FAST SUBLINEAR SKETCHES FOR ANN AND KDE

Anonymous authors
Paper under double-blind review

ABSTRACT

In modern applications, massive volumes of input data often arrive as dynamic streams. Storing the entire input can be costly and challenging. Sublinear-space sketches are therefore essential for processing such massive data efficiently. We study Approximate Nearest Neighbor (ANN) search, a fundamental and extensively studied problem in algorithms and machine learning. However, ANN faces a fundamental space barrier in the streaming setting. We establish a simple information-theoretic lower bound ruling out $o(n)$-bit sketches in the worst case, even for insertion-only streams. On the positive side, under natural assumptions on the input data distribution, we design a sketch that retains an $n^{-\eta}$ fraction of the input and uses $\mathcal{O}(n^{(1-\eta)(1+\rho)})$ space for fixed locality-sensitive hashing (LSH) collision probabilities. Here, ρ is the LSH exponent and $0 < \eta < 1$; the space bound is sublinear whenever $\eta > \rho/(1+\rho)$. Our sketch preserves classical LSH approximation guarantees for the full stream while matching its space and query bounds on the retained sample. It supports batch queries and turnstile updates under bounded local deletions. These space–approximation trade-offs complement long line of prior work on dynamic ANN that focused primarily on update time. Experiments on real-world and synthetic datasets demonstrate high approximate recall with compact sketches. On the evaluated real-world benchmarks, our method improves the recall–space trade-off over RACE-CMS (Coleman et al., ICML 2020) and achieves substantially higher query throughput. As a further application of LSH-based sketching, we consider Approximate Kernel Density Estimation (A-KDE), another fundamental problem in data analysis. By incorporating exponential histograms, we obtain the first sublinear-space sketch with provable approximation guarantees for A-KDE in the sliding-window model. While earlier sketches support insertions and deletions in the turnstile model, our construction handles data expiration explicitly while supporting efficient updates and queries. Experiments on real-world and synthetic datasets demonstrate accuracy comparable to RACE (Coleman and Shrivastava, WWW 2020) with a small memory footprint.

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 Har-Peled et al. (2012), Har-Peled et al. presented fully dynamic solutions to ANN, though such

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

methods require at least linear space and are not suited for the streaming model. In parallel, locality-sensitive hashing–based frameworks (e.g., Panigrahy (2006); Andoni & Indyk (2008); Andoni et al. (2014); Andoni & Razenshteyn (2015); Andoni et al. (2017); Ahle (2017)) have driven much of the progress on optimizing space–time trade-offs in the primarily static and 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 $(1+\epsilon)$ 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 REPEATED ARRAY OF COUNTS ESTIMATOR (in short, RACE) Coleman & Shrivastava (2020), which compresses high-dimensional vectors into compact counters; TAKDE Wang et al. (2023), evaluated experimentally 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.

We study the problems under well-studied streaming models, namely, *insertion-only*, *strict turnstile*, and *sliding window*. (See Appendix A.1.5 for definitions, model specifications, and related work.)

1.1 PROBLEM DEFINITIONS

Problem 1 (*Streaming* (c,r)-*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. Maintain a sketch storing only $\mathcal{O}(n^{1-\eta})$ points for $\eta \in (0,1]$, such that for any query $\mathbf{q} \in \mathcal{X}$ with $\mathcal{D}_{\mathcal{P}}(\mathbf{q}) \leq r$, the sketch returns some $\mathbf{p}' \in \mathcal{P} \cap B(\mathbf{q}, cr)$ with probability at least $1-\delta$.

Problem 2 (*Sliding-Window A-KDE*). Let $\{\mathbf{x}_t\}_{t\geq 1}$ be a data stream, where the sampling density is potentially time-varying. 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 $\mathcal{T}_t$ is the current window of size N. Maintain a compact sketch that supports *A-KDE* over the sliding window, enabling efficient updates and queries.

1.2 OUR CONTRIBUTIONS

Our main results concern streaming Approximate Nearest Neighbor (ANN) search (Problem 1). We establish a worst-case space lower bound and show how natural distributional assumptions enable sublinear-space sketches with efficient updates and queries. As a further application of LSH-based sketching, we obtain the first sublinear-space sketch with provable approximation guarantees for

Approximate Kernel Density Estimation (A-KDE) in the sliding-window model (Problem 2). We complement these theoretical results with extensive experiments on real-world and synthetic datasets. Below, we highlight our key contributions.

1.2.1 Our contributions to ANN (Problem 1)

Streaming ANN faces a fundamental space barrier. We prove an $\Omega(n/m)$-bit lower bound for one-pass ANN with per-query success probability at least $2/3$ when nonempty queried r-balls contain $\Theta(m)$ points, which rules out $o(n)$-bit sketches for $m = O(1)$ even in insertion-only streams.

We show that efficient sketching becomes possible under a natural assumption on local data density. Specifically, we model neighborhood occupancy as a Poisson random variable with mean $m \geq Cn^{\eta}$, for a sufficiently large constant $C > 0$ and $0 < \eta < 1$. This assumption is motivated by Poisson point processes, a classical and well-studied model in stochastic geometry Penrose (2003); Daley & Vere-Jones (2003). Our approach complements earlier work that exploits structure in the queries or data Mou & Wang (2017); Coleman et al. (2020). We exploit the classical LSH framework Indyk & Motwani (1998) in a novel manner, combining it with uniform sampling to obtain the following guarantees:

1. **Sublinear-space streaming ANN.** Our sketch retains only $\mathcal{O}(n^{1-\eta})$ points while preserving the (c, r)-ANN approximation guarantee for the full stream. Its space and query bounds match classical LSH on the retained sample. For fixed collision probabilities, the index uses $\mathcal{O}(n^{(1-\eta)(1+\rho)})$ words, which is sublinear when $\eta > \rho/(1+\rho)$, where ρ is the LSH exponent (Theorem 3.4).
2. **Turnstile robustness.** We extend the guarantees to strict turnstile streams with bounded adversarial deletions within each r-ball (Theorem A.8).
3. **Parallel batch queries.** The sketch supports batch queries that can be processed independently in parallel (Corollary A.5).

Empirical validation. Experiments on `sift1m`, `fashion-mnist`, and a synthetic Poisson dataset, `syn-32`, demonstrate high approximate recall with compact sketches. At $\epsilon = 0.5$, the measured sketch sizes are consistent with sublinear scaling for the tested values $\eta \geq 0.5$. Our method improves approximate recall and (c, r)-ANN accuracy over the Johnson–Lindenstrauss (JL) baseline beyond $\epsilon \approx 0.7$–0.8 on `sift1m` and $\epsilon \approx 0.9$ on `fashion-mnist`. It also achieves higher query throughput than JL across the tested workloads. Compared with RACE-CMS Coleman et al. (2020), our method improves the recall–space trade-off and delivers substantially higher query throughput on the evaluated real-world benchmarks.

1.2.2 Our contributions to A-KDE (Problem 2)

The RACE sketch of Coleman & Shrivastava (2020) supports KDE under insertions and deletions in the turnstile model. However, it does not track temporal information and therefore cannot, by itself, expire points outside the most recent N updates.

We address this limitation by replacing each RACE counter with an exponential histogram (EH) Datar et al. (2002).[1] Each histogram estimates how many active-window points hash to the same LSH bucket as the query $\boldsymbol{q}$. By controlling the resulting estimation error, we obtain the *first sublinear-space sketch with provable approximation guarantees for sliding-window A-KDE*, while supporting efficient updates and queries. For fixed approximation accuracy, each histogram uses $\mathcal{O}(\log^2 N)$ bits to maintain the required temporal information. The construction also supports fixed-size batch updates, with the window covering the most recent N batches.

Empirical validation. We evaluate the sketch on `News Headlines`, `ROSIS Hyperspectral Images`, and synthetic multivariate Gaussian datasets. Experiments varying sketch and window sizes show mean relative errors well below the theoretical bound, even with few sketch rows, and accuracy comparable to RACE while supporting data expiration.

For brevity, proofs of the Lemmas and Theorems marked with $(\star)$ are deferred to the Appendix.

[1] An exponential histogram maintains approximate aggregates over the last N stream elements; in particular, it estimates the number of 1's in a binary sliding window within a prescribed relative error.

2 PRELIMINARIES

We briefly summarize the required preliminaries below, deferring the full details to Appendix A.1.

Locality-Sensitive Hashing (LSH): LSH families map points to discrete buckets so that nearby points are much more likely to *collide* than distant points. A useful LSH family therefore has collision probability decreasing with distance. In this work, we use two standard constructions: SRP (Angular) LSH Charikar (2002) and p-stable LSH for Euclidean distances Datar et al. (2004).

Approximate Nearest Neighbor: Har-Peled et al. (2012) present a fully dynamic scheme for the (c, r)-ANN problem with sublinear query and update complexity, building on an (r, cr, p_1, p_2)-sensitive LSH family (Definition A.1). They amplify the gap between near and far collision probabilities by concatenating independent hash functions, store only non-empty buckets across L hash tables, and at query time return the closest point among a small candidate set of colliding points.

Repeated Array-of-Counts Estimator (RACE): Coleman & Shrivastava (2020) propose RACE, an efficient sketching algorithm for KDE on high-dimensional streaming data. RACE compresses a dataset into a 2-D array of integer counters, where each row is an ACE Luo & Shrivastava (2018) structure. For each data point, the hash values are evaluated and the corresponding counters of the array are incremented. The KDE of a query can be estimated by averaging over hash values for all rows of the array. ACE is an unbiased estimator for KDE.

Exponential Histogram (EH): In the BASIC COUNTING problem, over a sliding window of the last N elements in a binary data stream, the goal is to estimate the number of 1's. Datar et al. (2002) proposed the EH algorithm, which gives a $(1+\epsilon)$-approximation of this count at all times. EH groups the 1's into buckets and dynamically updates the bucket size. The query time is $\mathcal{O}(1)$. The algorithm maintains invariants that ensure the relative error is at most ϵ, and space-complexity $\mathcal{O}(\frac{1}{\epsilon}\log^2 N)$.

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.

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. The S-ANN algorithm as given in Algorithm 1 describes a scheme that inserts points from the stream into our sketch and employs the *Query Processing* routine to solve the (c, r)-ANN problem for any query point $\boldsymbol{q}$. The insert and query mechanisms for our sketch are illustrated in Figure 1.

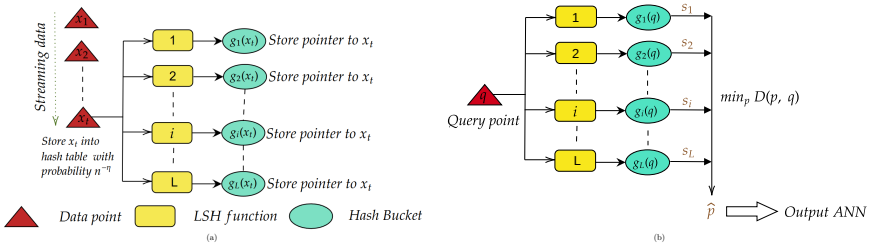


Figure 1: Algorithm 1 (a) Insert mechanism, where with probability $n^{-\eta}$, we store point $\boldsymbol{x}_t$ into the hash table using L independent hash functions $g_i(\boldsymbol{x})$. (b) Query mechanism which retrieves the closest point to the query vector $\boldsymbol{q}$ from a *candidate set* $\mathbb{S}$ constructed as the union of all collision sets, *i.e.* $\mathbb{S} = \bigcup_{i=1}^{L} s_i$, with s_i denoting the set of points that collide with $\boldsymbol{q}$ in the ith hash table.

Algorithm 1 Streaming ANN (S-ANN)

Input: data stream $\mathbb{D}$, query point $\boldsymbol{q}$, LSH family $\mathcal{H}$, parameters k, L, sampling parameter η
Output: approximate nearest neighbor of $\boldsymbol{q}$ or **NULL**
Initialize $\{g_j\}_{j=1}^{L}$ where each $g_j : \mathcal{X} \to \mathcal{U}^k$ concatenates k independent draws from $\mathcal{H}$
for each point $\boldsymbol{p} \in \mathbb{D}$ **do**
  Draw $x \sim \text{Bernoulli}(n^{-\eta})$
  if $x = 1$ **then**
    Store $\boldsymbol{p}$ ▷ *Random sampling to decide whether or not point stored*
    for $j = 1$ **to** L **do**
      Insert $\boldsymbol{p}$ into bucket $g_j(\boldsymbol{p})$ ▷ *Insert point into all L independent hash tables*
    end for
  end if
end for

Query Processing:
Initialize candidate set $\mathbb{C} \leftarrow \emptyset$ ▷ $\mathbb{C}$ *stores points colliding with $\boldsymbol{q}$ across L independent hash tables*
for $j = 1$ **to** L **do**
  $\mathbb{C} \leftarrow \mathbb{C} \cup g_j(\boldsymbol{q})$
  if $|\mathbb{C}| \geq 3L$ **then**
    break ▷ *Enough candidates gathered, no need to query remaining tables*
  end if
end for
Remove duplicates from $\mathbb{C}$
$p^* \leftarrow \arg\min_{\boldsymbol{p} \in \mathbb{C}} D(\boldsymbol{p}, \boldsymbol{q})$ ▷ *Retrieve the nearest point from the candidate set*
if $D(p^*, \boldsymbol{q}) \leq cr$ **then**
  return p^*
else
  return NULL
end if

3.1 Analysis and Guarantees

Correctness. We establish correctness of the Streaming (c, r)-ANN sketch via two events whose joint occurrence implies query success. Lemma 3.1 shows that these events are sufficient. We then prove that both events hold with high probability while storing only a sublinear uniform sample of the stream. Lemma 3.2 fixes the amplification parameter k to make collisions with far points negligible after sampling. Lemma 3.3, under a local-density assumption, fixes the number of hash tables L so that a near neighbor survives sampling and collides with the query with high probability. These bounds combine to yield Theorem 3.4, which provides a sublinear Streaming (c, r)-ANN sketch.

We first establish terminology used throughout this section. An LSH-*amenable* metric space, $(\mathcal{X}, \mathcal{D})$, is one that admits an (r, cr, p_1, p_2)-sensitive LSH family $\mathcal{H}$. We define $\rho := \frac{\log(1/p_1)}{\log(1/p_2)}$ as a measure of hash family quality: p_1 and p_2 are the collision probabilities for near and far points respectively, so a smaller ρ corresponds to a sharper, more discriminating hash family. In Lemma 3.1 below, we decompose correctness into two events: $\mathbf{E_1}$, that at least one genuine near neighbor survives sub-sampling and collides with the query in some hash table; and $\mathbf{E_2}$, that false candidates (points beyond distance cr) are few enough that a true one can still be found. When both hold, the algorithm is guaranteed to return a point within distance cr of the query.

Lemma 3.1 ($\star$). *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 $\mathbb{P} - B(\boldsymbol{q}, cr)$ 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

Note that the point $\mathbf{p}'$ need not be the nearest neighbor: any point within distance r that survives sampling and collides with the query suffices. By Lemma 3.1, we bound the failure probabilities of $\mathbf{E_1}$ and $\mathbf{E_2}$ separately. For $\mathbf{E_2}$, sampling reduces the number of far points, while increasing k lowers their collision probabilities. Tuning k to η keeps the expected number of spurious collisions per hash table bounded by a constant, regardless of the data distribution.

Lemma 3.2 ($\star$). *Setting $k = \lceil (1-\eta)\log_{1/p_2} n \rceil$, for a query $\boldsymbol{q}$, if we store $\mathcal{O}(n^{1-\eta})$ input points, event $\mathbf{E_2}$ succeeds with probability $P_2 \geq \frac{2}{3}$, independent of the data distribution.*

To guarantee $\mathbf{E_1}$, a near neighbor must both survive sub-sampling and collide with the query in at least one hash table. The Poisson occupancy assumption makes the first requirement tractable by giving a clean characterization of how many near neighbors are retained, enabling us to argue at least one survives with good probability. The choice of L ensures collision via a standard LSH argument.

Lemma 3.3 ($\star$). *Assume that the number of points in any r-neighborhood follows a $\mathrm{Poisson}(m)$ distribution. Let $k = \lceil (1-\eta)\log_{1/p_2} n \rceil$ and $L = n^{(1-\eta)\rho}/p_1$. Then, for any fixed 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.*

Having established the failure bounds for $\mathbf{E_1}$ and $\mathbf{E_2}$ in Lemmas 3.3 and 3.2, and their sufficiency for query success in Lemma 3.1, we can now conclude our main guarantee.

Theorem 3.4. *Let $(\mathcal{X}, \mathcal{D})$ be an* LSH-amenable *metric space. Assume $\mathrm{Poisson}(m)$ occupancy for all r-neighborhoods, with $m \geq Cn^{\eta}$. Then, for any point set $\mathbb{P} \subseteq \mathcal{X}$ with $|\mathbb{P}| \leq n$, there exists a sketch supporting Streaming (c, r)-ANN with the following guarantees:*

- *The sketch stores only $\mathcal{O}(n^{1-\eta})$ points from the stream and uses at most $\mathcal{O}(n^{(1-\eta)(1+\rho)}/p_1)$ words of space. This is sub-linear in n if $\eta > \frac{\rho}{1+\rho}$.*
- *Each query and update requires at most $\mathcal{O}\big(n^{\rho(1-\eta)}/p_1\big)$ distance computations and $\mathcal{O}\Big((1-\eta)\frac{n^{\rho(1-\eta)}}{p_1} \cdot \log_{1/p_2} n\Big)$ evaluations of hash functions from $\mathcal{H}$.*

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

We note that the above ideas readily extends to the batch setting[2] (see Appendix A.2.4).

ANN in the Turnstile Model. We now extend our scheme to the strict turnstile model. Handling deletions introduces a fundamental new difficulty: an adversary can remove points from within a query's r-neighborhood, potentially eliminating every near neighbor from the sketch and making correct retrieval impossible. As discussed in Section 1.2.1, this is not a limitation of our approach but rather an information-theoretic barrier: any sublinear-space algorithm must contend with the possibility that adversarial deletions erase all local geometric structure. To circumvent this, we impose a mild bounded-deletion assumption: the adversary may delete at most d points from any r-neighborhood, ensuring that local structure is not completely erased. Under this assumption and the Poisson occupancy model, we show that sufficiently many near neighbors survive both sub-sampling and adversarial deletion so that at least one witness remains available at query time.

Theorem 3.5 (Informal). *Under the same Poisson occupancy condition as Theorem 3.4, and provided an adversary deletes at most d points from any r-neighborhood with d smaller than the expected number of survivors, there exists a turnstile streaming sketch that matches the space, query, and update guarantees of Theorem 3.4 with constant failure probability.*

We state the result formally in Theorem A.8 and defer the complete details to Appendix A.2.5.

3.2 Lower bound

We formalize the space barrier outlined in Section 1.2.1 using the (r, cr) decision problem on n points: given a query $\boldsymbol{q}$, distinguish whether some point lies within distance r or all points lie beyond cr, with one of these cases promised. We assume that each nonempty queried r-ball contains $\Theta(m)$ points.

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

Theorem 3.6 ($\star$)**.** *Any one-pass streaming algorithm for this problem with per-query success probability at least* $2/3$ *requires* $\Omega(n/m)$ *bits of memory.*

Remark 3.7 (Consequences of the space lower bound). Theorem 3.6 implies that sparse neighborhoods, with $m = O(1)$, require $\Omega(n)$ bits of memory, ruling out $o(n)$-bit one-pass algorithms in the worst case. More generally, when $m = \Theta(n^\eta)$, the lower bound becomes $\Omega(n^{1-\eta})$ bits. This bound applies to arbitrary retention policies, including adaptive and data-dependent ones, and does not rely on the Poisson assumption.

The dependence on n coincides with that of our sample size, although the lower bound is measured in bits rather than retained points. Up to logarithmic factors and point-representation costs, the gap between our space bound and this lower bound is the $n^{\rho(1-\eta)}$ factor contributed by the LSH tables. This overhead reflects the repeated-table construction of Har-Peled et al. (2012). Reducing this gap while preserving our query and update guarantees is a natural direction for future work.

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 Sliding window AKDE, in short SW-AKDE, which uses a modified RACE structure to make it suitable for a sliding window model by using EH 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.

4.1 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 Datar et al. (2002), where the incoming stream of data is 1 if the counter is incremented at a time instant, otherwise 0. We will store an EH for each cell of RACE. On querying the EH, we will get an estimate of the count in a particular cell. In the RACE sketch, the estimator is computed using the median of means procedure. For SW-AKDE, we will take the average of ACE estimates over L independent repetitions.

We defer the correctness analysis and the approximation, space, and query-time guarantees of SW-AKDE to the Appendix A.3.1. The extension to batch updates is discussed in Appendix A.3.2.

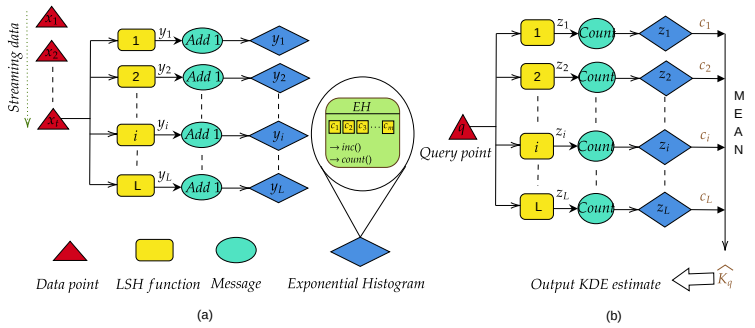


Figure 2: Algorithm 2 (a) Update mechanism where y_i is the output of the LSH function i on x_t. For the ith row, we add 1 to the EH at index y_i (b) Query mechanism where z_i is the output of the LSH function i on q. For the ith row, c_i is the count estimate of the EH at index z_i.

Algorithm 2 Approximate KDE Sketch for Sliding Window (SW-AKDE)

Input: data set $\mathbb{D}$, LSH family $\mathcal{H}$ (range W), parameters k, L, N
Output: approximate KDE value at query $\boldsymbol{q}$
Initialize L hash functions $\{h_1, \ldots, h_L\}$, each formed by concatenating p independent hashes from $\mathcal{H}$
Initialize sketch $A \leftarrow \emptyset$ where $A \in \mathbb{R}^{L \times W^p}$, timestamp $t \leftarrow 0$
for each point $\boldsymbol{p} \in \mathbb{D}$ **do**
  for $i = 1$ **to** L **do**
    $j \leftarrow h_i(\boldsymbol{p})$
    if $A[i, j]$ is empty **then**
      Create an exponential histogram with parameters k, N at $A[i, j]$ with timestamp t
    else
      Insert a count of 1 into the exponential histogram at $A[i, j]$ with timestamp t
    end if
  end for
  $t \leftarrow t + 1$
end for
Initialize KDE estimate $y \leftarrow 0$
for $i = 1$ **to** L **do**
  if $A[i, h_i(\boldsymbol{q})]$ is not empty **then**
    Let c be the estimated count from the exponential histogram at $A[i, h_i(\boldsymbol{q})]$
    $y \leftarrow y + c$
  end if
end for
$y \leftarrow y/L$
return y

5 EXPERIMENTS

We now present the empirical evaluation of our proposed sketching algorithms to validate their theoretical guarantees and to assess their practical performance across standard benchmark datasets[3].

5.1 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 $\eta > \rho$. Finally, we assess the query time performance of our scheme in comparison to the baseline to understand its computational efficiency.

Datasets. Experiments were conducted on two standard ANN benchmarks Aumüller et al. (2020): `sift1m` Jégou et al. (2011) (1M vectors, 128-dimensions) and `fashion-mnist` Xiao et al. (2017) (60,000 images, 784-dimensions), and a synthetic dataset[4]: `syn-32` (100,000 points, 32-dimensions).

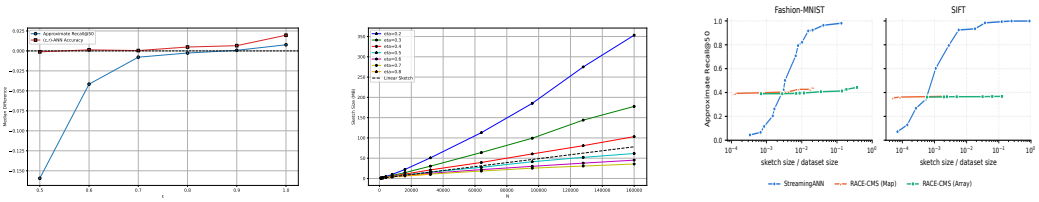
Implementation. All data structures were implemented in Python, assuming `float32` vectors. Compression is measured relative to $N \times d \times 4/1024^2$ MB. No additional memory optimizations were applied. We used the p-stable scheme described in Datar et al. (2004) for hashing.

Experimental Setup. We measure the number of points within distance r of a typical query (Table 2), choosing r so that at most 5% of the dataset lies within the acceptance radius cr to keep queries selective. We compare against RACE-CMS (Algorithms 1–2 in Coleman et al. (2020)) on SIFT ($n = 2 \cdot 10^5$–10^6) and Fashion-MNIST ($n = 5.5 \cdot 10^4$), using a shared insertion/query split, exact ground truth, and separate hyperparameter sweeps to obtain each method's Pareto frontier. For the JL comparison, we vary ϵ from 0.5 to 1 and control compression through k (JL) and η (S-ANN). Each run uses $50{,}000$ stored points and $5{,}000$ queries with $r = 0.5$, measuring approximate recall@50, (c, r)-ANN accuracy, and memory usage. A separate timing comparison fixes $\epsilon = 0.5$

[3] Our codes are available in https://github.com/VedTheKnight/Streaming-ANN-and-Sliding-Window-KDE.git

[4] A synthetic dataset generated by sampling points, uniformly randomly from a Poisson point process (PPP).

with 10,000 stored points and 100 queries, again varying k and η and measuring average recall and query throughput (queries per second). Finally, we examine sketch-size scaling on `sift1m` at $\epsilon = 0.5$, varying η from 0.2 to 0.8 and N from 1,000 to 160,000.



(a) (b) (c)

Figure 3: (a) Median difference plots for `sift1m` (b) Memory requirements vs stream size (N) for fixed ϵ for `sift1m` (c) Comparison of Recall vs. sketch size between S-ANN and RACE-CMS across datasets.

Table 1: Query throughput (queries/second, single-query, identical workload).

Dataset	n	RACE-CMS	Ours	speed-up
Fashion-MNIST	$5.5 \cdot 10^4$	413	9,300	23×
SIFT	$2 \cdot 10^5$	121	10,600	87×
SIFT	$5 \cdot 10^5$	47	10,400	222×
SIFT	10^6	24	9,500	398×

Discussion. The occupancy measurements in Table 2 reveal populated query neighborhoods despite high ambient dimension, supporting the local-density premise of our analysis. Across the tested regimes, S-ANN achieves high recall with compact sketches whose measured growth is consistent with sublinear scaling (Figure 4). It improves the recall–space trade-off over RACE-CMS and delivers substantially higher query throughput, which remains nearly stable across the evaluated input sizes (Figure 3c and Table 1). The methods exploit different structural properties: RACE-CMS relies on recoverability of the similarity vector, whereas our sketch uses local density and direct LSH probing. Against JL, S-ANN achieves higher recall and ANN accuracy in the tested larger-ϵ regimes and higher throughput across the evaluated workloads. See the Appendix A.4 for the complete discussion, and Appendix A.5 for the A-KDE experiments.

6 Conclusion

We presented new sketching algorithms for two fundamental streaming problems: Approximate Nearest Neighbor (ANN) search and Approximate Kernel Density Estimation (A-KDE). For ANN, we showed that under Poisson-distributed inputs, storing only a sublinear fraction of the stream suffices to preserve (c, r)-ANN guarantees, yielding simple, turnstile-robust sketches that support both single and batch queries and outperform the Johnson–Lindenstrauss baseline in accuracy and throughput. For A-KDE, we delicately integrated RACE with EH to design the first sketch that works in the general sliding-window model, providing provable error guarantees and empirical performance comparable to RACE while efficiently handling data expiration.

Several open directions arise from this work. For ANN, it would be valuable to understand how the guarantees extend beyond the Poisson model to other distributional settings, and to characterize finer trade-offs between space, time, and approximation quality. Another promising direction is to explore data-aware variants of LSH Andoni & Razenshteyn (2015); Andoni et al. (2018) that adapt sketch parameters to the underlying distribution. For A-KDE, our experiments (Fig. 9) show that the sliding-window size significantly influences performance, raising the natural question of how to select this parameter optimally—potentially as a function of the relative error of the EH Datar et al. (2002), the sketch width, or the observed data dynamics. Developing adaptive mechanisms for adjusting the window size based on the evolving data distribution remains an intriguing direction for future work.

AI USE STATEMENT

We used generative AI tools to assist with language polishing and minor editorial revisions to improve the clarity and readability of the manuscript. The authors take full responsibility for the final content, including all technical claims and experimental results.

ETHICS STATEMENT

We do not identify any ethical concerns specific to this work.

REPRODUCIBILITY STATEMENT

Our codes are available in https://github.com/VedTheKnight/Streaming-ANN-and-Sliding-Window-KDE.git. In this repository, we have implemented the algorithms mentioned in the paper, generated synthetic datasets and have used real-world datasets with proper references.

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. doi: 10.1109/FOCS.2017.91.

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. doi: 10.1145/237814.237823.

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. doi: 10.1145/1327452.1327494.

Alexandr Andoni and Ilya P. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pp. 793–801. ACM, 2015. doi: 10.1145/2746539.2746553.

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. doi: 10.1137/1.9781611973402.76.

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. doi: 10.1137/1.9781611974782.4.

Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya P. Razenshteyn, and Erik Waingarten. Data-dependent hashing via nonlinear spectral gaps. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pp. 787–800. ACM, 2018. doi: 10.1145/3188745.3188846.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020. doi: 10.1016/J.IS.2019.02.006.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pp. 380–388, 2002. doi: 10.1145/509907.509965.

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. doi: 10.1145/3366423.3380244.

Benjamin Coleman, Richard G Baraniuk, and Anshumali Shrivastava. Sub-linear memory sketches for near neighbor search on streaming data. In *Proceedings of the 37th International Conference on Machine Learning,ICML 2020, 13-18 July 2020, Virtual Event*, pp. 2089–2099. PMLR, 2020.

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. doi: 10.1016/j.jalgor.2003.12.001.

Daryl J. Daley and David Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, 2003. doi: 10.1137/1032025.

Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002. doi: 10.1137/S0097539701398363.

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. doi: 10.1145/997817.997857.

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. doi: 10.1007/978-1-4419-8339-8\-13.

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. ISBN 1558606157. URL http://www.vldb.org/conf/1999/P49.pdf.

Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.*, 2012. doi: 10.4086/TOC.2012.V008A014.

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. doi: 10.1145/276698.276876.

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. doi: 10.1109/TPAMI.2010.57.

Rohit Kulkarni. A million news headlines, 2017. URL https://www.kaggle.com/datasets/therohk/million-headlines.

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. doi: 10.1145/3178876.3186056.

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. doi: 10.1137/1.9781611974775.18.

S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005. doi: 10.1561/0400000002.

Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pp. 1186–1195. ACM Press, 2006.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962. doi: 10.1214/aoms/1177704472.

Mathew Penrose. *Random Geometric Graphs*. Oxford University Press, 2003. doi: 10.1093/acprof:oso/9780198506263.001.0001.

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. doi: 10.1109/TKDE.2016.2626441.

David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015. doi: 10.1002/9781118575574.

Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018. doi: 10.1201/9781315140919.

V. Sowmya, K. P. Soman, and M. Hassaballah. *Hyperspectral Image: Fundamentals and Advances*, pp. 401–424. Springer International Publishing, Cham, 2019. doi: 10.1007/978-3-030-03000-1_16.

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. doi: 10.1109/TPAMI.2023.3297950.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. URL `https://arxiv.org/abs/1708.07747`.

A Appendix

A.1 Missing Parts of Preliminaries

A.1.1 Locality Sensitive Hashing

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

Definition A.1. A family $\mathcal{H} = \{h : \mathbb{X} \to \mathbb{U}\}$ is (r_1, r_2, p_1, p_2)-*sensitive* for $(\mathbb{X}, D)$ 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), described in Charikar (2002) and (2) p-stable LSH, described in Datar et al. (2004).

A.1.2 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.2 (Har-Peled et al. (2012))**.** *Suppose there is an (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 an (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 pre-processing, 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.1.3 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 Arrays of (locality-sensitive) Counts Estimator(ACE) 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. If each hash function $h_i(\boldsymbol{x})$ is constructed by concatenating p independent hashes from LSH family $\mathcal{H}$ with finite range $[1, W]$, then $R = W^p$ is the range of $h_i(\boldsymbol{x})$.

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 (2018) is:

Theorem A.3 (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 $\boldsymbol{A}$ 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. (2020) have shown a tight bound on the variance.

Theorem A.4. *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.1.4 Exponential Histogram

We want to solve the Basic Counting problem for data streams with respect 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 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 EH to maintain the count of active 1's in that they are present within the last N elements. Every bucket in the histogram

[5]ACE is a swift 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.

maintains the timestamp of the most recent 1, and the number of 1's is 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 ith 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 least $\frac{k}{2}$ and at most $\frac{k}{2} + 1$ 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.1.5 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 (in short, EH) 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 3.1

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 $j \in \{1, \ldots, L\}$, meaning this p' will be added to

the candidate set unless the search loop terminates early. If the loop does terminate early due to the condition $|\mathbb{C}| \geq 3L$, Event $\mathbf{E_2}$ guarantees that the total number far points across all L tables is strictly less than $3L$, by the pigeonhole principle $\mathbb{C}$ must contain at least one point that is within distance cr. Since we ultimately return a point that is closest to the query point $\boldsymbol{q}$, we are guaranteed to return $\boldsymbol{p}'$ or another valid near neighbour in either scenario.

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

A.2.2 Proof of Lemma 3.2

For any $\boldsymbol{p}' \in \mathbb{P} - B(\boldsymbol{q}, cr)$, the probability of collision under a fixed g_j is at most $p_2^k \leq p_2^{(1-\eta)\log_{1/p_2} n} = n^{-(1-\eta)}$. Since only $n^{1-\eta}$ points are stored, the expected number of such collisions is at most $n^{1-\eta} \cdot n^{-(1-\eta)} = 1$ for every g_j, and at most $L \cdot 1 = L$ across all L functions. Let $\mathbf{N}$ denote the total number of collisions. Using Markov's Inequality, we can say that the probability that more than $3L$ such collisions occur is $P(\mathbf{N} \geq 3L) \leq \frac{\mathbb{E}[\mathbf{N}]}{3L} \leq \frac{L}{3L} = \frac{1}{3}$. Thus, $\mathbf{E_2}$ succeeds with probability $P_2 = 1 - P(\mathbf{N} \geq 3L) \geq \frac{2}{3}$.

A.2.3 Proof of Lemma 3.3

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:

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

where $p = n^{-\eta}$ is the probability that we keep 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:

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

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

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

By setting $L = n^{(1-\eta)\rho}/p_1$,

$$
\begin{aligned}
P(\mathbf{E_1}) = P_1 &\geq (1 - e^{-mp})(1 - (1 - p_1 n^{-(1-\eta)\rho})^{n^{(1-\eta)\rho}/p_1}) \\
&\geq (1 - e^{-mp})\{1 - (e^{-p_1 n^{-(1-\eta)\rho}})^{n^{(1-\eta)\rho}/p_1}\} \\
&= (1 - e^{-mp})(1 - 1/e)
\end{aligned}
$$

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

A.2.4 Extension to Batch Queries

The result of Theorem 3.4 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 3.4 apply to each of them. Moreover, the structure admits straightforward parallelization, making batch queries especially efficient in practice.

Corollary A.5. *The Streaming (c, r)-ANN sketch of Theorem 3.4 extends to batch queries $\mathbb{Q} = \{\boldsymbol{q}_i\}_{i=1}^{B}$. The sketch stores $\mathcal{O}(n^{1-\eta})$ points and uses $\mathcal{O}(n^{(1-\eta)(1+\rho)}/p_1)$ space. Each batch requires $\mathcal{O}(Bn^{(1-\eta)\rho}/p_1)$ distance computations and $\mathcal{O}\Big(Bn^{(1-\eta)\rho}/p_1 \cdot (1-\eta)\log_{1/p_2} n\Big)$ hash evaluations. The failure probability for a batch query is at most $B\big(\frac{1}{3} + \frac{e^{mp}+e-1}{e^{mp+1}}\big)$.*

A.2.5 Details of ANN in the Turnstile Model

We now extend our scheme to the strict turnstile model as defined in Appendix A.1.5. Handling deletions introduces a fundamental new difficulty: an adversary can remove points from within a query's r-neighborhood, potentially eliminating every near neighbor from the sketch and making correct retrieval impossible. As discussed in Section 1.2.1, this is not a limitation of our approach but rather an information-theoretic barrier: any sublinear-space algorithm must contend with the possibility that adversarial deletions erase all local geometric structure.

To circumvent this, we impose a mild bounded-deletion assumption: the adversary may delete at most d points from any r-neighborhood. This ensures that local structure is not completely erased, which is necessary for any sublinear-space ANN algorithm to provide correctness guarantees in this setting. Under this assumption, we show that the Poisson occupancy model guarantees sufficiently many near neighbors survive both sub-sampling and adversarial deletion, so that at least one witness remains available to the query. The argument proceeds in a manner similar to analysis of the *insertion-only* case, with two additional ingredients to handle the deletion: a tail bound controlling the probability that too few points survive inside the query's r-ball, and the Poisson thinning property, which characterizes the distribution of survivors after sub-sampling. We state both results formally below.

Lemma A.6. *(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.7. *(Poisson Thinning) Let* $\boldsymbol{K} \sim \text{Poisson}(m)$ *be the number of points in a ball. Suppose each point is kept independently with probability* $p \in [0,1]$ *during random sampling. Let* $\boldsymbol{S}$ *denote the number of points that remain after sampling. Then* $\boldsymbol{S} \sim \text{Poisson}(mp)$.

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

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

Thus, we can obtain the probability density function of $\mathbf{S}$ 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 $\text{Poisson}(mp)$. Therefore $\mathbf{S} \sim \text{Poisson}(mp)$. □

Having established the two auxiliary lemmas, we now state and prove the main guarantee for Streaming (c,r)-ANN in the turnstile model.

Theorem A.8. *Let* $(\mathcal{X}, \mathcal{D})$ *be an* LSH-amenable *metric space. Assume* $\text{Poisson}(m)$ *occupancy for all* r*-neighborhoods, with* $m \geq Cn^\eta$*. 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* $\mathbb{P} \subseteq \mathcal{X}$ *with* $|\mathbb{P}| \leq n$*, there exists a sketch for turnstile Streaming* (c,r)*-ANN search that inherits all the space, query and update guarantees of Theorem 3.4 with failure probability at most* $\frac{1}{3} + \frac{1}{e} + e^{d-mp+d\ln\frac{mp}{d}}(1-\frac{1}{e})$*, which is less than 1 for an appropriate choice of* m *and* d*.*

Proof. The proof does 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 follow 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.7 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 d 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:

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

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

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

Now, similar to Lemma 3.3, we can set $L = n^{(1-\eta)\rho}/p_1$ 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 d and L, the guarantees of our data structure hold with failure probability $\frac{1}{3} + \frac{1}{e} + e^{d-mp+d\ln\frac{mp}{d}}(1 - \frac{1}{e})$. □

A.2.6 Proof of Theorem 3.6

Let $k = \lfloor n/m \rfloor$. We work in $\mathbb{R}^d$ with $d = \Theta(\log k)$, which admits $3k$ points at pairwise distance greater than $2cr$ (e.g. a suitably scaled lattice); fix such a set and use it for the locations below. Given an arbitrary bit string $x \in \{0,1\}^k$, for each $i \in [k]$ fix a query point $\boldsymbol{q}_i$, a nearby location a_i within distance r of $\boldsymbol{q}_i$, and a far location b_i. For each bit x_i, insert a cluster of m points near a_i if $x_i = 1$, and near b_i if $x_i = 0$. The stream then contains at most n points, and every populated r-ball contains $\Theta(m)$, satisfying the promise. By the $2cr$ separation, $\boldsymbol{q}_i$ has a data point within distance r if and only if $x_i = 1$; when $x_i = 0$, every data point lies at distance greater than cr from $\boldsymbol{q}_i$, since a point near a_j for $j \neq i$ is at distance greater than $2cr - r > cr$ from $\boldsymbol{q}_i$.

This construction yields a one-way communication protocol for INDEX. Alice runs the streaming algorithm on the point set encoding x and sends its memory state to Bob. Given an index i, Bob issues the single query $\boldsymbol{q}_i$; a correct near/far answer reveals x_i. (The algorithm returns an actual stored point and can verify $D(\boldsymbol{p}^*, \boldsymbol{q}) \leq cr$, so it solves the decision problem; our sketch does exactly this.) Since Bob makes only one query, the protocol inherits the algorithm's constant success probability, and any randomized one-way protocol for INDEX on k bits requires $\Omega(k)$ bits. Hence the streaming algorithm must use $\Omega(k) = \Omega(n/m)$ bits of memory.

A.3 Missing Parts and Proofs of Section 4

A.3.1 Correctness

Consider the ACE Estimator. We know that if we have the actual counts (by using a counter instead of EH), then $X = \boldsymbol{A}[h(\boldsymbol{q})]$ is an unbiased estimator for the Kernel Density with bounded variance(A.4). So, $\mathbf{K} = \mathbb{E}[X]$ is the Kernel Density estimate. We will now demonstrate that, by using EH, the new estimator for a single ACE instance approximates $\mathbf{K}$ with a certain level of accuracy.

Lemma A.9. *Let Y be the new estimator obtained from querying the* EH *at* $\boldsymbol{A}[h(\boldsymbol{q})]$*. Then,* $\mathbb{E}[Y] \leq (1+\epsilon')\boldsymbol{K}$*.*

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

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

□

We use L independent ACE instances to estimate the kernel density. Hence, the KDE estimator in the current setting is,

$$\hat{Y} = \frac{1}{L}\sum_{i=1}^{L} Y_i \quad \text{where } Y_i \text{ is the EH estimate for } \boldsymbol{A}[h_i(\boldsymbol{q})]$$

The expectation of $\hat{Y}$ is $\mathbb{E}Y_i$ and the variance of $\hat{Y}$ is $\frac{1}{L}Var(Y_i)$. Now we will show the bounds of the estimator $\hat{Y}$ and derive the necessary bounds on L.

Lemma A.10. $|\hat{Y} - \mathbb{E}\hat{Y}| < \epsilon'\mathbb{E}[\hat{Y}]$ *holds with probability* $1-\delta$ *if:*

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

where ϵ' *is the relative error of* EH*,* L *is the number of repetitions of* ACE *(or the number of rows in the* SW-AKDE *sketch).*

Proof. From 3, $|Y_i - X_i| \leq \epsilon' X_i \implies X_i(1-\epsilon') \leq Y_i \leq X_i(1+\epsilon') \forall i \in \{1, 2, \cdots, L\}$. Here, X_i refers to the actual count in a cell in the last N time steps, if we had used a counter instead of an EH for the ith hash function.

Note that, in order to estimate the lower bound for L, we need the maximum of X_i's for $i \in [L]$. Therefore, we can only arrive at the solution using an iterative technique, which can be made faster through usual binary trick. The procedure is illustrated in algorithm 3. This implies that the *for* loop in the subroutine repeats $\log L$ times, where L is the solution. Hence, we incur a logarithmic factor in the running time.

It follows from *Hoeffding's inequality*, where we define ϵ as $\epsilon'\mathbb{E}[\hat{Y}]$,

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

Algorithm 3 Find optimal L for A-KDE sketch

Require: Relative error of EH ϵ', failure probability δ, dataset $\mathcal{D}$, LSH family $\mathcal{H}$ of range W, parameters p
Ensure: Optimal number of rows $\hat{L}$
  Initialize $L \leftarrow l_0$ ▷ *e.g.,* $l_0 = 1$
  for $i = 1, 2, \ldots$ **do**
    $A \leftarrow \text{PREPROCESSING}(\mathcal{D}, \mathcal{H}, p = 1, L)$ ▷ *Use counters instead of* EH *in the sketch update step*
    if $L > \dfrac{2\max\{X_i\}^2}{(1-\epsilon')^2\mathbf{K}^2}\log\left(\dfrac{2}{\delta}\right)$ **then**
      break
    else
      $L \leftarrow 2L$ ▷ *Double the value of* L
    end if
  end for
  return L

To bound this probability by δ, we need :

$$
\begin{aligned}
& 2\exp\left(-\frac{L\mathbb{E}[Y_i]^2}{2\max\{X_i\}^2}\right) \leq \delta \\
\implies & L \geq \frac{2\max\{X_i\}^2}{\mathbb{E}[Y_i]^2}\log\left(\frac{2}{\delta}\right) \\
\implies & L \geq \frac{2\max\{X_i\}^2}{(1-\epsilon')^2\mathbf{K}^2}\log\left(\frac{2}{\delta}\right) \text{ (using lower bound of } \mathbb{E}[Y_i] \text{ from Lemma A.9)}
\end{aligned}
$$

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

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

Lemma A.11. *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 SW-AKDE algorithm is $\hat{Y}$. Then,

$$
\begin{aligned}
|\hat{Y} - \mathbf{K}| &\leq |\hat{Y} - \mathbb{E}\hat{Y}| + |\mathbb{E}\hat{Y} - \mathbf{K}| \quad \text{(using triangle inequality)} \\
&\leq \epsilon'\mathbb{E}\hat{Y} + \epsilon'\mathbf{K} \quad \text{(using lemma A.10 and penultimate inequality of equation 3)} \\
&\leq \epsilon'(1+\epsilon')\mathbf{K} + \epsilon'\mathbf{K} \quad \text{(using last inequality of equation 3)} \\
&\leq (2\epsilon' + \epsilon'^2)\mathbf{K} \\
&= \epsilon\mathbf{K} \quad \text{(substituting } \epsilon = 2\epsilon' + \epsilon'^2\text{)} \\
\implies |\hat{Y} - \mathbf{K}| &\leq \epsilon\mathbf{K}
\end{aligned}
$$

Note that the bound from Lemma A.10 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 A.12. *The proposed SW-AKDE has space complexity $\mathcal{O}\left(LW^p \cdot \frac{1}{\sqrt{1+\epsilon}-1}\log^2 N\right)$ where L is the number of rows, W is the range of the hash function, p is the number of hash functions concatenated, ϵ is the relative error for KDE, N is the window size.*

Proof. The number of cells in SW-AKDE sketch is LW^p. Each cell is represented by an EH of space complexity $\mathcal{O}(\frac{1}{\epsilon'}\log^2 N)$ where ϵ' is the relative error of the EH. The relative error for KDE ϵ

is related to ϵ' as (using lemma A.11): $\epsilon = 2\epsilon' + \epsilon'^2 \implies \epsilon' = \sqrt{1+\epsilon} - 1$.
Hence, the total space requirement for SW-AKDE is:

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

□

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

Theorem A.13. *Suppose we are given an* LSH *function with range* W*. Then the proposed* SW-AKDE *sketch with* $L = \mathcal{O}\left(\frac{2\max\{X_i\}^2}{(1-\epsilon')^2\boldsymbol{K}^2}\log\left(\frac{2}{\delta}\right)\right)$ *independent repetitions of the hash function provides a* $1+\epsilon$ *multiplicative approximation to* $\boldsymbol{K}$ *(which is the* KDE*) with probability* $1-\delta$*, using space* $\mathcal{O}\left(LW^p \cdot \frac{1}{\sqrt{1+\epsilon}-1}\log^2 N\right)$ *where N is the window size.*

A.3.2 Batch query setting

We define the batch 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, instead of N data points.

Our algorithm can be extended for this setting accordingly. We have to modify only the update step in the EH. The EH has to estimate the number of elements in the last N batches which hash to $h_i(\boldsymbol{q})$. The maximum increment for an EH in a cell at a given time step is R. This will happen when all the elements in the current batch of size R hash to the same LSH bucket. Datar et al. (2002) show that the EH algorithm can be generalized for this problem using at most $(k/2+1)(\log(\frac{2NR}{k}+1)+1)$ buckets, where $k = \lceil 1/\epsilon' \rceil$. The memory requirement for each bucket is $\log N + \log(\log N + \log R)$ bits. Hence, we get the following corollary.

Corollary A.14. *In the batch setting, the proposed* SW-AKDE *sketch with* $L = \mathcal{O}\left(\frac{2\max\{X_i\}^2}{(1-\epsilon)^2\boldsymbol{K}^2}\log\left(\frac{2}{\delta}\right)\right)$ *rows provides a* $1+\epsilon$ *multiplicative approximation to the* KDE *with probability* $1-\delta$*, using space* $\mathcal{O}(LW^p \cdot \frac{1}{\sqrt{1+\epsilon}-1}(\log(2NR\sqrt{1+\epsilon})(\log N + \log(\log N + \log R)))$.

A.4 Experiments and discussion for S-ANN

Detailed discussion. We demonstrate through the first experiment that the data in these popular benchmarks is clustered in a manner which can be modeled conveniently by the Poisson occupancy assumption. From the measurements in Table 2, we observe that the premise that a fixed-radius ball in high dimensions is essentially empty is usually not observed on standard benchmarks: at the above radius, a typical query has a large number of points within r across datasets. The exponential growth is governed by the number of directions the data actually varies in (around 15-20 here) rather than by the length of the vectors, which is 128–784. Image embeddings concentrate near much lower-dimensional structures, which is why neighborhoods stay populated. At this scale, this comfortably enables hyperparameter settings that attain sublinear space bounds. Additionally, The condition gets easier with scale. r is fixed in advance while points keep arriving, so occupancy grows in proportion to n, and the same measurements give a usable η of about 0.6 at a million points and 0.7 at 10^8. The assumption is therefore tightest at benchmark scale and comfortable at the stream sizes that actually motivate the problem, consistent with sublinear sketches attaining good accuracy from $\eta \geq 0.5$ (see Fig. 4). Uniformly random 128-dimensional points have nothing within r, this is a sparse regime. From the lower bound in Theorem 3.6, it is a regime where no one-pass algorithm succeeds, ours included. The occupancy condition is therefore not something we impose for convenience, it is the line between instances that are solvable in sublinear space and instances that provably are not.

Prior to our work, RACE-CMS Coleman et al. (2020) is the only known algorithm for streaming ANN. We show through these experiments that our method operates in a different regime - it recovers the full similarity vector $s(q)$ by compressed sensing, which requires $s(q)$ to be sparse. On the dense benchmarks we consider it separates from our sketch on both axes. As shown in Fig. 3c, RACE-CMS achieves only marginal improvements in recall across the full range of sketch sizes. In contrast, S-ANN consistently exceeds this apparent performance ceiling for every tested value of ϵ

and n, without requiring additional memory. It attains near-perfect approximate recall while retaining a compressed representation of the data. To verify that the RACE-CMS plateau is not caused by insufficient hyperparameter tuning, we additionally evaluate its recall over sufficiently broad ranges of the LSH power K and kernel bandwidth values. The difference is also structural at query time, and Table 1 shows it widening with n. RACE-CMS stores no vectors, so every query must score all n identities, $\hat{s}[i] = \min_j \widehat{\mathrm{CMS}}[j, h_j(i)]$, costing $O(n \cdot \mathrm{rows})$ regardless of sketch size. Its throughput therefore falls by $17\times$ over the range we test, while ours is nearly flat in n, because a query probes a bounded number of buckets rather than the whole dataset. Hence, we see that within this regime our sketch is more accurate without requiring too much more memory, and one to two orders of magnitude faster in terms of the query complexity.

Remark A.15. Both sketches need an assumption on the query neighbourhood, and the two are not comparable. Compressed-sensing recovery has error scaling with the total mass of $s(q)$, so RACE-CMS is accurate when few items are related to the query which is the case on its set and graph-similarity benchmarks. On image embeddings most items are loosely related, the mass stays large at any sketch size, and recall plateaus. We never recover $s(q)$: Poisson occupancy constrains the local density without requiring the neighbourhood to be sparse, and candidates come from LSH probing, so additional memory enlarges the candidate set and recall keeps improving.

The median difference plots [6] in Figure 5a and Figure 5b show that S-ANN 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 Figure 6. Figure 4 shows us that for a fixed 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.

Figure 7 compares recall and query throughput for the JL baseline (left column) and our S-ANN scheme (right column) across three datasets (rows). For a fixed workload, the figure shows that increasing JL's projection dimension k, steadily improves recall but offers little to no throughput benefit. Similarly for S-ANN, lowering the sampling rate η improves recall but doesn't convincingly benefit the query throughput, just as one would expect. However the throughput for our scheme is significantly superior to that for the baseline for all datasets over all choice of parameters for this fixed workload.

Table 2: Measured r-neighbourhood occupancy.

dataset	vector length	effective dim.	points within r	largest usable η
fashion-mnist	784	15.4	177	0.47
mnist	784	13.9	229	0.50
sift1m	128	22.7	805	0.48
uniform random	128	63.9	0	—

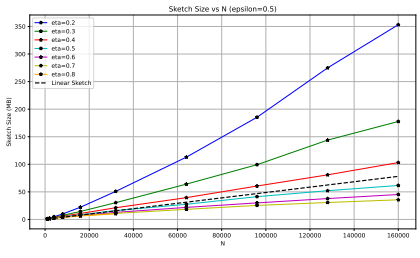
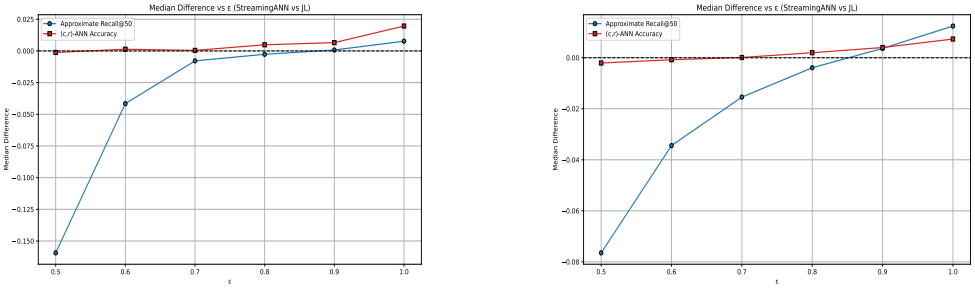


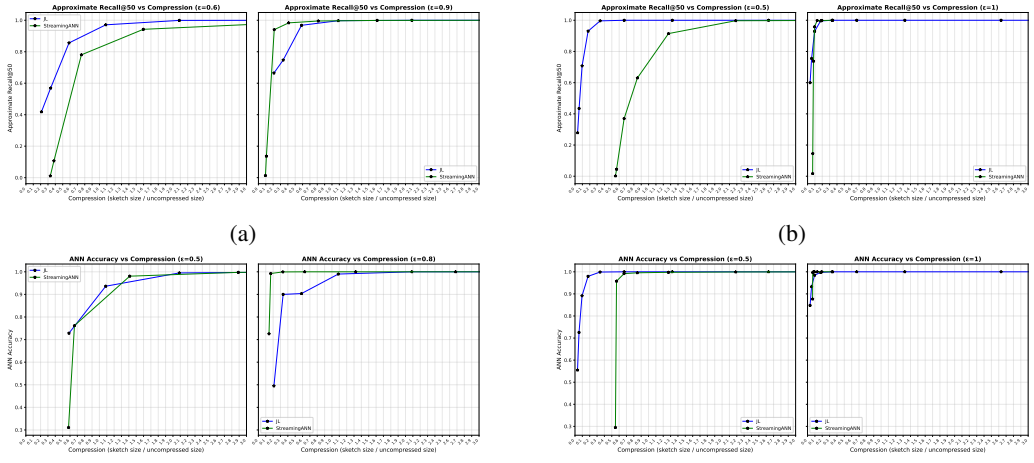
Figure 4: Memory requirements scale with stream size N for fixed $\epsilon = 0.5$ for the `sift1m` dataset.

[6] 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 outperforming the baseline.



(a) (b)

Figure 5: Median differences in Approximate Recall and (c, r)-ANN Accuracy over varying ϵ for `sift1m` (left) and `fashion-mnist` (right).



(a) (b)

(c) (d)

Figure 6: Effect of ϵ on retrieval performance: Left column shows results on `sift1m`, Right column shows results on `fashion-mnist`. (a) and (b) show Recall vs. Compression Rate for two values of ϵ, and (c) and (d) show (c, r)-ANN Accuracy vs. Compression Rate for the same settings.

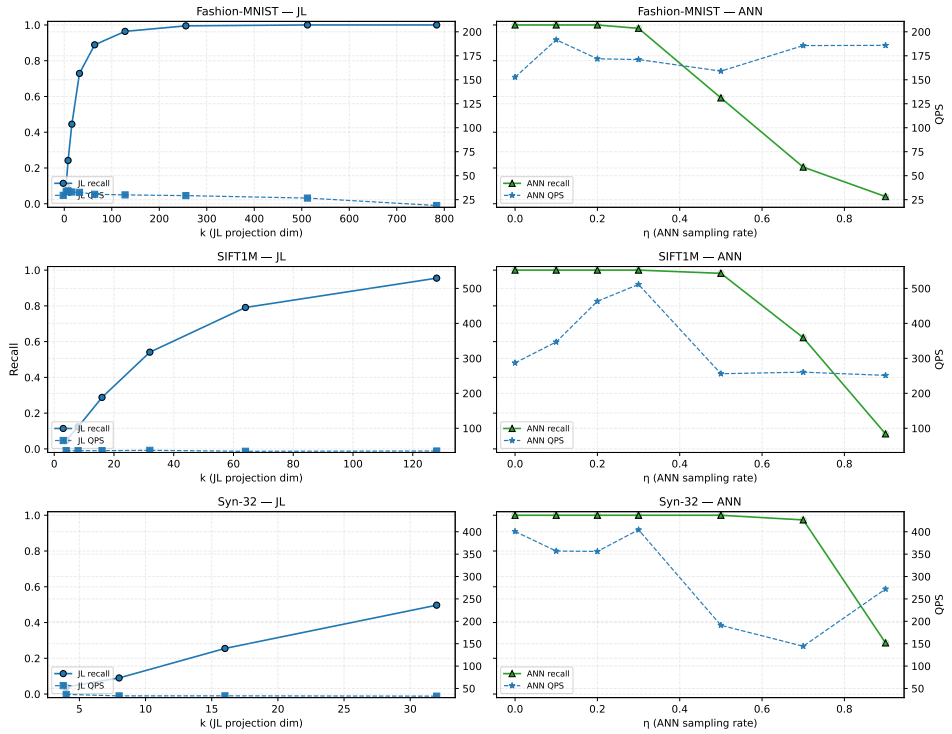


Figure 7: Query throughput (QPS) and Recall variation for both JL (left) and S-ANN (right) across datasets. Each row corresponds to a dataset (`fashion-mnist`, `sift1m`, and `syn-32`), showing Recall (solid) and Queries-per-Second (dashed, right axis) as parameters k and η vary.

A.5 Experiments for A-KDE

We investigate the impact of various parameters, including sketch size and sliding window size, on the accuracy of our algorithm for different datasets. We also compare SW-AKDE with RACEColeman & Shrivastava (2020).

Datasets. We conduct experiments on both synthetic and real-world datasets. For the synthetic dataset, we generate $10{,}000$ 200-dimensional vectors sampled from 10 different multivariate gaussian distributions. For every 1000 datapoints, we use a specific gaussian distribution. We generate this data set 50 times at random and conduct *Monte Carlo* simulations to study the performance of our algorithm. The real-world datasets used are: (a) News Headlines Kulkarni (2017) — It consists of news headlines published over eighteen years by a reputable news agency. We select approximately $80{,}000$ headlines and convert them into 384-dimensional text embeddings using the sentence transformer **all-MiniLM-L6-v2**[7]. (b) ROSIS Hyperspectral Data Sowmya et al. (2019) — It contains hyperspectral images captured by the *ROSIS* sensor. Each image pixel is treated as a high-dimensional data vector (of dimension 103 in this case) corresponding to a frequency spectrum. We use around $42{,}000$ data.

Implementation. SW-AKDE is implemented using two different LSH kernels: (a) Angular LSH, and (b) p-stable Euclidean LSH. To bound the range of the p-stable LSH functions, we employ rehashing. We have used Python to implement all the algorithms. The bandwidth parameter, denoted by p in Algorithm 2, was set to 1 for all experiments. All results reported here correspond to the single-query setting. For the SW-AKDE, the relative error of the EH was taken as 0.1. Hence, the upper bound on the theoretical relative error for the A-KDE estimate is 0.21 (follows from Lemma A.11). It is observed that the experimental errors are much less than this theoretical value, even for a small number of rows of the sketch.

[7]Kaggle link

Baseline. We have used the RACE algorithm proposed in Coleman & Shrivastava (2020) for comparison. This algorithm works for general streaming settings, whereas our algorithm works in a sliding window model.

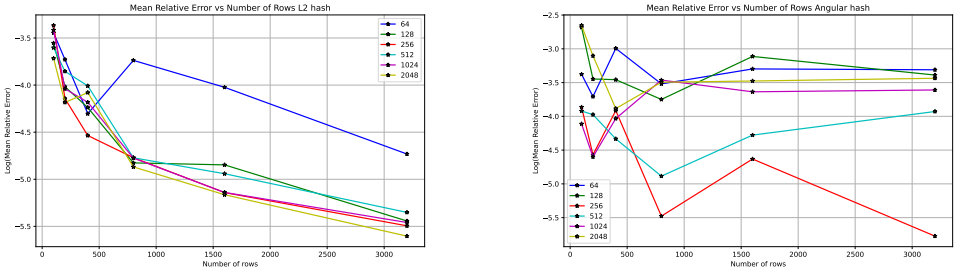
Experimental Setup. We have performed the following experiments to show the effect of various parameters on the performance of our algorithm.

1. **Sketch Size vs. Error:** We show the variation of the log of the mean relative error with sketch size(Fig. 8). We fixed the EH relative error as $\epsilon = 0.1$ and the window size as 450. We ran the experiment for 10,000 streaming data and measured the mean error on 1000 query data. We did separate experiments on both real-world and synthetic datasets using Euclidean hash and angular hash. We varied the row size exponentially as: $100, 200, 400, 800, 1600, 3200$.

2. **Window Size Effect:** We demonstrate the effect of varying the sliding window size on the mean relative error for the real-world datasets(Fig. 9). We have taken the window size as: $64, 128, 256, 512, 1024, 2048$. For each window size, we have plotted the log of the mean relative error against the number of rows. The row sizes were varied similarly to those of Experiment 1.

3. **Comparison with RACE:** We compared SW-AKDE algorithm using angular hash and a window size of 260 with RACE (Fig. 10) for both the real-world and synthetic datasets. We have plotted the log of the mean relative error against the row size, where row size has been taken similarly to Experiment 1.

Discussion: For SW-AKDE, the mean error reduces with an increase in sketch size for Euclidean kernels(Fig. 8a). In case of angular hash, the mean error decreases in general for text data but is erratic for image data(Fig. 8b). For synthetic data, both kernels show a regular decrease with increasing sketch size(Figs. 8d, 8c). It is observed that higher window sizes minimize error for text data(Fig. 9a) while for image data the mean relative error is minimized for window size=256(Fig. 9b). When compared to RACE, our algorithm gives similar performance(Fig. 10) for both real-world and synthetic datasets. These results validate our theoretical guarantees and demonstrate practical effectiveness across tasks.



(a) (b)

Figure 9: Effect of the window size on mean relative error for SW-AKDE with (a) Euclidean hash on news headlines data (b) angular hash on ROSIS image data.

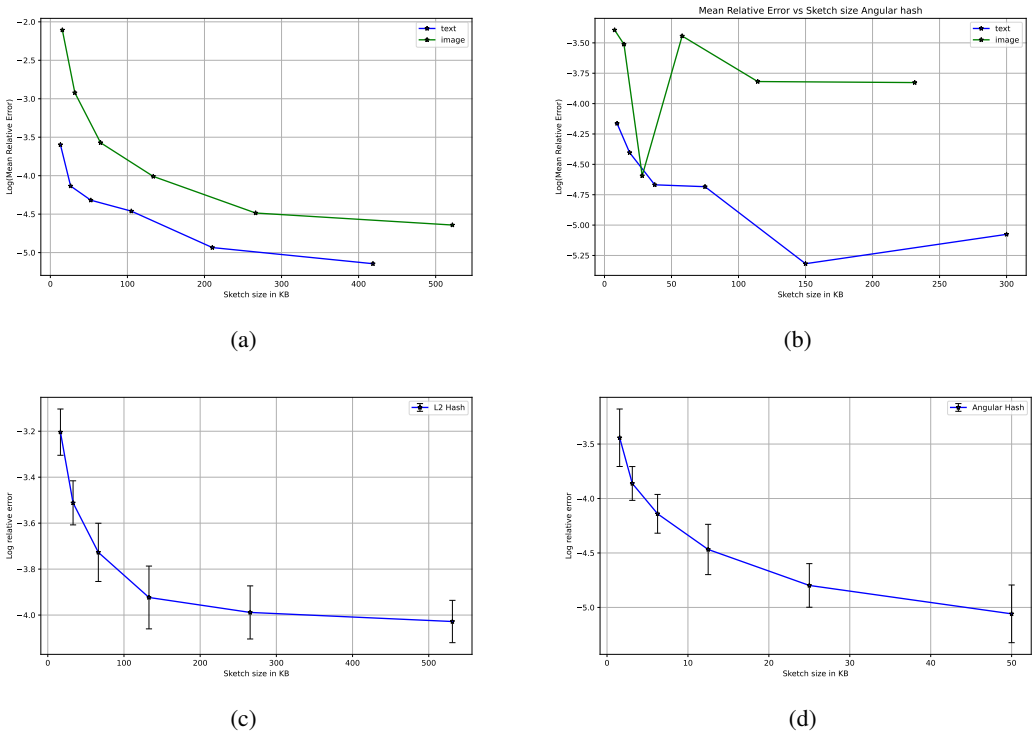


Figure 8: Effect of sketch size on KDE estimates (a) Real-world data with p-stable hash (b) Real-world data with Angular hash (c) Synthetic data with p-stable hash (d) Synthetic data with Angular hash

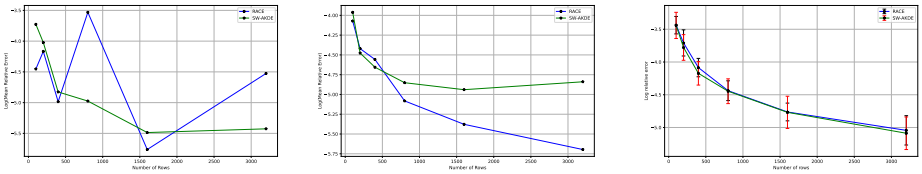


Figure 10: Comparison between RACE structure with Angular Hash and SW-AKDE with Angular Hash (a) `ROSIS Hyperspectral data` (b) `News headlines` (c) Synthetic data