) is equal to

$$\begin{aligned}
& |(\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(X, C \cup C^{\text{aux}})) - (\text{cost}_z(S_{\text{in}}, C) - \text{cost}_z(S, C \cup C^{\text{aux}}))| \\
& \leq |(\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(X_{\text{in}}, C \cup C^{\text{aux}})) - (\text{cost}_z(S_{\text{in}}, C) - \text{cost}_z(S_{\text{in}}, C \cup C^{\text{aux}}))| \\
& \quad + |\text{cost}_z(X_{\text{out}}, C \cup C^{\text{aux}}) - \text{cost}_z(S_{\text{out}}, C \cup C^{\text{aux}})|
\end{aligned} \tag{12}$$

by using the fact that $\text{cost}_z(X, C \cup C^{\text{aux}}) = \text{cost}_z(X_{\text{in}}, C \cup C^{\text{aux}}) + \text{cost}_z(X_{\text{out}}, C \cup C^{\text{aux}})$ and $\text{cost}_z(S, C \cup C^{\text{aux}}) = \text{cost}_z(S_{\text{in}}, C \cup C^{\text{aux}}) + \text{cost}_z(S_{\text{out}}, C \cup C^{\text{aux}})$, along with the triangle inequality. We separately upper-bound the two terms.

Lemma 4.13 (Error Analysis for the First Term). *It holds that*

$$\begin{aligned}
& |(\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(X_{\text{in}}, C \cup C^{\text{aux}})) - (\text{cost}_z(S_{\text{in}}, C) - \text{cost}_z(S_{\text{in}}, C \cup C^{\text{aux}}))| \\
& \leq O(\varepsilon) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \varepsilon^{1-z} \cdot \lambda^z
\end{aligned}$$

Proof. **TOPROVE 9** □

Lemma 4.14 (Error Analysis for the Second Term). *It holds that*

$$|\text{cost}_z(X_{\text{out}}, C \cup C^{\text{aux}}) - \text{cost}_z(S_{\text{out}}, C \cup C^{\text{aux}})| \leq O(\varepsilon) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \varepsilon^{1-z} \cdot \lambda^z.$$

Proof. **TOPROVE 10** □

By plugging the bounds from Lemma 4.13 and Lemma 4.14 into (12), we obtain

$$\begin{aligned} & |(\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(X, C \cup C^{\text{aux}})) - (\text{cost}_z(S_{\text{in}}, C) - \text{cost}_z(S, C \cup C^{\text{aux}}))| \\ & \leq O(\varepsilon) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \varepsilon^{1-z} \cdot \lambda^z. \end{aligned} \quad (13)$$

Concluding Lemma 4.3 By combining (11), (13) and the triangle inequality, we have

$$\begin{aligned} & \left| \sum_{i=1}^t \text{cost}_z^{(h_i)}(X_i, C) - \sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) \right| \\ & = |\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(S_{\text{in}}, C)| \\ & \leq |(\text{cost}_z(X_{\text{in}}, C) - \text{cost}_z(X, C \cup C^{\text{aux}})) - (\text{cost}_z(S_{\text{in}}, C) - \text{cost}_z(S, C \cup C^{\text{aux}}))| \\ & \quad + |\text{cost}_z(X, C \cup C^{\text{aux}}) - \text{cost}_z(S, C \cup C^{\text{aux}})| \\ & \leq O(\varepsilon) \cdot \text{cost}_z(X_{\text{in}}, C) + 2^{O(z \log(z+1))} \cdot m \cdot \varepsilon^{1-z} \cdot \lambda^z \\ & = O(\varepsilon) \cdot \sum_{i=1}^t \text{cost}_z^{(h_i)}(X_i, C) + 2^{O(z \log(z+1))} \cdot m \cdot \varepsilon^{1-z} \cdot \lambda^z. \end{aligned}$$

Since above inequality holds for all $C \in V^k$ and all real numbers $h_1, \dots, h_t \geq 0$ with $\sum_{i=1}^t h_i \leq m$, we have proved that S satisfies the condition of Lemma 4.8, with $\eta = 2^{O(z \log(z+1))} \cdot m \varepsilon^{1-z} \lambda^z$. Therefore, we conclude that S is an $(O(\varepsilon), 2^{O(z \log(z+1))} \cdot m \varepsilon^{1-z} \lambda^z)$ -coreset of X for (k, z, m) -CLUSTERING. □

4.3 Correctness of Algorithm 2: Proof of Lemma 4.6

Lemma 4.6 (Correctness of Algorithm 2). *If there exists a $\frac{\varepsilon\mu}{1000z\Gamma}$ -bounded partition of X of size t ($t \geq 1$) and k' satisfies that $k' \geq (k+t)\Lambda$, then Algorithm 2 returns a weighted set $S \subseteq X$ and a μ -bounded partition $\mathcal{Q} = (X_1, \dots, X_l)$ of X with $l \leq t\Lambda$ such that*

1. S is an ε -coreset of X for (k, z) -CLUSTERING, and
2. $\forall i \in [l], w_S(S \cap X_i) \in (1 + \varepsilon) \cdot |X_i|$.

Write $\mathcal{P} = (P_1, \dots, P_t)$ where \mathcal{P} is a λ -bounded partition for $\lambda := \frac{\varepsilon\mu}{1000z\Gamma}$, and let $k' \geq (k+t)\Lambda$ be an integer, as in the premise. Let $M_{l,w}^{\text{dup}} = (V \times [l], \text{dist}_{l,w})$ be the w -separated l -duplication of M (see Definition 4.1) used in Line 2 of Algorithm 2, where $w = 200z\varepsilon^{-1} \cdot \text{diam}(X) \cdot n^{1/z}$.

We first have the following claim that bounds the size of \mathcal{Q} according to the sparsity property of \mathcal{Q} .

Claim 4.15. *It holds that $|\mathcal{Q}| = l \leq t\Lambda$.*

Proof. **TOPROVE 11** □

Then, we separately prove the two properties of S in the following.

Size-preserving Property of S Notice that for every $i \in [l]$, $|X_i| = |X'_i|$ and $w_S(S \cap X_i) = w_{S'}(S' \cap X'_i)$. Here, S' is an ε -coreset of X' for (k', z) -CLUSTERING on the metric space $M_{l,w}^{\text{dup}}$, constructed in Line 3 of Algorithm 2. Therefore, we aim to prove the size-preserving property of S' , i.e., $\forall i \in [l]$, $w_{S'}(S' \cap X'_i) \in (1 \pm \varepsilon) \cdot |X'_i|$, which is equivalent to $w_S(S \cap X_i) \in (1 \pm \varepsilon) \cdot |X_i|$.

For every $i \in [l]$, we pick an arbitrary point $x_i \in X_i$ and let $x'_i := (x_i, i) \in X'_i$. Recall that $w \geq \text{diam}(X)$, the separation property of $M_{l,w}^{\text{dup}} = (V \times [l], \text{dist}_{l,w})$ (see Definition 4.1) implies the following fact.

Fact 4.16. *For every $i \in [l]$ and every $p \in X_i$, let $p' := (p, i) \in X'_i$. Then it holds that $\min_{j \in [l]} \text{dist}_{l,w}(p', x'_j) = \text{dist}_{l,w}(p', x'_i) = \text{dist}(p, x_i)$.*

Then, for a fixed index $i \in [l]$, we consider the center set $C := \{x'_j : j \in [l], j \neq i\} \subseteq V \times [l]$, which contains all x'_j for $j \in [l]$ except x'_i . By Fact 4.16, we have

$$\begin{aligned} \text{cost}_z(X', C) &= \sum_{j \in [l] \setminus \{i\}} \sum_{p' \in X'_j} \text{dist}_{l,w}^z(p', x'_j) + \sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C) \\ &= \sum_{j \in [l] \setminus \{i\}} \sum_{p \in X_j} \text{dist}^z(p, x_j) + \sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C) \end{aligned}$$

For the first term, we have $\sum_{j \in [l] \setminus \{i\}} \sum_{p \in X_j} \text{dist}^z(p, x_j) \leq n \cdot (\text{diam}(X))^z \leq \varepsilon \cdot w^z / 200$. For the term $\sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C)$, we have $\sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C) \geq w^z$. Hence,

$$\text{cost}_z(X', C) \in (1 \pm \varepsilon / 100) \sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C). \quad (14)$$

Moreover, for every $p' = (p, i) \in X'_i$, by the triangle inequality, we have

$$|\text{dist}_{l,w}(p', C) - \text{dist}_{l,w}(x'_i, C)| \leq \text{dist}_{l,w}(p', x'_i) \leq \text{diam}(X) \leq \frac{\varepsilon w}{200z} \leq \frac{\varepsilon}{200z} \cdot \text{dist}_{l,w}(x'_i, C).$$

As a result, we obtain $\text{dist}_{l,w}^z(p', C) \in (1 \pm \varepsilon / 50) \cdot \text{dist}_{l,w}^z(x'_i, C)$ using the fact that $1 - 2\alpha \leq (1 - \alpha/z)^z, (1 + \alpha/z)^z \leq 1 + 2\alpha$ for $\alpha \in (0, 1)$. Therefore, we have

$$\sum_{p' \in X'_i} \text{dist}_{l,w}^z(p', C) \in (1 \pm \varepsilon / 50) \cdot |X'_i| \cdot \text{dist}_{l,w}^z(x'_i, C).$$

Combined with (14), we have

$$\text{cost}_z(X', C) \in (1 \pm \varepsilon / 10) \cdot |X'_i| \cdot \text{dist}_{l,w}^z(x'_i, C). \quad (15)$$

For $\text{cost}_z(S', C)$, let $C^{\text{full}} := \{x'_j : j \in [l]\}$ and we have

$$\text{cost}_z(S', C) = \sum_{j \in [l] \setminus \{i\}} \sum_{p' \in S' \cap X'_j} w_{S'}(p') \cdot \text{dist}_{l,w}^z(p', x'_j) + \sum_{p' \in S' \cap X'_i} w_{S'}(p') \cdot \text{dist}_{l,w}^z(p', C).$$

The first term can be upper bounded by

$$\begin{aligned} \sum_{j \in [l] \setminus \{i\}} \sum_{p' \in S' \cap X'_j} w_{S'}(p') \cdot \text{dist}_{l,w}^z(p', x'_j) &\leq \sum_{j \in [l]} \sum_{p' \in S' \cap X'_j} w_{S'}(p') \cdot \text{dist}_{l,w}^z(p', x'_j) \\ &= \text{cost}_z(S', C^{\text{full}}) \\ &\leq 2 \text{cost}_z(X', C^{\text{full}}) \\ &\leq \varepsilon \cdot w^z / 100. \end{aligned}$$

where the second inequality is due to the coreset guarantee of S' , and the last inequality is due to $\text{cost}_z(X', C^{\text{full}}) \leq \varepsilon \cdot w^z/200$ as discussed above. Then, by a similar argument, we can obtain the same result for $\text{cost}_z(S', C)$:

$$\text{cost}_z(S', C) \in (1 \pm \varepsilon/10) \cdot w_{S'}(S' \cap X'_i) \cdot \text{dist}_{l,w}^z(x'_i, C). \quad (16)$$

Finally, since $|C| = l - 1$ and S' is an $\varepsilon/10$ -coreset for (k', z) -CLUSTERING with $k' = (t + k)\Lambda \geq l$ by Claim 4.15, we have $\text{cost}_z(S', C) \in (1 \pm \varepsilon/10) \cdot \text{cost}_z(X', C)$. Hence, we obtain that $w_{S'}(S' \cap X'_i) \in (1 \pm \varepsilon) \cdot |X'_i|$, which is equivalent to $w_S(S \cap X_i) \in (1 \pm \varepsilon) \cdot |X_i|$.

Coreset Guarantee of S Consider any center set $C \in V^k$. Our strategy is to construct a “bridge” center set $C' \subseteq V \times [l]$ such that

$$\text{cost}_z(X, C) \in (1 \pm \varepsilon/10) \cdot \text{cost}_z(X', C'), \quad \text{cost}_z(S, C) \in (1 \pm \varepsilon/10) \cdot \text{cost}_z(S', C'). \quad (17)$$

If there indeed exists such a C' with $|C'| \leq k' = (t + k)\Lambda$, then as the coreset guarantee of S' states that $\text{cost}_z(S', C') \in (1 \pm \varepsilon/10) \cdot \text{cost}_z(X', C')$, we conclude that $\text{cost}_z(X, C) \in (1 \pm \varepsilon) \cdot \text{cost}_z(S, C)$.

Hence, it remains to give the construction of the bridge center set, which utilizes the λ -bounded partition $\mathcal{P} = (P_1, \dots, P_t)$ (recalling that $\lambda = \frac{\varepsilon\mu}{1000z\Gamma}$). For every $j \in [t]$, we define $C_j^{\text{close}} := \{c \in C : \text{dist}(c, P_j) \leq \frac{\mu}{4\Gamma}\}$, $c_j^{\text{far}} := \arg \min_{c \in C \setminus C_j^{\text{close}}} \text{dist}(c, P_j)$, and let $C_j := C_j^{\text{close}} \cup \{c_j^{\text{far}}\}$. We have the following lemma.

Lemma 4.17. *For every $x \in P_j$, it holds that*

$$\text{dist}^z(x, C) \leq \text{dist}^z(x, C_j) \leq (1 + \varepsilon/50) \cdot \text{dist}^z(x, C).$$

Proof. **TOPROVE 12** □

Then, for every $i \in [l]$, let $D_i := \bigcup_{j \in [t]: P_j \cap X_i \neq \emptyset} C_j$. Now consider any $x \in X_i$ and let $j \in [t]$ be an integer such that $x \in P_j$. We have $P_j \cap X_i \neq \emptyset$, which means that $C_j \subseteq D_i$. Therefore, by Lemma 4.17, we have

$$\text{dist}^z(x, C) \leq \text{dist}^z(x, D_i) \leq \text{dist}^z(x, C_j) \leq (1 + \varepsilon/50) \cdot \text{dist}^z(x, C). \quad (18)$$

This implies that

$$\text{cost}_z(X, C) = \sum_{i=1}^l \text{cost}_z(X_i, C) \in (1 \pm \varepsilon/50) \cdot \sum_{i=1}^l \text{cost}_z(X_i, D_i), \quad (19)$$

and

$$\text{cost}_z(S, C) = \sum_{i=1}^l \text{cost}_z(S \cap X_i, C) \in (1 \pm \varepsilon/50) \cdot \sum_{i=1}^l \text{cost}_z(S \cap X_i, D_i). \quad (20)$$

Let $D'_i := \{(c, i) \in V \times [l] : c \in D_i\}$ and $C' := \bigcup_{i=1}^l D'_i$ for every $i \in [l]$. For every $i, j \in [l]$ with $i \neq j$, the separation property of $M_{l,w}^{\text{dup}}$ (see Definition 4.1) implies that

$$\text{cost}_z(X'_i, D'_j) \geq \text{cost}_z(X_i, D_j) \geq \text{cost}_z(X_i, C).$$

By (18), we further deduce that

$$\text{cost}_z(X'_i, D'_j) \geq \text{cost}_z(X_i, C) \geq (1 + \varepsilon/50)^{-1} \cdot \text{cost}_z(X_i, D_i) = (1 + \varepsilon/50)^{-1} \cdot \text{cost}_z(X'_i, D'_i),$$

implying that $\text{cost}_z(X'_i, D'_i) \in (1 \pm \varepsilon/50) \cdot \text{cost}_z(X'_i, C')$. Therefore, we have

$$\sum_{i=1}^l \text{cost}_z(X_i, D_i) = \sum_{i=1}^l \text{cost}_z(X'_i, D'_i) \in (1 \pm \varepsilon/50) \cdot \text{cost}_z(X', C').$$

Similarly, we can obtain

$$\sum_{i=1}^l \text{cost}_z(S \cap X_i, D_i) \in (1 \pm \varepsilon/50) \cdot \text{cost}_z(S', C').$$

Combined with (19), (20), we confirm that C' satisfies the bridge property stated in (17). It remains to prove the following Lemma 4.18 regarding the size of C' , which would finish the proof of Lemma 4.6.

Lemma 4.18. *It holds that $|C'| = \sum_{i=1}^l |D'_i| \leq (t + k)\Lambda$.*

Proof. TOPROVE 13 □

5 Streaming Implementations

In this section, we implement our reduction algorithms (both Theorem 3.1 and Theorem 4.2) in the dynamic streaming setting. We consider the standard geometric streaming model proposed by Indyk [Ind04], where the input is from a discrete set $[\Delta]^d$ for some integer $\Delta \geq 1$ and is presented as a stream of point insertion and deletions. We obtain two black-box reductions, stated together in Theorem 5.1, that turns a streaming coresets construction algorithm for (k, z) -CLUSTERING to the one for (k, z, m) -CLUSTERING, with guarantee similar to Theorem 3.1 and Theorem 4.2, respectively.

Theorem 5.1 (Streaming Coresets for (k, z, m) -CLUSTERING). *Assume there exists a streaming algorithm that, given $0 < \varepsilon, \delta < 1$, integers $k, z, d, \Delta \geq 1$, and a dataset from $[\Delta]^d$ presented as a dynamic stream, uses space $W(d, \Delta, k, \varepsilon^{-1}, \delta^{-1})$ to construct an ε -coreset for (k, z) -CLUSTERING with a failure probability of at most δ . Then there exists a streaming algorithm that, given $0 < \varepsilon, \delta < 1$, integers $k, z, m, d, \Delta \geq 1$, and a dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, constructs an ε -coreset of X for (k, z, m) -CLUSTERING. This algorithm has a failure probability of at most δ and uses space $\min\{W_1, W_2\} \cdot \text{poly}(d \log(\delta^{-1} \Delta))$, where*

$$\begin{aligned} W_1 &= 2^{O(z \log z)} \cdot \tilde{O}((k + d^z)m\varepsilon^{-1}) + W(d, \Delta, k, O(\varepsilon^{-1}), O(\delta^{-1}zd \log \Delta)), \\ W_2 &= 2^{O(z \log z)} \cdot \tilde{O}(k + m(d/\varepsilon)^{2z}) + W(d + 1, z\varepsilon^{-1}\Delta^{\text{poly}(d)}, k \text{poly}(d), O(\varepsilon^{-1}), O(\delta^{-1}zd \log \Delta)). \end{aligned}$$

5.1 $W_1 \cdot \text{poly}(d \log(\delta^{-1} \Delta))$ -Space: Streaming Implementation of Theorem 3.1

Algorithm Overview The first space complexity $W_1 \cdot \text{poly}(d \log \Delta)$ is derived by implementing Theorem 3.1 in the dynamic streaming setting. Recall that the coreset construction of Theorem 3.1 is based on a λ -bounded partition, where the value of λ depends on the optimal objective $\text{OPT}_z^{(m)}(X)$ (which we can “guess” using small additional space via standard approaches). In the following, we assume that λ is fixed and proceed to present the subroutines we will use in the algorithm’s design. The main challenge lies in the streaming construction of almost-dense λ -bounded partition as shown in Lemma 3.4, which inherently requires storing the entire dataset using $\Omega(n)$ space. To address this, our strategy is to set up a partition of the space $[\Delta]^d$ into buckets before

the stream begins, which can be achieved efficiently (in space) by a geometric decomposition technique called *consistent hashing* (see Lemma 5.2). The non-empty buckets, which contain points from the dataset, form a partition of X . Although there may be numerous non-empty buckets, we show (in Lemma 5.3) that, similar to Claim 4.7 (which is a refinement of Lemma 3.4), there exists a set F of isolated points with a bounded size that can be efficiently extracted in a dynamic stream. Moreover, after removing this set from the dataset, the number of non-empty buckets significantly decreases. Hence, $X \setminus F$ admits a small-sized partition (defined by the consistent hashing). Next, we employ a two-level sparse recovery approach to identify all buckets containing a small number of points (see Lemma 5.5), thereby obtaining the sparse subset X_S . The dense subset is then $X \setminus (X_S \cup F)$, and we construct an ε -coreset S_D for (k, z) -CLUSTERING by independently running a streaming coreset construction on a stream that represents $X \setminus (X_S \cup F)$, which comprises the dataset stream of X with the removals corresponding to $X_S \cup F$. Finally, the coreset for (k, z, m) -CLUSTERING of X is formed as $S_D \cup X_S \cup F$.

Defining λ -Bounded Partition via Consistent Hashing We need to use a geometric hashing technique called *consistent hashing*. The definition of consistent hashing is actually equivalent to that of a sparse partition of the space \mathbb{R}^d , except that it is data oblivious, and requires little space to store and evaluate for every point x the ID of the part that x belongs to. Consistent hashing was first applied in the dynamic streaming setting in a recent work [CFJ+22], and we summarize their result in the following lemma.

Lemma 5.2 ([CFJ+22, Theorem 5.1]). *For any $\lambda > 0$, there exists a (deterministic) hash $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that $\{\varphi^{-1}(y) : y \in \varphi(\mathbb{R}^d)\}$ forms a (μ, Γ, Λ) -sparse partition (see Definition 4.4) of \mathbb{R}^d in Euclidean space, where $\Gamma = O(d)$ and $\Lambda = O(d \log d)$. Furthermore, φ can be described using $O(d^2 \log^2 d)$ bits and one can evaluate $\varphi(X)$ for any point $x \in \mathbb{R}^d$ in space $O(d^2 \log^2 d)$.*

A consistent hash $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$, obtained from Lemma 5.2 in a data-oblivious way, maps each point $x \in X$ to the bucket $\varphi(x)$, and these buckets can be used to create a partition. Formally, we can define a partition $\mathcal{P}_\varphi(X) := \{X \cap \varphi^{-1}(y) : y \in \varphi(X)\}$ of X based on φ . The partition satisfies that $|\mathcal{P}_\varphi(X)| = |\varphi(X)|$ and, according to the diameter property in Definition 4.4, $\mathcal{P}_\varphi(X)$ is λ -bounded. In the following lemma, we demonstrate the existence of a small isolated set F similar to Claim 4.7.

Lemma 5.3. *For a consistent hash φ with a diameter bound λ and a dataset $X \subseteq \mathbb{R}^d$, there exists a subset $F \subseteq X$ of X with $|F| \leq m + (\Gamma/\lambda)^z \cdot \text{OPT}_z^{(m)}(X)$ such that $|\varphi(X \setminus F)| \leq k\Lambda$.*

Proof. **TOPROVE 14** □

Extract Isolated Points F After fixing a mapping φ , the next step is to extract most of the isolated points, for which we have Lemma 5.4. Here we aim to achieve a slightly different goal: extracting a set G from X such that $|\varphi(X \setminus G)|$ approximates $|\varphi(X \setminus Y)|$ for any fixed set Y with $|Y| \leq T$ (where T is a parameter to be determined). This result suffices for our purposes because, as guaranteed by Lemma 5.3, there exists a small-sized set F with $|\varphi(X \setminus F)| \leq k\Lambda = O(kd \log d)$. Therefore, by carefully choosing T , we can obtain a set G with nearly the same result: $|\varphi(X \setminus G)| \leq 2k\Lambda$.

Lemma 5.4 (Extract isolated points). *There exists a randomized algorithm that, given $0 < \delta < 1$, an integer $T > 0$, a hash $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that the value for any point in \mathbb{R}^d can be evaluated in space $\text{poly}(d)$ and a dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, uses $\tilde{O}(T \text{poly}(d \log(\delta^{-1} \Delta)))$*

space to sample a random subset $G \subseteq X$ of size $\tilde{O}(T \text{poly}(d \log(\delta^{-1} \Delta)))$ such that, for any subset $Y \subseteq X$ with $|Y| \leq T$, $\Pr[|\varphi(X \setminus G)| \leq 2 \cdot |\varphi(X \setminus Y)|] \geq 1 - \delta$.

The proof of Lemma 5.4, provided in Appendix C.1, relies on a two-level sampling procedure as follows: we first choose a non-empty bucket $y \in \varphi(X)$ uniformly at random (u.a.r.), then choose u.a.r. a point $x \in \varphi^{-1}(y) \cap X$. The set G is then constructed by repeating the two-level sampling to sample points from X *without replacement*. Let Y be a set such that $|Y| \leq T$, for simplicity, we assume that $\varphi(Y) \cap \varphi(X \setminus Y) = \emptyset$. The high-level idea is that, if $|\varphi(Y)| \geq |\varphi(X \setminus Y)|$, meaning that Y occurs the majority of buckets, then the two-level sampling will, with constant probability, return a point that is from Y . Thus, by repeating two-level sampling without replacement enough times to obtain G , it is likely that G contains most of the points from Y , resulting in $|\varphi(Y \setminus G)| \leq |\varphi(X \setminus Y)|$. Finally, we have $|\varphi(X \setminus G)| \leq |\varphi(X \setminus Y)| + |\varphi(Y \setminus G)| \leq 2|\varphi(X \setminus Y)|$. It remains to implement the two-level sampling *without replacement* in the streaming setting, for which we employ a subroutine of two-level ℓ_0 -sampler proposed in [CFJ⁺22, Lemma 3.3].

Extract Sparse Subset We then present an algorithm that, given a mapping φ and a dataset X , if $|\varphi(X)|$ is bounded, then the algorithm recovers all the parts of $\mathcal{P}_\varphi(X)$ with small sizes from the data stream. The algorithm is an application of a two-level extension of the sparse recovery (from a frequency vector); see e.g. [CM06], and we provide a proof in Appendix C.2.

Lemma 5.5 (Identify Sparse Subsets). *There exists a streaming algorithm that, given integers $0 < \delta < 1$, $N, M > 0$, a mapping $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that the value for any point in \mathbb{R}^d can be evaluated in space $\text{poly}(d)$, and a dataset $X \subseteq [\Delta]^d$ represented as a dynamic stream, returns a collection of subsets of X or \perp . If $|\mathcal{P}_\varphi(X)| \leq N$, the algorithm returns $\{P \in \mathcal{P}_\varphi(X) : |P| \leq M\}$; otherwise, it returns \perp . The algorithm uses space $\tilde{O}(NM \cdot \text{poly}(d \log(\delta^{-1} \Delta)))$ and fails with probability at most δ .*

Remark 5.6. A straightforward corollary of Lemma 5.3 is that $|\mathcal{P}_\varphi(X)| = |\varphi(X)| \leq k\Lambda + m + (\Gamma/\lambda)^z \cdot \text{OPT}_z^{(m)}(X)$. Hence, one might consider the naive approach of directly identifying all the sparse parts in $\mathcal{P}_\varphi(X)$. However, we note that in the streaming setting, verifying whether a part $P \in \mathcal{P}_\varphi(X)$ contains more than $(1 + \varepsilon^{-1})m$ points may require $\Omega(m\varepsilon^{-1})$ space. This would result in a factor of m^2 in the total space, which is not acceptable.

Putting Things Together Now we are ready to prove Theorem 5.1 by combining all aforementioned subroutines together. Let us assume momentarily that we have a guess $\widehat{\text{OPT}}$ of $\text{OPT}_z^{(m)}(X)$, and we will remove this assumption later. For simplicity, let \mathcal{A} and \mathcal{B} denote the algorithm of Lemma 5.4 and the algorithm of Lemma 5.5, respectively. Moreover, let \mathcal{C} denote a streaming algorithm that can construct an ε -coreset for (k, z) -CLUSTERING of a dataset from $[\Delta]^d$ presented as a dynamic stream, using space $W(d, \Delta, k, \varepsilon^{-1}, \delta^{-1})$ and failing with probability δ . Then, our algorithm is described in Algorithm 4.

Clearly, Algorithm 4 uses space that is the cumulative space of \mathcal{A} , \mathcal{B} , and \mathcal{C} , amounting to $2^{O(z \log z)} \cdot \tilde{O}((k + d^z)m\varepsilon^{-1} \cdot \text{poly}(d \log \Delta)) + W(d, \Delta, k, \varepsilon^{-1}, \delta^{-1})$.

Algorithm 4 can be implemented in one pass. Specifically, We initially run all three algorithms \mathcal{A} , \mathcal{B} and \mathcal{C} on the input stream. Once the stream ends, we obtain G from \mathcal{A} , and proceed to run \mathcal{B} and \mathcal{C} on a stream composed of deletions of points in G . This is equivalent to running the algorithms on a stream representing $X \setminus G$, hence implementing Line 3. By applying the same approach, we can implement Line 5.

Algorithm 4 Coreset construction for (k, z, m) -CLUSTERING of X presented by a dynamic stream

Require: a guess $\widehat{\text{OPT}}$ of $\text{OPT}_z^{(m)}(X)$

- 1: construct a consistent hash φ with diameter bound $\lambda := (z+1)^{-\xi} \cdot \left(\frac{\varepsilon \cdot \widehat{\text{OPT}}}{m}\right)^{1/z}$ for sufficiently large constant $\xi > 0$ using Lemma 5.2
 - 2: run \mathcal{A} on inputs $\delta, T := 2^{O(z \log(z+1))} \cdot m d^z \varepsilon^{-1}, \varphi$ and the dynamic stream that represents X to obtain a subset $G \subseteq X$.
 - 3: run \mathcal{B} on inputs $\delta, N := O(kd \log d), M := (1 + \varepsilon^{-1})m, \varphi$ and a dynamic stream that represents $X \setminus G$
 - 4: if \mathcal{B} returns \perp , the algorithm returns \perp ; otherwise, let $\mathcal{S} \leftarrow$ the output of \mathcal{B} and let $X_{\mathcal{S}} \leftarrow \bigcup_{P \in \mathcal{S}} P$
 - 5: run \mathcal{C} on a dynamic stream that represents $X_{\mathcal{D}} := X \setminus (G \cup X_{\mathcal{S}})$ to obtain an ε -coreset $S_{\mathcal{D}}$ of $X_{\mathcal{D}}$
 - 6: **return** $S_{\mathcal{D}} \cup X_{\mathcal{S}} \cup G$ \triangleright algorithm fails if either \mathcal{B} or \mathcal{C} fails
-

Error Analysis Let us first assume that the guess $\widehat{\text{OPT}}$ satisfies that $\text{OPT}_z^{(m)}(X)/(z+1) \leq \widehat{\text{OPT}} \leq \text{OPT}_z^{(m)}(X)$, and we will remove this assumption later. Algorithm 4 fails to return a coreset if and only if \mathcal{B} or \mathcal{C} fails, or if \mathcal{B} returns \perp . Applying union bound, both algorithms \mathcal{B} and \mathcal{C} succeed simultaneously with probability $1 - 2\delta$. Furthermore, by Lemma 5.3, since $\widehat{\text{OPT}} \geq \text{OPT}_z^{(m)}(X)/(z+1)$, there exists a set F with $|F| \leq 2^{O(z \log(z+1))} \cdot m d^z \varepsilon^{-1}$ such that $|\varphi(X \setminus F)| \leq O(kd \log d)$. In this case, G returned by \mathcal{A} also satisfies that $|\varphi(X \setminus G)| \leq O(kd \log d)$ with probability $1 - \delta$. Given this condition and the success of \mathcal{B} , we can conclude, according to Lemma 5.5, that \mathcal{B} does not return \perp . Consequently, the probability of that Algorithm 4 succeeds in returning a coreset is at least $1 - 3\delta$.

Conditioned on the success of Algorithm 4, we have, as guaranteed by Lemma 5.5, that the set $X_{\mathcal{D}} := X \setminus (G \cup X_{\mathcal{S}})$ satisfies the property that, for every $P \in \mathcal{P}_{\varphi}(X)$ with $P \cap X_{\mathcal{D}} \neq \emptyset$, $|P \cap X_{\mathcal{D}}| \geq (1 + \varepsilon^{-1})m$. Hence by Lemma 3.3, $S_{\mathcal{D}}$ is also an $(O(\varepsilon), O(\varepsilon) \cdot \widehat{\text{OPT}})$ -coreset of $X_{\mathcal{D}}$ for (k, z, m) -CLUSTERING, implying that $S := G \cup X_{\mathcal{S}} \cup S_{\mathcal{D}}$ is an $(O(\varepsilon), O(\varepsilon) \cdot \widehat{\text{OPT}})$ -coreset of the original dataset X for (k, z, m) -CLUSTERING due to the composability of the coreset (Fact 2.3). Since the guess $\widehat{\text{OPT}}$ further satisfies that $\widehat{\text{OPT}} \leq \text{OPT}_z^{(m)}(X)$, S becomes an $O(\varepsilon)$ -coreset for (k, z, m) -CLUSTERING. It suffices to scale both ε and δ by a constant factor.

Removing Assumption of Knowing $\widehat{\text{OPT}}$ Finally, we remove the assumption of knowing $\widehat{\text{OPT}}$ which satisfies that $\text{OPT}_z^{(m)}(X)/(z+1) \leq \widehat{\text{OPT}} \leq \text{OPT}_z^{(m)}(X)$ in advance. We assume without loss of generality that X contains more than $k + m + 1$ distinct points from $[\Delta]^d$, since otherwise one could use a sparse recovery structure to fully recover all the data points. Then, we have $1 \leq \text{OPT}_z^{(m)}(X) \leq n \cdot (\sqrt{d}\Delta)^z$. We run in parallel $\tau := \lfloor \log_{z+1}(n \cdot (\sqrt{d}\Delta)^z) \rfloor + 1$ instances $\mathcal{D}_0, \dots, \mathcal{D}_{\tau-1}$ of Algorithm 4 (resulting in a space increase by a factor of only $\tau \leq O(zd \log \Delta)$), where the \mathcal{D}_i tries $\widehat{\text{OPT}}_i := (z+1)^i$. We set the failure probability of each instance to be δ/τ . Then, with probability $1 - \delta$, all instances succeeds, i.e., return a coreset or \perp . We select the output of the \mathcal{D}_j , where $j \geq 0$ is the smallest index such that \mathcal{D}_j does not return \perp , as the final output. Condition on that all instances do not fail, let $i^* := \lfloor \log_{z+1}(\text{OPT}_z^{(m)}(X)) \rfloor$, we know that the instance \mathcal{D}_{i^*} must return a coreset. Hence, we have $i \leq i^*$, which implies that $\widehat{\text{OPT}}_i \leq \text{OPT}_z^{(m)}(X)$, and thus, \mathcal{D}_i returns an ε -coreset.

5.2 $W_2 \cdot \text{poly}(d \log(\delta^{-1} \Delta))$ -Space: Streaming Implementation of Theorem 4.2

We present the streaming implementation of Theorem 4.2 and achieve the $W_2 \cdot \text{poly}(d \log \Delta)$ space complexity. As an important step in Theorem 4.2, one needs to first run Algorithm 2 to turn a vanilla coreset to a coreset with a weaker size-preserving property. Hence, we also start with the streaming implementation of this Algorithm 2. Then similar to Theorem 4.2, we calibrate its weight to ensure it is truly size-preserving, and we can conclude that this resultant coreset works for (k, z, m) -CLUSTERING by Lemma 4.3. We would reuse the gadgets of streaming algorithm developed in Section 5.1 without mentioning how they are implemented again.

Streaming Implementation of Algorithm 2 Recall that the input for Algorithm 2 consists of a dataset X , which is presented as a dynamic point stream in this case, along with two parameters $\mu \geq 0$ and $k' \geq 1$. We would then discuss how each line of Algorithm 2 is implemented.

Line 1 In the streaming setting, we cannot expect to explicitly compute and store a sparse partition of the dataset. Therefore, instead, we compute a consistent hashing φ using Lemma 5.2 with a diameter bound μ at the very beginning, which takes $\text{poly}(d \log \Delta)$ space to store the description of φ . According to Lemma 5.2 and Definition 4.4, we know that the partition $\mathcal{P}_\varphi(X) = \{\varphi^{-1}(y) \cap X : y \in \varphi(X)\}$ is a (μ, Λ, Γ) -sparse partition with $\Lambda = O(d)$ and $\Gamma = O(d \log d)$. Moreover, recall from Lemma 5.2 that for any give point $x \in \mathbb{R}^d$, the value of $\varphi(x)$ can be computed using space $\text{poly}(d)$. Hence, the image $\varphi([\Delta]^d)$ has a size of at most $\Delta^d \cdot 2^{\text{poly}(d)} \leq \Delta^{\text{poly}(d)}$, since we can encode the input point and the computation process of $\varphi(x)$ by a binary string of length $O(d \log \Delta) + \text{poly}(d)$. Hence, we assume that the image of φ is $[\Delta^{\text{poly}(d)}]$ instead of \mathbb{R}^d .

Line 2 and Line 3 The main difficulty arises in implementing Line 2 and Line 3, where we map the input points into an $O(z\varepsilon^{-1} \cdot \text{diam}(X) \cdot |X|^{1/z})$ -separated $|\mathcal{P}_\varphi(X)|$ -duplicated space M^{dup} of the discrete Euclidean space and construct a vanilla coreset on M^{dup} .

To accomplish this in the streaming setting, we prove in Lemma B.1 that it suffices to map each point x to some $(x, \varphi(x) \cdot w) \in \mathbb{R}^{d+1}$ where $w = O(z\varepsilon^{-1} \cdot \text{diam}(X) \cdot |X|^{1/z})$, and then construct a vanilla coreset on \mathbb{R}^{d+1} . Notice that $\text{diam}(X) \leq d\Delta$ and $|X| \leq \Delta^d$, we pick $w' := z\varepsilon^{-1} \cdot \Delta^{c \cdot d}$ for sufficiently large constant $c \geq 1$. Then, our implementation simply converts each insertion/deletion of a point x to the insertion/deletion of the point $(x, \varphi(x) \cdot w')$, and feeds these resulting insertions/deletions to a streaming algorithm for constructing an $O(\varepsilon)$ -coreset for (k', z) -CLUSTERING. Since $\varphi(x) \cdot w' \leq z\varepsilon^{-1} \cdot \Delta^{\text{poly}(d)}$, we have $(x, \varphi(x) \cdot w') \in [z\varepsilon^{-1} \cdot \Delta^{\text{poly}(d)}]^{d+1}$. Hence, the implementation of Line 2 and Line 3 uses space $W(d+1, z\varepsilon^{-1} \Delta^{\text{poly}(d)}, k', O(\varepsilon^{-1}), O(\delta^{-1}))$.

Line 4 Line 4 can be directly implemented after the stream ends. Hence, we complete the implementation of Algorithm 2 with a space complexity of $W(d+1, z\varepsilon^{-1} \Delta^{\text{poly}(d)}, k', O(\varepsilon^{-1}), O(\delta^{-1})) + \text{poly}(d \log \Delta)$. We denote by \mathcal{A} the streaming version of Algorithm 2.

Streaming Implementation of Theorem 4.2 Similar to the steps in Section 5.1, we assume that we have a guess $\widehat{\text{OPT}} = \Theta(\text{OPT}_z^{(m)}(X))$. Set $\mu := 2^{-O(\log(z+1))} \cdot \varepsilon \cdot \left(\frac{\widehat{\text{OPT}}}{m}\right)^{1/z}$, and let $\lambda := O\left(\frac{\varepsilon \mu}{z \Gamma}\right) = 2^{-O(\log(z+1))} \cdot \varepsilon^2 \cdot \left(\frac{\widehat{\text{OPT}}}{m}\right)^{1/z} \cdot d^{-1}$. Let φ and φ' be two consistent hashing with diameter bounds μ and λ , respectively. According to Lemma 5.3, there exists a set $F \subseteq X$ with $|F| \leq 2^{O(z \log z)} \cdot O(m(d/\varepsilon)^{2z})$ such that $|\varphi'(X \setminus F)| \leq k\Lambda$. Therefore, we first apply the algorithm of Lemma 5.4 to extract $G \subseteq X$ such that $|\varphi'(X \setminus G)| \leq 2k\Lambda$ using space $2^{O(z \log z)} \cdot \tilde{O}(m(d/\varepsilon)^{2z})$.

$\text{poly}(d \log(\delta^{-1} \Delta))$. Then we have $X \setminus G$ admits a λ -bounded partition of size at most $2k\Lambda$. Moreover, for every $y \in \varphi'(X)$, since $\text{diam}(\varphi'^{-1}(y)) \leq \lambda \leq \mu/\Gamma$, we have that $|\varphi(\varphi'^{-1}(y))| \leq \Lambda$. This implies that $|\varphi(X \setminus G)| \leq |\varphi'(X \setminus G)| \cdot \Lambda \leq 2k\Lambda^2$.

Set $k' = (k + 2k\Lambda^2 + 2k\Lambda)\Lambda = k \text{poly}(d)$. We run \mathcal{A} , the streaming version of Algorithm 2, on input consisting of a stream representing $X \setminus G$, μ , and k' . The implementation of running \mathcal{A} on a stream representing $X \setminus G$ is the same as that in the first algorithm of Theorem 5.1. Specifically, we first run \mathcal{A} and an algorithm identifying G in parallel. After the stream ends, we continue to run \mathcal{A} on a stream consisting of deletions of points in G . By doing so, we obtain a weighted set S such that S is an ε -coreset of $X \setminus G$ for $(k + 2k\Lambda^2, z)$ -CLUSTERING and is nearly size-preserving with respect to $\mathcal{P}_\varphi(X \setminus G)$ with $|\mathcal{P}_\varphi(X \setminus G)| \leq 2k\Lambda^2$. This step uses space $W(d + 1, z\varepsilon^{-1}\Delta^{\text{poly}(d)}, k \text{poly}(d), O(\varepsilon^{-1}), O(\delta^{-1})) + \text{poly}(d \log(\delta^{-1} \Delta))$.

In addition, we run a sparse recovery algorithm (Lemma C.3) in parallel with \mathcal{A} on a stream representing $\varphi(X \setminus G)$, which uses space $\tilde{O}(k \cdot \text{poly}(d \log(\delta^{-1} \Delta)))$ to return the frequencies of each $y \in \varphi(X \setminus G)$ if $|\varphi(X \setminus G)| \leq 2k\Lambda^2$ and fails with a probability of at most δ .

Then, we can calibrate the weight of S so that S is exactly size-preserving with respect to $\mathcal{P}_\varphi(X \setminus G)$. By Lemma 4.3, we have that S is an $O(\varepsilon)$ -coreset of $X \setminus G$ for (k, z, m) -CLUSTERING. Finally, by composability of coresets, $S \cap G$ is the desired $O(\varepsilon)$ -coreset of X for (k, z, m) -CLUSTERING. It remains to scale ε by a constant. The overall space is

$$2^{O(z \log z)} \cdot \tilde{O}(m(d/\varepsilon)^{2z}) \cdot \text{poly}(d \log(\delta^{-1} \Delta)) + W\left(d + 1, z\varepsilon^{-1}\Delta^{\text{poly}(d)}, k \text{poly}(d), O(\varepsilon^{-1}), O(\delta^{-1})\right).$$

We complete the proof by removing the assumption of knowing $\widehat{\text{OPT}}$ using a similar approach as in the first algorithm of Theorem 5.1, which needs to replace δ with $\delta/O(zd \log \Delta)$ and results in a space increase by a $O(zd \log \Delta)$ factor.

References

- [BBH⁺20] Daniel N. Baker, Vladimir Braverman, Lingxiao Huang, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in graphs of bounded treewidth. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 569–579. PMLR, 2020.
- [BCJ⁺22] Vladimir Braverman, Vincent Cohen-Addad, Shaofeng H.-C. Jiang, Robert Krauthgamer, Chris Schwiegelshohn, Mads Bech Tofttrup, and Xuan Wu. The power of uniform sampling for coresets. In *FOCS*, pages 462–473. IEEE, 2022.
- [BEL13] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general communication topologies. In *NIPS*, pages 1995–2003, 2013.
- [BFL⁺17] Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 576–585. PMLR, 2017.
- [BFS21] Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and euclidean spaces and their applications. In *ICALP*, volume 198 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

- [BJKW19] Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for ordered weighted clustering. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 744–753. PMLR, 2019.
- [BJKW21a] Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering in excluded-minor graphs and beyond. In *SODA*, pages 2679–2696. SIAM, 2021.
- [BJKW21b] Vladimir Braverman, Shaofeng H.-C. Jiang, Robert Krauthgamer, and Xuan Wu. Coresets for clustering with missing values. In *NeurIPS*, pages 17360–17372, 2021.
- [BVX19] Aditya Bhaskara, Sharvaree Vadgama, and Hong Xu. Greedy sampling for approximate clustering in the presence of outliers. In *NeurIPS*, pages 11146–11155, 2019.
- [CDR⁺25] Vincent Cohen-Addad, Andrew Draganov, Matteo Russo, David Saulpic, and Chris Schwiegelshohn. A tight vc-dimension analysis of clustering coresets with applications. In *SODA*, pages 4783–4808. SIAM, 2025.
- [CFJ⁺22] Artur Czumaj, Arnold Filtser, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via geometric hashing. *arXiv preprint arXiv:2204.02095*, 2022. The latest version has additional results compared to the preliminary version in [CJK⁺22]. [arXiv:2204.02095](https://arxiv.org/abs/2204.02095).
- [Che09] Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.
- [CJK⁺22] Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. Streaming facility location in high dimension via geometric hashing. In *FOCS*, pages 450–461. IEEE, 2022.
- [CKLV17] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *NIPS*, pages 5029–5037, 2017.
- [CKMN01] Moses Charikar, Samir Khuller, David M. Mount, and Giri Narasimhan. Algorithms for facility location problems with outliers. In *SODA*, pages 642–651. ACM/SIAM, 2001.
- [CL19] Vincent Cohen-Addad and Jason Li. On the fixed-parameter tractability of capacitated clustering. In *ICALP*, volume 132 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [CLS⁺22] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, Chris Schwiegelshohn, and Omar Ali Sheikh-Omar. Improved coresets for euclidean k-means. In *NeurIPS*, 2022.
- [CLSS22] Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In *STOC*, pages 1038–1051. ACM, 2022.
- [CM06] Graham Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. In *CISS*, pages 198–201. IEEE, 2006.

- [CSS21] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coresets framework for clustering. In *STOC*, pages 169–182. ACM, 2021.
- [CSS23] Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. Deterministic clustering in high dimensional spaces: Sketches and approximation. In *FOCS*, pages 1105–1130. IEEE, 2023.
- [Fel20] Dan Feldman. Core-sets: An updated survey. *WIREs Data Mining Knowl. Discov.*, 10(1), 2020.
- [Fil20] Arnold Filtser. Scattering and sparse partitions, and their applications. In *ICALP*, volume 168 of *LIPIcs*, pages 47:1–47:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [FKRS19] Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. *ACM Trans. Algorithms*, 15(2):26:1–26:26, 2019.
- [FL11] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578. ACM, 2011. <https://arxiv.org/abs/1106.1379>.
- [FS12] 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*, pages 1343–1354. SIAM, 2012.
- [FSS20] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca, and projective clustering. *SIAM J. Comput.*, 49(3):601–657, 2020.
- [GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543. IEEE Computer Society, 2003.
- [HHHW23] Lingxiao Huang, Ruiyuan Huang, Zengfeng Huang, and Xuan Wu. On coresets for clustering in small dimensional euclidean spaces. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 13891–13915. PMLR, 2023.
- [HJLW18] Lingxiao Huang, Shaofeng H.-C. Jiang, Jian Li, and Xuan Wu. Epsilon-coresets for clustering (with outliers) in doubling metrics. In *FOCS*, pages 814–825. IEEE Computer Society, 2018.
- [HJLW23] Lingxiao Huang, Shaofeng H.-C. Jiang, Jianing Lou, and Xuan Wu. Near-optimal coresets for robust clustering. In *ICLR*. OpenReview.net, 2023.
- [HJV19] Lingxiao Huang, Shaofeng H.-C. Jiang, and Nisheeth K. Vishnoi. Coresets for clustering with fairness constraints. In *NeurIPS*, pages 7587–7598, 2019.
- [HK07] Sarel Har-Peled and Akash Kushal. Smaller coresets for k -median and k -means clustering. *Discret. Comput. Geom.*, 37(1):3–19, 2007.
- [HK20] Monika Henzinger and Sagar Kale. Fully-dynamic coresets. In *ESA*, volume 173 of *LIPIcs*, pages 57:1–57:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

- [HLLW25] Lingxiao Huang, Jian Li, Pinyan Lu, and Xuan Wu. Coresets for constrained clustering: General assignment constraints and improved size bounds. In *SODA*, pages 4732–4782. SIAM, 2025.
- [HLW24] Lingxiao Huang, Jian Li, and Xuan Wu. On optimal coreset construction for euclidean (k, z) -clustering. In *STOC*, pages 1594–1604. ACM, 2024.
- [HM04] Sarel Har-Peled and Soham Mazumdar. On coresets for k -means and k -median clustering. In *STOC*, pages 291–300. ACM, 2004. <https://arxiv.org/abs/1810.12826>.
- [HSYZ18] Wei Hu, Zhao Song, Lin F. Yang, and Peilin Zhong. Nearly optimal dynamic k -means clustering for high-dimensional data. *CoRR*, abs/1802.00459, 2018.
- [HV20] Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In *STOC*, pages 1416–1429. ACM, 2020.
- [Ind04] Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC*, pages 373–380. ACM, 2004.
- [JKS08] T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory Comput.*, 4(1):129–135, 2008.
- [JLN⁺05] Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *STOC*, pages 386–395. ACM, 2005.
- [KN97] Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- [KNR99] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Comput. Complex.*, 8(1):21–49, 1999.
- [KPS00] Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k -center problems. *Theor. Comput. Sci.*, 242(1-2):237–245, 2000.
- [LSF22] Sagi Lotan, Ernesto Evgeniy Sanches Shayda, and Dan Feldman. Coreset for line-sets clustering. In *NeurIPS*, 2022.
- [MF19] Yair Marom and Dan Feldman. k -means clustering of lines for big data. In *NeurIPS*, pages 12797–12806, 2019.
- [MMR19] Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k -means and k -medians clustering. In *STOC*, pages 1027–1038. ACM, 2019.
- [MS18] Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *Künstliche Intell.*, 32(1):37–53, 2018.
- [SSS19] Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Fair coresets and streaming algorithms for fair k -means. In *WAOA*, volume 11926 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2019.

- [SW18] Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *FOCS*, pages 802–813. IEEE Computer Society, 2018.
- [TWZ⁺22] Murad Tukan, Xuan Wu, Samson Zhou, Vladimir Braverman, and Dan Feldman. New coresets for projective clustering and applications. In *AISTATS*, volume 151 of *Proceedings of Machine Learning Research*, pages 5391–5415. PMLR, 2022.

Appendices

A Facts about Coresets

Fact 2.3 (Composability of coresets). *For $0 < \varepsilon < 1, \eta_1, \eta_2 \geq 0$, and two datasets $X, Y \subseteq V$, if S_X is an (ε, η_1) -coreset of X for (k, z, m) -CLUSTERING, and S_Y is an (ε, η_2) -coreset of Y for (k, z, m) -CLUSTERING, then $S_X \cup S_Y$ is an $(\varepsilon, \eta_1 + \eta_2)$ -coreset of $X \cup Y$ for (k, z, m) -CLUSTERING.*

Proof. **TOPROVE 15** □

Lemma 4.8. *For some $0 < \varepsilon < 1$ and $\eta > 0$, if for all $C \in V^k$ and real numbers $h_1, \dots, h_t \geq 0$ with $\sum_{i=1}^t h_i \leq m$, it holds that*

$$\left| \sum_{i=1}^t \text{cost}_z^{(h_i)}(S_i, C) - \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) \right| \leq \varepsilon \cdot \sum_{i=1}^t \text{cost}_z^{(h_i)} \text{cost}_z(X_i, C) + \eta, \quad (4)$$

then S is an (ε, η) -coreset (see Definition 2.2) of X for (k, z, m) -CLUSTERING.

Proof. **TOPROVE 16** □

B Separated Duplication of Various Metric Families

In this section, we discuss how Theorem 4.2 can be applied in various metric spaces. The main challenge lies in demonstrating that there exists a w -separated h -duplication $M_{h,w}^{\text{dup}}$ of M has a “dimension” that is almost equal to that of M for any w and h . We accomplish this on a case-by-case basis, examining metric spaces for which the construction of coresets is well-studied, including Euclidean space, doubling metric spaces, and graph metrics.

Without loss of generality, in the following presentation, we assume that both the coreset size N and the runtime T are monotonic with respect to n, k, ε^{-1} .

B.1 ℓ_p Metrics

We begin by discussing the ℓ_p metric space for $p \geq 1$, which includes Euclidean space by setting $p = 2$. The ℓ_p metric space $M = (\mathbb{R}^d, \ell_p)$ is defined on \mathbb{R}^d , with distance function computed using ℓ_p norm. Coresets for clustering have been extensively studied in ℓ_p metric spaces, especially in Euclidean space. The state-of-the-art coreset construction has already achieved a size bound of $\text{poly}(k\varepsilon^{-1})$, which is independent of the input size and dimension (see, e.g., [CSS21]).

To apply Theorem 4.2 to ℓ_p metric spaces $M = (\mathbb{R}^d, \ell_p)$, we demonstrate that for any h and w , there exists a w -separated h -duplication of M that can be embedded, with no distortion, into the metric space $(\mathbb{R}^{d+1}, \ell_p)$, as stated in Lemma B.1.

Lemma B.1. *Let $M = (\mathbb{R}^d, \ell_p)$ be an ℓ_p metric space, for integer $d \geq 1$ and real number $p \geq 1$. For every integer $h \geq 1$ and real number $w \geq 0$, there exists a w -separated h -duplication M^{dup} of M that can be embedded into $(\mathbb{R}^{d+1}, \ell_p)$ with no distortion. Moreover, for any given point from M^{dup} , its image under the embedding can be computed in $O(d)$ time.*

Proof. TOPROVE 17 □

According to Lemma B.1, if an algorithm \mathcal{A} can construct vanilla coreset for $(\mathbb{R}^{d+1}, \ell_p)$, then there exists a family $\mathcal{M}^{\text{dup}} = \left\{ M_{h,w}^{\text{dup}} \right\}_{h \geq 1, w \geq 0}$ of separated duplication of M and an algorithm \mathcal{B} such that \mathcal{B} can construct vanilla coreset for every $M_{h,w}^{\text{dup}} \in \mathcal{M}^{\text{dup}}$ by embedding the input dataset into $(\mathbb{R}^{d+1}, \ell_p)$, running algorithm \mathcal{A} , and then returning the pre-image of the constructed coreset. Since Lemma B.1 also guarantees that the embedding can be computed in $O(d)$ time, the additional runtime is only $O(nd)$.

Therefore, we obtain the following corollary, where for simplicity, we assume that the size bound N only depends on d, k and ε^{-1} .

Corollary B.2 (Reduction on ℓ_p metric space). *Let $p \geq 1$ be a real number. Assume there exists an algorithm that given $0 < \varepsilon < 1$, integers $d, k, z \geq 1$ and an n -point dataset $X \subseteq \mathbb{R}^d$ as input, runs in time $T(d, n, k, \varepsilon^{-1})$ to construct an ε -coreset of X for (k, z) -CLUSTERING on (\mathbb{R}^d, ℓ_p) of size $N(d, k, \varepsilon^{-1})$.*

Then, there exists an algorithm that, given $0 < \varepsilon < 1$, integers $d, k, z, m \geq 1$, an n -point dataset $X \subseteq \mathbb{R}^d$ and a $(2^{O(z)}, O(1), O(1))$ -approximation solution C^ to (k, z, m) -CLUSTERING on X as input, runs in time*

$$\tilde{O}(nkd) + \text{poly}(kdm\varepsilon^{-1}) + 2 \cdot T(d+1, n, O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1}))$$

to compute an ε -coreset of X for (k, z) -CLUSTERING of size

$$2^{O(z \log z)} \cdot O(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1})) + 2 \cdot N(d+1, O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1})).$$

B.2 Doubling Metrics

We then consider the doubling metric space, which is an important generalization of ℓ_p metric spaces. Here, an important concept is the *doubling dimension* [GKL03], defined as the least integer $t \geq 0$, such that every ball can be covered by at most 2^t balls of half the radius. A metric space with a bounded doubling dimension is called a doubling metric, and we denote by $\text{ddim}(M)$ the doubling dimension of the doubling metric M . Coresets in doubling metrics have been studied in previous works [HJLW18, CSS21], which achieve a size that depends only on the doubling dimension, k , and ε^{-1} .

For a doubling metric $M = (V, \text{dist})$, we have the following lemma, demonstrating that for every h and w , there is a w -separated h -duplication of M that has a doubling dimension $O(\text{ddim}(M))$.

Lemma B.3. *For a doubling metric $M = (V, \text{dist})$, integer $h \geq 1$ and real number $w \geq 0$, there exists a w -separated h -duplication $M^{\text{dup}} = (V \times [h], \text{dist}')$ of M such that $\text{ddim}(M^{\text{dup}}) \leq 2 \text{ddim}(M) + 2$. Moreover, we can evaluate the distance $\text{dist}'(x, y)$ in constant time for any $x, y \in V \times [h]$.*

Proof. TOPROVE 18 □

Corollary B.4. *Assume that there exists an algorithm such that, for every $0 < \varepsilon < 1$, integers $d, k, z \geq 1$, metric space M with $\text{ddim}(M) \leq d$ and n -point dataset from M , it computes in time $T(d, n, k, \varepsilon^{-1})$ an ε -coreset of size $N(d, k, \varepsilon^{-1})$ for (k, z) -CLUSTERING on M .*

Then, there is an algorithm that, given $0 < \varepsilon < 1$, integers $d, k, z, m \geq 1$, a metric space M with $\text{ddim}(M) \leq d$, an n -point dataset from M and a $(2^{O(z)}, O(1), O(1))$ -approximation solution C^* to (k, z, m) -CLUSTERING on X , computes in time

$$\tilde{O}(nk) + \text{poly}(km\varepsilon^{-1}) + 2 \cdot T(O(d), n, O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1}))$$

an ε -coreset for (k, z, m) -CLUSTERING on M of size

$$2^{O(z \log z)} \cdot O(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1})) + 2 \cdot N(O(d), O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1})).$$

Remark B.5. A special case of a doubling metric is the general metric $M = (V, \text{dist})$ with finite ambient size, for which we have a well-known fact that $\text{ddim}(M) \leq O(\log |V|)$. Although this fact enables us to use doubling dimension to indicate the relationship between M and its separated duplication, we still hope to establish the relationship based on the ambient size. To achieve this, observe that in Theorem 4.2, we actually only apply the assumed algorithm to $M_{h,w}^{\text{dup}}$ with h less than the data size. Therefore, it suffices to restrict the family \mathcal{M}^{dup} to include only those $M_{h,w}^{\text{dup}}$ with $1 \leq h \leq |V|$, which has an ambient size at most $|V| \cdot h \leq |V|^2$.

B.3 Graph Metrics

Given an edge-weighted graph $G = (V, E)$, we consider the metric space $M = (V, \text{dist})$, where dist is the shortest-path distance on the graph G . Coresets for graph datasets have also gained researchers' interest in recent years [BBH⁺20, BJKW21a, CSS21, CDR⁺25], where they relate the coreset size to some complexity measures of the graph, such as treewidth and the size of the excluded minor. Here, we hope to establish reductions from robust coresets to vanilla coresets on graph metrics.

Given a graph $G = (V, E)$, an integer $h \geq 1$, and a real number $w \geq 0$, the following lemma provides a construction of a new graph $G' = (V', E')$ such that the shortest-path metric of G' is a w -separated h -duplication of that of G . Furthermore, G' has similar complexity to G in terms of both treewidth and the size of the excluded minor.

Lemma B.6. *For any graph $G = (V, E)$, integer $h \geq 1$ and real number $w \geq 0$, there exists a graph $G' = (V \times [h], E')$ such that the shortest-path metric of G' is a w -separated h -duplication of the shortest-path metric of G . Moreover, it holds that*

1. $\text{tw}(G') \leq \text{tw}(G)$, where $\text{tw}(G)$ denotes the treewidth of G ; and
2. if G excludes a fixed minor H , then G' excludes the same minor H .

Proof. TOPROVE 19 □

According to Lemma B.6, Theorem 4.2 is applicable to graphs with bounded treewidth and minor-free graphs. Here, we state the reduction result only for graphs with bounded treewidth. The result for minor-free graphs is nearly identical, with the only difference lying in the dependence of N and T .

Corollary B.7. *Assume that there exists an algorithm that, given $0 < \varepsilon < 1$, integers $t, k, z \geq 1$, a graph $G = (V, E)$ with $\text{tw}(G) \leq t$ and an n -point dataset $X \subseteq V$ as input, runs in time $T(t, n, k, \varepsilon^{-1})$ to compute an ε -coreset of size $N(t, k, \varepsilon^{-1})$ for (k, z) -CLUSTERING on the shortest-path metric space of G .*

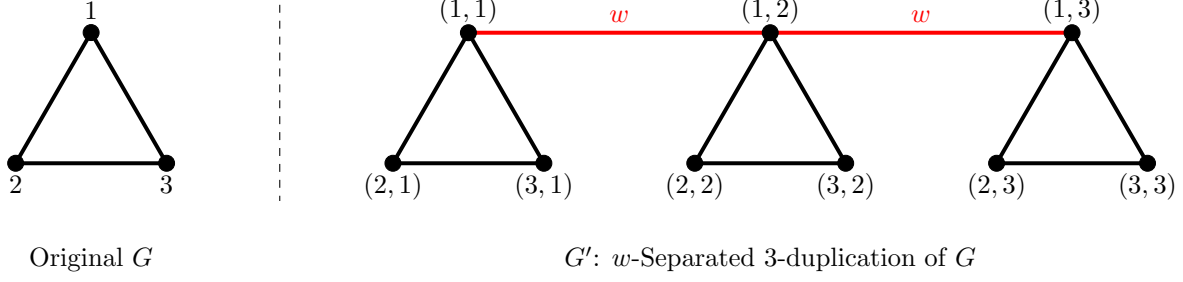


Figure 1: Illustration of the construction of separated duplication of a graph with the weights of edges omitted. On the left, we show an original graph G , using a triangle graph as an example. On the right, we demonstrate our construction for the w -separated 3-duplication of G , where the three triangle graphs connected by black edges represent the s of G , and red edges are weighted by w to satisfy the separation requirement.

Then, there is an algorithm that, given $0 < \varepsilon < 1$, integers $t, k, z \geq 1$, a graph $G = (V, E)$ with $\text{tw}(G) \leq t$, an n -point dataset $X \subseteq V$ and a $(2^{O(z)}, O(1), O(1))$ -approximation solution C^* to (k, z, m) -CLUSTERING on X as input, runs in time

$$\tilde{O}(nk) + \text{poly}(km\varepsilon^{-1}) + 2 \cdot T(t, n, O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1}))$$

to construct an ε -coreset of size

$$2^{O(z \log z)} \cdot O(m\varepsilon^{-2z} \log^z(km\varepsilon^{-1})) + 2 \cdot N(t, O(k \log^2(km\varepsilon^{-1})), O(\varepsilon^{-1}))$$

for (k, z, m) -CLUSTERING on the shortest-path metric of G .

C Missing Proofs of Section 5.1

C.1 Proof of Lemma 5.4

Lemma 5.4 (Extract isolated points). *There exists a randomized algorithm that, given $0 < \delta < 1$, an integer $T > 0$, a hash $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that the value for any point in \mathbb{R}^d can be evaluated in space $\text{poly}(d)$ and a dataset $X \subseteq [\Delta]^d$ presented as a dynamic stream, uses $\tilde{O}(T \text{poly}(d \log(\delta^{-1} \Delta)))$ space to sample a random subset $G \subseteq X$ of size $\tilde{O}(T \text{poly}(d \log(\delta^{-1} \Delta)))$ such that, for any subset $Y \subseteq X$ with $|Y| \leq T$, $\Pr[|\varphi(X \setminus G)| \leq 2 \cdot |\varphi(X \setminus Y)|] \geq 1 - \delta$.*

Firstly, let us consider an *offline* two-level sampling procedure: we first choose $y \in \varphi(X)$ uniformly at random (u.a.r.), then choose u.a.r. a point $x \in \varphi^{-1}(y) \cap X$. Then, G is constructed by repeating the two-level sampling to sample points from X *without replacement*.

Lemma C.1. *Let G contain $O(T \log(T\delta^{-1}))$ points, where the points are sampled by the two-level uniform sampling on X without replacement. Then for any subset $F \subseteq X$ with $|F| \leq T$, it holds that $\Pr[|\varphi(X \setminus G)| \leq 2 \cdot |\varphi(X \setminus F)|] \geq 1 - \delta$.*

Proof. **TOPROVE 20** □

Streaming Implementation We utilize an implementation proposed by [CFJ⁺22].

Lemma C.2 (Two-level ℓ_0 -sampler, [CFJ⁺22, Lemma 3.3]). *There is a randomized algorithm, that given as input a matrix $M \in \mathbb{R}^{M \times N}$, with $M \leq N$ and integer entries bounded by $\text{poly}(N)$, that is present a stream of additive entry-wise updates, returns an index-pair (i, j) of M , where i is chosen u.a.r. from the non-zero rows, and then j is chosen u.a.r. from the non-zero columns in that row i . The algorithm uses space $\text{poly}(\log(\delta^{-1}N))$ and fails with probability at most δ .*

Recall that for any point $x \in \mathbb{R}^d$, the value of $\varphi(x)$ can be computed in space $\text{poly}(d)$. Hence, the image $\varphi([\Delta]^d)$ has a size of at most $\Delta^d \cdot 2^{\text{poly}(d)} \leq \Delta^{\text{poly}(d)}$, since we can encode the input point and the computation process of $\varphi(x)$ by a binary string of length $O(d \log \Delta) + \text{poly}(d)$. Therefore, we assume without loss of generality that the image of φ is $[\Delta^{\text{poly}(d)}]$ instead of \mathbb{R}^d .

We convert each update to the dataset given by the stream into an update to a frequency matrix M , where rows correspond to all images of $\varphi([\Delta]^d)$, and columns correspond to all points in $[\Delta]^d$. This leads to $M \leq N \leq \Delta^{\text{poly}(d)}$. To achieve sampling $\sigma := O(T \log(T\delta^{-1}))$ points without replacement, we maintain σ independent two-level ℓ_0 -samplers l_1, \dots, l_σ during the stream simultaneously using Lemma C.2 with each failing with probability δ/σ . At the end of stream, we begin to sample. For the i -th sampling, we use l_i to obtain an index pair $(\varphi(x_i), x_i)$ (in case l_i fails, no action is taken), and then we update l_{i+1}, \dots, l_σ by decreasing the frequency of $(\varphi(x_i), x_i)$ by 1, which means we remove the point x_i from the dataset. By union bound, the probability that all samplers succeed is at least $1 - \delta$, and hence, by Lemma C.1, the sampled $G := \{x_1, \dots, x_\sigma\}$ satisfies $|\varphi(X \setminus G)| \leq 2 \cdot |\varphi(X \setminus F)|$ for a fixed F with probability at least $1 - 2\delta$. We finish the proof by rescaling δ .

C.2 Proof of Lemma 5.5

Lemma 5.5 (Identify Sparse Subsets). *There exists a streaming algorithm that, given integers $0 < \delta < 1$, $N, M > 0$, a mapping $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that the value for any point in \mathbb{R}^d can be evaluated in space $\text{poly}(d)$, and a dataset $X \subseteq [\Delta]^d$ represented as a dynamic stream, returns a collection of subsets of X or \perp . If $|\mathcal{P}_\varphi(X)| \leq N$, the algorithm returns $\{P \in \mathcal{P}_\varphi(X) : |P| \leq M\}$; otherwise, it returns \perp . The algorithm uses space $\tilde{O}(NM \cdot \text{poly}(d \log(\delta^{-1}\Delta)))$ and fails with probability at most δ .*

Similarly, we assume that the image of φ is $[\Delta^{\text{poly}(d)}]$ instead of \mathbb{R}^d . We first present an offline algorithm in Algorithm 5 and then we discuss how to implement it in dynamic streams. Assume the algorithm succeeds in returning a collection \mathcal{S} of subsets, as can be seen in Line 9, we have that, for every $y \in \varphi(X)$, there exists $i \in [w]$ such that $B_{h_i(y)}^{(i)} = \varphi^{-1}(y) \cap X$. This implies that we recover the partition $\mathcal{P}_\varphi(X) = \{\varphi^{-1}(y) \cap X : y \in \varphi(X)\}$ of X induced by φ exactly, and return $\mathcal{S} := \{P \in \mathcal{P}_\varphi(X) : |P| \leq M\}$, hence achieving the guarantee of Lemma 5.5.

Streaming Implementation Before the stream starts, the algorithm builds the 2-universal hash functions h_1, \dots, h_w , which can be implemented using space $\text{poly}(\log(\Delta^{\text{poly}(d)})) = \text{poly}(d \log \Delta)$. Once the hash functions have been constructed, it becomes straightforward to update and maintain $B_j^{(i)}$ for $i \in [w], j \in [2N]$ and $\varphi(X)$ during the stream if there is no space limit. When aiming to manage them with a limited space, we first notice that we actually need a subroutine that exactly maintains $B_j^{(i)}$ (or $\varphi(X)$) only when its size does not exceed M (or N , respectively). In cases where the size exceeds the threshold, it suffices to instead return a symbol, such as \perp , to indicate this. Therefore, we can similarly consider the task of maintaining $B_j^{(i)}$ as a sparse recovery problem, where the goal is to exactly recover a vector $v \in [0, 1]^X$ with $v_x = \mathbf{1}[x \in B_j^{(i)}]$ ⁵ when the support of

⁵ $\mathbf{1}[\mathcal{E}]$ is an indicator variable which equals 1 if the event \mathcal{E} happens, and equals 0 otherwise.

Algorithm 5 Identify light parts (offline)

```

1: let  $w \leftarrow \Theta(\log(\delta^{-1}N))$ 
2: let  $h_1, \dots, h_w : [\Delta^{\text{poly}(d)}] \rightarrow [2N]$  be independent 2-universal hash functions.
3: for  $i \in [w], j \in [2N]$  do
4:   let  $B_j^{(i)} \leftarrow \{x \in X : h_i(\varphi(x)) = j\}$ 
5: end for
6: return  $\perp$  if  $|\varphi(X)| > N$ 
7:  $\mathcal{S} \leftarrow \emptyset$ 
8: for  $y \in \varphi(X)$  do
9:   find  $i \in [w]$  s.t.  $\forall y' \in \varphi(X), y' \neq y$ , it holds that  $h_i(y') \neq h_i(y)$ 
10:  algorithm fails if no such  $i$  exists
11:  if  $|B_{h_i(y)}^{(i)}| \leq M$ , let  $\mathcal{S} \leftarrow \mathcal{S} \cup \{B_{h_i(y)}^{(i)}\}$   $\triangleright B_{h_i(y)}^{(i)} = \varphi^{-1}(y)$ 
12: end for
13: return  $X_{\mathcal{S}}$ 

```

v is no greater than M .

Lemma C.3 (Sparse recovery, [CM06]). *There exists a streaming algorithm that, given $0 < \delta < 1$, integers $I, J, K \geq 1$, and a frequency vector $v \in [-J, J]^I$ presented as a dynamic stream, where we denote its support by $\text{supp}(v) := \{i \in [I], v_i \neq 0\}$, uses space $O(K \cdot \text{poly}(\log(\delta^{-1}IJ)))$. If $|\text{supp}(v)| \leq K$, then with probability $1 - \delta$, the algorithm returns all the elements in the support $\text{supp}(v)$ and their frequencies. Otherwise, it returns \perp .*

As a result, we can run $2Nw$ instances of the sparse recovery algorithm of Lemma C.3 in parallel, with each instance corresponding to one of the sets $B_i^{(j)}$. Similarly, we can employ a separate instance for the purpose of maintaining $\varphi(X)$. The total space is $2Nw \cdot M \text{poly}(d \log(\delta^{-1}\Delta)) + N \text{poly}(d \log(\delta^{-1}\Delta)) = \tilde{O}(NM \text{poly}(d \log(\delta^{-1}\Delta)))$. The remaining steps from Line 7 to Line 13 are easy to implement after the stream ends.

Success Probability The algorithm fails if and only if one of the instances of sparse recovery fails, or it fails to find an $i \in [w]$ satisfying the condition of Line 9 for some $y \in \varphi(X)$. For the former, the probability that one instance of sparse recovery fails can be reduced to $\delta/(2Nw + 1)$, at the cost of increasing the space by factor of $O(\log(Nw)) = O(\log N)$. Therefore, applying union bound, the probability that one of the $(2Nw + 1)$ instances fails is at most δ .

As for the latter, we first consider a fixed $y \in \varphi(X)$, and $i \in [w]$, the probability of the collision of $h_i(y) = h_i(y')$ for some $y' \in \varphi(X) \setminus \{y\}$ is

$$\Pr[h_i(y) = h_i(y')] \leq \frac{1}{2N}.$$

By union bound over all $y' \in \varphi(X) \setminus \{y\}$, we have

$$\Pr[\exists y' \in \varphi(X) \setminus \{y\}, h_i(y) = h_i(y')] \leq \frac{|\varphi(X)| - 1}{2N} \leq \frac{1}{2}$$

Recall that h_1, \dots, h_w are independent hash function, then the probability of that such event happens for every $i \in [w]$ simultaneously is at most $(1/2)^w \leq \delta/N$. As a result, we can find a desired $i \in [w]$ for $y \in \varphi(X)$ that satisfies the condition of Line 9 with probability at least $1 - \delta/N$. Applying union bound again, we have that with probability at least $1 - \delta$, we can find a desired $i \in [w]$ for every $y \in \varphi(X)$ simultaneously. Overall, the success probability of the streaming version of Algorithm 5 is at least $1 - 2\delta$. It remains to scale δ by a constant.

D Streaming Lower Bounds Based on INDEX

Claim D.1. *For every integer $m \geq 10$, any algorithm that, with constant probability, computes an m -approximation $g > 0$ to the optimal objective for $(1, m)$ -MEDIAN of a dataset $X \subseteq [2m^{10}]^2$ presented as an insertion-only point stream must use space $\Omega(m)$.*

We prove Claim [D.1](#) based on the INDEX problem, as stated below.

Definition D.2 (INDEX problem). Alice is given a vector $x \in \{0, 1\}^n$, and Bob is given an index $i \in [n]$. Alice can send Bob exactly one message M , and Bob needs to use his input i and this message M to compute $x_i \in \{0, 1\}$.

The well-known fact is that achieving constant probability success in the INDEX problem requires a communication complexity of at least $|M| = \Omega(n)$ (see e.g., [[KN97](#), [KNR99](#), [JKS08](#)]).

Proof. [TOPROVE 21](#) □