Adversarial Robustness of Streaming Algorithms through Importance Sampling

Anonymous Author(s)
Affiliation
Address
email

Abstract

Robustness against adversarial attacks has recently been at the forefront of algorithmic design for machine learning tasks. In the adversarial streaming model, an adversary gives an algorithm a sequence of adaptively chosen updates $u_1, \ldots, u_n$ as a data stream. The goal of the algorithm is to compute or approximate some predetermined function for every prefix of the adversarial stream, but the adversary may generate future updates based on previous outputs of the algorithm. In particular, the adversary may gradually learn the random bits internally used by an algorithm to manipulate dependencies in the input. This is especially problematic as many important problems in the streaming model require randomized algorithms, as they are known to not admit any deterministic algorithms that use sublinear space. In this paper, we introduce adversarially robust streaming algorithms for central machine learning and algorithmic tasks, such as regression and clustering, as well as their more general counterparts, subspace embedding, low-rank approximation, and coreset construction. For regression and other numerical linear algebra related tasks, we consider the row arrival streaming model. Our results are based on a simple, but powerful, observation that many importance sampling-based algorithms give rise to adversarial robustness which is in contrast to sketching based algorithms, which are very prevalent in the streaming literature but suffer from adversarial attacks. In addition, we show that the well-known merge and reduce paradigm in streaming is adversarially robust. Since the merge and reduce paradigm allows coreset constructions in the streaming setting, we thus obtain robust algorithms for k-means, k-median, k-center, Bregman clustering, projective clustering, principal component analysis (PCA) and non-negative matrix factorization. To the best of our knowledge, these are the first adversarially robust results for these problems yet require no new algorithmic implementations. Finally, we empirically confirm the robustness of our algorithms on various adversarial attacks and demonstrate that by contrast, some common existing algorithms are not robust.

1 Introduction

Robustness against adversarial attacks have recently been at the forefront of algorithmic design for machine learning tasks [GSS15, CW17, AEIK18, MMS+18, TSE+19]. We extend this line of work by studying adversarially robust streaming algorithms.

In the streaming model, data points are generated one at a time in a stream and the goal is to compute some meaningful function of the input points while using a limited amount of memory, typically *sublinear* in the total size of the input. The streaming model is applicable in many algorithmic and ML related tasks where the size of the data far exceeds the available storage. Applications of the streaming model include monitoring IP traffic flow, analyzing web search queries [LMV+16],

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

processing large scientific data, feature selection in machine learning [HZZ19, GRB+19, WYWD10], and estimating word statistics in natural language processing [GDC12] to name a few. Streaming algorithms have also been implemented in popular data processing libraries such as Apache Spark which have implementations for streaming tasks such as clustering and linear regression [ZXW+16a].

In the adversarial streaming model [MBN+17, BMSC17, AMYZ19, BY20, BJWY20, HKM+20, WZ20, ABD+21, KMNS21], an adversary gives an algorithm a sequence of adaptively chosen updates $u_1, \ldots, u_n$ as a data stream. The goal of the algorithm is to compute or approximate some predetermined function for every prefix of the adversarial stream, but the adversary may generate future updates based on previous outputs of the algorithm. In particular, the adversary may gradually learn the random bits internally used by an algorithm to manipulate dependencies in the input. This is especially problematic as many important problems in the streaming model require randomized algorithms, as they are known to not admit any deterministic algorithms that use sublinear space. Studying when adversarially robust streaming algorithms are possible is an important problem in lieu of recent interest in adversarial attacks in ML with applications to adaptive data analysis.

Related Works. Adversarial robustness of streaming algorithms has been an important topic of recent research. On the positive note, [BJWY20] gave a robust framework for estimating the L_p norm of points in a stream in the insertion-only model, where previous stream updates cannot later be deleted. Their work thus shows that deletions are integral to the attack of [HW13]. Subsequently, [HKM+20] introduced a new algorithmic design for robust L_p norm estimation algorithms, by using differential privacy to protect the internal randomness of algorithms against the adversary. Although [WZ20] tightened these bounds, showing that essentially no losses related to the size of the input n or the accuracy parameter ε were needed, [KMNS21] showed that this may not be true in general. Specifically, they showed a separation between oblivious and adversarial streaming in the adaptive data analysis problem.

[BY20] showed that sampling is not necessarily adversarially robust; they introduce an exponentially sized set system where a constant number of samples, corresponding to the VC-dimension of the set system, may result in a very unrepresentative set of samples. However, they show that with an additional logarithmic overhead in the number of samples, then Bernoulli and or reservoir sampling are adversarially robust. This notion is further formalized by [ABD+21], who showed that the classes that are online learnable requires essentially sample-complexity proportional to the Littlestone's dimension of the underlying set system, rather than VC dimension. However, these sampling procedures are uniform in the sense that each item in the stream is sampled with the same probability. Thus the sampling probability of each item is *oblivious* to the identity of the item. By contrast, we show the robustness for a variety of algorithms based on *non-oblivious* sampling, where each stream item is sampled with probability roughly proportional to the "importance" of the item.

1.1 Our Contributions

Our main contribution is a powerful yet simple statement that algorithms based on non-oblivious sampling are adversarially robust if informally speaking, the process of sampling each item in the stream can be viewed as using fresh randomness independent of previous steps, even if the sampling probabilities depend on previous steps.

Let us describe, very informally, our meta-approach. Suppose we have an adversarial stream of elements given by $u_1, \ldots, u_n$. Our algorithm $\mathcal{A}$ will maintain a data structure A_t at time t which updates as the stream progresses. $\mathcal{A}$ will use a function $g(A_t, u_t)$ to determine the probability of sampling item u_t to update A_t to A_{t+1}. The function g measures the 'importance' of the element u_t to the overall problem that we wish to solve. For example, if our application is k-means clustering and u_t is a point far away from all previously seen points so far, we want to sample it with a higher probability. We highlight that even though the sampling probability for u_t given by $g(A_t, u_t)$ is adversarial, since the adversary designs u_t and previous streaming elements, the *coin toss* performed by our algorithm $\mathcal{A}$ to keep item u_t is *independent* of any events that have occurred so far, including the adversary's actions. This new randomness introduced by the independent coin toss is a key conceptual step in the analysis for all of the applications listed in Figure 1.

Contrast this to the situation where a "fixed" data structure or sketch is specified upfront. In this case, we would not be adaptive to which inputs u_t the adversary designs to be "important" for our problem which would lead us to potentially disregard such important items rendering the algorithm ineffective.

As applications of our meta-approach, we introduce adversarially robust streaming algorithms for two central machine learning tasks, regression and clustering, as well as their more general counterparts, subspace embedding, low-rank approximation, and coreset construction.

We show that several methods from the streaming algorithms "toolbox", namely merge and reduce, online leverage score sampling, and edge sampling are adversarially robust "for free." As a result, *existing* (and future) streaming algorithms that use these tools are robust as well. We discuss our results in more detail below and provide a summary of our results and applications in Figure 1.

Meta-approach	Applications
Merge and reduce (Theorem 1.1)	Coreset construction, support vector machine, Gaussian mixture models, k-means clustering, k-median clustering, projective clustering, principal component analysis, M-estimators, Bayesian logistic regression, generative adversarial networks (GANs), k-line center, j-subspace approximation, Bregman clustering
Row sampling (Theorem 1.2)	Linear regression, generalized regression, spectral approximation, low-rank approximation, projection-cost preservation, L_1-subspace embedding
Edge sampling (Theorem 1.3)	Graph sparsification

Fig. 1: Summary of our robust sampling frameworks and corresponding applications

We first show that the well-known merge and reduce paradigm is adversarially robust. Since the merge and reduce paradigm defines coreset constructions, we thus obtain robust algorithms for k-means, k-median, Bregman clustering, projective clustering, principal component analysis (PCA), non-negative matrix factorization (NNMF) [LK17].

Theorem 1.1 (Merge and reduce is adversarially robust) *Given an offline ε-coreset construction, the merge and reduce framework gives an adversarially robust streaming construction for an ε-coreset with high probability.*

For regression and other numerical linear algebra related tasks, we consider the row arrival streaming model, in which the adversary generates a sequence of row vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$ in d-dimensional vector space. For $t \in [n]$, the t-th prefix of the stream induces a matrix $\mathbf{A}_t \in \mathbb{R}^{t \times d}$ with rows $\mathbf{a}_1, \ldots, \mathbf{a}_t$. We denote this matrix as $\mathbf{A}_t = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_t$ and define κ to be an upper bound on the largest condition number[1] of the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_n$.

Theorem 1.2 (Row sampling is adversarially robust) *There exists a row sampling based framework for adversarially robust streaming algorithms that at each time $t \in [n]$:*

(1) Outputs a matrix $\mathbf{M}_t$ such that $(1-\varepsilon)\mathbf{A}_t^\top \mathbf{A}_t \preceq \mathbf{M}_t^\top \mathbf{M}_t \preceq (1+\varepsilon)\mathbf{A}_t^\top \mathbf{A}_t$, while sampling $\mathcal{O}\left(\frac{d^2\kappa}{\varepsilon^2} \log n \log \kappa\right)$ rows (spectral approximation/subspace embedding/linear regression/generalized regression).

(2) Outputs a matrix $\mathbf{M}_t$ such that for all rank k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d \times d}$,

$$(1-\varepsilon)\left\|\mathbf{A}_t - \mathbf{A}_t\mathbf{P}\right\|_F^2 \leq \left\|\mathbf{M}_t - \mathbf{M}_t\mathbf{P}\right\|_F^2 \leq (1+\varepsilon)\left\|\mathbf{A}_t - \mathbf{A}_t\mathbf{P}\right\|_F^2,$$

while sampling $\mathcal{O}\left(\frac{dk\kappa}{\varepsilon^2} \log n \log^2 \kappa\right)$ rows (projection-cost preservation/low-rank approximation).

(3) Outputs a matrix $\mathbf{M}_t$ such that $(1-\varepsilon)\left\|\mathbf{A}_t\mathbf{x}\right\|_1 \leq \left\|\mathbf{M}_t\mathbf{x}\right\|_1 \leq (1+\varepsilon)\left\|\mathbf{A}_t\mathbf{x}\right\|_1$, while sampling $\mathcal{O}\left(\frac{d^2\kappa}{\varepsilon^2} \log^2 n \log \kappa\right)$ rows (L_1 subspace embedding).

Finally, we show that our analysis also applies to algorithms for graph sparsification for in which edges are sampled according to their "importance". Define κ as the ratio of the largest and the smallest cut sizes in G (see Section 4 and Supplementary Section C for exact details).

[1] the ratio of the largest and smallest nonzero singular values

Theorem 1.3 *Given a weighted graph $G = (V, E)$ with $|V| = n$ whose edges $e_1, \ldots, e_m$ arrive sequentially in a stream, there exists an adversarially robust streaming algorithm that outputs a $1 \pm \varepsilon$ cut sparsifier with $\mathcal{O}\left(\frac{\kappa^2 n \log n}{\varepsilon^2}\right)$ edges with probability $1 - 1/\operatorname{poly}(n)$.*

Sketching vs Sampling Algorithms. A central tool for randomized streaming algorithms is the use of linear sketches. These methods maintain a data structure f such that after the $(i+1)$-th input x_i, we can update f by computing a linear function of x_i. Typically, these methods employ a random matrix. For example, if the input consists of vectors, sketching methods will use a random matrix to project the vector into a much smaller dimension space. In [HW13], it was proved no linear sketch can approximate the L_2-norm within a polynomial multiplicative factor against such an adaptive adversary. In general, streaming algorithms that use sketching are highly susceptible to the type of attack described in [HW13] where the adversary can effectively 'learn' the kernel of the linear function used and send inputs along the kernel. For example, if an adversary knows the kernel of the random matrix used to project the input points, then by sending points that lie on the kernel of the matrix as inputs, the adversary can render the whole streaming algorithm useless.

One the other hand, we employ a different family of streaming algorithms that are based on *sampling* the input rather than *sketching* it. Surprisingly, this simple change allows one to automatically get many adversarially robust algorithms either "for free" or *without* new algorithmic overheads. For more information, see Section 1.1. We emphasize that while our techniques are not theoretically sophisticated, we believe its power lies in its simple message that **sampling is often superior to sketching for adversarial robustness**. In addition to downstream algorithmic and ML applications, this provides an interesting separation and trade-offs between the two paradigms; for non adversarial inputs sketching often gives similar or better performance guarantees for many tasks [BYKS01].

2 Merge and Reduce

We show that the general merge and reduce paradigm is adversarially robust. Merge and reduce is widely used for the construction of a coreset, which provides dimensionality reduction on the size of an underlying dataset, so that algorithms for downstream applications can run more efficiently:

Definition 2.1 (ε **coreset**) *Let $P \subset X$ be a set of elements from a universe X, $z \geq 0$, $\varepsilon \in (0, 1)$, and $(P, \operatorname{dist}, Q)$ be a query space. Then a subset C equipped with a weight function $w : P \to \mathbb{R}$ is called an ε-coreset with respect to the query space $(P, \operatorname{dist}, Q)$ if*

$$(1-\varepsilon) \sum_{\mathbf{p} \in P} \operatorname{dist}(\mathbf{p}, Q)^z \leq \sum_{\mathbf{p} \in C} w(p) \operatorname{dist}(\mathbf{p}, Q)^z \leq (1+\varepsilon) \sum_{\mathbf{p} \in P} \operatorname{dist}(\mathbf{p}, Q)^z.$$

The study of efficient offline coreset constructions for a variety of geometric and algebraic problems forms a long line of active research. For example, offline coreset constructions are known for linear regression, low-rank approximation, L_1-subspace embedding, k-means clustering, k-median clustering, k-center, support vector machine, Gaussian mixture models, M-estimators, Bregman clustering, projective clustering, principal component analysis, k-line center, j-subspace approximation, and so on. Thus, our result essentially shows that using the merge and reduce paradigm, these offline coreset constructions can be extended to obtain robust and accurate streaming algorithms. The merge and reduce paradigm works as follows. Suppose we have a stream $p_1, \ldots, p_n$ of length $n = 2^k$ for some

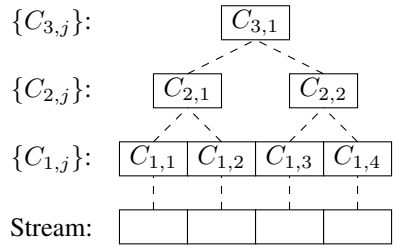

Fig. 2: Merge and reduce framework. Each $C_{1,j}$ is an $O(\varepsilon/\log n)$-coreset of the corresponding partition of the substream and each $C_{i,j}$ is an ε-coreset of $C_{i-1,2j-1}$ and $C_{i-1,2j}$ for $i > 1$.

integer k, without loss of generality (otherwise we can use a standard padding argument to increase the length of the stream). Define $C_{0,j} = p_j$ for all $j \in [n]$. Consider k levels, where each level $i \in [k]$ consists of $\frac{n}{2^i}$ coresets $C_{i,1}, \ldots, C_{i,n/2^i}$ and each coreset $C_{i,j}$ is an $\frac{\varepsilon}{2k}$-coreset of $C_{i-1,2j-1}$ and $C_{i-1,2j}$. Note that this approach can be implemented efficiently in the streaming model, since each $C_{i,j}$ can be built immediately once $C_{i-1,2j-1}$ and $C_{i-1,2j}$ are constructed, and after $C_{i,j}$ is constructed, then both $C_{i-1,2j-1}$ and $C_{i-1,2j}$ can be discarded. For an illustration of the merge and reduce framework, see Figure 2, though we defer all formal proofs to the supplementary material. Using the coresets of [BDM+20], Theorem 1.1 gives the following applications:

Theorem 2.2 *There exists a merge-and-reduce row sampling based framework for adversarially robust streaming algorithms that at each time $t \in [n]$:*

(1) Outputs a matrix $\mathbf{M}_t$ such that $(1-\varepsilon)\mathbf{A}_t^\top \mathbf{A}_t \preceq \mathbf{M}_t^\top \mathbf{M}_t \preceq (1+\varepsilon)\mathbf{A}_t^\top \mathbf{A}_t$, while sampling $\mathcal{O}\left(\frac{d^2}{\varepsilon^2}\log^4 n \log \kappa\right)$ rows (spectral approximation/subspace embedding/linear regression/generalized regression).

(2) Outputs a matrix $\mathbf{M}_t$ such that for all rank k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d\times d}$,

$$(1-\varepsilon)\left\|\mathbf{A}_t - \mathbf{A}_t\mathbf{P}\right\|_F^2 \leq \left\|\mathbf{M}_t - \mathbf{M}_t\mathbf{P}\right\|_F^2 \leq (1+\varepsilon)\left\|\mathbf{A}_t - \mathbf{A}_t\mathbf{P}\right\|_F^2,$$

while sampling $\mathcal{O}\left(\frac{k}{\varepsilon^2}\log^4 n \log^2 \kappa\right)$ rows (projection-cost preservation/low-rank approximation).

(3) Outputs a matrix $\mathbf{M}_t$ such that $(1-\varepsilon)\left\|\mathbf{A}_t\mathbf{x}\right\|_1 \leq \left\|\mathbf{M}_t\mathbf{x}\right\|_1 \leq (1+\varepsilon)\left\|\mathbf{A}_t\mathbf{x}\right\|_1$, while sampling $\mathcal{O}\left(\frac{d}{\varepsilon^2}\log^5 n \log \kappa\right)$ rows (L_1 subspace embedding).

Using coresets of [HV20], then Theorem 1.1 also gives applications for (k,z)-clustering such as k-median for $z=1$ and k-means for $z=2$. Moreover, [LK17] noted that constructions of [FL11] give coresets for Bregman clustering, which handles μ-similar Bregman divergences such as the Itakura-Saito distance, KL-divergence, Mahalanobis distance, etc.

Theorem 2.3 *There exists a merge-and-reduce importance sampling based framework for adversarially robust streaming algorithms that at each time t:*

(1) Outputs a set of centers that gives a $(1+\varepsilon)$-approximation to the optimal (k,z)-clustering, k-means clustering ($z=2$), and k-median clustering ($z=1$), while storing $\mathcal{O}\left(\frac{1}{\varepsilon^{2z+2}} k \log^{2z+2} n \log k \log \frac{k\log n}{\varepsilon}\right)$ points.

(2) Outputs a set of centers that gives a $(1+\varepsilon)$-approximation to the optimal k-Bregman clustering, while storing $\mathcal{O}\left(\frac{1}{\varepsilon^2} dk^3 \log^3 n\right)$ points.

Using the sensitivity bounds of [VX12a, VX12b] and the coreset constructions of [BFL16], then Theorem 1.1 also gives applications for the following shape fitting problems:

Theorem 2.4 *There exists a merge-and-reduce importance sampling based framework for adversarially robust streaming algorithms that at each time t:*

(1) Outputs a set of lines that gives a $(1+\varepsilon)$-approximation to the optimal k-lines clustering, while storing $\mathcal{O}\left(\frac{d}{\varepsilon^2} f(d,k) k^{f(d,k)} \log^4 n\right)$ points of $\mathbb{R}^d$, for a fixed function $f(d,k)$.

(2) Outputs a subspace that gives a $(1+\varepsilon)$-approximation to the optimal dimension j subspace approximation, while storing $\mathcal{O}\left(\frac{d}{\varepsilon^2} g(d,j) k^{g(d,j)} \log^4 n\right)$ points of $\mathbb{R}^d$, for a fixed function $g(d,j)$.

(3) Outputs a set of subspaces that gives a $(1+\varepsilon)$-approximation to the optimal (j,k)-projective clustering, while storing $\mathcal{O}\left(\frac{d}{\varepsilon^2} h(d,j,k) \log^3 n (\log n)^{h(d,j,k)}\right)$ points of $\mathbb{R}^d$s, for a fixed function $h(d,j,k)$, for a set of input points with integer coordinates.

Adversarially robust approximation algorithms for Bayesian logistic regression, Gaussian mixture models, generative adversarial networks (GANs), and support vector machine can be obtained

from Theorem 1.1 and coreset constructions of [HCB16, FKW19, SZG+20, TBFR20]; a significant number of additional applications of Theorem 1.1 using coreset constructions can be seen from recent surveys on coresets, e.g., see [LK17, Fel20].

3 Adversarial Robustness of Subspace Embedding and Applications

We use $[n]$ to represent the set $\{1, \ldots, n\}$ for an integer $n > 0$. We typically use bold font to denote vectors and matrices. For a matrix $\mathbf{A}$, we use $\mathbf{A}^{-1}$ to denote the Moore-Penrose inverse of $\mathbf{A}$. We first formally define the goals of our algorithms:

Problem 3.1 (Spectral Approximation) *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and an approximation parameter $\varepsilon > 0$, the goal is to output a matrix $\mathbf{M} \in \mathbb{R}^{m\times d}$ with $m \ll n$ such that $(1-\varepsilon)\|\mathbf{A}\mathbf{x}\|_2 \leq \|\mathbf{M}\mathbf{x}\|_2 \leq (1+\varepsilon)\|\mathbf{A}\mathbf{x}\|_2$ for all $\mathbf{x} \in \mathbb{R}^d$ or equivalently, $(1-\varepsilon)\mathbf{A}^\top\mathbf{A} \preceq \mathbf{M}^\top\mathbf{M} \preceq (1+\varepsilon)\mathbf{A}^\top\mathbf{A}$.*

We note that linear regression is a well-known specific application of spectral approximation.

Problem 3.2 (Projection-Cost Preservation) *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, a rank parameter $k > 0$, and an approximation parameter $\varepsilon > 0$, the goal is to find a matrix $\mathbf{M} \in \mathbb{R}^{m\times d}$ with $m \ll n$ such that for all rank k orthogonal projection matrices $\mathbf{P} \in \mathbb{R}^{d\times d}$,*

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

Note if $\mathbf{M}$ is a projection-cost preservation of $\mathbf{A}$, then its best low-rank approximation can be used to find a projection matrix that gives an approximation of the best low-rank approximation to $\mathbf{A}$.

Problem 3.3 (Low-Rank Approximation) *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, a rank parameter $k > 0$, and an approximation parameter $\varepsilon > 0$, find a rank k matrix $\mathbf{M} \in \mathbb{R}^{n\times d}$ such that $(1-\varepsilon)\left\|\mathbf{A}-\mathbf{A}_{(k)}\right\|_F^2 \leq \|\mathbf{A}-\mathbf{M}\|_F^2 \leq (1+\varepsilon)\left\|\mathbf{A}-\mathbf{A}_{(k)}\right\|_F^2$, where $\mathbf{A}_{(k)}$ for a matrix $\mathbf{A}$ denotes the best rank k approximation to $\mathbf{A}$.*

Problem 3.4 (L_1-Subspace Embedding) *Given a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and an approximation parameter $\varepsilon > 0$, the goal is to output a matrix $\mathbf{M} \in \mathbb{R}^{m\times d}$ with $m \ll n$ such that $(1-\varepsilon)\|\mathbf{A}\mathbf{x}\|_1 \leq \|\mathbf{M}\mathbf{x}\|_1 \leq (1+\varepsilon)\|\mathbf{A}\mathbf{x}\|_1$ for all $\mathbf{x} \in \mathbb{R}^d$.*

We consider the general class of row sampling algorithms, e.g., [CMP16, BDM+20]. Here we maintain a L_p subspace embedding of the underlying matrix by approximating the online L_p sensitivities of each row as a measure of importance to perform sampling. For more details, see Algorithm 1.

Definition 3.5 (Online L_p Sensitivities) *For a matrix $\mathbf{A} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_n \in \mathbb{R}^{n\times d}$, the online sensitivity of row $\mathbf{a}_i$ for each $i \in [n]$ is the quantity $\max_{\mathbf{x}\in\mathbb{R}^d} \frac{|\langle \mathbf{a}_i, \mathbf{x}\rangle|^p}{\|A_i\mathbf{x}\|_p^p}$, where $\mathbf{A}_{i-1} = \mathbf{a}_1 \circ \ldots \circ \mathbf{a}_{i-1}$.*

Algorithm 1 Row sampling based algorithms, e.g., [CMP16, BDM+20]

Input: A stream of rows $\mathbf{a}_1, \ldots, \mathbf{a}_n \in \mathbb{R}^d$, parameter $p > 0$, and an accuracy parameter $\varepsilon > 0$
Output: A $(1+\varepsilon)$ L_p subspace embedding.

```
M ← ∅
α ← (Cd/ε²) log n with sufficiently large parameter C > 0
for each row a_i, i ∈ [n] do
    if a_i ∈ span(M) then
        τ_i ← 2 · max_{x∈ℝ^d, x∈span(M)} |⟨a_i, x⟩|^p / (||Mx||_p^p + |⟨a_i, x⟩|^p)      ▷See Remark 3.6
    else
        τ_i ← 1
    p_i ← min(1, ατ_i)
    With probability p_i, M ← M ∘ a_i / p_i^{1/p}      ▷Online sensitivity sampling
return M
```

We remark on standard computation or approximation of the online L_p sensitivities, e.g., see [CEM+15, CMP16, CMM17, BDM+20].

Remark 3.6 *We note that for $p = 1$, a constant fraction approximation to any online L_p sensitivity τ_i such that $\tau_i > \frac{1}{\mathrm{poly}(n)}$ can be computed in polynomial time using (offline) linear programming while for $p = 2$, τ_i is equivalent to the online leverage score of $\mathbf{a}_i$, which has the closed form expression $\mathbf{a}_i^\top (\mathbf{A}_i^\top \mathbf{A}_i)^{-1} \mathbf{a}_i$, which can be approximated by $\mathbf{a}_i^\top (\mathbf{M}^\top \mathbf{M})^{-1} \mathbf{a}_i$, conditioned on $\mathbf{M}$ being a good approximation to $\mathbf{A}_{i-1}$ when $\mathbf{a}_i$ is in the span of $\mathbf{M}$. Otherwise, τ_i takes value 1 when $\mathbf{a}_i$ is not in the span of $\mathbf{M}$.*

Lemma 3.7 (Adversarially Robust L_p Subspace Embedding and Linear Regression) *Given $\varepsilon > 0$, $p \in \{1, 2\}$, and a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ whose rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$ arrive sequentially in a stream with condition number at most κ, there exists an adversarially robust streaming algorithm that outputs a $(1 + \varepsilon)$ spectral approximation with high probability. The algorithm samples $\mathcal{O}\left(\frac{d^2 \kappa^2}{\varepsilon^2} \log n \log \kappa\right)$ rows for $p = 2$ and $\mathcal{O}\left(\frac{d^2 \lambda^2}{\varepsilon^2} \log^2 n \log \kappa\right)$ rows for $p = 1$, with high probability, where λ is a ratio between upper and lower bounds on $\|\mathbf{A}\|_1$.*

We also show robustness of row sampling for low-rank approximation by using online ridge-leverage scores. Together, Lemma 3.7 and Lemma 3.8 give Theorem 1.2.

Lemma 3.8 (Adversarially Robust Low-Rank Approximation) *Given accuracy parameter $\varepsilon > 0$, rank parameter $k > 0$, and a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ whose rows $\mathbf{a}_1, \ldots, \mathbf{a}_n$ arrive sequentially in a stream with condition number at most κ, there exists an adversarially robust streaming algorithm that outputs a $(1 + \varepsilon)$ low-rank approximation with high probability. The algorithm samples $\mathcal{O}\left(\frac{kd\kappa^2}{\varepsilon^2} \log n \log \kappa\right)$ rows with high probability.*

4 Graph Sparsification

In this section, we highlight how the sampling paradigm gives rise to an adversarially robust streaming algorithm for graph sparsification. First, we motivate the problem of graph sparsification. Massive graphs arise in many theoretical and applied settings, such as in the analysis of large social or biological networks. A key bottleneck in such analysis is the large computational resources, in both memory and time, needed. Therefore, it is desirable to get a representation of graphs that take up far less space while still preserving the underlying "structure" of the graph. Usually the number of vertices is much fewer than the number of edges; for example in typical real world graphs, the number of vertices can be several orders of magnitude smaller than the number of edges (for example, see the graph datasets in [RA15]). Hence, a natural benchmark is to reduce the number of edges to be comparable to the number of vertices.

The most common notion of graph sparsification is that of preserving the value of *all* cuts in the graph by keeping a small weighted set of edges of the original graph. More specifically, suppose our graph is $G = (V, E)$ and for simplicity assume all the edges have weight 1. A cut of the graph is a partition of $V = (C, V \setminus C)$ and the value of a cut, $\mathrm{Val}_G(C)$, is defined as the number of edges that cross between the vertices in C and $V \setminus C$. A graph H on the same set of vertices as V is a sparsifier if it preserves the value of every cut in G and has a few number of weighted edges. For a precise formulation, see Problem 4.1.

In addition to being algorithmically tractable, this formulation is natural since it preserves the underlying cluster structure of the graph. For example, if there are two well connected components separated by a sparse cut, i.e. two distinct communities, then the sparsifier according to the definition above will ensure that the two communities are still well separated. Conversely, by considering any cut within a well connected component, it will also ensure that any community remains well connected (for more details, see [SPR11] and references therein). Lastly, graph sparsification has been considered in other frameworks such as differential privacy [EKKL20], distributed optimization [WWLZ18], and even learning graph sparsification using deep learning methods [ZZC+20]. The formal problem definition of graph sparsification is as follows.

Problem 4.1 (Graph Sparsification) *Given a graph weighted $G = (V, E)$ with $|V| = n, |E| = m$, and an approximation parameter $\varepsilon > 0$, compute a weighted subgraph H of G on the same set of vertices such that*

(1) every cut in H has value between $1-\varepsilon$ and $1+\varepsilon$ times its value in G: $(1-\varepsilon)\mathrm{Val}_G(C) \leq \mathrm{Val}_H(C) \leq (1+\varepsilon)\mathrm{Val}_G(C)$ for all cuts C where $\mathrm{Val}_G(C), \mathrm{Val}_H(C)$ denotes the cost of the cut in the graphs G and H respectively and for the latter quantity, the edges are weighted,

(2) the number of edges in H is $\mathcal{O}\left(\frac{n \log n}{\varepsilon^2}\right)$.

Ignoring dependence on ε, there are previous results that already get sparsifiers H with $O(n \log n)$ edges [BK96, SS08]. Their setting is when the *entire* graph is present up-front in memory. In contrast, we are interested in the streaming setting where future edges can depend on past edges as well as revealed randomness of an algorithm while processing the edges.

Our main goal is to show that the streaming algorithm from [AG09] (presented in Algorithm 2 in the supplementary section), which uses a sampling procedure to sample edges in a stream, is adversarially robust, albeit with a slightly worse guarantee for the number of edges. Following the proof techniques of the non streaming algorithm given in [BK96], it is shown in [AG09] that Algorithm 2 outputs a subgraph H such that H satisfies the conditions of Problem 4.1 with probability $1 - 1/\operatorname{poly}(n)$ where the probability can be boosted by taking a larger constant C. We must show that this still holds true if the edges of the stream are adversarially chosen, i.e., when **new edges in the stream depend on the previous edges and the randomness used by the algorithm so far.** We thus again use a martingale argument; the full details are given in Supplementary Section C. As in Section 3, we let κ_1 and κ_2 to be deterministic lower/upper bounds on the size of any cut in G and define $\kappa = \kappa_2/\kappa_1$.

Theorem 1.3 *Given a weighted graph $G = (V, E)$ with $|V| = n$ whose edges $e_1, \ldots, e_m$ arrive sequentially in a stream, there exists an adversarially robust streaming algorithm that outputs a $1 \pm \varepsilon$ cut sparsifier with $\mathcal{O}\left(\frac{\kappa^2 n \log n}{\varepsilon^2}\right)$ edges with probability $1 - 1/\operatorname{poly}(n)$.*

5 Experiments

To illustrate the robustness of Importance-Sampling-Based streaming algorithms we devise adversarial settings for clustering and linear regression. With respect to our adversarial setting, we show that the performance of a Merge-and-Reduce based streaming K-Means algorithm is robust while a popular streaming K-Means implementation (not based on importance sampling) is not. Similarly, we show the robustness superiority of a Streaming Linear Regression algorithm based on Row-Sampling over a popular Streaming Linear Regression implementation and over Sketching.

Streaming k-means In this adversarial clustering setting we consider a series of point batches where all points except those in the last batch are randomly sampled from a two dimensional standard normal distribution and points in the last batch similarly sampled but around a distant center (see the data points realization in both panels of Figure 3). We then feed the point sequence to `StreamingKMeans`, the streaming K-Means implementation of Spark [ZXW+16b] the popular big-data processing framework. As illustrated in the left panel of Figure 3, the resulting two centers are both within the origin. Now, this result occurs regardless of the last batch's distance from the origin, implying that the per-sample loss performance of the algorithm can be made arbitrarily large. Alternatively, we used a Merge-and-Reduce based streaming K-Means algorithm and show that one of the resulting cluster centers is at the distant cluster (as illustrated in the right panel of Figure 3) thereby keeping the resulting per sample loss at the desired minimum. Specifically we use `Streamkm` an implementation of StreamKM++ [AMR+12] from the ClusOpt Core library [Mac20].

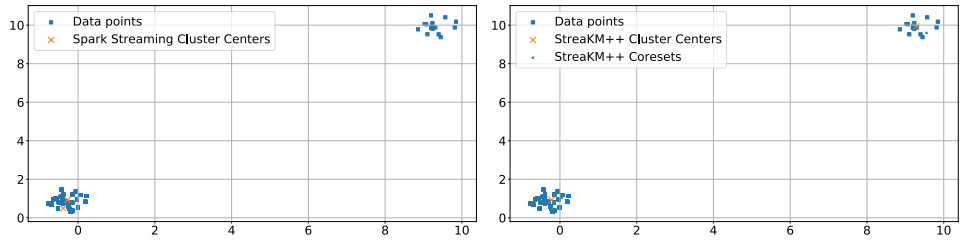

Fig. 3: Resulting cluster centers (x) on our adversarial datapoints setting for the popular Spark implementation (left) and StreamKM++ (right).

Streaming linear regression. Similar to the clustering setting, in the adversarial setting for Streaming Linear Regression all batches except the last one are sampled around a constellation of four points in the plane such that the optimal regression line is of -1 slope through the origin (see the leftmost panel of Figure 4). The last batch of points however, is again far from the origin (L, L) such that the resulting optimal regression line is of slope 1 through the origin[2]. We compare the performance of `LinearRegression` from the popular streaming machine learning library River [MHM+20] to our own Row-Sampling based implementation of streaming linear regression along the lines of Algorithm 1 and observe the following: Without the last batch of points, both implementations result in the optimal regression line, however, the River implementation reaches that line only after several iterations, while our implementation is accurate throughout (This is illustrated in the second-left panel of Figure 4). When the last batch is used, nevertheless, Algorithm 1 picks up the drastic change and adapts immediately to a line of the optimal slope (the blue line of the second right panel of Figure 4) while the River implementation update merely moves the line in the desired direction (the orange line in that same panel) but is far from catching up. Finally, the rightmost panel of Figure 4) details the loss trajectory for both implementations. While the River loss skyrockets upon the last batch, the loss of Algorithm 1 remains relatively unaffected, illustrating its adversarial robustness.

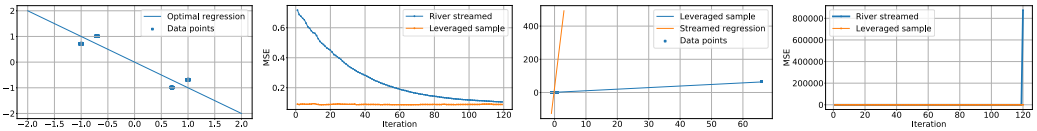

Fig. 4: Streaming Linear Regression experiment (from left to right): points constellation without last batch, loss trajectory up to (not including) last batch, resulting regression lines upon training with last batch, and loss trajectory including the last batch.

Note that in both the clustering and linear regression settings above the adversary was not required to consider the algorithms internal randomization to achieve the desired effect (this is due to the local nature of the algorithms computations). This is no longer the case in the following last setting.

Sampling vs. sketching. Finally, we compare the performance of the leverage sampling Algorithm 1 to Sketching. In this setting, for a random unit sketching matrix S (that is, each of its elements is sampled from $\{-1, 1\}$ with equal probability), we create an adversarial data stream A such that its columns are in the null space of S. As a result, the linear regression as applied to the sketched data $S \cdot A$ as a whole is unstable and might significantly differ from the resulting linear regression applied to streamed prefixes of the sketched data. As illustrated in Figure 5, this is not the case when applying the linear regression to the original streamed data A using Algorithm 1. Upon the last batch, the performance of the sketching-based regression deteriorates by orders of magnitude, while the performance of Algorithm 1 is not affected. Moreover, the data reduction factor achieved by leveraged sampling[3] is almost double compared to the data reduction factor achieved by sketching.

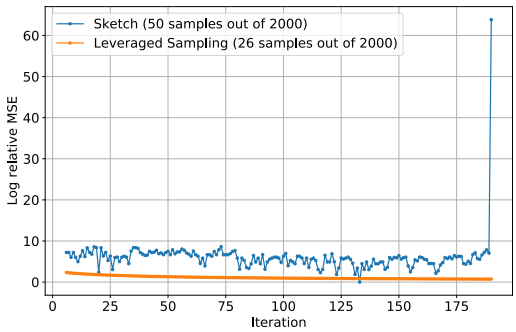

Fig. 5: Comparing the performance trajectory of leveraged sampling Algorithm 1 to Sketching, for an adversarial data stream tailored to a sketching Matrix. Sketching performance deteriorates catastrophically upon the last batch, while leveraged sampling remains robust.

[2]For MSE loss, this occurs for L at least the square root of the number of batches.

[3]The original stream A contained 2000 samples, each of dimension 10.

References

[ABD+21] Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. Adversarial laws of large numbers and optimal regret in online classification. *CoRR*, abs/2101.09054, 2021. 2

[AEIK18] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 284–293, 2018. 1

[AG09] Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Automata, Languages and Programming*, pages 328–338, 2009. 8, 19, 20, 21

[AMR+12] Marcel R. Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *ACM J. Exp. Algorithmics*, 17, May 2012. 8

[AMYZ19] Dmitrii Avdiukhin, Slobodan Mitrovic, Grigory Yaroslavtsev, and Samson Zhou. Adversarially robust submodular maximization under knapsack constraints. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 148–156, 2019. 2

[BDM+20] Vladimir Braverman, Petros Drineas, Cameron Musco, Christopher Musco, Jalaj Upadhyay, David P. Woodruff, and Samson Zhou. Near optimal linear algebra in the online and sliding window models. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 517–528, 2020. 5, 6, 17, 18, 19

[BFL16] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889, 2016. 5

[BJWY20] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*, pages 63–80, 2020. 2

[BK96] A. Benczúr and D. Karger. Approximating s-t minimum cuts in õ(n2) time. In *STOC '96*, 1996. 8, 20, 21

[BMSC17] Ilija Bogunovic, Slobodan Mitrovic, Jonathan Scarlett, and Volkan Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, volume 70, pages 508–516, 2017. 2

[BY20] Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS*, pages 49–62, 2020. 2

[BYKS01] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, page 266–275, 2001. 4

[CEM+15] Michael B. Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank approximation. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC*, pages 163–172, 2015. 6

[CMM17] Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1758–1777, 2017. 6, 19

[CMP16] Michael B. Cohen, Cameron Musco, and Jakub W. Pachocki. Online row sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 7:1–7:18, 2016. 6, 18

[CW17] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP*, pages 39–57. IEEE Computer Society, 2017. 1

[EKKL20] Marek Eliás, Michael Kapralov, Janardhan Kulkarni, and Yin Tat Lee. Differentially private release of synthetic graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 560–578, 2020. 7

[Fel20] Dan Feldman. Introduction to core-sets: an updated survey. *CoRR*, abs/2011.09384, 2020. 6

[FKW19] Dan Feldman, Zahi Kfir, and Xuan Wu. Coresets for gaussian mixture models of any shape. *CoRR*, abs/1906.04895, 2019. 6

[FL11] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC*, pages 569–578. ACM, 2011. 5

[Fre75] David A. Freedman. On tail probabilities for martingales. *the Annals of Probability*, 3(1):100–118, 1975. 16

[GDC12] Amit Goyal, Hal Daumé, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, page 1093–1103, 2012. 2

[GKK10] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. Graph sparsification via refinement sampling. *CoRR*, abs/1004.4915, 2010. 19

[GRB+19] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and João Gama. Machine learning for streaming data: State of the art, challenges, and opportunities. *SIGKDD Explor. Newsl.*, 21(2):6–22, 2019. 2

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015. 1

[HCB16] Jonathan H. Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems*, pages 4080–4088, 2016. 6

[HKM+20] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020. 2

[HV20] Lingxiao Huang and Nisheeth K. Vishnoi. Coresets for clustering in euclidean spaces: importance sampling is nearly optimal. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1416–1429, 2020. 5

[HW13] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *Symposium on Theory of Computing Conference, STOC*, pages 121–130, 2013. 2, 4

[HZZ19] B. Hou, L. Zhang, and Z. Zhou. Learning with feature evolvable streams. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019. 2

[KMNS21] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. Separating adaptive streaming from oblivious streaming. *CoRR*, abs/2101.10836, 2021. 2

[KNST19] Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. *CoRR*, abs/1903.12150, 2019. 20

[LK17] Olivier Bachem Mario Lucic and Andreas Krause. Practical coreset constructions for machine learning. *CoRR*, abs/1703.06476, 2017. 3, 5, 6, 15

[LMV+16] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, page 101–114, 2016. 1

[Mac20] Giuliano Oliveira De Macedo. Github, 2020. 8

[MBN+17] Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4557–4566, 2017. 2

[MHM+20] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: machine learning for streaming data in python, 2020. 9

[MMS+18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR*, 2018. 1

[RA15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. 7

[SPR11] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, page 721–732, 2011. 7

[SS08] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, page 563–568, 2008. 8

[SZG+20] Samarth Sinha, Han Zhang, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, and Augustus Odena. Small-gan: Speeding up GAN training using core-sets. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, pages 9005–9015, 2020. 6

[TBFR20] Murad Tukan, Cenk Baykal, Dan Feldman, and Daniela Rus. On coresets for support vector machines. In *Theory and Applications of Models of Computation, 16th International Conference, TAMC Proceedings*, pages 287–299, 2020. 6

[TSE+19] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *7th International Conference on Learning Representations, ICLR*, 2019. 1

[VX12a] Kasturi R. Varadarajan and Xin Xiao. A near-linear algorithm for projective clustering integer points. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1329–1342, 2012. 5

[VX12b] Kasturi R. Varadarajan and Xin Xiao. On the sensitivity of shape fitting problems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 486–497, 2012. 5

[WWLZ18] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, 2018. 7

[WYWD10] X. Wu, Kui Yu, Hao Wang, and W. Ding. Online streaming feature selection. In *ICML*, 2010. 2

[WZ20] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. *CoRR*, abs/2011.07471, 2020. 2

[ZXW+16a] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016. 2

[ZXW+16b] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. 8

[ZZC+20] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. Robust graph representation learning via neural sparsification. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11458–11468, 2020. 7

Checklist

(1) For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

(b) Did you describe the limitations of your work? [Yes]

(c) Did you discuss any potential negative societal impacts of your work? [Yes] See Supplementary Section D.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

(2) If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] Full assumptions are included in each theorem/lemma statements

(b) Did you include complete proofs of all theoretical results? [Yes] All omitted proofs appear in the attached Supplementary sections

(3) If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] We are illustrating (qualitative) failure rather than (quantitative) performance advantage. As such, the result of an experiment is merely binary (e.g., when our algorithm does not fail but the competing algorithm does). Therefore, error bars are not relevant in our settings.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A] Our experiments are about adversarial robustness using simple demonstrations and we did not use any GPUs or cloud computing.

(4) If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite and name all code libraries we used in Section 5.

(b) Did you mention the license of the assets? [N/A] The libraries we used are open source and their code can be found in Github (see cited links).

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Our code is included in the supplementary files.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

(5) If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]