SUBLINEAR SKETCHES FOR APPROXIMATE NEAREST NEIGHBOR AND KERNEL DENSITY ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Approximate Nearest Neighbor (ANN) search and Approximate Kernel Density Estimation (A-KDE) are fundamental problems at the core of modern machine learning, with broad applications in data analysis, information systems, and large-scale decision making. In massive and dynamic data streams, a central challenge is to design compact sketches that preserve essential structural properties of the data while enabling efficient queries.

In this work, we develop new sketching algorithms that achieve sublinear space and query time guarantees for both ANN and A-KDE for a dynamic stream of data. For ANN in the streaming model, under natural assumptions, we design a sublinear sketch that requires only $\mathcal{O}(n^{1+\rho-\eta})$ memory by storing only a sublinear $(n^{-\eta})$ fraction of the total inputs, where ρ is a parameter of the LSH family, and $0 < \eta < 1$. Our method supports sublinear query time, batch queries, and extends to the *Turnstile* model. While earlier works have focused on Exact NN, this is the first result on ANN that achieves near-optimal trade-offs between memory size and approximation error. Next, for A-KDE in the *Sliding-Window* model, we propose a sketch of size $\mathcal{O}\left(RW \cdot \frac{1}{\sqrt{1+\epsilon}-1} \log^2 N\right)$, where R is the number of sketch rows, W is the LSH range, N is the window size, and ϵ is the approximation error. This, to the best of our knowledge, is the first theoretical sublinear sketch guarantee for A-KDE in the *Sliding-Window* model.

We complement our theoretical results with experiments on various real-world datasets, which show that the proposed sketches are lightweight and achieve consistently low error in practice.

1 INTRODUCTION

Modern machine learning and data analysis often require answering similarity and density queries over massive and evolving datasets. As data grows in scale and arrives in dynamic streams, it becomes infeasible to store or process everything explicitly, and the central challenge is to design compact sketches that preserve essential structure while supporting efficient queries. This challenge has been highlighted in seminal works on sublinear algorithms, streaming, and high-dimensional data processing (Alon et al., 1996; Indyk & Motwani, 1998; Gionis et al., 1999; Muthukrishnan, 2005; Cormode & Muthukrishnan, 2005). Two central problems in this setting are Approximate Nearest Neighbor (ANN) search and Approximate Kernel Density Estimation (A-KDE).

Approximate Nearest Neighbor (ANN): The Approximate Nearest Neighbor (ANN) problem formalizes similarity search in high-dimensional spaces. In its standard (c, r)-formulation, given a point set $\mathbb{P}$ in a metric space $(\mathcal{X}, \mathcal{D})$, the task is to preprocess $\mathbb{P}$ so that, for any query $\boldsymbol{q} \in \mathcal{X}$, if the distance to its nearest neighbor in $\mathbb{P}$ is at most r, then with probability at least $1 - \delta$ the algorithm returns a point within distance cr of $\boldsymbol{q}$. This formulation corresponds to the modern version of the (r_1, r_2)-PLEB problem introduced by Indyk & Motwani (1998), with a relaxed variant due to Har-Peled et al. (2012) that permits null or arbitrary answers if no point lies within distance r_1. In the seminar work of Har-Peled et al. (2012), Har-Peled et al. obtained fully dynamic solutions to ANN, though such methods require at least linear space and are not designed for the streaming model. Subsequent advances, particularly those based on locality-sensitive hashing (e.g., Panigrahy (2005); Andoni & Indyk (2008); Andoni et al. (2014); Andoni & Razenshteyn (2015); Andoni et al. (2017);

Ahle (2017)), have achieved strong trade-offs between space and query time, though again primarily in static or dynamic settings rather than streaming.

Approximate Kernel Density Estimation (A-KDE): Kernel density estimation (KDE) is a classical non-parametric method for estimating probability distributions from data (Davis et al., 2011; Parzen, 1962; Silverman, 2018; Scott, 2015). Given a sequence of i.i.d. random variables $x_1, \ldots, x_n \in \mathbb{R}^d$, the density at a query x is estimated as $\hat{p}(x; \sigma) = \frac{1}{n} \sum_{i=1}^{n} K_\sigma(x - x_i)$, where $K_\sigma(\cdot)$ is a kernel function with bandwidth σ. While highly effective, exact KDE becomes computationally prohibitive for large-scale or streaming datasets, which motivated the approximate version (namely, A-KDE), where the goal is to return, for any query q, a $(1 \pm \epsilon)$ multiplicative approximation to the true density with probability at least $1 - \delta$. A-KDE thus provides a principled framework for large-scale density estimation, balancing accuracy and efficiency. Notable advances include RACE (Coleman & Shrivastava, 2020), which compresses high-dimensional vectors into compact counters; TAKDE (Wang et al., 2023), optimal in the sliding-window setting; and KDE-TRACK (Qahtan et al., 2016), designed for spatiotemporal streams.

Streaming Applications: Consider a personalized news agent or financial assistant powered by large language models: vast streams of articles or market updates arrive dynamically, yet the system must provide timely, personalized insights without storing or processing the entire corpus. Approximate nearest neighbor (ANN) search enables real-time matching of a user's evolving interests to relevant news items or market updates, while approximate kernel density estimation (A-KDE) captures shifts in topical or market distributions to adapt recommendations. A similar challenge arises in large-scale image and video platforms, where streams of photos or frames arrive continuously. ANN supports fast similarity search for recommendation, moderation, or retrieval in these large-scale systems, while A-KDE tracks distributional changes such as emerging styles, anomalies, or trending categories.

Such scenarios are not limited to text or vision: related needs appear in personalization, anomaly detection, and monitoring of high-volume data streams (Alon et al., 1996; Indyk & Motwani, 1998; Muthukrishnan, 2005; Cormode & Muthukrishnan, 2005; Jégou et al., 2011; Coleman & Shrivastava, 2020; Qahtan et al., 2016; Wang et al., 2023). Across these domains, storing or processing all data explicitly is infeasible, making compact sketches essential for balancing efficiency and accuracy in large-scale retrieval problems.

A central question is how to efficiently perform approximate nearest neighbor (ANN) search and approximate kernel density estimation (A-KDE) on massive, dynamically evolving data streams.

In this work, we use three most commonly used models of streaming, namely, insertion only, turnstile, and sliding window (see Appendix A.1.4 for their formal definitions and related work).

1.1 FORMAL PROBLEM DEFINITIONS

Problem 1 (Streaming (c, r)-Approximate Near Neighbor (ANN))**.** *Let $(\mathcal{X}, \mathcal{D})$ be a metric space, and let $\mathcal{P} \subseteq \mathcal{X}$ be a stream of at most n points. The goal is to maintain a data structure over the stream such that, for any query point $\mathbf{q} \in \mathcal{X}$:*

- *If $\mathcal{D}_\mathcal{P}(\mathbf{q}) \leq r$, then with probability at least $1 - \delta$ (for $0 < \delta \leq 1$), the data structure returns some $\mathbf{p}' \in \mathcal{P} \cap B(\mathbf{q}, cr)$.*
- *The data structure stores only a sublinear fraction of the stream, i.e., $\mathcal{O}(n^{1-\eta})$ points (for $0 < \eta \leq 1$), while supporting efficient updates and queries.*

We refer to this task as the Streaming (c, r)-Approximate Near Neighbor Problem *with failure probability δ.*

Problem 2 (Sliding-Window Approximate Kernel Density Estimation (A-KDE))**.** *Let $\{\mathbf{x}_t\}_{t \geq 1}$ be a data stream, where each $\mathbf{x}_t$ is drawn from a (potentially time-varying) density $p_t(\mathbf{x})$. Let N denote the window size, and let $\mathcal{T}_t = \{t - N + 1, \ldots, t\}$ denote the indices of points in the current window.*

Given a query $\mathbf{x}$*, the sliding-window KDE at time* t *is defined as:*

$$\hat{h}(\mathbf{x}; \sigma_t) = \frac{1}{N} \sum_{j \in \mathcal{T}_t} K_{\sigma_t}(\mathbf{x} - \mathbf{x}_j),$$

where K_{σ_t} *is a kernel with bandwidth* σ_t*.*

The goal is to maintain a compact sketch that supports Approximate KDE (A-KDE) *over the sliding window, enabling efficient updates and queries.*

1.2 Our Contributions

This work makes progress on two central problems in large-scale data analysis: streaming Approximate Nearest Neighbor (ANN) search (Problem 1) and Approximate Kernel Density Estimation (A-KDE) (Problem 2). We design new sketching algorithms that provably achieve sublinear space while supporting efficient queries, and we validate their effectiveness through extensive experiments. Below, we highlight our key contributions.

Our contributions to ANN (Problem 1): At first glance, maintaining even approximate solutions for ANN in the streaming model with *sublinear sketches* appears rather hopeless: an adversary can force any algorithm to store nearly all the data by giving inputs from scaled multidimensional lattices. However, real-world data is far from adversarial and often follows natural distributional assumptions (Mou & Wang, 2017; Coleman et al., 2019). Leveraging this, we prove that under a Poisson point process model—a well-studied and practically relevant distribution—ANN in the streaming setting *does* admit efficient sketching.

Our approach revisits the classical Motwani–Indyk framework (Indyk & Motwani, 1998) and shows that, under Poisson distributed inputs, it suffices to retain only a sublinear fraction of the stream, namely $\mathcal{O}(n^{1-\eta})$ points obtained by uniform sampling. This leads to a simple yet powerful sketching scheme with the following guarantees:

1. **Streaming ANN with sublinear space.** Our sketch provides (c, r)-ANN guarantees while storing only a vanishing fraction of the input.
2. **Turnstile robustness.** We extend our guarantees to the Turnstile model, assuming only mild restrictions on adversarial deletions within any unit ball.
3. **Parallel batch queries.** Our scheme naturally supports batch queries, which can be executed in parallel to achieve significant speedups.

To our knowledge, this is the first work to obtain such guarantees for ANN in the streaming model under realistic assumptions. Importantly, the simplicity of our scheme makes it broadly applicable and easy to generalize.

Empirical validation. We complement our theory with experiments on real-world datasets. The results demonstrate that our sketches are lightweight, achieve consistently low error, and provide *truly sublinear* space usage in high-ϵ regimes without compromising accuracy. In particular:

1. We show that for $\epsilon = 0.5$, we obtain sublinear sketches for all $\eta \geq 0.5$. More generally, for every sufficiently large ϵ, there exists a threshold η^* such that for all $\eta > \eta^*$, our scheme guarantees sublinear sketches without compromising on performance.
2. Our method outperforms the Johnson–Lindenstrauss (JL) baseline: beyond $\epsilon \approx 0.7$–0.8 on `sift1m`, and beyond $\epsilon \approx 0.9$ on `fashion-mnist`.

Our contributions to A-KDE (Problem 2): The RACE algorithm of Coleman & Shrivastava (2020) provides an elegant sketch for KDE in dynamic data streams and naturally supports the Turnstile model, thanks to its ability to handle both insertions and deletions. However, RACE lacks the mechanism to manage temporal information explicitly, making it unsuitable for the *sliding-window* model where data must expire once it falls outside the most recent N updates.

To address this challenge, we incorporate the classical EXPONENTIAL HISTOGRAM[1] result (Datar et al. (2002)) into each RACE structure. EXPONENTIAL HISTOGRAM is a powerful tool for maintaining aggregates over the most recent N updates with provable accuracy guarantees, and here they enable us to count, with bounded error, how many elements in the active window hash to the same LSH bucket as the query $\boldsymbol{q}$. This delicate combination allows us to design the *first sketch* for the **A-KDE in the sliding-window model**, which explicitly handles expiration of old data while retaining the efficiency of RACE. Our construction does incur an extra $\log^2 N$ factor in space compared to plain RACE, but it uniquely enables sliding-window guarantees. We further extend this approach to handle *batch updates*, where data arrives in mini-batches; here, the window consists of N batches, and the EXPONENTIAL HISTOGRAM is naturally adapted to this setting.

Empirical validation. We evaluate our sliding-window A-KDE sketch on real-world datasets. The results highlight that:

1. The empirical relative error of KDE estimates is significantly smaller than the worst-case theoretical bound, even with a small number of rows in the sketch.
2. Our sliding-window A-KDE achieves accuracy comparable to RACE (Coleman & Shrivastava, 2020) on `News Headlines` and `ROSIS Hyperspectral Images`, while uniquely supporting explicit expiration and batch updates.

Due to space paucity, proofs of the Lemmas and Theorems marked with $(\star)$ are deferred to the Appendix.

2 PRELIMINARIES

Locality Sensitive Hashing: Let $\mathbb{X}$ be a metric space and $D : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is a distance metric.

Definition 2.1. *A family $\mathcal{H} = \{h : \mathbb{X} \to \mathbb{U}\}$ is (r_1, r_2, p_1, p_2)-sensitive for $(\mathbb{X}, \mathbb{D})$ if for any $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{X}$, we have:*

- *If $D(\boldsymbol{p}, \boldsymbol{q}) \leq r_1$, then $P_{h\in\mathcal{H}}[h(\boldsymbol{q}) = h(\boldsymbol{p})] \geq p_1$.*
- *If $D(\boldsymbol{p}, \boldsymbol{q}) \geq r_2$, then $P_{h\in\mathcal{H}}[h(\boldsymbol{q}) = h(\boldsymbol{p})] \leq p_2$.*

For a locality-sensitive family to be useful, it must satisfy the inequalities $\boldsymbol{p}_1 > \boldsymbol{p}_2$ and $r_1 < r_2$. Two points $\boldsymbol{p}$ and $\boldsymbol{q}$ are said to collide, if $h(\boldsymbol{p}) = h(\boldsymbol{q})$. We denote the collision probability by $k(\boldsymbol{x}, \boldsymbol{y})$. Note that $k(.,.)$ is bounded and symmetric *i.e.* $0 \leq k(\boldsymbol{x}, \boldsymbol{y}) \leq 1, k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{y}, \boldsymbol{x})$, and $k(\boldsymbol{x}, \boldsymbol{x}) = 1$. It is known that if there exists a hash function $h(\boldsymbol{x})$ with $k(\boldsymbol{x}, \boldsymbol{y})$ and range $[1, W]$, the same hash function can be independently concatenated p times to obtain a new hash function $H(.)$ with collision probability $k^p(\boldsymbol{x}, \boldsymbol{y})$ for any positive integer p. The range of the new hash function will be $[1, W^p]$. In particular, We use two such hash families in our analysis: (1) SRP-LSH (also known as, Angular LSH) as described in Charikar (2002) and (2) p-stable LSH, described in Datar et al. (2004).

We also require the concepts of the EXPONENTIAL HISTOGRAM (Datar et al., 2002), the Repeated Array-of-Counts Estimator (RACE) (Coleman & Shrivastava, 2020), and the static (c, r)-ANN result of Har-Peled et al. (2012). Due to space constraints, these preliminaries are deferred to Appendix A.1.

3 STREAMING (C,R)-APPROXIMATE NEAR NEIGHBOR

We generalize the classical Motwani–Indyk scheme Indyk & Motwani (1998) to the streaming setting by formulating the (c, r)-ANN problem under the assumption that the number of points inside any ball follows a Poisson distribution with an appropriate mean parameter. Within this framework, we prove that it suffices to retain only a sublinear sample of the data stream, specifically $\mathcal{O}(n^{1-\eta})$ points obtained through uniform random sampling.

[1] An exponential histogram can maintain aggregates over data streams with respect to the last N data elements. For example, it can estimate up to a certain error the number of 1's seen within the last N elements, assuming data is in the form of 0s and 1s.

The Algorithm: Let N be an upper bound on the size of the data stream $\mathbb{D}$. We initialize a family of hash functions $\mathcal{H}$ with parameters k and L, chosen as functions of N and ϵ. Below, we describe a scheme that inserts points from the stream into our data structure and employs the *Query Processing* routine to solve the (c, r)-ANN problem for any query point $\boldsymbol{q}$ (see Algorithm 1).

Algorithm 1 StreamingANN

Require: Data stream $\mathbb{D}$, query point $\boldsymbol{q}$, LSH family $\mathcal{H}$, parameters k and L, sampling parameter η

Initialize L independent hash functions $\{g_1, g_2, \ldots, g_L\}$
for each point $\boldsymbol{p} \in \mathbb{D}$ **do**
    decide whether to drop or store $\boldsymbol{p}$ ▷ Use uniform sampling to store approximately $O(n^{1-\eta})$ elements
    for $j = 1$ to L **do**
        Insert $\boldsymbol{p}$ into bucket $g_j(\boldsymbol{p})$
    end for
end for
Query processing:
Initialize candidate list $\mathbb{C} \leftarrow \emptyset$
for $j = 1$ to L **do**
    Retrieve all points from bucket $g_j(\boldsymbol{q})$ and add to $\mathbb{C}$
    if $|\mathbb{C}| \geq 3L$ **then**
        break
    end if
end for
Remove duplicates from $\mathbb{C}$
$p^* \leftarrow \arg\min_{\boldsymbol{p}_j \in \mathbb{C}} D(\boldsymbol{p}_j, \boldsymbol{q})$
if $D(p^*, \boldsymbol{q}) \leq r_2$ **then**
    return p^*
end if
return NULL

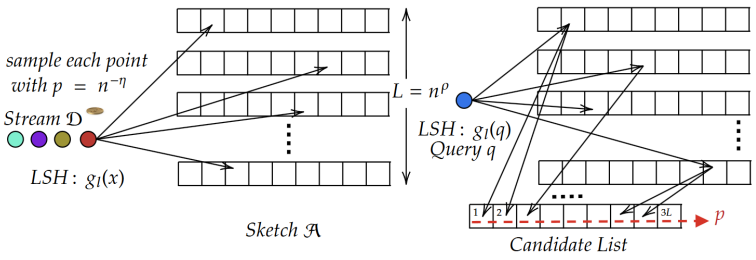


Figure 1: Illustration of the insertion (left) and querying (right) parts of StreamingANN

Correctness: A query $\boldsymbol{q}$ can succeed when certain events take place, which are mentioned here.

Lemma 3.1. *Let $\mathbb{P}$ be the set of points at the time when the query is executed. Define $B(\boldsymbol{p}, r)$ as a ball of radius r surrounding a point $\boldsymbol{p}$. We define the following two events for any query point $\boldsymbol{q}$:*

- $\mathbf{E_1}$: *$\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r)$ such that $g_j(\boldsymbol{p}') = g_j(\boldsymbol{q})$ for some $j \in \{1, \ldots, L\}$.*
- $\mathbf{E_2}$: *The number of points from $\mathbb{P} - B(\boldsymbol{q}, r_2)$ which hash to the same bucket as $\boldsymbol{q}$ is less than $3L$ i.e.*

$$\sum_{j=1}^{L} |(\mathbb{P} - B(\boldsymbol{q}, r_2)) \cap g_j^{-1}(g_j(\boldsymbol{q}))| < 3L.$$

If $\mathbf{E_1}$ and $\mathbf{E_2}$ hold, then the query $\boldsymbol{q}$ succeeds

Proof. Denote the nearest neighbor of query $\boldsymbol{q}$ as $\boldsymbol{p}^*$. We have two cases:

Case 1 $\exists \boldsymbol{p}^* \in B(\boldsymbol{q}, r)$: Since the algorithm stores $3L$ candidate points, $\mathbf{E_2}$ holding implies that the candidate set has a non-zero number of vacant spots. Now, if $\mathbf{E_1}$ also holds, we know that there exists $\boldsymbol{p}' \in B(\boldsymbol{q}, r)$ such that $g_j(\boldsymbol{p}') = g_j(\boldsymbol{q})$ for some $j \in \{1, \ldots, L\}$. So this p' must get added to the candidate set. Since we ultimately return a point that is closest to the query point $\boldsymbol{q}$, we guarantee that we return $\boldsymbol{p}'$ or a closer point. But since $\boldsymbol{p}' \in B(\boldsymbol{q}, r)$, trivially we have that $\boldsymbol{p}' \in B(\boldsymbol{q}, cr)$, hence the query $\boldsymbol{q}$ succeeds.

Case 2 $\nexists \boldsymbol{p}^* \in B(\boldsymbol{q}, r)$: In this case, the algorithm can return $null$ or any point in $\mathbb{P}$, so the query $\boldsymbol{q}$ succeeds trivially. □

It is to be noted that the aforementioned p' does not need to be the nearest neighbor of the query point. It suffices to have a point present in a ball of radius r surrounding the query after the random sampling. In the following lemma, we show how we can set the parameter k to guarantee the success of $\mathbf{E_2}$ with high probability.

Lemma 3.2 ($\star$). *If we set $k = \lceil \log_{1/p_2} n \rceil$, for a certain query $\boldsymbol{q}$ event $\mathbf{E_2}$ succeeds with probability $P_2 \geq 1 - \frac{1}{3n^\eta}$ if we store $\mathcal{O}(n^{1-\eta})$ points from the stream independent of how the data is distributed.*

Now, we show that if we set k as per Lemma 3.2, assuming that our data is obtained from a Poisson point process, we can guarantee the success of $\mathbf{E_1}$ with high probability for an appropriate choice of L.

Lemma 3.3. *Assume that the points are distributed in such a manner that the number of points in every ball of radius r is distributed as a Poisson random variable with mean m. Given that $k = \lceil \log_{1/p_2} n \rceil$, if we set $L = \frac{n^\rho}{p_1}$, for a certain query $\boldsymbol{q}$ event $\mathbf{E_1}$ succeeds with probability $P_1 \geq (1 - e^{-mp})(1 - 1/e)$ on sampling $\mathcal{O}(n^{1-\eta})$ points from the stream*

Proof. We require that $\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r)$ such that $g_j(\boldsymbol{p}') = g_j(\boldsymbol{q})$ for some $j \in \{1, \ldots, L\}$. First, consider the probability that there is at least one point retained in a ball of radius r surrounding the query after the uniform sampling, *i.e.* $\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r)$. We know that the data follows a Poisson distribution, so if we say that the number of points in a ball of radius r surrounding a query is a Poisson random variable $\mathbf{K}$ with mean m:

$$\begin{aligned}\mathbb{P}(\text{No points in the } r\text{-ball after uniform sampling}) &= \mathbb{E}[(1-p)^{\mathbf{K}}] \\ &= e^{-mp}\end{aligned}$$

where $p = n^{-\eta}$ is the probability that we choose to store the point in the data structure while uniformly sampling. This implies that the probability of having at least one point ($\boldsymbol{p}'$) close to the query is $(1 - e^{-mp})$.

Now, given that there exists $\boldsymbol{p}' \in B(\boldsymbol{q}, r)$, we can lower bound the probability that $g_j(\boldsymbol{p}') = g_j(\boldsymbol{q})$ for some $j \in \{1, \ldots, L\}$ as follows:

$$P(g_j(\boldsymbol{p}') = g_j(\boldsymbol{q})) \geq p_1^k \geq p_1^{\log_{1/p_2} n + 1} = p_1 n^{-\log_{1/p_2}(1/p_1)} = p_1 n^{-\rho} \tag{1}$$

Combining these two statements, the probability of success of event $\mathbf{E_1}$ is lower bounded as:

$$\begin{aligned}P(\mathbf{E_1}) &\geq (1 - e^{-mp})(1 - (1 - p_1 n^{-\rho})^L) \text{ (using equation 1)} \\ &\geq (1 - e^{-mp})(1 - 1/e) \text{ (setting } L = n^\rho / p_1)\end{aligned}$$

(See Appendix A.2.2 for the detailed proof)

□

We can now use the above Lemmas (Lemmas 3.1, 3.2, 3.3) to prove the following theorem.

Theorem 3.1 ($\star$). *Let $(\mathcal{X}, \mathcal{D})$ be a metric space, and suppose that there exists a (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$, with $p_1, p_2 \in (0, 1)$, and define $\rho = \frac{\log(\frac{1}{p_1})}{\log(\frac{1}{p_2})}$. We further assume that the number of points contained in any ball of radius r can be modeled as a Poisson random variable with mean m, where $m \geq Cn^\eta$ for some constant $C > 0$. Then, for a point set $\mathbb{P} \subseteq \mathcal{X}$ comprising at most n points, there exists a data structure for streaming (c, r)-nearest neighbor search with the following guarantees:*

- *The data structure stores only $\mathcal{O}(n^{1-\eta})$ points from the stream.*
- *Each query requires at most $\mathcal{O}(n^{\rho}/p_1)$ distance computations and $\mathcal{O}\left(\frac{n^{\rho}}{p_1} \cdot \log_{1/p_2} n\right)$ evaluations of hash functions from $\mathcal{H}$. The same bounds hold for updates.*
- *The data structure uses at most $\mathcal{O}(n^{1+\rho-\eta}/p_1)$ words of space, in addition to the space required to store $\mathbb{P}$.*

The probability of failure is at most $\frac{1}{3n^{\eta}} + \frac{e^{mp}+e-1}{e^{mp+1}}$, which is less than 1 for an appropriate choice of C.

The result of Theorem 3.1 naturally extends to the batch queries[2] setting (see Appendix A.2.4).

ANN in the Turnstile Model: In the turnstile setting, arbitrary deletions can break ANN guarantees if the nearest neighbor within a query ball is removed. To mitigate this, we assume an adversary can delete at most d points from any ball of radius r. Under this restriction, the earlier sublinear-sample guarantees hold by bounding the probability that a ball contains at least $d+1$ points. We prove that, under natural assumptions on point distributions in a stream and limiting deletions per region, we can maintain a sublinear-sized data structure that supports efficient approximate nearest neighbor queries with low failure probability, handling both insertions and deletions (see Theorem A.4). The complete description of this section is provided in Appendix A.2.5.

4 Sliding-Window Approximate Kernel Density Estimation (A-KDE)

In Coleman & Shrivastava (2020), the authors propose RACE, an efficient sketching technique for kernel density estimation on high-dimensional streaming data. We have seen that we can get low relative errors using a larger number of repetitions in RACE. We propose a modified RACE structure to make it suitable for a sliding window model by using EXPONENTIAL HISTOGRAM (Datar et al., 2002). We give bounds on the number of repetitions *i.e.* , the number of rows, to obtain a good estimate of the KDE with high probability.

The Algorithm: In RACE, we increment $\boldsymbol{A}[i, h_i(\boldsymbol{x})]$ for every new element $\boldsymbol{x}$ coming from dataset $\mathbb{D}$. In the sliding window model, we are interested in the last N elements, assuming that we get an element every time step. Hence, we have to find the number of times a counter has been incremented in the last N time steps. This problem is similar to the BASIC COUNTING problem in section A.1.3, where the incoming stream of data is 1 if the counter is incremented at a time instant, otherwise 0. We will store an EXPONENTIAL HISTOGRAM for each cell of RACE. On querying the EXPONENTIAL HISTOGRAM, we will get an estimate of the count in a particular cell. In the RACE sketch, the estimator is computed by the median of means procedure. For our purposes, we will take the average of ACE estimates over L independent repetitions. The algorithm to construct the modified RACE array and estimate KDE for a given query is given in 2.

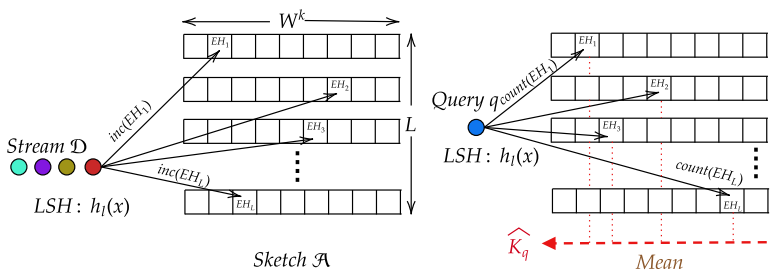


Figure 2: Illustration of the pre-processing (left) and querying (right) parts of Algorithm 2

[2]A query consists of a set of points $\mathbb{Q} = \{\boldsymbol{q}_i\}_{i=1}^{B}$, and each batch can be viewed as B independent queries

Algorithm 2 Approximate KDE sketch construction and querying for sliding window

Preprocessing:

Require: Data set $\mathbb{D}$, LSH family $\mathcal{H}$ of range W, parameters k and L

Initialize L independent hash functions $\{h_1, h_2, \cdots, h_L\}$ where each h_i is constructed by concatenating p independent hashes from $\mathcal{H}$
$A \leftarrow$*empty* ▷ RACE structure
$t \leftarrow 0$ ▷ timestamp initialized to 0
for $\boldsymbol{p} \in \mathbb{D}$ **do**
  for i in 1 to L **do**
    $j \leftarrow h_i(\boldsymbol{p})$
    if $A[i, j]$ is empty **then**
      Create an Exponential Histogram at $A[i, j]$ with timestamp t
    else
      Add a 1 to the exponential histogram at $A[i, j]$ with timestamp t
    end if
  end for
  $t \leftarrow t + 1$
end for

Query processing:

Require: RACE sketch A, Query $\boldsymbol{q}$, Hash functions $\{h_1, h_2, \cdots, h_L\}$ initialized in preprocessing
$y \leftarrow 0$ ▷ initialize the KDE estimate to 0
for i = 1 to L **do**
  if $A[i, h_i(\boldsymbol{q})]$ is not empty **then**
    $c \leftarrow$ the estimate of count in the exponential histogram at $A[i, h_i(\boldsymbol{q})]$
    $y \leftarrow y + c$
  end if
end for
$y \leftarrow y/L$ ▷ compute the approximate KDE
return y

Correctness: The algorithm is illustrated in Fig. 2. Consider the ACE Estimator. We know that if we have the actual counts, then $X = \boldsymbol{A}[h(\boldsymbol{q})]$ is an unbiased estimator for the Kernel Density with bounded variance(A.3). So, $\mathbf{K} = \mathbb{E}[X]$ is the Kernel Density estimate. We will now show that by using EXPONENTIAL HISTOGRAM, the new estimator for a single ACE instance approximates $\mathbf{K}$ up to a certain error.

Lemma 4.1 ($\star$)**.** *Let Y be the new estimator obtained from querying the* EXPONENTIAL HISTOGRAM *at* $\boldsymbol{A}[h(\boldsymbol{q})]$*. Then,* $\mathbb{E}[Y] \leq (1 + \epsilon')\boldsymbol{K}$

We use r independent instances of the ACE array to estimate the kernel density. Hence, the KDE estimator in the current setting is, $\hat{Y} = \frac{1}{r}\sum_{i=1}^{r} Y_i$. The expectation of $\hat{Y}$ is $\mathbb{E}Y_i$ and the variance of $\hat{Y}$ is $\frac{1}{r}Var(Y_i)$, where Y_i is the i^{th} independent instance of ACE. Now we will show the bounds of the estimator $\hat{Y}$ and derive the necessary bounds on r.

Lemma 4.2 ($\star$)**.** $|\hat{Y} - \mathbb{E}\hat{Y}| < \epsilon'\mathbb{E}[\hat{Y}]$ *holds with probability* $1 - \delta$ *if:*

$$r \geq \frac{2\max\{X_i\}^2}{(1+\epsilon')^2\boldsymbol{K}^2}\log\left(\frac{2}{\delta}\right)$$

where r *is the number of repetitions of* ACE *(or the number of rows in the* RACE *array data structure).*

Now we will show that $\hat{Y}$ gives a multiplicative approximation of the KDE with probability $1 - \delta$.

Lemma 4.3. *The estimator* $\hat{Y}$ *gives a* $(1 + \epsilon)$ *approximation of the Kernel density with probability* $1 - \delta$*.*

Proof. Let the KDE be given by $\mathbf{K}$. The estimator from the modified RACE algorithm is $\hat{Y}$. Then,

$$\begin{aligned}
|\hat{Y} - \mathbf{K}| &\leq |\hat{Y} - \mathbb{E}\hat{Y}| + |\mathbb{E}\hat{Y} - \mathbf{K}| \text{ (using triangle inequality)} \\
&\leq \epsilon' \mathbb{E}\hat{Y} + \epsilon' \mathbf{K} \text{ (using lemma 4.2 and penultimate inequality of equation 3)} \\
&\leq \epsilon'(1+\epsilon')\mathbf{K} + \epsilon' \mathbf{K} \text{ (using last inequality of equation 3)} \\
&\leq (2\epsilon' + \epsilon'^2)\mathbf{K} \\
&= \epsilon \mathbf{K} \text{ (substituting } \epsilon = 2\epsilon' + \epsilon'^2) \\
\implies |\hat{Y} - \mathbf{K}| &\leq \epsilon \mathbf{K}
\end{aligned}$$

Note that the bound from Lemma 4.2 holds with probability $1 - \delta$. Hence, this result holds with probability $1 - \delta$. □

Let us compute the space requirement of the sketch proposed for our algorithm.

Lemma 4.4 ($\star$)**.** *The proposed* RACE *data structure has space complexity* $\mathcal{O}\left(RW \cdot \frac{1}{\sqrt{1+\epsilon}-1} \log^2 N\right)$ *where R is the number of rows, W is the range of the hash function, ϵ is the relative error for* KDE*, N is the window size.*

Using the above lemmas, we can state the main theorem as follows

Theorem 4.1. *Suppose we are given an* LSH *function with range W. Then the proposed sliding window* RACE *data structure with*

$$R = \mathcal{O}\left(\frac{2\max\{X_i\}^2}{(1+\epsilon)\boldsymbol{K}^2} \log\left(\frac{2}{\delta}\right)\right)$$

independent repetitions of the hash function provide a $1 \pm \epsilon$ multiplicative approximation to $\boldsymbol{K}$ (which is the KDE*) with probability $1 - \delta$, using space $\mathcal{O}\left(RW \cdot \frac{1}{\sqrt{1+\epsilon}-1} \log^2 N\right)$ where N is the window size.*

We can extend the result of Theorem 4.1 to batch queries setting (see Appendix A.3.4)

5 Experiments

Experiments for ANN: We evaluate the efficacy of our streaming ANN approach on standard benchmarks, focusing on the trade-offs between sampling aggressiveness (parameter η) and the approximate recall/accuracy of (c, r)-ANN queries. We also investigate the interplay between ϵ and η, demonstrating that for sufficiently large ϵ, sub-linear sketch sizes are attainable with $\eta < \rho$.

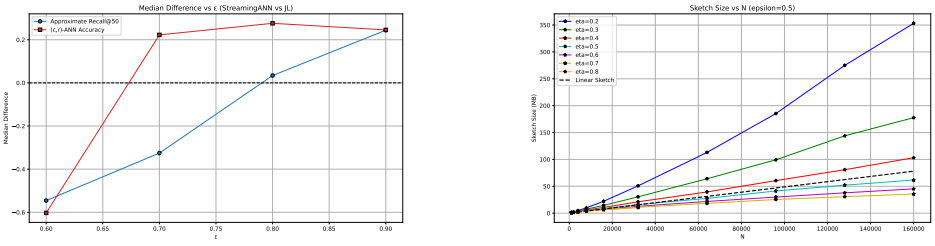
Datasets. Experiments were conducted on two standard ANN benchmarks Aumüller et al. (2020): `sift1m` Jegou et al. (2010) (1M vectors, 128-dimensions) and `fashion-mnist` Xiao et al. (2017) (60,000 images, 784-dimensions).

Implementation. All data structures were implemented in Python, assuming `float32` vectors. Compression is measured relative to $N \times d \times 4/1024^2$ MB. No additional memory optimizations were applied. We used the p-stable scheme described in Datar et al. (2004) for hashing.

Baseline. We compare against Johnson-Lindenstrauss (JL) projection Johnson et al. (1984), the only known strict one-pass solution for (c, r)-ANN.

Experimental Setup. Two experiments were performed: **(1) Comparison with JL:** We compared our method and the JL baseline by sweeping over $\epsilon = 0.5$ to 1 and adjusting compression rates via k (JL) and η (ours). Each run stored 50,000 points and issued 5,000 queries with $r = 0.5$. Metrics included approximate recall@50, (c, r)-ANN accuracy, and memory usage. **(2) Sketch Size Scaling:** Using the `sift1m` dataset, we fixed $\epsilon = 0.5$ and varied η (0.2 to 0.8) and dataset size N (1,000 to 160,000), measuring sketch size.

Experiments for A-KDE: We evaluate the effect of row count in the sliding-window A-KDE sketch on mean relative KDE error.



(a) (b)

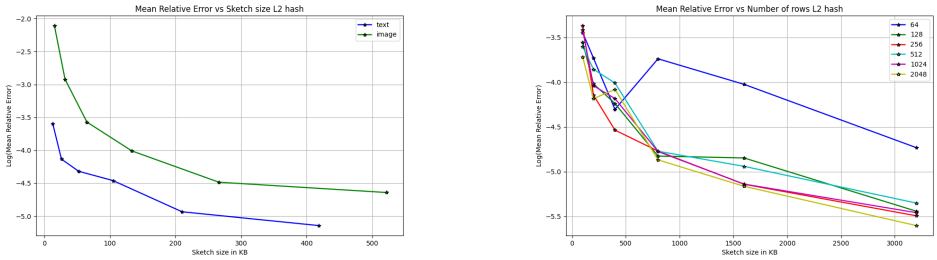
Figure 3: (a) Summary of the median performance difference between StreamingANN and JL across ϵ, and (b) Memory requirements scale with stream size N for fixed $\epsilon = 0.5$ for the `sift1m` dataset.

Datasets. (a) `News Headlines` Kulkarni (2017): 80k headlines embedded into 384-dim vectors using **all-MiniLM-L6-v2**[3]. (b) `ROSIS Hyperspectral Data` Sowmya et al. (2019): each pixel as a spectral vector.

Implementation. Sliding-window A-KDE uses (a) angular LSH and (b) p-stable Euclidean LSH (rehashing used for range-bounding). Bandwidth parameter $p = 1$. Reported results use a single-query setting. The theoretical relative error is taken as 0.21, while the experimental error is observed to be much lower.

Baseline. We compare against RACE Coleman & Shrivastava (2020), which works for general streaming setting.

Experimental Setup. Three experiments: **(1) Sketch Size vs. Error:** Log mean relative error vs. sketch size (100 to 3200 rows) for window size 450, $\epsilon = 0.21$, using both hashes and datasets. **(2) Window Size Effect:** Log mean error vs. rows, for L2 hash and angular hash, for different window sizes 64 to 2048. **(3) Comparison with RACE:** Compared A-KDE using Angular hash and window size 260 with `RACE`.



(a) (b)

Figure 4: (a) Effect of sketch size on KDE estimates using Euclidean hash (b) Effect of the window size on mean relative error for Sliding window A-KDE with Euclidean hash on news headlines data

Discussion: The median difference plot [4] in Figure 3a shows that StreamingANN outperforms JL on both metrics beyond certain values of ϵ. We show a more detailed comparison of approximate recall/accuracy vs compression rate for both datasets in Appendix A.4. Figure 3b shows us that for this choice of ϵ, we can obtain sub-linear sketches for $\eta \geq 0.5$. Putting these experiments together, we can see that it is possible to attain sublinear sketches for Problem 1 for an appropriate choice of ϵ with good performance.

For A-KDE, increasing sketch size reduces mean error for Euclidean kernels(Fig. 4a), while angular hashing shows dataset-specific behavior. Higher window sizes minimize error for text data(Fig. 4b), with A-KDE performing similar to RACE (Fig. 7). These results validate our theoretical guarantees and demonstrate practical effectiveness across tasks.

[3] Kaggle link

[4] Median difference is the median value of the difference in the respective metric (approximate recall/accuracy) as we vary compression rates. So a positive median difference corresponds to our scheme consistently out-performing the baseline.

REFERENCES

Thomas Dybdahl Ahle. Optimal las vegas locality sensitive data structures. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 938–949. IEEE, 2017.

Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29, 1996.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 793–801, 2015.

Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1018–1028. SIAM, 2014.

Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms*, pp. 47–66. SIAM, 2017.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, 2002.

Benjamin Coleman and Anshumali Shrivastava. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *Proceedings of The Web Conference 2020*, pp. 1739–1749, 2020.

Benjamin Coleman, Richard G Baraniuk, and Anshumali Shrivastava. Sub-linear memory sketches for near neighbor search on streaming data. *arXiv preprint arXiv:1902.06687*, 2019.

Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, 2004.

Richard A Davis, Keh-Shin Lii, and Dimitris N Politis. Remarks on some nonparametric estimates of a density function. In *Selected Works of Murray Rosenblatt*, pp. 95–100. Springer, 2011.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, volume 99, pp. 518–529, 1999.

Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. 2012.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Rohit Kulkarni. A million news headlines. `https://www.kaggle.com/datasets/therohk/million-headlines`, 2017. CSV data file; over 1.2 million news headlines from ABC (Australian Broadcasting Corporation), CC0 Public Domain license. Accessed: 2025-09-21.

Chen Luo and Anshumali Shrivastava. Arrays of (locality-sensitive) count estimators (ace) anomaly detection on the edge. In *Proceedings of the 2018 World Wide Web Conference*, pp. 1439–1448, 2018.

Wenlong Mou and Liwei Wang. A refined analysis of lsh for well-dispersed data points. In *2017 Proceedings of the Fourteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pp. 174–182. SIAM, 2017.

S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. *arXiv preprint cs/0510019*, 2005.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Abdulhakim Qahtan, Suojin Wang, and Xiangliang Zhang. Kde-track: An efficient dynamic density estimator for data streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):642–655, 2016.

David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

V. Sowmya, K. P. Soman, and M. Hassaballah. *Hyperspectral Image: Fundamentals and Advances*, pp. 401–424. Springer International Publishing, Cham, 2019.

Yinsong Wang, Yu Ding, and Shahin Shahrampour. Takde: temporal adaptive kernel density estimator for real-time dynamic density estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11):13831–13843, 2023.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

A Appendix

A.1 Missing Parts of Preliminaries

A.1.1 Approximate Nearest Neighbor:

In Har-Peled et al. (2012), the authors give a scheme to solve the (c,r)-ANN problem with the following guarantees:

Theorem A.1 (Har-Peled et al. (2012))**.** *Suppose there is a (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$ for $(\mathcal{X}, \mathcal{D})$, where $p_1, p_2 \in (0,1)$ and let $\rho = \frac{\log(\frac{1}{p_1})}{\log(\frac{1}{p_2})}$. Then there exists a fully dynamic data structure for the (c, r)-Approximate Near Neighbor Problem over a set $\mathbb{P} \subset \mathcal{X}$ of at most n points, such that:*

- *Each query requires at most $\mathcal{O}\left(\frac{n^\rho}{p_1}\right)$ distance computations and $\mathcal{O}\left(\frac{n^\rho}{p_1} \cdot \log_{\frac{1}{p_2}} n\right)$ evaluations of hash functions from $\mathcal{H}$. The same bounds hold for updates.*

- *The data structure uses at most $\mathcal{O}(\frac{n^{1+\rho}}{p_1})$ words of space, in addition to the space required to store $\mathbb{P}$.*

The failure probability of the data structure is at most $\frac{1}{3} + \frac{1}{e} < 1$.

Given a (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$ of hash functions, the authors amplify the gap between the "high" probability p_1 and "low" probability p_2 by concatenating several functions. For a specified parameter k, they define a function family $\mathcal{G} = \{g : \mathcal{X} \to \mathcal{U}^k\}$ such that

$$g(\boldsymbol{p}) = (h_{i_1}(\boldsymbol{p}), h_{i_2}(\boldsymbol{p}), \ldots, h_{i_k}(\boldsymbol{p}))$$

where $h_i \in \mathcal{H}$ and $I = \{i_1, \ldots, i_k\} \subset \{1, \ldots, |\mathcal{H}|\}$. For an integer L, they choose L functions $g_1, \ldots, g_L$ from $\mathcal{G}$ independently and uniformly at random. During preprocessing, they store a pointer to each $\boldsymbol{p} \in \mathbb{P}$ in the buckets $g_1(\boldsymbol{p}), \ldots, g_L(\boldsymbol{p})$. Since the total number of buckets may be large, they retain only the non-empty buckets by resorting to "standard" hashing.

A.1.2 Repeated Array-of-Counts Estimator (RACE)

In Coleman & Shrivastava (2020), the authors propose RACE, an efficient sketching algorithm for kernel density estimation on high-dimensional streaming data. The RACE algorithm compresses a dataset $\mathbb{D}$ into a 2-dimensional array $\boldsymbol{A}$ of integer counters of size $L \times R$ where each row is an ACE (Arrays of (locality-sensitive) Counts Estimator) data structure[5](Luo & Shrivastava, 2018). To add an element $\boldsymbol{x} \in \mathbb{D}$ we compute the hash of $\boldsymbol{x}$ using L independent LSH functions $h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), ..., h_L(\boldsymbol{x})$. Then we increment the counters at $\boldsymbol{A}[i, h_i(\boldsymbol{x})]$ for all $i \in [1, ..., L]$. So each array cell stores the number of data elements that have been hashed to the corresponding LSH bucket.

The KDE of a query is roughly a measure of the number of nearby elements in the dataset. Hence, it can be estimated by averaging over hash values for all rows of RACE:

$$\hat{K}(\boldsymbol{q}) = \frac{1}{L} \sum_{i=1}^{L} \boldsymbol{A}[i, h_i(\boldsymbol{q})]$$

For a query $\boldsymbol{q}$ the RACE sketch computes the KDE using the median of means procedure rather than the average to bound the failure probability of the randomized query algorithm. The key result of Luo & Shrivastava is:

Theorem A.2 (ACE estimator)**.** *Given a dataset $\mathbb{D}$ and an* LSH *family $\mathcal{H}$ with finite range $[1, W]$, construct a hash function $h : x \to [1, W^p]$ by concatenating p independent hashes from $\mathcal{H}$. Suppose an ACE array $\mathcal{A}$ is created by hashing each element of $\mathbb{D}$ using $h(.)$. Then for any query $\boldsymbol{q}$,*

$$\mathbb{E}[\boldsymbol{A}[h(\boldsymbol{q})]] = \sum_{\boldsymbol{x} \in \mathbb{D}} k^p(\boldsymbol{x}, \boldsymbol{q})$$

ACE is useful for KDE because it is an unbiased estimator. Moreover, Coleman et al. have shown a tight bound on the variance.

Theorem A.3. *Given a query $\boldsymbol{q}$, the variance of* ACE *estimator $\boldsymbol{A}[h(\boldsymbol{q})]$ is bounded as:*

$$var(\boldsymbol{A}[h(\boldsymbol{q})]) \leq \left(\sum_{\boldsymbol{x} \in \mathbb{D}} k^{p/2}(\boldsymbol{x}, \boldsymbol{q}) \right)^2$$

This implies that we can estimate KDE using repeated ACE or RACE with very low relative error given a sufficient number of repetitions p.

A.1.3 Exponential Histogram

We want to solve the Basic Counting problem for data streams with regard to the last N elements seen so far. To be specific, we want to count the number of 1's in the last N elements given a stream

[5]ACE is an extremely fast and memory efficient algorithm used for unsupervised anomaly detection which does not require to store even a single data sample and can be dynamically updated.

of data elements containing 0 or 1. Datar et al. (2002) proposed an algorithm for this problem, which provides a $(1+\epsilon)$ estimate of the actual value at every instant. They use an EXPONENTIAL HISTOGRAM (EH) to maintain the timestamp of active 1's in that they are present within the last N elements. Every bucket in the histogram maintains the timestamp of the most recent 1 and the number of 1's called the bucket size. The buckets are indexed as $1, 2, ...$ in decreasing order of their arrival times *i.e.* the most recent bucket is indexed 1. Let the size of the i^{th} bucket be denoted as C_i. When the timestamp of a bucket expires, we delete that bucket. The estimate for the number of 1's at any instant is given by subtracting half of the size of the last bucket from the total size of the existing buckets, *i.e.* $(TOTAL - LAST)/2$. To guarantee counts with relative error of at most ϵ, the following invariants are maintained by the algorithm (define k as $\lceil 1/\epsilon \rceil$) :

1. **Invariant 1:** Bucket sizes $c_1, c_2, \ldots, c_m$ are such that $\forall j \leq m$ we have $\frac{c_j}{2(1+\Sigma_{i=1}^{j-1} c_i)} \leq \frac{1}{k}$.

2. **Invariant 2:** Bucket sizes are non-decreasing, i.e. $c_1 \leq c_2 \leq c_3 \leq \cdots \leq c_m$. Further, they are constrained to only powers of 2. Finally, for every bucket other than the last bucket, there are at most $\frac{k}{2} + 1$ and at least $\frac{k}{2}$ buckets of that size.

It follows from invariant 2 that to cover all active 1's, we need no more than $n \leq (\frac{k}{2}+1) \cdot (\log(2\frac{N}{k} + 1) + 1)$ buckets. The bucket size takes at most $\log N$ values, which can be stored using $\log \log N$ bits, and the timestamp requires $\log N$ bits. So the memory requirement for an EH is $\mathcal{O}(\frac{1}{\epsilon} \log^2 N)$. By maintaining a counter each for $TOTAL$ and $LAST$, the query time becomes $\mathcal{O}(1)$.

A.1.4 STREAMING MODELS

Insertion-Only: In the insertion-only streaming model, data arrives sequentially and can only be appended to the dataset; deletions are not allowed. This model captures many practical scenarios, such as log analysis, clickstreams, and sensor readings, where storing all data explicitly is infeasible. The goal is to maintain a compact summary that supports approximate queries using sublinear memory. Classical sketches such as Count-Min (Cormode & Muthukrishnan, 2005), AMS (Alon et al., 1996), and their extensions to similarity search and density estimation (Indyk & Motwani, 1998; Coleman & Shrivastava, 2020) have demonstrated the effectiveness of insertion-only algorithms for both high-dimensional similarity search and approximate kernel density estimation.

Turnstile: The Turnstile model generalizes insertion-only streams by allowing both additions and deletions of data elements. This model is essential in settings where the dataset evolves dynamically or counts need to be adjusted, such as network traffic monitoring, dynamic graphs, and streaming recommendation updates. Maintaining sublinear sketches under Turnstile updates is more challenging, but prior work has shown that linear sketches, hash-based methods, and RACE-style counters can provide provable guarantees on query accuracy while supporting deletions efficiently (Indyk & Motwani, 1998; Cormode & Muthukrishnan, 2005; Coleman & Shrivastava, 2020). For ANN, fully dynamic LSH and entropy-based methods have also been developed to handle updates in this model (Har-Peled et al., 2012; Andoni et al., 2017).

Sliding Window: In many applications, only the most recent data is relevant. The sliding-window model maintains a succinct summary of the last W updates, automatically expiring older elements. This model is particularly suitable for time-evolving datasets such as streaming video, sensor networks, or financial transactions, where queries should adapt to current trends. Exact computation of statistics like kernel density is often infeasible in this setting, motivating the development of approximate sketches. Techniques such as the EXPONENTIAL HISTOGRAM (Datar et al., 2002), combined with RACE (Coleman & Shrivastava, 2020) or other linear sketches, allow efficient approximation of both ANN and KDE over sliding windows (Wang et al., 2023; Qahtan et al., 2016). These methods balance memory efficiency, update speed, and approximation guarantees, making them practical for large-scale streaming environments.

A.2 Missing Parts and Proofs of Section 3

A.2.1 Proof of Lemma 3.2

Proof. For any $\boldsymbol{p}' \in \mathbb{P} - B(\boldsymbol{q}, cr)$, the probability of collision under a fixed g_j is at most $p_2^k \leq p_2^{\log_{1/p_2} n} = \frac{1}{n}$. Since we store only $n^{1-\eta}$ points, the expected number of such collisions is at most $n^{-\eta}$ for one g_j, and at most $L \cdot n^{-\eta}$ across all L functions. Let $\mathbf{N}$ denote the random variable for the number of collisions. Using Markov's Inequality, we can say that the probability that more than $3L$ such collisions occur is:

$$P(\mathbf{N} \geq 3L) \leq \mathbb{E}\mathbf{N}/3L \leq \frac{L \cdot n^{-\eta}}{3L} = \frac{1}{3n^{\eta}}$$

Thus the success probability for event $\mathbf{E_2}$, $P_2 = 1 - P(\mathbf{N} \geq 3L) \geq 1 - \frac{1}{3n^{\eta}}$. □

A.2.2 Missing details in proof of Lemma 3.3

Probability of no points in the r-ball after uniform sampling

$$\begin{aligned}
\mathbb{P}(\text{No points in the r-ball after uniform sampling}) &= \mathbb{E}[(1-p)^{\mathbf{K}}] \quad \mathbf{K} \sim Poisson(m) \\
&= \sum_{0}^{\infty} (1-p)^k \cdot e^{-m} \frac{m^k}{k!} \\
&= e^{-m} \sum_{0}^{\infty} \frac{(m(1-p))^k}{k!} \\
&= e^{-m} e^{m(1-p)} \\
&= e^{-mp}
\end{aligned}$$

Upper bound on success probability of $\mathbf{E_1}$

$$\begin{aligned}
P(\mathbf{E_1}) &= P(\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r) \text{ such that } g_j(\boldsymbol{p}') = g_j(\boldsymbol{q}) \text{ for some j } \in \{1, \ldots, L\}) \\
&= P(\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r) \wedge g_j(\boldsymbol{p}') = g_j(\boldsymbol{q}) \text{ for some j } \in \{1, \ldots, L\}) \\
&= P(\exists \boldsymbol{p}' \in B(\boldsymbol{q}, r)) \cdot P(g_j(\boldsymbol{p}') = g_j(\boldsymbol{q}) \text{ for some j } \in \{1, \ldots, L\}) \\
&= (1 - e^{-mp}) \cdot P\left(\bigcup_{j=1}^{L} g_j(\boldsymbol{p}') = g_j(\boldsymbol{q}) \right) \\
&= (1 - e^{-mp}) \cdot \left(1 - P\left(\bigcap_{j=1}^{L} g_j(\boldsymbol{p}') \neq g_j(\boldsymbol{q}) \right) \right) \text{ (using } \textit{De Morgan's Law}\text{)} \\
&= (1 - e^{-mp}) \cdot \left(1 - \prod_{j=1}^{L} P(g_j(\boldsymbol{p}') \neq g_j(\boldsymbol{q})) \right) \\
&= (1 - e^{-mp}) \cdot \left(1 - \prod_{j=1}^{L} (1 - P(g_j(\boldsymbol{p}') \neq g_j(\boldsymbol{q}))) \right) \\
&\geq (1 - e^{-mp})(1 - (1 - p_1 n^{-\rho})^L) \text{ (using equation 1)}
\end{aligned}$$

By setting $L = n^{\rho}/p_1$,

$$\begin{aligned}
P_1 &\geq (1 - e^{-mp})(1 - (1 - p_1 n^{-\rho})^{n^{\rho}/p_1}) \\
&\geq (1 - e^{-mp})\{1 - (e^{-p_1 n^{-\rho}})^{n^{\rho}/p_1}\} \\
&= (1 - e^{-mp})(1 - 1/e)
\end{aligned}$$

In the second inequality, we have used $1 - x \leq e^{-x}$.

A.2.3 PROOF OF THEOREM 3.1

Proof. Assume that there exists $\boldsymbol{p}^* \in B(\boldsymbol{q}, r)$ (otherwise, there is nothing to prove). From Lemma 3.1, we can see that for a query to succeed, we require that events $\mathbf{E_1}$ and $\mathbf{E_2}$ occur with constant probability.

From Lemma 3.2 and Lemma 3.3, we can infer that for appropriate choice of k and L, the query fails with probability at most $\frac{1}{3n^\eta} + \frac{e^{mp}+e-1}{e^{mp+1}}$. This proves the failure probability of the theorem.

□

A.2.4 EXTENSION TO BATCH QUERIES

The result extends naturally to the batch streaming setting, where a query consists of a set of points $\mathbb{Q} = \{\boldsymbol{q}_i\}_{i=1}^{B}$. Each batch can be viewed as B independent queries, and the guarantees of Theorem 3.1 apply to each. Moreover, the structure admits straightforward parallelization, making batch queries especially efficient in practice.

Corollary 1. *The Streaming ANN data structure extends to the batch streaming setting, where a query consists of a set $\mathbb{Q} = \{\boldsymbol{q}_i\}_{i=1}^{B}$. In this case:*

- *The data structure stores only $\mathcal{O}(n^{1-\eta})$ points from the stream.*
- *Each batch requires at most $\mathcal{O}\Big(B \cdot \frac{n^\rho}{p_1}\Big)$ distance computations and $\mathcal{O}\Big(B \cdot \frac{n^\rho}{p_1} \cdot \log_{1/p_2} n\Big)$ evaluations of hash functions from $\mathcal{H}$.*
- *The data structure uses at most $\mathcal{O}(n^{1+\rho-\eta}/p_1)$ words of space, in addition to the space required to store $\mathbb{P}$.*

The probability of failure of the batch is at most $B(\frac{1}{3n^\eta} + \frac{e^{mp}+e-1}{e^{mp+1}})$, with each independent query failing with probability at most $\frac{1}{3n^\eta} + \frac{e^{mp}+e-1}{e^{mp+1}}$.

A.2.5 ANN IN TURNSTILE MODEL

For the turnstile model, an arbitrary deletion of points from the data structure may not be effective because an adversary could remove all points except the nearest neighbor within a ball surrounding the query. If our random sampling does not retain this point, the subsequent query would fail. Hence, a natural assumption would be that an adversary is allowed to delete at most d points from any ball of radius r. Now, we can retain the earlier guarantees by bounding the probability that there are at least $d+1$ points in an r-ball surrounding the query. We begin by stating some smaller results that we will require to establish the main result for the turnstile case.

Lemma A.1. *(Tail Bound for a Poisson random variable) Let $\boldsymbol{S} \sim \text{Poisson}(\lambda)$ and $d \leq \lambda$. Then $P(\boldsymbol{S} \leq d) \leq e^{d-\lambda+d\ln\frac{\lambda}{d}}$.*

Proof. For any $t \geq 0$, we can say $P(\mathbf{S} \leq d) = P\big(e^{-t\mathbf{S}} \geq e^{-td}\big) \leq e^{td}\,\mathbb{E}\big[e^{-t\mathbf{S}}\big]$ using Markov's inequality. We also know that the MGF of a Poisson random variable is $\mathbb{E}\big[e^{-tS}\big] = e^{\lambda(e^{-t}-1)}$. Thus, putting it all together, we can say that,

$$P(\mathbf{S} \leq d) \leq e^{\lambda(e^{-t}-1)+td}$$

Define $\varphi(t) := \lambda(e^{-t}-1) + td$. Differentiating and setting $\varphi'(t) = 0$ gives $e^{-t} = d/\lambda$. Since $d \leq \lambda$, we obtain the optimum as $t^\star = \ln\frac{\lambda}{d}$. Substituting $t^\star$ into $\varphi(t)$ yields

$$\varphi(t^\star) = \lambda\big(\tfrac{d}{\lambda} - 1\big) + d\ln\frac{\lambda}{d} = d - \lambda + d\ln\frac{\lambda}{d}$$

Hence, we obtain the result stated in the lemma

$$P(\mathbf{S} \leq d) \leq e^{d-\lambda+d\ln\frac{\lambda}{d}}$$

□

Lemma A.2. *(Poisson Thinning) Let $\boldsymbol{K} \sim \mathrm{Poisson}(m)$ be the number of points in a ball. Suppose each point is kept independently with probability $p \in [0, 1]$. Let $\boldsymbol{S}$ denote the number of points that remain after sampling. Then $\boldsymbol{S} \sim \mathrm{Poisson}(mp)$.*

Proof. Conditioning on $\mathbf{K}$, given $\mathbf{K} = k$, we know that the number of retained points $\mathbf{S}$ follows a $\mathrm{Binomial}(k, p)$ distribution:

$$P(\mathbf{S} = s \mid \mathbf{K} = k) = \binom{k}{s} p^s (1-p)^{k-s}$$

Thus, we can obtain the unconditional probability as

$$\begin{aligned} P(\mathbf{S} = s) &= \sum_{k=s}^{\infty} P(\mathbf{K} = k) P(\mathbf{S} = s \mid \mathbf{K} = k) \\ &= \sum_{k=s}^{\infty} e^{-m} \frac{m^k}{k!} \binom{k}{s} p^s (1-p)^{k-s} \\ &= \sum_{k=s}^{\infty} e^{-m} \frac{(mp)^s}{s!} \cdot \frac{\big(m(1-p)\big)^{k-s}}{(k-s)!} \end{aligned}$$

Summing over $k \geq s$ gives a Poisson tail series that sums to an exponential:

$$P(\mathbf{S} = s) = e^{-m} \frac{(mp)^s}{s!} \sum_{t=0}^{\infty} \frac{\big(m(1-p)\big)^t}{t!} = e^{-m} \frac{(mp)^s}{s!} e^{m(1-p)} = e^{-mp} \frac{(mp)^s}{s!},$$

which is the probability mass function for $\mathrm{Poisson}(mp)$. Therefore $\mathbf{S} \sim \mathrm{Poisson}(mp)$. □

We use these results to prove theorem for ANN under the turnstile model.

Theorem A.4. *Let $(\mathcal{X}, \mathcal{D})$ be a metric space, and suppose there exists an (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$, with $p_1, p_2 \in (0, 1)$, and define $\rho = \frac{\log(\frac{1}{p_1})}{\log(\frac{1}{p_2})}$. We further assume that the number of points contained in any ball of radius r can be modeled as a Poisson random variable with mean m, where $m \geq Cn^{\eta}$ for some constant $C > 0$. Assume that an adversary may delete up to d points from any r-ball (strict turnstile) such that $d \leq mp$. Then, for a point set $\mathbb{P} \subseteq \mathcal{X}$ comprising at most n points, there exists a data structure for turnstile streaming (c, r)-nearest neighbor search with the following guarantees:*

- *The data structure stores only $\mathcal{O}(n^{1-\eta})$ points from the stream.*
- *Each query requires at most $\mathcal{O}(n^{\rho}/p_1)$ distance computations and $\mathcal{O}\left(\frac{n^{\rho}}{p_1} \cdot \log_{1/p_2} n\right)$ evaluations of hash functions from $\mathcal{H}$. The same bounds hold for updates.*
- *The data structure supports arbitrary deletion of points as per the strict turnstile model (up to d points from each r-ball)*
- *The data structure uses at most $\mathcal{O}(n^{1+\rho-\eta}/p_1)$ words of space, in addition to the space required to store $\mathbb{P}$.*

The failure probability is at most $\frac{1}{3n^{\eta}} + \frac{1}{e} + e^{d - mp + d \ln \frac{mp}{d}}(1 - \frac{1}{e})$, which is less than 1 for an appropriate choice of C and d.

Proof. The proof doesn not vary too much from that of the vanilla streaming case. For correctness, we still require Lemma 3.1 to hold. It is easy to see that $\mathbf{E_2}$ as defined in Lemma 3.1, holds trivially on deletion of points under the turnstile model, because the probability of hashing far-away points strictly decreases on deleting points from the data structure.

We need to show that $\mathbf{E_1}$ still holds with sufficiently high probability. We follow a similar approach to Lemma 3.3 to show that after deletion of up to k points, $\exists \boldsymbol{p'} \in B(\boldsymbol{q}, r)$ such that $g_j(\boldsymbol{p'}) = g_j(\boldsymbol{q})$

for some $j \in \{1, \cdots, L\}$. We know that the original data follows a Poisson distribution, so if we say that the number of points in a ball of radius r surrounding a query is a Poisson random variable $\mathbf{K}$ with mean m, we can use Lemma A.2 to say that the number of retained points follows a Poisson distribution with mean mp, where $p = n^{-\eta}$ is the probability that every point is retained independently. Denote this distribution by $\mathbf{S}$.

$$P(\text{At most } d \text{ points lie in an r-ball surrounding the query point}) = P(\mathbf{S} \leq d) \leq e^{d-mp+d\ln\frac{mp}{d}}$$

This implies that the probability of having at least $d+1$ points close to the query is $(1 - e^{d-mp+d\ln\frac{mp}{d}})$.

Now, given that there exists $\boldsymbol{p'} \in B(\boldsymbol{q}, r)$ even on deleting k points in the worst case, we can lower bound the probability that $g_j(\boldsymbol{p'}) = g_j(\boldsymbol{q})$ for some $j \in \{1, \ldots, L\}$ as follows:

$$P(g_j(\boldsymbol{p'}) = g_j(\boldsymbol{q})) \geq p_1^k \geq p_1^{\log_{1/p_2} n+1} = p_1 n^{-\log_{1/p_2}(1/p_1)} = p_1 n^{-\rho} \tag{2}$$

So, the worst case probability of success of event $\mathbf{E_1}$ is:

$$P(\mathbf{E_1}) \geq (1 - e^{d-mp+d\ln\frac{mp}{d}})(1 - (1 - p_1 n^{-\rho})^L) \text{ (using equation 2)}$$

Now, similar to Lemma 3.3, we can set $L = n^\rho/p_1$ to obtain $P_1 = (1 - e^{d-mp+d\ln\frac{mp}{d}})(1 - \frac{1}{e})$. So now, using Lemma 3.2 and the success probability derived above, we can say that for an appropriate choice of k and L, the guarantees of our data structure hold with failure probability $\frac{1}{3n^\eta} + \frac{1}{e} + e^{d-mp+d\ln\frac{mp}{d}}(1 - \frac{1}{e})$. □

A.3 Missing Parts and Proofs of Section 4

A.3.1 Proof of Lemma 4.1

Proof. Suppose the relative error of the estimate from the EXPONENTIAL HISTOGRAM algorithm is ϵ'. So,

$$\begin{aligned} & |Y - X| \leq \epsilon' X \\ & \text{Taking expectation on both sides,} \\ \implies & \mathbb{E}[|Y - X|] \leq \epsilon' \mathbb{E}[X] = \epsilon' \mathbf{K} \text{ (since } \mathbf{K} = \mathbb{E}[X]) \\ \implies & |\mathbb{E}[Y] - \mathbb{E}[X]| \leq \mathbb{E}[|Y - X|] \leq \epsilon' \mathbf{K} \\ \implies & |\mathbb{E}[Y] - \mathbf{K}| \leq \epsilon' \mathbf{K} \\ \implies & \mathbb{E}[Y] \leq (1 + \epsilon')\mathbf{K} \end{aligned} \tag{3}$$

□

A.3.2 Proof of Lemma 4.2

Proof. From 3, $|Y_i - X_i| \leq \epsilon' X_i \implies X_i(1-\epsilon') \leq Y_i \leq X_i(1+\epsilon') \forall i \in \{1, 2, \cdots, r\}$. It follows from ***Hoeffding's inequality***, where we define ϵ as $\epsilon' \mathbb{E}[\hat{Y}]$,

$$\begin{aligned} \mathbb{P}(|\hat{Y} - \mathbb{E}\hat{Y}| > \epsilon' \mathbb{E}[\hat{Y}]) &\leq 2\exp\left(-\frac{2r^2\epsilon^2}{\sum_{i=1}^r (b_i - a_i)^2}\right) \\ &\leq 2\exp\left(-\frac{2\epsilon'^2 \mathbb{E}[\hat{Y}]^2 r^2}{\sum_{i=1}^r (2\epsilon' X_i)^2}\right) \text{ (using } \epsilon = \epsilon' \mathbb{E}[\hat{Y}]) \\ &\leq 2\exp\left(-\frac{2\epsilon'^2 \mathbb{E}[\hat{Y}]^2 r^2}{r(2\epsilon' \max\{X_i\})^2}\right) \\ &\leq 2\exp\left(-\frac{r\mathbb{E}Y_i^2}{2\max\{X_i\}^2}\right) \text{ (using } \mathbb{E}[\hat{Y}] = \mathbb{E}[Y_i]) \end{aligned}$$

To bound this probability by δ, we need :

$$2\exp\left(-\frac{r\mathbb{E}Y_i^2}{2\max\{X_i\}^2}\right) \leq \delta$$
$$\implies r \geq \frac{2\max\{X_i\}^2}{\mathbb{E}[Y_i]^2}\log\left(\frac{2}{\delta}\right) \geq \frac{2\max\{X_i\}^2}{(1+\epsilon')^2\mathbf{K}^2}\log\left(\frac{2}{\delta}\right)$$

Thus $|\hat{Y} - \mathbb{E}\hat{Y}| \leq \epsilon'\mathbb{E}[\hat{Y}]$ holds with probability $1-\delta$ if r satisfies the aforesaid bound. □

A.3.3 Proof of Lemma 4.4

Proof. The number of cells in the modified RACE is RW. Each cell is represented by an EXPONENTIAL HISTOGRAM of space complexity $\mathcal{O}(\frac{1}{\epsilon'}\log^2 N)$ where ϵ' is the relative error of the EXPONENTIAL HISTOGRAM. The relative error for KDE ϵ is related to ϵ' as (using lemma 4.3):

$$\epsilon = 2\epsilon' + \epsilon'^2 \implies \epsilon' = \sqrt{1+\epsilon} - 1$$

Hence, the total space requirement for modified RACE is:

$$RW \cdot \mathcal{O}(\frac{1}{\epsilon'}\log^2 N) = RW \cdot \mathcal{O}(\frac{1}{\sqrt{1+\epsilon}-1}\log^2 N) = \mathcal{O}\left(RW\frac{1}{\sqrt{1+\epsilon}-1}\log^2 N\right)$$

□

A.3.4 Extension to batch queries

We define the dynamic streaming dataset where a batch of data points at a new timestamp t is denoted by $\mathbb{X}^{(t)} = \{\boldsymbol{x}_i^{(t)} \in \mathbb{R}\}_{i=1}^{n_t}$. Let the batch size ($n_t$) be a constant, say R. For the sliding window setting, we will consider the last N batches for the KDE estimation, rather than the last N data points.

Our algorithm can be extended for this setting accordingly. We have to modify the only update step in the EXPONENTIAL HISTOGRAM. The EXPONENTIAL HISTOGRAM has to estimate the number of elements in the last N batches which hash to $h_i(\boldsymbol{q})$. The maximum increment for an EXPONENTIAL HISTOGRAM in a RACE cell at a given time step is R[6]. Datar et al. show that the EXPONENTIAL HISTOGRAM algorithm can be generalized for this problem using at most $(k/2+1)(\log(\frac{2NR}{k}+1)+1)$ buckets, where $k = \lceil\epsilon\rceil$. The memory requirement for each bucket is $\log N + \log(\log N + \log R)$ bits. Hence, we get the following corollary from theorem 4.1.

Corollary 2. *Suppose we are given an* LSH *function with range W. The data comes in batches of size R at every time step. Then the proposed sliding window* RACE *data structure with*

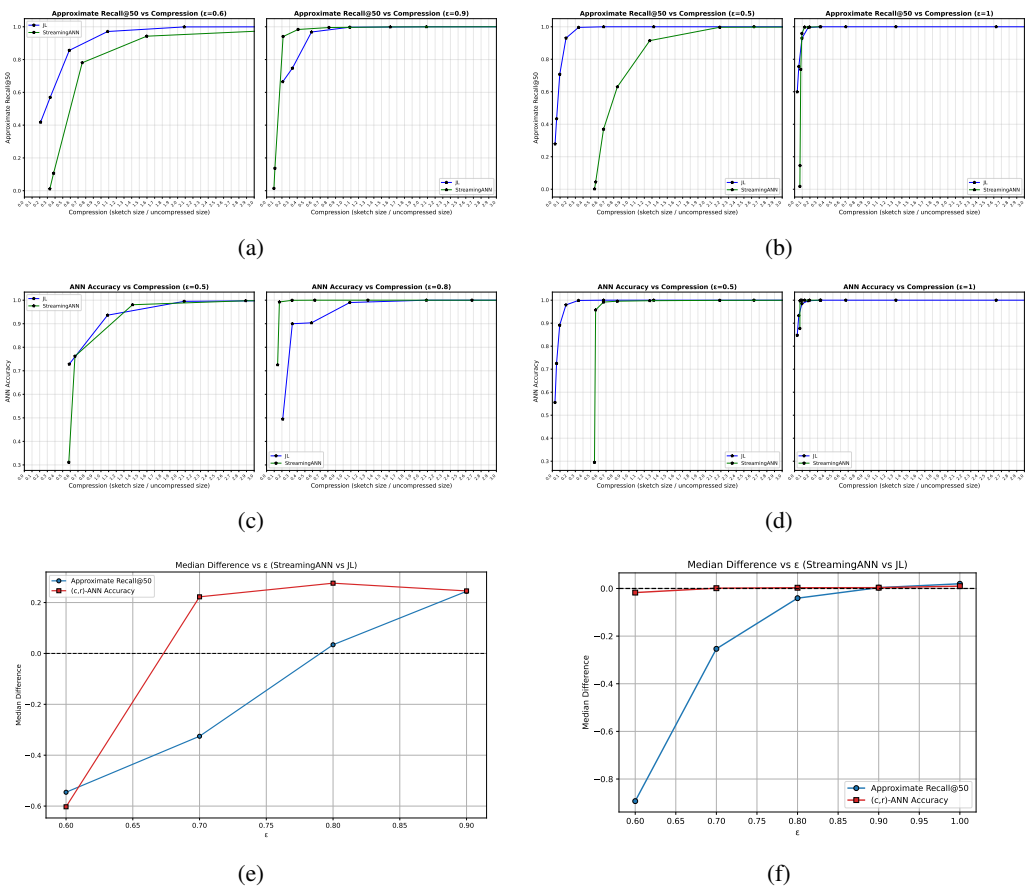
$$r = \mathcal{O}\left(\frac{2\max\{X_i\}^2}{(1+\epsilon)\boldsymbol{K}^2}\log\left(\frac{2}{\delta}\right)\right)$$

independent repetitions of the hash function provide a $1 \pm \epsilon$ multiplicative approximation to $\boldsymbol{K}$ (which is the KDE*) with probability $1-\delta$, using space $\mathcal{O}\left(rW \cdot \frac{1}{\sqrt{1+\epsilon}-1}(\log(2NR\sqrt{1+\epsilon})(\log N + \log(\log N + \log R))\right)$ where N is the window size.*

A.4 Additional Plots from Section 5

Streaming ANN. From Figures 5e and 5f, we see that beyond an appropriate value for ϵ, our algorithm outperforms the JL baseline in terms of both approximate recall and ANN accuracy for both datasets. From 5a, 5b, 5c and 5d, we illustrate the crossing over of curves as we increase epsilon, with both methods achieving similar approximate recalls and accuracies for lower epsilon, but with our algorithm outperforming the baseline for higher epsilon. We can also see that our scheme achieves a reasonable trade-off between performance and sketch size, achieving compression rate < 1 while still maintaining good recall and accuracy for both datasets. Note that in case our parameters allow a sublinear sketch, the compression rates only improve as we scale N.

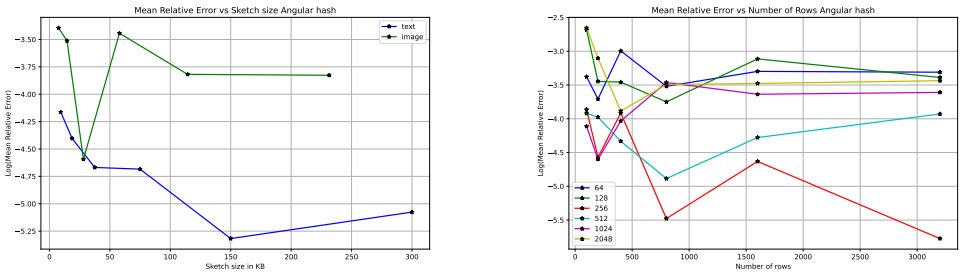
[6]this will happen when all the elements in the current batch of size R hash to the same LSH bucket



(a) (b) (c) (d) (e) (f)

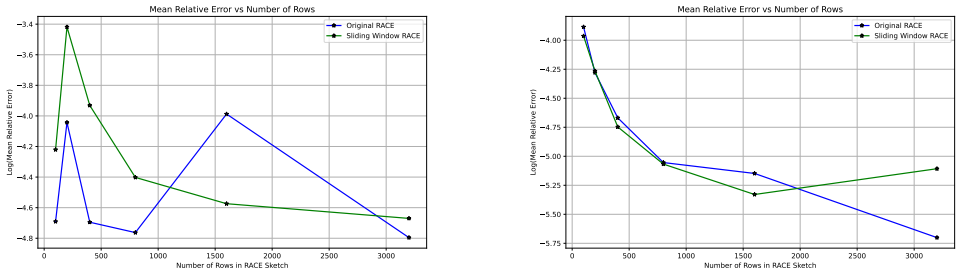
Figure 5: Effect of ϵ on the performance of our scheme. Left column: results for `sift1m`, Right column: results for `fashion-mnist`. (a) and (b) Recall vs Compression rate for 2 values of ϵ, (c) and (d) (c, r)-ANN Accuracy vs Compression Rate for 2 values of ϵ. (e) and (f) Median differences in Approximate Recall and ANN Accuracy over varying ϵ.

A-KDE. From fig. 6a, it is observed that the behaviour of mean relative error is erratic for A-KDE with angular hash, especially with `ROSIS` images. Fig. 6b shows the dependency of mean relative error on varying window sizes: 64,128,256,512,1024,2048. For $N = 256$, the error minimizes, whereas it flattens out for other window sizes. These discrepancies in the plots can be attributed to the underlying data distribution.



(a) (b)

Figure 6: KDE experiments: (a) Effect of sketch size on KDE estimates using Angular hash, (b) Effect of sliding window size on mean relative error for Sliding window A-KDE with angular hash on ROSIS image data.



(a) (b)

Figure 7: Comparison between RACE structure with Angular Hash and Sliding window A-KDE with Angular Hash on two datasets (a) `ROSIS Hyperspectral data` (b) `News headlines`