Dynamic Maintenance of Kernel Density Estimation Data Structure: From Practice to Theory

Jiehao Liang[1] **Zhao Song**[1,*] **Zhaozhuo Xu**[2] **Junze Yin**[3,†] **Danyang Zhuo**[4]

[1]University of California, Berkeley, [2]Stevens Institute of Technology
[3]Boston University, [4]Duke University
[*]magic.linuxkde@gmail.com,[†]junze@bu.edu

Abstract

Kernel density estimation (KDE) stands out as a challenging task in machine learning. The problem is defined in the following way: given a kernel function $f(x, y)$ and a set of points $\{x_1, x_2, \cdots, x_n\} \subset \mathbb{R}^d$, we would like to compute $\frac{1}{n}\sum_{i=1}^n f(x_i, y)$ for any query point $y \in \mathbb{R}^d$. Recently, there has been a growing trend of using data structures for efficient KDE. However, the proposed KDE data structures focus on static settings. The robustness of KDE data structures over dynamic changing data distributions is not addressed. In this work, we focus on the dynamic maintenance of KDE data structures with robustness to adversarial queries. Especially, we provide a theoretical framework of KDE data structures. In our framework, the KDE data structures only require subquadratic spaces. Moreover, our data structure supports the dynamic update of the dataset in sublinear time. Furthermore, we can perform adaptive queries with the potential adversary in sublinear time.

1 INTRODUCTION

Kernel density estimation (KDE) is a well-known machine learning approach with wide applications in biology Fleming and Calabrese [2017], Cantrell et al. [2018], physics Hallin et al. [2022], Cranmer [2001] and law Cho et al. [2020]. The KDE is defined as follows: given a kernel function $f(x, y)$ and a dataset $\{x_1, x_2, \cdots, x_n\}$, we would like to estimate $\frac{1}{n}\sum_{i=1}^n f(x_i, y)$ for a query y. It is standard to assume the kernel f to be positive semi-definite. From a statistics perspective, we regard KDE as estimating the density of a probability distribution provided by a mapping. Recently, there has been a growing trend in applying probabilistic data structures for KDE Charikar and Siminelakis [2017], Backurs et al. [2019], Siminelakis et al. [2019], Alman et al. [2020], Charikar et al. [2020], Coleman and Shrivastava [2020], Karppa et al. [2022]. The general idea is to transform the kernel function into a distance measure and then apply similarity search data structures such as locality sensitive hashing and sketching. This co-design of data structure and KDE is of practical importance: (1) the computational efficiency is taken into consideration when we design KDE algorithms; (2) the capacity of traditional probabilistic data structures is extended from search to sampling. As a result, we obtain a series of KDE algorithms with both sample efficiency and running time efficiency. However, current KDE data structures focus on static settings where the dataset is fixed and the queries and independent of each other. More practical settings should be taken into consideration. In some applications of KDE, the dataset is dynamically changing. For instance, in time series modeling Moon et al. [1995], He and Li [2018], the KDE estimators should be adaptive to the insertion and deletion in the dataset. In semi-supervised learning Wang et al. [2009, 2019], KDE data structures should handle the update of the kernel function. Moreover, in the works that apply KDE in optimization, the data structures should be robust over adversarial queries. As a result, the dynamic maintenance of KDE data structures should be emphasized in the research of machine learning.

In this paper, we argue that there exists a practice-to-theory gap for the dynamic maintenance of KDE data structures. Although there are existing work Chan et al. [2021] that supports insertion and deletion in KDE data structures, these operations' impact on the quality of KDE is not well-addressed. Moreover, the robustness of KDE data structures over adversaries has recently been raised as a concern. Thus, a formal theoretical analysis is required to discuss the robustness of KDE data structures in a dynamic setting. We present a theoretical analysis of the efficient maintenance of KDE for dynamic datasets and adversarial queries. Specifically, we present the first data structure design that can quickly adapt to updated input data and is robust to adversarial queries.

Proceedings of the 41st Conference on Uncertainty in Artificial Intelligence (UAI 2025), PMLR 244:2552–2581.

We call our data structure and the corresponding algorithms *adaptive kernel density estimation.* Our data structure only requires subquadratic spaces, and each update to the input data only requires sublinear time, and each query can finish in sublinear time.

Notation We use $\mathbb{R}$, $\mathbb{R}_+$, $\mathbb{N}_+$ to denote the set of real numbers, positive real numbers, and positive integers. For a set X, we use $|X|$ to denote its cardinality. Let $n \in \mathbb{N}_+$ and $r \in \mathbb{R}$. We define $[n] := \{1, 2, 3, \ldots, n\}$ and $\lceil r \rceil$ to be the ceiling of r. Let $\mathbb{R}^n$ be the set of all n-dimensional vectors whose entries are all real numbers. $\|x\|_2$ represents the ℓ_2 norm of x. $\Pr[\cdot]$ represents the probability, and $\mathbb{E}[\cdot]$ represents the expectation. We define $\exp_2(r) := 2^r$.

1.1 RELATED WORK

Efficient Kernel Density Estimation The naive KDE procedure takes a linear scan of the data points. This is prohibitively expensive for large-scale datasets. Thus, it is of practical significance to develop efficient KDE algorithms. A series of traditional KDE algorithms, namely kernel merging Heinz and Seeger [2008], Cao et al. [2012], is to perform clustering on the dataset so that the KDE is approximated by a weighted combination of centroids. However, these algorithms do not scale to high-dimensional datasets. Also, there is a trend of sampling-based KDE algorithms. The goal is to develop efficient procedures that approximate KDE with fewer data samples. Starting from random sampling Muandet et al. [2017], sampling procedures such as Herding Chen et al. [2010] and k-centers Cortes and Scott [2016] are introduced in KDE. Some work also incorporates sampling with the coreset concept Phillips and Tai [2020] and provides a KDE algorithm by sampling on an optimized subset of data points. Recently, there has been a growing interest in applying hash-based estimators (HBE) Charikar and Siminelakis [2017], Backurs et al. [2019], Siminelakis et al. [2019], Coleman and Shrivastava [2020], Coleman et al. [2022], Spring and Shrivastava [2021] for KDE. The HBE uses Locality Sensitive Hashing (LSH) functions. The collision probability of two vectors in terms of an LSH function is monotonic to their distance measure. Using this feature, HBE performs efficient importance sampling by LSH functions and hash table type data structures. However, current HBEs are built for static settings and thus, are not robust to incremental changes in the input data. As a result, their application in large-scale online learning is limited. Except for LSH based KDE literature, there are also other KDE work based polynomial methods Alman et al. [2020], Alman and Song [2023, 2025a,b]. The dynamic type of KDE has also been considered in Deng et al. [2022], Brand et al. [2024]. Deng et al. [2023b] presents both randomized and deterministic algorithms for approximating a symmetric KDE computation.

Adaptive Data Structure Recently, there has been a growing trend of applying data structures Song [2019], Chen et al. [2020], Shrivastava et al. [2021], Xu et al. [2021b,a], Song et al. [2022b], Zhang [2022], Song et al. [2022a] to improve running time efficiency in machine learning. These adaptive data structures have also extended their success to many fields, such as optimization Lee et al. [2019], Cohen et al. [2019], Brand et al. [2020], Jiang et al. [2021], Qin et al. [2023] and differential privacy Hassidim et al. [2022], Cherapanamjeri et al. [2023], Song et al. [2023], Feng et al. [2025]. However, there exists a practice-to-theory gap between data structures and learning algorithms. Most data structures assume queries to be independent and provide theoretical guarantees based on this assumption. On the contrary, the query to data structures in each iteration of learning algorithms is mutually dependent. As a result, the existing analysis framework for data structures could not provide guarantees in optimization. To bridge this gap, quantization strategies Shrivastava et al. [2021], Xu et al. [2021b], Song et al. [2022b,a] are developed for adaptive queries in machine learning to quantize each query into its nearest vertex on the ϵ-net. Therefore, the failure probability of the data structures could be upper bounded by a standard ϵ-net argument. Although quantization methods demonstrate their success in machine learning, this direct combination does not fully enable the power of both data structure and learning algorithms. In our work, we aim at a co-design of data structure and machine learning for efficiency improvements in adaptive KDE.

1.2 PROBLEM FORMULATION

In this work, we would like to study the following problem.

Definition 1.1 (Dynamic KDE)**.** *Let $f : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ denote a kernel function. Let $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$ denote a dataset. Let $f^*_{\mathsf{KDE}} := f(X, q) := \frac{1}{|X|} \sum_{x \in X} f(x, q)$ define the kernel density estimate of a query $q \in \mathbb{R}^d$ with respect to X. Our goal is to design a data structure that efficiently supports any sequence of the following operations:*

- INITIALIZE($f : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1], X \subset \mathbb{R}^d, \epsilon \in (0, 1), f_{\mathsf{KDE}} \in [0, 1]$). *The data structure takes kernel function f, data set $X = \{x_1, x_2, \ldots, x_n\}$, accuracy parameter ϵ and a known quantity f_{KDE} satisfying $f_{\mathsf{KDE}} \geq f^*_{\mathsf{KDE}}$ as input for initialization.*
- UPDATE($z \in \mathbb{R}^d, i \in [n]$). *Replace the i'th data point of data set X with z.*
- QUERY($q \in \mathbb{R}^d$). *Output $\widetilde{d} \in \mathbb{R}$ such that $(1 - \epsilon) f^*_{\mathsf{KDE}}(X, q) \leq \widetilde{d} \leq (1 + \epsilon) f^*_{\mathsf{KDE}}(X, q)$.*

We note that in the QUERY procedure do not assume i.i.d queries. Instead, we take adaptive queries and provide theoretical guarantees.

1.3 OUR RESULT

In this work, we provide theoretical guarantees for the dynamic KDE data structures defined in Definition 1.1. We summarize our main result as below:

Theorem 1.2 (Main result)**.** *Given a function K and a set of points set $X \subset \mathbb{R}^d$. Let $\mathrm{cost}(f)$ be defined as Definition 2.7. For any accuracy parameter $\epsilon \in (0, 0.1)$, there is a data structure using space $O(\epsilon^{-2} n \cdot \mathrm{cost}(f))$ (Algorithm 5, 6 and 7) for the Dynamic KDE Problem (Definition 1.1) with the following procedures:*

- INITIALIZE($f : \mathbb{R}^d \times \mathbb{R}^d \to [0,1], X \subset \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0, 0.1]$). *Given a kernel function f, a dataset P, an accuracy parameter ϵ and a quantity f_{KDE} as input, the data structure* DYNAMICKDE *preprocess in time*

$$O(\epsilon^{-2} n^{1+o(1)} \mathrm{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n) \quad (1)$$

- UPDATE($z \in \mathbb{R}^d, i \in [n]$). *Given a new data point $z \in \mathbb{R}^d$ and index $i \in [n]$, the* UPDATE *operation take z and i as input and update the data structure in time*

$$O(\epsilon^{-2} n^{o(1)} \mathrm{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n) \quad (2)$$

- QUERY($q \in \mathbb{R}^d$). *Given a query point $q \in \mathbb{R}^d$, the* QUERY *operation takes q as input and approximately estimate kernel density at q in Eq.* (2) *time and output $\widetilde{d}$ such that: $(1-\epsilon) f(X, q) \leq \widetilde{d} \leq (1+\epsilon) f(X, q)$.*

We prove the main result in Lemma 3.4, Lemma 3.6 and Lemma 3.9.

1.4 TECHNICAL OVERVIEW

In this section, we introduce an overview of our technique that leads to our main result.

Density Constraint. We impose an upper bound on the true kernel density for query q. We also introduce geometric level sets, so the number of data points that fall into each level is upper bounded.

Importance Sampling. To approximate kernel density efficiently, we adopt the importance sampling technique. We sample each data point with different probability according to their contribution to the estimation, the higher the contribution, the higher the probability to be sampled. Then, we can construct an unbiased estimator based on sampled points and sampling probability. The main problem is how to evaluate the contribution to KDE for each point. We explore the geometry property of the kernel function and estimate the contribution of each point based on their distance from the query point.

Locality Sensitive Hashing. One problem with importance sampling is that when preprocessing, we have no access to the query point. It is impossible to estimate the contribution of each point for a future query. We make use of LSH to address this issue. To be specific, LSH preprocesses data points and finds the near neighbors for a query with high probability. With this property, we can first sample all data points in several rounds with geometrically decaying sampling probability. We design LSH for each round that satisfies the following property: given a query, LSH recovers points that have contributions proportional to the sampling probability in that round. Then we can find sampled points and proper sampling probability when a query comes.

Dynamic Update. Since the previous techniques, i.e. importance sampling and LSH, are independent of the coordinate value of the data point itself. This motivates us to support updating data points dynamically. Since LSH is a hash-based structure, given a new data point $z \in \mathbb{R}^d$ and index i indicating the data point to be replaced, we search for a bucket where x_i was hashed, replace it with new data point z and update the hash table. Such an update will not affect the data structure for estimating kernel density.

Robustness. To make our data structure robust to adaptive queries, we take the following steps to obtain a robust data structure. Starting from the constant success probability, we first repeat the estimation several times and take the median. This will provide a high probability of obtaining a correct answer for a fixed point. Then we push this success probability to the net points of a unit ball. Finally, we generalize to all query points in $\mathbb{R}^d$. Thus we have a data structure that is robust to adversary query.

Roadmap In Section 2, we describe some basic definitions and lemmas that are frequently used. In Section 3, we demonstrate our data structure in detail, including the algorithm and the running time analysis. We perform an analysis of our data structures over the adversary in Section 4. Finally, we draw our conclusion in Section 5.

2 PRELIMINARIES

The goal of this section is to introduce the basic definitions and lemmas that will be used to prove our main result. We first introduce a collection of subsets called geometric weight levels.

Definition 2.1 (Geometric Weight Levels)**.** *Fix $R \in \mathbb{N}_+$ and $q \in \mathbb{R}^d$. We define $w_i := f(x_i, q)$. For any fix $r \in [R] := \{1, 2, \cdots, R\}$, we define $L_r := \{x_i \in X \mid w_i \in (2^{-r+1}, 2^{-r}]\}$. We define the corresponding distance levels as $z_r := \max_{\mathrm{s.t.} f(z) \in (2^{-r}, 2^{-r+1}]} z$, where $f(z) := f(x, q)$*

for $z = \|x - q\|_2$. In addition, we define $L_{R+1} := X \setminus \bigcup_{r \in [R]} L_r$.

Geometric weight levels can be visualized as a sequence of circular rings centered at query q. The contribution of each level to kernel density at q is mainly determined by the corresponding distance level. Next, we introduce the important sampling technique to accelerate the query procedure.

Definition 2.2 (Importance Sampling)**.** *Let $x_1, \ldots, x_n \subset \mathbb{R}^d$ be a given set of data points. Suppose each point x_i is sampled independently with probability $p_i > 0$. The importance sampling estimator for a quantity of interest is given by:*

$$T := \sum_{i=1}^{n} \frac{\chi_i}{p_i} x_i,$$

where $\chi_i = 1$ is defined to be the event that point p_i gets sampled and recovered in the phase corresponding to its weight level, and $\chi_i = 0$ is defined to the contrary.

To apply importance sampling, we need to evaluate the contribution of each point. We sample each point that has a high contribution with a high probability. A natural question arose: when preprocessing, we have no access to the query, so we cannot calculate distance directly. Locality Sensitive Hashing is a practical tool to address this problem.

Definition 2.3 (Locally Sensitive Hash Indyk and Motwani [1998])**.** *A family $\mathcal{H}$ is called $(p_{\mathrm{near}}, p_{\mathrm{far}}, z, c)$-sensitive where $p_{\mathrm{near}}, p_{\mathrm{far}} \in [0, 1], z \in \mathbb{R}, c \geq 1$, if for any $x, q \in \mathbb{R}^d$, $\Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \leq z] \geq p_{\mathrm{near}}$, and $\Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \geq cz] \leq p_{\mathrm{far}}$.*

The next lemma shows the existence of the LSH family and its evaluation time.

Lemma 2.4 (Lemma 3.2 in page 6 of Andoni and Indyk [2006])**.** *Let $(x, q) \in \mathbb{R}^d \times \mathbb{R}^d$. Define*

$$p_{\mathrm{near}} := p_1(z) := \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \leq z]$$

and

$$p_{\mathrm{far}} := p_2(z, c) := \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \geq cz].$$

Then, if we fix z to be positive, we can have a hash family $\mathcal{H}$ satisfying

$$\rho := \frac{\log 1/p_{\mathrm{near}}}{\log 1/p_{\mathrm{far}}} \leq \frac{1}{c^2} + O(\frac{\log t}{t^{\frac{1}{2}}}),$$

for any $c \geq 1, t > 0$, where $p_{\mathrm{near}} \geq e^{-O(\sqrt{t})}$ and it requires $dt^{O(t)}$ time for every evaluation.

Remark 2.5. *We set $t = \log^{\frac{2}{3}} n$, which results in $n^{o(1)}$ evaluation time and $\rho = \frac{1}{c^2} + o(1)$. Note that if $c = O(\log^{\frac{1}{7}} n)$, then $\frac{1}{\frac{1}{c^2} + O(\log t / t^{\frac{1}{2}})} = c^2(1 - o(1))$ [1].*

Next, we assign the LSH family to each geometric weight level (Definition 2.1) and show how well these families can distinguish points from different levels.

Lemma 2.6 (Probability bound for separating points in different level sets, informal version of Lemma A.5)**.** *Given kernel function f and $r \in [R]$, let L_r be the weight level set and z_r be the corresponding distance level (Definition 2.1). For any query $q \in \mathbb{R}^d$, any integer pair $(i, r) \in [R+1] \times [R]$, satisfying $i > r$, let $x \in L_r$ and $x' \in L_i$. Let $c_{i,r} := \min\{\frac{z_{i-1}}{z_r}, \log^{1/7} n\}$. We set up Andoni-Indyk LSH family (Definition 2.3) $\mathcal{H}$ with near distance z_r and define*

$$p_{\mathrm{near},r} := \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \leq z]$$

and

$$p_{\mathrm{far},r} := \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \geq cz].$$

Then, for any $k \geq 1$, it is sufficient to show $\Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \geq p_{\mathrm{near},r}^k$ and $\Pr_{h^* \sim \mathcal{H}^k}[h^*(x') = h^*(q)] \leq p_{\mathrm{near},r}^{kc_{i,r}^2(1-o(1))}$

This lemma suggests that we can apply LSH several times to separate points in different level sets. It is useful for recovering points in a specific level set when estimating the "importance" of a point based on its distance from the query point. We will discuss more in Section 3. We use a similar definition for the cost of the kernel in Charikar et al. [2020].

Definition 2.7 (Kernel cost)**.** *Given a kernel f, which has geometric weight levels L_r's and distance levels z_r's defined in Definition 2.1. For any $r \in [R]$, we define the kernel cost f for L_r as*

$$\mathrm{cost}(f, r) := \exp_2(\max_{i \in \{r+1, \cdots, R+1\}} \lceil \frac{i - r}{c_{i,r}(1 - \mathrm{o}(1))} \rceil),$$

where

$$c_{i,r} := \min\{\frac{z_{i-1}}{z_r}, \log^{\frac{1}{7}} n\}.$$

Then we define the general cost of a kernel f as

$$\mathrm{cost}(f) := \max_{r \in [R]} \mathrm{cost}(f, r).$$

Note that when f is Gaussian kernel, the $\mathrm{cost}(f)$ is $(\frac{1}{f_{\mathsf{KDE}}})^{(1+o(1))\frac{1}{4}}$ Charikar et al. [2020].

[1] The above three $o(1)$ can be $\frac{\log \log^{\frac{2}{3}} n}{\log^{\frac{1}{3}} n}$, $\frac{\log \log^{\frac{2}{3}} n}{\log^{\frac{1}{3}} n}$, $\frac{\log \log^{\frac{2}{3}} n}{\log^{\frac{1}{21}} n}$ respectively.

3 OUR DATA STRUCTURES

Our algorithm's high-level idea is the following. We apply importance sampling (Definition 2.2) to approximate kernel density at a query point q efficiently. We want to sample data points with probability according to their contribution to the estimation. Ideally, given query point $q \in \mathbb{R}^d$ and data set $X \subset \mathbb{R}^d$, we can sample each data point in X with probability proportional to the inverse of the distance from query q. Unfortunately, we have no access to query points when preprocessing. Hence we make use of LSH (Definition 2.3) to overcome this problem. In general, given a query q, LSH can recover its near points with high probability while the probability of recovering far points is bounded by a small quantity. To apply LSH, we first run $R = \lceil \frac{1}{f_{\mathsf{KDE}}} \rceil$ rounds sampling, in which we sample each data point with probability $\frac{1}{2^r n f_{\mathsf{KDE}}}$ in r-th round. Then we obtain R subsampled data sets. Given query q, we use LSH to recover those points both in level set L_r and the r-th subsampled data sets. Hence we get the sampled data points and the corresponding sampling rates (in other words "importance"). Then we construct the estimator as in Definition 2.2. Finally, we repeat the estimation process independently and take the median to get $(1 \pm \epsilon)$ approximation with high probability.

3.1 LSH DATA STRUCTURE

In this section, we present our LSH data structure with the following procedures:

Initialize. Given a data set $\{x_1, \cdots, x_n\} \in \mathbb{R}^d$ and integral parameters k, L, it first invokes private procedure CHOOSEHASHFUNC. The idea behind this is to amplify the "sensitivity" of hashing by concatenating k basic hashing functions from the family $\mathcal{H}$ (Algorithm 8 line 9) into new functions. Thus we obtain a family of "augmented" hash function $\mathcal{H}_l, l \in [L]$ (Algorithm 1 line 7). We follow by CONSTRUCTHASHTABLE in which we hash each point x_i using the hashing function $\mathcal{H}_l$. Then we obtain L hash tables corresponding to L hash functions which can be updated quickly.

Recover. Given a query $q \in \mathbb{R}^d$, it finds the bucket where q is hashed by $\mathcal{H}_l$ and retrieves all the points in the bucket according to hashtable $\mathcal{T}_l$. This operation applies to all L hashtables.

UpdateHashTable. Given a new data point $z \in \mathbb{R}^d$ and index $i \in [n]$, it repeats following operations for all $l \in [L]$: find bucket $\mathcal{H}_l(z)$ and insert point z; find bucket $\mathcal{H}_l(x_i)$ and delete point x_i.

Note that traditional LSH data structure only has INITIALIZE and RECOVER procedures. To make it a dynamic structure, we exploit its hash storage. We design UPDATEHASHTABLE procedure so that we can update the hash table on the fly. This procedure provides guarantee for dynamic kernel density estimation.

3.2 INITIALIZE PART OF DATA STRUCTURE

Algorithm 1 LSH, private procedures

```
1: data structure LSH
2:
3: private
4: procedure CHOOSEHASHFUNC(k, L ∈ ℕ_+)
5:     for l ∈ [L] do
6:             ▷ Amplify hash functions by concatenating
7:         H_l ← sample k hash functions (f_{1,l}, f_{2,l}, ··· , f_{k,l}) from H
8:     end for
9: end procedure
10:
11: procedure CONSTRUCTHASHTABLE({x_i}_{i∈[n]} ⊂ ℝ^d)
12:     for l ∈ [L] do
13:         for i ∈ [n] do
14:             H_l(x_i).INSERT(x_i)
15:             T_l ← T_l ∪ H_l(x_i)        ▷ Creat hashtable by aggregating buckets
16:         end for
17:     end for
18: end procedure
19: end data structure
```

In this section, we present the initialize part of our data structure. We start by analyzing the space storage for LSH and DYNAMICKDE. Then we state the result of running time for LSH in Lemma 3.3 and DYNAMICKDE in Lemma 3.4. We first show the space storage of LSH part in our data structure.

Lemma 3.1 (Space storage of LSH)**.** *Let $X = \{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$ be a dataset of n points in d-dimensional space. Consider an* LSH*-based data structure that constructs L independent hash tables using k-wise concatenated hash functions from an* LSH *family $\mathcal{H}$.*

The space required to initialize and store the LSH *data structure, including hash functions and the constructed hashtables, is:*

$$O(Lkdn^{o(1)} + Ln).$$

Proof. The space storage comes from two parts: CHOOSEHASHFUNC and CONSTRUCTHASHTABLE in Algorithm 1.

Part 1. CHOOSEHASHFUNC (line 4) takes L, k as input. It has a for loop with L iterations. In each iteration, it samples k functions (line 7) from hash family $\mathcal{H}$ to create $\mathcal{H}_l$, which uses $O(kdn^{o(1)})$ space. Thus the total space usage of CHOOSEHASHFUNC is $L \cdot O(kdn^{o(1)}) = O(Lkdn^{o(1)})$.

Part 2. CONSTRUCTHASHTABLE (line 11) takes data set $\{x_i\}_{i\in[n]}$ and parameter L as input. It has two recursive for loops.

- The first for loop repeats L iterations.
- The second for loop repeats n iterations.

The space storage of the inner loop comes from line 28 and line 15, which is $O(1)$. Thus the total space storage of CONSTRUCTHASHTABLE is $L \cdot n \cdot O(1) = O(Ln)$.

The final space storage of INITIALIZE is

$$\mathbf{Part1} + \mathbf{Part2} = O(Lkdn^{o(1)} + Ln).$$

□

Thus, the LSH data structure maintains subquadratic storage complexity while enabling efficient approximate nearest-neighbor search.

Algorithm 2 Dynamic KDE, members and initialize part, informal version of Algorithm 5

1: **data structure** DYNAMICKDE ▷ Theorem 1.2
2: **members**
3: For $i \in [n]$,$x_i \in \mathbb{R}^d$ ▷ dataset X
4: $K_1, R, K_2 \in \mathbb{N}_+$ ▷ Number of repetitions
5: For $a \in [K_1]$, $r \in [R]$, $\mathcal{H}_{a,r} \in$ LSH ▷ Instances from LSH class
6: **end members**
7: **procedure** INITIALIZE($X \subset \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1]$) ▷ Lemma 3.2
8: Initialize K_1, R as in Section C
9: **for** $a = 1, 2, \cdots, K_1$ **do**
10: **for** $r = 1, 2, \cdots, R$ **do**
11: Compute $K_{2,r}, k_r$ as in Section C
12: $P_j \leftarrow$ sample each element in X with probability $\min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$.
13: $\mathcal{H}_{a,r}$.INITIALIZE($P_r, k_r, K_{2,r}$)
14: **end for**
15: $\widetilde{P}_a \leftarrow$ sample elements from X, each one has sample probability $\frac{1}{n}$ ▷ Store $\widetilde{P}_a$
16: **end for**
17: **end procedure**
18: **end data structure**

Then, we can prove the total space storage of DYNAMICKDE structure in the following lemma.

Lemma 3.2 (Space storage part of Theorem 1.2, informal version of Lemma C.2)**.** *The* INITIALIZE *of the data structure* DYNAMICKDE *(Algorithm 2) uses space*

$$O(\epsilon^{-2}(\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \cdot \log^2 n \cdot (\frac{1}{f_{\mathsf{KDE}}} + n^{o(1)} \cdot \log^2 n))$$

Proof sketch. The space usage of the INITIALIZE procedure of DYNAMICKDE mainly comes from $K_1 \cdot R$ copies of the LSH data structure $\mathcal{H}$. By Lemma 3.1, the space usage of each $\mathcal{H}$ is:

$$\begin{aligned} & O(Lkdn^{o(1)} + Ln) \\ = & O(\mathrm{cost}(f) n^{o(1)} \cdot \log^3 n + \mathrm{cost}(f)(1/f_{\mathsf{KDE}}) \cdot \log n). \end{aligned}$$

Since there are $K_1 \cdot R$ copies of $\mathcal{H}$, the total space usage of INITIALIZE is:

$$\begin{aligned} & K_1 \cdot R \cdot O(\mathrm{cost}(K) \log n \cdot (1/f_{\mathsf{KDE}} + \log n)) \\ = & O(\epsilon^{-2}(1/f_{\mathsf{KDE}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \\ & \cdot \log^2 n \cdot (1/f_{\mathsf{KDE}} + n^{o(1)} \cdot \log^2 n)) \end{aligned}$$

which is by $K_1 = O(\epsilon^{-2}(1/f_{\mathsf{KDE}})^{o(1)} \cdot \log n)$ and $R = O(\log(1/f_{\mathsf{KDE}}))$. □

For the running time, we again start with the LSH part.

Lemma 3.3 (Upper bound on running time of INITIALIZE of the data-structure LSH, informal version of Lemma C.3)**.** *Given input data points $\{x_i\}_{i\in[n]} \subset \mathbb{R}^d$, parameters $k, L \in \mathbb{N}_+$, LSH parameters $p_{\mathrm{near}}, p_{\mathrm{far}} \in [0,1], c \in [1,\infty), r \in \mathbb{R}_+$ and kernel K, the* INITIALIZE *of the data-structure* LSH *(Algorithm 8) runs in time $O(L \cdot (kdn^{o(1)} + dn^{1+o(1)} + n \log n))$.*

Having shown the running time of LSH, we now move on to prove the total running time of INIT in our data structure by combining the above result in the LSH part.

Lemma 3.4 (The initialize part of Theorem 1.2, informal version of Lemma C.4)**.** *Given $(K : \mathbb{R}^d \times \mathbb{R}^d \to [0,1], X \subset \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1])$, the* INITIALIZE *of the data-structure* DYNAMICKDE *(Algorithm 2) runs in Eq.* (1) *time.*

Proof sketch. The INITIALIZE procedure of DYNAMICKDE has two nested for loops. The outer loop repeats $K_1 = O(\epsilon^{-2} \log(n) \cdot f_{\mathsf{KDE}}^{-o(1)})$ times and the inner loop repeats $R = O(\log 1/f_{\mathsf{KDE}})$ times. The running time of each iteration of the inner loop consists of $O(\log(1/f_{\mathsf{KDE}}))$ time for lines 20-line 25, $O(n)$ time for line 26, and $O(n^{1+o(1)} \mathrm{cost}(K) \cdot \log^2 n)$ time for line 27, by Lemma 3.3. Thus, the total time for each iteration of the inner loop is $O(n^{1+o(1)} \mathrm{cost}(K) \cdot \log^2 n)$. Since the inner loop repeats R times and the outer loop repeats K_1 times, the total running time is: $K_1 \cdot R \cdot O(n^{1+o(1)} \mathrm{cost}(K) \cdot \log^2 n) = O(\epsilon^{-2} n^{1+o(1)} \mathrm{cost}(f) \cdot (1/f_{\mathsf{KDE}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n)$ (Eq. (1)) where the last step uses $K_1 = O(\epsilon^{-2} \cdot f_{\mathsf{KDE}}^{-o(1)} \cdot \log n)$ and $R = O(\log(1/f_{\mathsf{KDE}}))$. □

Algorithm 3 Dynamic KDE, update part, informal version of Algorithm 6

1: **data structure** DYNAMICKDE ▷ Theorem 1.2
2:
3: **procedure** UPDATE($v \in \mathbb{R}^d, f_{\mathsf{KDE}} \in [0,1], i \in [n]$) ▷ Lemma 3.6
4: **for** $a = 1, 2, \cdots, K_1$ **do**
5: **for** $r = 1, 2, \cdots, R$ **do**
6: Update the hashtables in $\mathcal{H}_{a,r}$ with v
7: **end for**
8: **end for**
9: Replace x_i with v
10: **end procedure**
11:
12: **end data structure**

3.3 UPDATE PART OF DATA STRUCTURE

We move to the update part of our data structure. We show how to update the LSH data structure. Then we can extend the update procedure to DYNAMICKDE structure to prove Lemma 3.6.

Lemma 3.5 (Update time of LSH, informal version of Lemma C.5)**.** *Given a data point $v \in \mathbb{R}^d$ and index $i \in [n]$, the* UPDATEHASHTABLE *of the data-structure* LSH *(Algorithm 8) runs in (expected) time $O(n^{o(1)} \log(n) \cdot \mathrm{cost}(f))$.*

UPDATE in LSH structure is a key part in UPDATE for DYNAMICKDE. Next, we show the running time of UPDATE for DYNAMICKDE by combining the above results.

Lemma 3.6 (The update part of Theorem 1.2, informal version of Lemma C.6)**.** *Given an update $v \in \mathbb{R}^d$, $i \in [n]$, the* UPDATE *of the data-structure* DYNAMICKDE *(Algorithm 3) runs Eq.* (2) *time.*

Proof sketch. The UPDATE procedure of DYNAMICKDE has two nested for loops. The outer loop repeats $K_1 = O(\epsilon^{-2} \log(n) \cdot f_{\mathsf{KDE}}^{-o(1)})$ times and the inner loop repeats $R = O(\log 1/f_{\mathsf{KDE}})$ times. The running time of each iteration of the inner loop is dominated by line 6, which takes $O(n^{o(1)} \log(n) \cdot \mathrm{cost}(f))$ time by Lemma 3.5. Since the inner loop repeats R times and the outer loop repeats K_1 times, the total running time is: $K_1 \cdot R \cdot O(n^{o(1)} \mathrm{cost}(f) \cdot \log^2 n) = O(\epsilon^{-2} n^{o(1)} \mathrm{cost}(f) \cdot (1/f_{\mathsf{KDE}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n)$. □

3.4 QUERY PART OF DATA STRUCTURE

Finally, we come to the query part. The goal of this section is to prove Lemma 3.9, which shows the running time of QUERY procedure for DYNAMICKDE.

Algorithm 4 Dynamic KDE, query part, informal version of Algorithm 7

1: **data structure** DYNAMICKDE ▷ Theorem 1.2
2:
3: **procedure** QUERY($q \in \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1]$)
4: **for** $a = 1, 2, \cdots, K_1$ **do**
5: **for** $r = 1, 2, \cdots, R$ **do**
6: Recover near neighbours of q using $\mathcal{H}_{a,r}$
7: Store them into $\mathcal{S}$
8: **end for**
9: **for** $x_i \in \mathcal{S}$ **do**
10: $w_i \leftarrow f(x_i, q)$
11: **if** $x_i \in L_r$ for some $r \in [R]$ **then**
12: $p_i \leftarrow \min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$
13: **end if**
14: **end for**
15: $T_a \leftarrow \sum_{x_i \in \mathcal{S}} \frac{w_i}{p_i}$
16: **end for**
17: **return** $\mathrm{Median}_{a \in K_1}\{T_a\}$
18: **end procedure**
19:
20: **end data structure**

The running time of QUERY procedure depends on two parts: the number of recovered points in each weight level and the recovery time of LSH. We first show the expected number of recovered points.

Lemma 3.7 (Expected number of points in level sets, informal version of Lemma C.7)**.** *Given a query $q \in \mathbb{R}^d$ and fix $r \in [R]$. For any $i \in [R+1]$, weight level L_i contributes at most* 1 *point to the hash bucket of query q.*

Next, we show the running time for LSH to recover points.

Lemma 3.8 (Running time for recovering points given a query, informal version of Lemma C.8)**.** *Given a query $q \in \mathbb{R}^d$ and $L, R, k \in \mathbb{N}_+$, the* RECOVER *of the data-structure* LSH *runs in (expected) time $O(Lkn^{o(1)} + LR)$.*

Combining two lemmas above, we prove the total running time of QUERY in DYNAMICKDE structure.

Lemma 3.9 (Query part of Theorem 1.2, informal version of Lemma C.9)**.** *Given a query $q \in \mathbb{R}^d$, the* QUERY *of the data-structure* DYNAMICKDE *(Algorithm 7) runs in Eq.* (2) *time.*

4 ROBUSTNESS TO ADVERSARY

In this section, we will turn the QUERY algorithm into a robust one. In other words, we want the following thing to happen with high probability: the algorithm responds to all query points correctly. We achieve this goal by taking three steps. We start with constant success probability for the

QUERY procedure, which we have proved in the previous section. In the first step, we boost this constant probability to a high probability by applying the median technique. We note that the current algorithm succeeds with high probability only for one fixed point but we want it to respond to arbitrary query points correctly.

It is not an easy task to generalize directly from a fixed point to infinite points in the whole space. Thus we take a middle step by introducing unit ball and ϵ-net. We say a unit ball in $\mathbb{R}^d$ is a collection of points whose norm is less than or equal to 1. An ϵ-net is a finite collection of points, called net points, that has the "covering" property. To be more specific, the union of balls that centered at net points with radius ϵ covers the unit ball. In the second step, we show that given a net of the unit ball, we have the correctness on all net points. Finally, we show the correctness of the algorithm from net points to all points in the whole space. Then we obtain a robust algorithm.

Starting Point In Section D, we have already obtained a query algorithm with constant success probability for a fixed query point.

Lemma 4.1 (Starting with constant probability)**.** *Given $\epsilon \in (0, 0.1)$, a query point $q \in \mathbb{R}^d$ and a set of data points $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$, let*

$$f^*_{\mathsf{KDE}}(q) := \frac{1}{|X|} \sum_{x \in X} f(x, q)$$

be an estimator $\mathcal{D}$ can answer the query satisfing $(1 - \epsilon) \cdot f^*_{\mathsf{KDE}}(q) \leq \mathcal{D}.\textsc{query}(q, \epsilon) \leq (1 + \epsilon) \cdot f^*_{\mathsf{KDE}}(q)$ with probability 0.9.

Boost the constant probability to high probability. Next, we begin to boost the success probability by repeating the query procedure and taking the median output.

Lemma 4.2 (Boost the constant probability to high probability)**.** *Let $\delta_1 \in (0, 0.1)$ denote the failure probability. Let $\epsilon \in (0, 0.1)$ denote the accuracy parameter. Given $L = O(\log(1/\delta_1))$ estimators $\{\mathcal{D}_j\}_{j=1}^L$. For each fixed query point $q \in \mathbb{R}^d$, the median of queries from L estimators satisfies that:*

$$\begin{aligned}(1 - \epsilon) \cdot f^*_{\mathsf{KDE}}(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f^*_{\mathsf{KDE}}(q)\end{aligned}$$

with probability $1 - \delta_1$.

From each fixed point to all the net points. So far, the success probability of our algorithm is still for a fixed point. We will introduce ϵ-net on a unit ball and show the high success probability for all the net points.

Fact 4.3. *Let N denote the ϵ_0-net of*

$$\{x \in \mathbb{R}^d \mid \|x\|_2 \leq 1\}.$$

We use $|N|$ to denote the number of points in N. Then $|N| \leq (10/\epsilon_0)^d$.

This fact shows that we can bound the size of an ϵ-net with an inverse of ϵ. We use this fact to conclude the number of repetitions we need to obtain the correctness of QUERY on all net points.

Lemma 4.4 (From each fixed points to all the net points)**.** *Let N denote the ϵ_0-net of $\{x \in \mathbb{R}^d \mid \|x\|_2 \leq 1\}$. We use $|N|$ to denote the number of points in N. Given $L = \log(|N|/\delta)$ estimators $\{\mathcal{D}_j\}_{j=1}^L$. With probability $1 - \delta$, we have: for all $q \in N$, the median of queries from L estimators satisfies that:*

$$\begin{aligned}(1 - \epsilon) \cdot f^*_{\mathsf{KDE}}(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f^*_{\mathsf{KDE}}(q).\end{aligned}$$

From net points to all points. With Lemma 4.4, we are ready to extend the correctness for net points to the whole unit ball. We demonstrate that all query points $\|q\|_2 \leq 1$ can be answered approximately with high probability in the following lemma.

Lemma 4.5 (From net points to all points)**.** *Let $\epsilon \in (0, 0.1)$. Let $\mathcal{L} \geq 1$. Let $\delta \in (0, 0.1)$. Let $\tau \in [0, 1]$. Given $L = O(\log((\mathcal{L}/\epsilon\tau)^d/\delta))$ estimators $\{\mathcal{D}_j\}_{j=1}^L$, with probability $1 - \delta$, for all query points $\|p\|_2 \leq 1$, we have the median of queries from L estimators satisfies that: $\forall \|p\|_2 \leq 1$*

$$\begin{aligned}(1 - \epsilon) \cdot f^*_{\mathsf{KDE}}(p) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f^*_{\mathsf{KDE}}(p).\end{aligned}$$

where q is the closest net point of p.

Thus, we obtain an algorithm that could respond to adversary queries robustly.

5 CONCLUSION

Kernel density estimation is an important problem in machine learning. It has wide applications in similarity search and nearest neighbor clustering. Meanwhile, in many modern scenarios, input data can change over time, and queries can be provided by adversaries. In these scenarios, we need to build adaptive data structures such that incremental changes in the input data do not require our data structure to go through costly re-initialization. Also, queries provided by adversaries do not reduce the accuracy of the estimation. We call this problem the adaptive kernel density estimation. We present the first such adaptive data structure design for kernel density estimation. Our data structure is efficient. It only requires subquadratic spaces. Each update to the input data only requires sublinear time, and each query can finish in sublinear time. It should be observed that the trade-off between efficiency and effectiveness persists in our proposed algorithms. Ordinarily, to augment the execution speed, a slight compromise on the algorithm's performance becomes inevitable. Yet, we assert that the introduction of our groundbreaking data structures pushes the boundaries of this trade-off.

Acknowledgements

The authors would like to thank the anonymous reviewer of UAI 2025 for their highly insightful suggestions.

References

Josh Alman and Zhao Song. Fast attention requires bounded entries. In *NeurIPS*, 2023.

Josh Alman and Zhao Song. The fine-grained complexity of gradient computation for training large language models. In *NeurIPS*, 2024a.

Josh Alman and Zhao Song. How to capture higher-order correlations? generalizing matrix softmax attention to kronecker computation. In *ICLR*, 2024b.

Josh Alman and Zhao Song. Fast rope attention: Combining the polynomial method and fast fourier transform. *arXiv preprint arXiv:2505.11892*, 2025a.

Josh Alman and Zhao Song. Only large weights (and not skip connections) can prevent the perils of rank collapse. *arXiv preprint arXiv:2505.16284*, 2025b.

Josh Alman, Timothy Chu, Aaron Schild, and Zhao Song. Algorithms and hardness for linear algebra on geometric graphs. In *FOCS*, 2020.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, 2006.

Arturs Backurs, Piotr Indyk, and Tal Wagner. Space and time efficient kernel density estimation in high dimensions. In *NeurIPS*, 2019.

Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *STOC*, 2020.

Jan van den Brand, Zhao Song, and Tianyi Zhou. Algorithm and hardness for dynamic attention maintenance in large language models. In *ICML*, 2024.

Danielle L Cantrell, Erin E Rees, Raphael Vanderstichel, Jon Grant, Ramón Filgueira, and Crawford W Revie. The use of kernel density estimation with a bio-physical model provides a method to quantify connectivity among salmon farms: spatial planning and management with epidemiological relevance. *Frontiers in Veterinary Science*, 2018.

Yang Cao, Bo Chen, Xiaoyu Li, Yingyu Liang, Zhizhou Sha, Zhenmei Shi, Zhao Song, and Mingda Wan. Force matching with relativistic constraints: A physics-inspired approach to stable and efficient generative modeling. *arXiv preprint arXiv:2502.08150*, 2025a.

Yang Cao, Zhao Song, and Chiwun Yang. Video latent flow matching: Optimal polynomial projections for video interpolation and extrapolation. *arXiv preprint arXiv:2502.00500*, 2025b.

Yuan Cao, Haibo He, and Hong Man. Somke: Kernel density estimation over data streams by sequences of self-organizing maps. *IEEE transactions on neural networks and learning systems*, 2012.

Yuefan Cao, Xuyang Guo, Jiayan Huo, Yingyu Liang, Zhenmei Shi, Zhao Song, Jiahao Zhang, and Zhen Zhuang. Text-to-image diffusion models cannot count, and prompt refinement cannot help. *arXiv preprint arXiv:2503.06884*, 2025c.

Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Byron Choi, and Jianliang Xu. Sws: a complexity-optimized solution for spatial-temporal kernel density visualization. In *VLDB*, 2021.

Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *FOCS*, 2017.

Moses Charikar, Michael Kapralov, Navid Nouri, and Paris Siminelakis. Kernel density estimation through density constrained near neighbor search. In *FOCS*, 2020.

Beidi Chen, Tharun Medini, James Farwell, Charlie Tai, Anshumali Shrivastava, et al. Slide: In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In *MLSys*, 2020.

Bo Chen, Chengyue Gong, Xiaoyu Li, Yingyu Liang, Zhizhou Sha, Zhenmei Shi, Zhao Song, Mingda Wan, and Xugang Ye. Nrflow: Towards noise-robust generative modeling via high-order flow matching. In *UAI*, 2025a.

Bo Chen, Zhenmei Shi, Zhao Song, and Jiahao Zhang. Provable failure of language models in learning majority boolean logic via gradient descent. *arXiv preprint arXiv:2504.04702*, 2025b.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. In *UAI*, 2010.

Yeshwanth Cherapanamjeri, Sandeep Silwal, David P Woodruff, Fred Zhang, Qiuyi Zhang, and Samson Zhou. Robust algorithms on adaptive inputs from bounded adversaries. In *ICLR*, 2023.

Jaewoong Cho, Gyeongjo Hwang, and Changho Suh. A fair classifier using kernel density estimation. In *NeurIPS*, 2020.

Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

Benjamin Coleman and Anshumali Shrivastava. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *WWW*, 2020.

Benjamin Coleman, Benito Geordie, Li Chou, RA Leo Elworth, Todd Treangen, and Anshumali Shrivastava. One-pass diversified sampling with application to terabyte-scale genomic sequence streams. In *ICML*, 2022.

Efren Cruz Cortes and Clayton Scott. Sparse approximation of a kernel mean. *IEEE Transactions on Signal Processing*, 2016.

Kyle Cranmer. Kernel estimation in high-energy physics. *Computer Physics Communications*, 2001.

Yichuan Deng, Wenyu Jin, Zhao Song, Xiaorui Sun, and Omri Weinstein. Dynamic kernel sparsifiers. *arXiv preprint arXiv:2211.14825*, 2022.

Yichuan Deng, Zhihang Li, and Zhao Song. Attention scheme inspired softmax regression. *arXiv preprint arXiv:2304.10411*, 2023a.

Yichuan Deng, Sridhar Mahadevan, and Zhao Song. Randomized and deterministic attention sparsification algorithms for over-parameterized feature dimension. *arXiv preprint arXiv:2304.04397*, 2023b.

Yichuan Deng, Zhihang Li, Sridhar Mahadevan, and Zhao Song. Zero-th order algorithm for softmax attention optimization. In *IEEE BigData*, 2024.

Shiyuan Feng, Ying Feng, George Z. Li, Zhao Song, David P. Woodruff, and Lichen Zhang. On differential privacy for adaptively solving search problems via sketching. In *ICML*, 2025.

Christen H Fleming and Justin M Calabrese. A new kernel density estimator for accurate home-range and species-range area estimation. *Methods in Ecology and Evolution*, 2017.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *ICML*, 2023.

Ziwang Fu, Feng Liu, Jiahao Zhang, Hanyang Wang, Chengyi Yang, Qing Xu, Jiayin Qi, Xiangling Fu, and Aimin Zhou. Sagn: semantic adaptive graph network for skeleton-based human action recognition. In *ICMR*, 2021.

Yeqi Gao, Zhao Song, and Shenghao Xie. In-context learning for attention scheme: from single softmax regression to multiple softmax regression via a tensor trick. *arXiv preprint arXiv:2307.02419*, 2023.

Yeqi Gao, Zhao Song, Weixin Wang, and Junze Yin. A fast optimization view: Reformulating single layer attention in llm based on tensor and svm trick, and solving it in matrix multiplication time. In *UAI*, 2025.

Chengyue Gong, Yekun Ke, Xiaoyu Li, Yingyu Liang, Zhizhou Sha, Zhenmei Shi, and Zhao Song. On computational limits of flowar models: Expressivity and efficiency. *arXiv preprint arXiv:2502.16490*, 2025.

Xuyang Guo, Jiayan Huo, Zhenmei Shi, Zhao Song, Jiahao Zhang, and Jiale Zhao. T2vphysbench: A first-principles benchmark for physical consistency in text-to-video generation. *arXiv preprint arXiv:2505.00337*, 2025.

Anna Hallin, Joshua Isaacson, Gregor Kasieczka, Claudius Krause, Benjamin Nachman, Tobias Quadfasel, Matthias Schlaffer, David Shih, and Manuel Sommerhalder. Classifying anomalies through outer density estimation. *Physical Review D*, 2022.

Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. *Journal of the ACM*, 2022.

Yaoyao He and Haiyan Li. Probability density forecasting of wind power using quantile regression neural network and kernel density estimation. *Energy conversion and management*, 2018.

Christoph Heinz and Bernhard Seeger. Cluster kernels: Resource-aware kernel density estimators over streaming data. *IEEE Transactions on Knowledge and Data Engineering*, 2008.

Jerry Yao-Chieh Hu, Weimin Wu, Zhuoru Li, Sophia Pi, Zhao Song, and Han Liu. On statistical rates and provably efficient criteria of latent diffusion transformers (dits). *NeurIPS*, 2024.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *STOC*, 2021.

Matti Karppa, Martin Aumüller, and Rasmus Pagh. (deann): Speeding up kernel-density estimation using approximate nearest neighbor search. In *AISTATS*, 2022.

Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, 2019.

Xiaoyu Li, Yingyu Liang, Zhenmei Shi, Zhao Song, Wei Wang, and Jiahao Zhang. On the computational capability of graph neural networks: A circuit complexity bound perspective. *arXiv preprint arXiv:2501.06444*, 2025.

Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, and Yufa Zhou. Beyond linear approximations: A novel pruning approach for attention matrix. In *ICLR*, 2025.

Young-Il Moon, Balaji Rajagopalan, and Upmanu Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 1995.

Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, Bernhard Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 2017.

Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry*, 2020.

Lianke Qin, Zhao Song, Lichen Zhang, and Danyang Zhuo. An online and unified algorithm for projection matrix vector multiplication with application to empirical risk minimization. In *AISTATS*, 2023.

Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Yanyu Li, Yifan Gong, Kai Zhang, Hao Tan, Jason Kuen, Henghui Ding, et al. Lazydit: Lazy learning for the acceleration of diffusion transformers. In *AAAI*, 2025.

Anshumali Shrivastava, Zhao Song, and Zhaozhuo Xu. Sublinear least-squares value iteration via locality sensitive hashing. *arXiv preprint arXiv:2105.08285*, 2021.

Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Levis. Rehashing kernel evaluation in high dimensions. In *ICML*, 2019.

Zhao Song. *Matrix theory: optimization, concentration, and algorithms*. PhD thesis, The University of Texas at Austin, 2019.

Zhao Song, Zhaozhuo Xu, Yuanyuan Yang, and Lichen Zhang. Accelerating frank-wolfe algorithm using low-dimensional and adaptive data structures. *arXiv preprint arXiv:2207.09002*, 2022a.

Zhao Song, Zhaozhuo Xu, and Lichen Zhang. Speeding up sparsification using inner product search data structures. *arXiv preprint arXiv:2204.03209*, 2022b.

Zhao Song, Xin Yang, Yuanyuan Yang, and Lichen Zhang. Sketching meets differential privacy: fast algorithm for dynamic kronecker projection maintenance. In *ICML*, 2023.

Zhao Song, Junze Yin, and Lichen Zhang. Solving attention kernel regression problem via pre-conditioner. In *AISTATS*, 2024.

Ryan Spring and Anshumali Shrivastava. Mutual information estimation using LSH sampling. In *IJCAI*, 2021.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *EMNLP*, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Meng Wang, Xian-Sheng Hua, Tao Mei, Richang Hong, Guojun Qi, Yan Song, and Li-Rong Dai. Semi-supervised kernel density estimation for video annotation. *Computer Vision and Image Understanding*, 2009.

Qin Wang, Wen Li, and Luc Van Gool. Semi-supervised learning by augmented distribution alignment. In *ICCV*, 2019.

Binghui Xie, Yongqiang Chen, Jiaqi Wang, Kaiwen Zhou, Bo Han, Wei Meng, and James Cheng. Enhancing evolving domain generalization through dynamic latent representations. In *AAAI*, 2024.

Zhaozhuo Xu, Beidi Chen, Chaojian Li, Weiyang Liu, Le Song, Yingyan Lin, and Anshumali Shrivastava. Locality sensitive teaching. In *NeurIPS*, 2021a.

Zhaozhuo Xu, Zhao Song, and Anshumali Shrivastava. Breaking the linear iteration cost barrier for some well-known conditional gradient methods using maxip data-structures. In *NeurIPS*, 2021b.

Amir Zandieh, Insu Han, Majid Daliri, and Amin Karbasi. Kdeformer: Accelerating transformers via kernel density estimation. In *ICML*, 2023.

Jiahao Zhang. Graph unlearning with efficient partial retraining. In *WWW*, 2024.

Lichen Zhang. Speeding up optimizations via data structures: Faster search, sample and maintenance. Master's thesis, Carnegie Mellon University, 2022.

Dynamic Maintenance of Kernel Density Estimation Data Structure: From Practice to Theory (Supplementary Material)

Jiehao Liang[1] **Zhao Song**[1,*] **Zhaozhuo Xu**[2] **Junze Yin**[3,†] **Danyang Zhuo**[4]

[1]University of California, Berkeley, [2]Stevens Institute of Technology
[3]Boston University, [4]Duke University
[*]magic.linuxkde@gmail.com,[†]junze@bu.edu

Roadmap. Section A presents the basic definitions and lemmas. Section B presents the technical claims for the proof of our main result. Section C consists of four subsections: Initialize part, Update part, Query part, and LSH part of our data structure. Each section presents the corresponding algorithm and proof of running time. We provide a proof sketch for correctness in Section D. Section E presents the detailed proof of the correctness of our data structure. Section F presents how to change our algorithm into an adaptive algorithm. Section G presents the technical claims of this paper. Section H presents elaborate discussion for our work.

A PRELIMINARIES

The goal of this subsection is to introduce some basic Definitions (Section A.1) and Lemmas (Section A.2) that will be used to prove the main result.

A.1 DEFINITIONS

We start by recalling the definition of geometric weight level.

Definition A.1 (Restatement of Definition 2.1 Geometric Weight Levels)**.** *Fix $R \in \mathbb{N}_+$ and $q \in \mathbb{R}^d$. We define*

$$w_i := f(x_i, q)$$

For any fix $r \in [R] := \{1, 2, \cdots, R\}$, we define

$$L_r := \{x_i \in X \mid w_i \in (2^{-r+1}, 2^{-r}]\}$$

We define the corresponding distance levels as

$$z_r := \max_{\mathrm{s.t.} f(z) \in (2^{-r}, 2^{-r+1}]} z.$$

where $f(z) := f(x, q)$ for $z = \|x - q\|_2$.

In addition, we define $L_{R+1} := P \setminus \bigcup_{r \in [R]} L_r$

We restate the definition and some properties of Locality Sensitive Hashing.

Definition A.2 (Restatement of Definition 2.3, Locally Sensitive Hash)**.** *A family $\mathcal{H}$ is called $(p_{\mathrm{near}}, p_{\mathrm{far}}, z, c)$-sensitive where $p_{\mathrm{near}}, p_{\mathrm{far}} \in [0, 1], z \in \mathbb{R}, c \geq 1$, if for any $x, q \in \mathbb{R}^d$:*

- $\Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \leq r] \geq p_{\mathrm{near}}$
- $\Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \geq cr] \leq p_{\mathrm{far}}$

A.2 LEMMAS

Lemma A.3 (Lemma 3.2 in page 6 of Andoni and Indyk [2006]). *Let $(a, b) \in \mathbb{R}^d \times \mathbb{R}^d$. Fixed $z > 0$, there is a hash family $\mathcal{H}$ such that, if $p_{\mathrm{near}} := p_1(z) := \Pr_{h \sim \mathcal{H}}[h(a) = h(b) \mid \|a - b\|_2 \leq z]$ and $p_{\mathrm{far}} := p_2(z, c) := \Pr_{h \sim \mathcal{H}}[h(a) = h(b) \mid \|a - b\|_2 \geq cz]$, then*

$$\rho := \frac{\log 1/p_{\mathrm{near}}}{\log 1/p_{\mathrm{far}}} \leq \frac{1}{c^2} + O(\frac{\log t}{t^{\frac{1}{2}}})$$

for any $c \geq 1, t > 0$, where $p_{\mathrm{near}} \geq e^{-O(\sqrt{t})}$ and it requires $dt^{O(t)}$ time to evaluate.

Remark A.4. *We find an upper bound for our definition of ρ and evaluation time for hashing. For the rest part, we denote $t = \log^{\frac{2}{3}} n$. Thus we obtain $n^{o(1)}$ evaluation time and $\rho = \frac{1}{c^2} + o(1)$. Since $c = O(\log^{\frac{1}{7}} n)$, we have*

$$\frac{1}{\frac{1}{c^2} + O(\frac{\log t}{t^{\frac{1}{2}}})} = c^2(1 - o(1)).^1$$

In the next lemma, we show the existence of an LSH family that can separate the near points and the far points from the query with high probability.

Lemma A.5 (probability bound for separating points in different level sets, formal version of Lemma 2.6). *Given kernel function f, we have corresponding weight level sets L_r's and distance levels z_r's (Definition 2.1). Given query $q \in \mathbb{R}^d$ and integer $i \in [R + 1]$, $r \in [R]$ satisfying $i > r$, let $x \in L_r$, $x' \in L_i$, $c_{i,r} := \min\{\frac{z_{i-1}}{z_r}, \log^{1/7} n\}$. We set up an Andoni-Indyk LSH family $\mathcal{H}$ (Definition 2.3) with near distance z_r.*

We define

$$\begin{aligned} p_{\mathrm{near},r} &:= \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \leq z] \\ p_{\mathrm{far},r} &:= \Pr_{h \sim \mathcal{H}}[h(x) = h(q) \mid \|x - q\|_2 \geq cz] \end{aligned}$$

Then the following inequalities holds for any integer $k \in \mathbb{N}_+$

1. $\Pr_{h^* \sim \mathcal{H}^k}[\, h^*(x) = h^*(q) \,] \geq p_{\mathrm{near},r}^k$
2. $\Pr_{h^* \sim \mathcal{H}^k}[\, h^*(x') = h^*(q) \,] \leq p_{\mathrm{near},r}^{kc_{i,r}^2(1-o(1))}$

Proof. Since $x \in L_r$, by Lemma 2.1, we have

$$\|x - q\|_2 \leq z_r \tag{3}$$

For $x' \in L_i$, since we assume the f is decaying radial kernel, we have

$$\|x' - q\|_2 \geq z_{i-1} \geq c_{i,r} z_r \tag{4}$$

where the first step follows from Definition 2.1, the last step follows from $c_{i,r} \geq \widetilde{c}_r$.

By Lemma 2.4 and Eq. (3), Eq. (4), we have

1. $\Pr_{h \sim \mathcal{H}}[h(x) = h(q)] \geq p_{\mathrm{near},r}$
2. $\Pr_{h \sim \mathcal{H}}[h(x') = h(q)] \leq p_{\mathrm{far},r}$

By remark 2.5, we have

$$p_{\mathrm{far},r} \leq p_{\mathrm{near},r}^{c_{i,r}(1-o(1))} \tag{5}$$

Then for any integer $k > 1$, we have

$$\Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \;\geq\; p_{\mathrm{near},r}^k$$

$$\Pr_{h^* \sim \mathcal{H}^k}[h^*(x') = h^*(q)] \ \leq \ p_{\mathrm{far},r}^k$$

By Eq. (5), we obtain the final result

1. $\Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \geq p_{\mathrm{near},r}^k$
2. $\Pr_{h^* \sim \mathcal{H}^k}[h^*(x') = h^*(q)] \leq p_{\mathrm{near},r}^{kc_{i,r}^2(1-o(1))}$

Thus, we complete the proof. □

B TECHNICAL CLAIMS

In Section B.1, we show how to bound the size of geometric weight levels. In Section B.2, we explain how show the probability for sampled point recovery. In Section B.3, we give an expectation bound the number of recovered points.

B.1 SIZE OF GEOMETRIC WEIGHT LEVELS

The goal of this section is to prove Lemma B.1.

Lemma B.1 (Sizes of geometric weight levels)**.** *Given $r \in [R]$, we have*

$$|L_r| \leq 2^r n f_{\mathsf{KDE}}^* \leq 2^r n f_{\mathsf{KDE}}.$$

Proof. For any fix $r \in [R]$, $x, q \in \mathbb{R}^d$, we have

$$\begin{aligned} n \cdot f_{\mathsf{KDE}} \geq & \ n \cdot f_{\mathsf{KDE}}^* \\ = & \sum_{x \in X} f(x,q) \\ \geq & \sum_{p \in L_r} f(x,q) \\ \geq & \ |L_r| 2^{-r} \end{aligned}$$

where the first step follows from $f_{\mathsf{KDE}} \geq f_{\mathsf{KDE}}^*$, the second step follows from Definition 1.1, the third step follows from shrinking the number of summands, the last step follows from Definition 2.1.

Thus, we complete the proof. □

B.2 PROBABILITY FOR SAMPLED POINT RECOVERY

The goal of this section is to prove Lemma B.2.

Lemma B.2 (Probability for sampled point recovery)**.** *Suppose that we invoke* DYNAMICKDE.INITIALIZE*. Suppose when $a = a^*$ and $r = r^*$, we sample a point $x \in L_{r^*}$. Given a query q, we invoke* DYNAMICKDE.QUERY*. With probability at least $1 - \frac{1}{n^{10}}$, H_{a^*,r^*} recovered x.*

Proof. By Lemma A.5 we have

$$\Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \geq p_{\mathrm{near},r^*}^k$$

Now note that in LSH.RECOVER (line 6) procedure, we repeat this process for

$$K_{2,r^*} = 100 \log(n) p_{\mathrm{near},r^*}^{-k}$$

times. Thus, for any sampled point $p \in L_{r^*}$, it is recovered in one of the repetitions of phase $r = r^*$, with probability at least $1 - \frac{1}{n^{10}}$.

□

B.3 NUMBER OF RECOVERED POINTS IN EXPECTATION

The goal of this section is to prove Lemma B.3

Lemma B.3 (Upper bound on number of recovered points in expectation)**.** *Fix a query $q \in \mathbb{R}^d$. We define $R := \lceil \log \frac{1}{f_{\mathsf{KDE}}} \rceil$. Fix $r \in [R]$, we define $p := p_{\mathrm{near},r}$. For each $(i,r) \in [R] \times [R]$, we define $c_{i,r} := \min\{\frac{z_{i-1}}{z_r}, \log^{\frac{1}{7}} n\}$. There exists $k \in \mathbb{N}_+$*

$$k := k_r := \frac{1}{\log(1/p)} \max_{l \in \{r+1,\cdots,R+1\}} \lceil \frac{l-r}{c_{l,r}^2(1-o(1))} \rceil.$$

such that for any $i > j$

$$\mathop{\mathbb{E}}_{h^* \sim \mathcal{H}^k}[|\{x' \in L_i : h^*(x') = h^*(q)\}|] = O(1)$$

Proof. By Lemma A.5 we have

$$\Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \leq p^{kc_{i,r}^2(1-o(1))}$$

where $c_{i,j} := \min\{\frac{r_{i-1}}{r_j}, \log^{\frac{1}{7}} n\}, p := p_{\mathrm{near},j} \in (0,1)$(remark 2.5).

$$\begin{aligned}
& \mathop{\mathbb{E}}_{h^* \sim \mathcal{H}^k}[|x' \in L_i : h^*(x') = h^*(q)|] \\
\leq & 2^i n f_{\mathsf{KDE}} \cdot \frac{1}{2^r n f_{\mathsf{KDE}}} \Pr_{h^* \sim \mathcal{H}^k}[h^*(x) = h^*(q)] \\
\leq & 2^{i-r} \cdot p^{kc_{i,r}^2(1-o(1))}
\end{aligned}$$

where the first step follows from lemma B.1 and sampling probability (Algorithm 5 line 25), the second step follows from Lemma 2.6.

Note that for $i > j$, we have

$$k \geq \frac{1}{\log \frac{1}{p}} \lceil \frac{i-r}{c_{i,r}^2(1-o(1))} \rceil \tag{6}$$

Then

$$\begin{aligned}
& 2^{i-r} \cdot p^{kc_{i,r}^2(1-o(1))} \\
= & 2^{i-r} \cdot 2^{\log(p) \cdot kc_{i,r}^2(1-o(1))} \\
\leq & 2^{i-r} 2^{\log(p) \cdot \frac{1}{\log(\frac{1}{p})} \lceil \frac{i-r}{c_{i,r}^2(1-o(1))} \rceil c_{i,r}^2(1-o(1))} \\
\leq & 2^{i-r+r-i} \\
= & 1
\end{aligned}$$

where the first step follows from rewriting in exponential form, the second step follows from Eq. (6) , the third step follows from canceling the same term in both numerator and denominator, and the last step follows from canceling i and r.

Thus, we complete the proof. □

C OUR DATA STRUCTURES

In this section, we describe our data structures in detail. Starting with the initialize part in Section C.1, we state the result of space storage and running time for Initialize in DYNAMICKDE. In Section C.2, we demonstrate the running time for the update part in our data structure. Section C.3 presents the running time for the query procedure. Finally, we study the LSH data structure in Section C.4. It is an important member in the DYNAMICKDE structure and fundamental to the implementation of all three procedures above.

C.1 INITIALIZE PART OF DATA STRUCTURE

In this section, we describe the space storage and running time of INITIALIZE part of our data structure DYNAMICKDE.

We start by showing the space storage of LSH structure.

Lemma C.1 (Space storage of LSH, formal version of Lemma 3.1)**.** *Given data set* $\{x_i\}_{i\in[n]} \subset \mathbb{R}^d$*, parameter* $L, k \in \mathbb{N}_+$*, the* INITIALIZE *(Algorithm 8) of the data-structure* LSH *uses space*

$$O(Lkdn^{o(1)} + Ln)$$

Proof. The space storage comes from two parts: CHOOSEHASHFUNC and CONSTRUCTHASHTABLE.

Part 1. CHOOSEHASHFUNC(line 4) takes L, k as input.

It has a for loop with L iterations.

In each iteration, it samples k functions(line 7) from hash family $\mathcal{H}$ to create $\mathcal{H}_l$, which uses $O(kdn^{o(1)})$ space.

Thus the total space usage of CHOOSEHASHFUNC is $L \cdot O(kdn^{o(1)}) = O(Lkdn^{o(1)})$.

Part 2. CONSTRUCTHASHTABLE(line 11) takes data set $\{x_i\}_{i\in[n]}$ and parameter L as input.

It has two recursive for loops.

- The first for loop repeats L iterations.
- The second for loop repeats n iterations.

The space storage of the inner loop comes from line 28 and line 15, which is $O(1)$.

Thus the total space storage of CONSTRUCTHASHTABLE is $L \cdot n \cdot O(1) = O(Ln)$.

The final space storage of INITIALIZE is

$$\begin{aligned} &\mathbf{Part1} + \mathbf{Part2} \\ &= O(Lkdn^{o(1)} + Ln) \end{aligned}$$

Thus, we complete the proof. □

Using the above lemma, we state the space storage of our DYNAMICKDE structure.

Lemma C.2 (Space storage part of Theorem 1.2, formal version of Lemma 3.2)**.** *The* INITIALIZE *of the data structure* DYNAMICKDE *(Algorithm 5) uses space*

$$O(\epsilon^{-2}(\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \cdot \log^2 n \cdot (\frac{1}{f_{\mathsf{KDE}}} + n^{o(1)} \cdot \log^2 n))$$

Proof. The space storage mainly comes from $K_1 \cdot R$ copies of $\mathcal{H}$.

Now let's consider the space storage of $\mathcal{H}$. By Lemma 3.1, we replace $\{x_i\}_{i\in[n]}, L, k$ with P_r (line 26), $K_{2,r}$ (line 24), k_r (line 22) respectively. We have $|P_r| = O(\frac{1}{f_{\mathsf{KDE}}})$, $K_{2,r} = O(\mathrm{cost}(K) \cdot \log n)$ and $k_r = O(\log n)$. Thus the total space usage of $\mathcal{H}$ is

$$O(Lkdn^{o(1)} + Ln) \tag{7}$$

$$= O(\mathrm{cost}(f)n^{o(1)} \cdot \log^3 n + \mathrm{cost}(f)(\frac{1}{f_{\mathsf{KDE}}}) \cdot \log n) \tag{8}$$

The total space storage of INITIALIZE of the data structure DYNAMICKDE is

$$K_1 \cdot R \cdot O(Lk + Ln)$$

$$= O(K_1 \cdot R \cdot \mathrm{cost}(K) \log n \cdot (\frac{1}{f_{\mathsf{KDE}}} + \log n))$$
$$= O(\epsilon^{-2}(\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \cdot \log^2 n \cdot (\frac{1}{f_{\mathsf{KDE}}} + n^{o(1)} \cdot \log^2 n))$$

where the first step follows from Eq. (7), the last step follows from $K_1 = O(\epsilon^{-2}(\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \cdot \log n)$ and $R = O(\log(1/f_{\mathsf{KDE}}))$. Thus, we complete the proof. □

Next, we show an upper bound on running time for INITILIZE in LSH data structure.

Lemma C.3 (Upper bound on running time of INITIALIZE of the data-structure LSH, formal version of Lemma 3.3)**.** *Given input data points $\{x_i\}_{i \in [n]} \subset \mathbb{R}^d$, parameters $k, L \in \mathbb{N}_+$, LSH parameters $p_{\mathrm{near}}, p_{\mathrm{far}} \in [0,1], c \in [1, \infty), r \in \mathbb{R}_+$ and kernel f, the* INITIALIZE *of the data-structure* LSH*(Algorithm 8) runs in time*

$$O(L \cdot (kdn^{o(1)} + dn^{1+o(1)} + n \log n))$$

Proof. This procedure consists of three parts

Part 1. We invoke CHOOSEHASHTABLE procedure with parameters k, L (line 15). The CHOOSEHASHTABLE procedure has one for loop with L iterations.

Now let's consider the running time in line 7, which is the running time in each iteration. In line 7, we sample k hash functons from hash family $\mathcal{H}$, which takes $O(k \cdot dn^{o(1)})$ time.

Thus the total running time for **Part 1** is

$$O(Lkdn^{o(1)})$$

Part 2. We invoke CONSTRUCTHASHTABLE procedure with data set $\{x_i\}_{i \in [n]}$. This procedure has two recursive for loops.

- The first loops repeat L iterations
- The second loop repeats n iterations

Now let's consider the running time from line 14 to line 15, which is the time for each inner loop.

- line 14: We first evaluate $\mathcal{H}_l(x_i)$, which takes $O(dn^{o(1)})$. Then we insert x_i in the bucket $\mathcal{H}_l(x_i)$, which takes $O(\log n)$ time.
- line 15 takes $O(1)$ time.

The running time from line 14 to line 15 is $O(dn^{o(1)} + \log n)$

The total running time for **Part 2** is

$$O(Ln \cdot (dn^{o(1)} + \log n))$$

Putting it all together. We prove that the INITIALIZE of the data-structure LSH(Algorithm 8) runs in time

$$\begin{aligned} & \mathbf{Part1} + \mathbf{Part2} \\ = & O(Lkdn^{o(1)}) + O(Ln \cdot (dn^{o(1)} + \log n)) \\ = & O(L \cdot (kdn^{o(1)} + dn^{1+o(1)} + n \log n)) \end{aligned}$$

Thus, we complete the proof.

□

Combining the results above, we can demonstrate the running time of INITIALIZE in DYNAMICKDE in the following lemma.

Lemma C.4 (The initialize part of Theorem 1.2, formal version of Lemma 3.4)**.** *Given* $(f : \mathbb{R}^d \times \mathbb{R}^d \to [0,1], P \subset \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1])$*, the* INITIALIZE *of the data-structure* DYNAMICKDE *(Algorithm 5) runs in time*

$$O(\epsilon^{-2} n^{1+o(1)} \operatorname{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^2 n)$$

Proof. The INITIALIZE procedure has two recursive for loops.

- The first loops repeat $K_1 = O(\epsilon^{-2} \log(n) \cdot f_{\mathsf{KDE}}^{-o(1)})$ iterations
- The second loops repeats $R = O(\log \frac{1}{f_{\mathsf{KDE}}})$ iterations

Now let's consider the running time from line 20 to line 27, which is the running time of the inner loop.

- line 20 to line 25 takes $O(\log(1/f_{\mathsf{KDE}}))$ time.
- line 26 takes $O(n)$ time.
- line 27: By Lemma 3.3, we replace L with $K_{2,r} = O(\operatorname{cost}(K) \cdot \log n)$ and k with $k_r = O(\log n)$.
  Thus the running time of this line is

$$\begin{aligned} & O(L \cdot (kdn^{o(1)} + dn^{1+o(1)} + n \log n) \\ = & O(\operatorname{cost}(K) \cdot \log n \cdot (n^{o(1)} \cdot \log^2 n + n^{1+o(1)} \cdot \log n + n \cdot \log n)) \\ = & O(n^{1+o(1)} \operatorname{cost}(K) \cdot \log^2 n) \end{aligned}$$

where the first step follows from $K_{2,r} = O(\operatorname{cost}(K) \cdot \log n)$, $d = O(\log n)$ and $k_r = O(\log n)$, the second step follows from $O(n \log n) = O(n^{1+o(1)})$.

The running time from from line 20 to line 27 is

$$\begin{aligned} & O(\log(1/f_{\mathsf{KDE}})) + O(n) + O(n^{1+o(1)} \operatorname{cost}(K) \log n) \\ = & O(n^{1+o(1)} \operatorname{cost}(K) \log^2 n) \end{aligned}$$

The final running time for INITIALIZE procedure is

$$\begin{aligned} & K_1 \cdot R \cdot O(n^{1+o(1)} \operatorname{cost}(K) \cdot \log^2 n) \\ = & O(\epsilon^{-2} n^{1+o(1)} \operatorname{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \cdot \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n) \end{aligned}$$

where we use $K_1 = O(\epsilon^{-2} \cdot f_{\mathsf{KDE}}^{-o(1)} \cdot \log n)$ and $R = O(\log(1/f_{\mathsf{KDE}}))$.

Thus, we complete the proof. □

C.2 UPDATE PART OF DATA STRUCTURE

The goal of this section is to prove Lemma 3.6. Our Lemma C.6 in this section is the formal version of Lemma 3.6. We present an auxiliary Lemma C.5 and then show how to this this auxiliary lemma to prove Lemma C.6.

Lemma C.5 (Update time of LSH, formal version of Lemma 3.5)**.** *Given a data point* $z \in \mathrm{R}^d$ *and index* $i \in [n]$*, the* UPDATEHASHTABLE *of the data-structure* LSH *runs in (expected) time*

$$O(n^{o(1)} \log(n) \cdot \operatorname{cost}(f)).$$

Proof. This procedure has one for loop which repeats $L = O(\log n)$ iterations. Now let us consider the running time from line 28 to line 29, which is the time for each iteration.

- line 28 takes $O(dn^{o(1)} \operatorname{cost}(f))$

Algorithm 5 Dynamic KDE, members and initialize part

1: **data structure** DYNAMICKDE ▷ Theorem 1.2
2: **members**
3: $d \in \mathbb{N}_+$ ▷ Dimension of data point
4: For $i \in [n]$,$x_i \in \mathbb{R}^d$ ▷ dataset X
5: $K_1 \in \mathbb{N}_+$ ▷ Number of repetitions
6: $R \in \mathbb{N}_+$
7: For $a \in [K_1]$, $\widetilde{P}_a \subset \mathbb{R}^d$ ▷ Sampled data points
8: $K_2 \in \mathbb{N}_+$
9: For $a \in [K_1]$, $r \in [R]$, $H_{a,r} \in \mathrm{LSH}$ ▷ Instances from LSH class
10: **end members**
11:
12: **procedure** INITIALIZE($X \subset \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1]$) ▷ Lemma 3.2
13: ▷ f_{KDE} is a known quantity satisfy $f_{\mathsf{KDE}} \geq f^*_{\mathsf{KDE}}$
14: ▷ ϵ represents the precision of estimation
15: $K_1 \leftarrow C \cdot \epsilon^{-2} \log n \cdot f_{\mathsf{KDE}}^{-o(1)}$
16: $R \leftarrow \lceil \log 1/f_{\mathsf{KDE}} \rceil$
17: **for** $a = 1, 2, \cdots, K_1$ **do**
18: **for** $r = 1, 2, \cdots, R$ **do**
19: **for** $i = r+1, \cdots, R+1$ **do**
20: $c_{i,r} \leftarrow \min\{\frac{z_{i-1}}{z_r}, \log^{\frac{1}{7}} n\}$ ▷ z_r is defined in Definition 2.1
21: **end for**
22: $k_r \leftarrow \max_{i \in \{r+1, \cdots, R+1\}} \frac{1}{\log \frac{1}{p}} \lceil \frac{i-r}{\widetilde{c}_{i,r}(1-o(1))} \rceil$
23: $p_{\mathrm{near},r} \leftarrow p(z_r)$
24: $K_{2,r} \leftarrow 100 \log n \cdot p_{\mathrm{near},r}^{-k_r}$ ▷ $p_{\mathrm{near},r}, p_{\mathrm{far},r}$ are defined in Lemma 2.4
25: $p_{\mathrm{sampling}} \leftarrow \min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$
26: $P_r \leftarrow$ sample each element in X with probability p_{sampling}.
27: $\mathcal{H}_{a,r}$.INITIALIZE($P_r, k_r, K_{2,r}$)
28: **end for**
29: $\widetilde{P}_a \leftarrow$ sample each element in X with probability $\frac{1}{n}$ ▷ Store $\widetilde{P}_a$
30: **end for**
31: **end procedure**
32: **end data structure**

Algorithm 6 Dynamic KDE, update part

1: **data structure** DYNAMICKDE ▷ Theorem 1.2
2:
3: **procedure** UPDATE($v \in \mathbb{R}^d, f_{\mathsf{KDE}} \in [0,1], i \in [n]$) ▷ Lemma 3.6
4: **for** $a = 1, 2, \cdots, K_1$ **do**
5: **for** $r = 1, 2, \cdots, R$ **do**
6: $\mathcal{H}_{a,r}$.UPDATEHASHTABLE(v, i)
7: **end for**
8: **end for**
9: $x_i \leftarrow v$
10: **end procedure**
11: **end data structure**

- line 29 takes the same time as line 28

The final running time

$$L \cdot O(dn^{o(1)} \operatorname{cost}(f))$$
$$= O(n^{o(1)} \operatorname{cost}(f) \cdot \log^2 n).$$

where we use $L = O(\log n)$ and $d = O(\log n)$.

Thus, we complete the proof.

□

Lemma C.6 (The update part of Theorem 1.2, formal version of Lemma 3.6)**.** *Given an update $z \in \mathbb{R}^d$ and index $i \in [n]$, the* UPDATE *of the data-structure* DYNAMICKDE *(Algorithm 6) runs in (expected) time*

$$O(\epsilon^{-2} n^{o(1)} \operatorname{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n).$$

Proof. This algorithm has two recursive for loops

- The first loops repeat $K_1 = O(\epsilon^{-2} \log(n) \cdot f_{\mathsf{KDE}}^{-o(1)})$ iterations
- The second loops repeats $R = O(\log \frac{1}{f_{\mathsf{KDE}}})$ iterations

Now let's consider the running time in line 6, which is the time for each inner loop.

By Lemma 3.5, line 6 takes $O(n^{o(1)} \log(n) \cdot \operatorname{cost}(f))$ time.

The final running time

$$\begin{aligned} & K_1 \cdot R \cdot O(n^{o(1)} \operatorname{cost}(f) \cdot \log^2 n) \\ = & O(\epsilon^{-2} n^{o(1)} \operatorname{cost}(f) \cdot (\frac{1}{f_{\mathsf{KDE}}})^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \log^3 n) \end{aligned}$$

where we use $K_1 = O(\epsilon^{-2} \log(n) \cdot f_{\mathsf{KDE}}^{-o(1)})$ and $R = O(\log \frac{1}{f_{\mathsf{KDE}}})$.

Thus, we complete the proof. □

C.3 QUERY PART OF DATA STRUCTURE

The goal of this section is to prove Lemma C.9. Our Algorithm 7 is for querying the approximated kernel density at point q, and Lemma C.9 specifies the running time for the query operation. In order to prove this lemma, we list and prove a few auxiliary lemmas.

We start by showing a lemma that states the expected number of points in each level set.

Lemma C.7 (expected number of points in level sets, formal version of Lemma 3.7)**.** *Given a query $q \in \mathbb{R}^d$ and fix $r \in [R]$. For any $i \in [R+1]$, weight level L_i contributes at most 1 point to the hash bucket of query q.*

Proof. We consider 2 cases:

Case 1. $i \leq r$: By lemma B.1, we have $|L_i| \leq 2^i n f_{\mathsf{KDE}}$. In the r'th phase, we sample each point in the whole data set with probability $\min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$ to obtain a subset X_r (Algorithm 5 line 26). Then

$$\begin{aligned} & \mathbb{E}[|\{x : x \in L_i \cap X_r\}|] \\ \leq & |L_i| \cdot \frac{1}{2^r n f_{\mathsf{KDE}}} \\ \leq & 2^i n f_{\mathsf{KDE}} \cdot \frac{1}{2^r n f_{\mathsf{KDE}}} \\ = & 2^{i-r} \\ \leq & 1 \end{aligned}$$

where the first step follows from sampling probability $\min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$, the second step follows from $|L_i| \leq 2^i n f_{\mathsf{KDE}}$, the third step follows from canceling $n f_{\mathsf{KDE}}$, the last step follows from $i \leq r$.

Thus, there is at most 1 sampled point from L_i in expectation.

Algorithm 7 Dynamic KDE, query part

data structure DYNAMICKDE ▷ Theorem 1.2

procedure QUERY($q \in \mathbb{R}^d, \epsilon \in (0,1), f_{\mathsf{KDE}} \in [0,1]$)
  for $a = 1, 2, \cdots, K_1$ **do**
    for $r = 1, 2, \cdots, R$ **do**
      $\mathcal{H}_{a,r}$.RECOVER(q)
      $\mathcal{S} \leftarrow \mathcal{S} \cup (\mathcal{H}_{a,r}.\mathcal{R} \cap L_r)$
    end for
    $\mathcal{R}_{R+1} \leftarrow$ recover points in $L_{R+1} \cap \widetilde{P}_a$ ▷ Recover by calculating w directly.
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{R}_{R+1}$
    for $x_i \in \mathcal{S}$ **do**
      $w_i \leftarrow f(x_i, q)$
      if $x_i \in L_r$ for some $r \in [R]$ **then**
        $p_i \leftarrow \min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$
      else if $x_i \in X \setminus \bigcup_{r \in [R]} L_r$ **then**
        $p_i \leftarrow \frac{1}{n}$
      end if
    end for
    $T_a \leftarrow \sum_{x_i \in \mathcal{S}} \frac{w_i}{p_i}$
  end for
  return $\mathrm{Median}\{T_a\}$
end procedure
end data structure

Then L_i contributes at most 1 point in the bucket of query q in expectation.

Case 2. $i = r+1, \cdots, R+1$: By Lemma 2.6, we have $|L_i| \leq 2^i n f_{\mathsf{KDE}}$. The sampling rate in r'th phase is $\min\{\frac{1}{2^r n f_{\mathsf{KDE}}}, 1\}$(Algorithm 5 line 26). Then there are at most 2^{i-r} sampled points from L_i in expectation. We set up LSH function such that the near distance is z_r (Definition 2.1). Also, we use (Algorithm 5 line 22)

$$k := k_r := \frac{1}{\log \frac{1}{p}} \max_{i=r+1,\cdots,R+1} \lceil \frac{i-r}{c_{i,r}(1-o(1))} \rceil.$$

as the number of concatenations. By Lemma B.3, L_i contributes at most 1 point in the bucket of query q in expectation.

The total number of points that L_i contributes to hash bucket of q is $\max\{$**Case 1**,**Case 2**$\} = 1$ in expectation.

Thus, we complete the proof. □

Next, we present the running time of RECOVER procedure in the LSH data structure, which is an important part of QUERY procedure in DYNAMICKDE.

Lemma C.8 (running time for recover points given a query, formal version of Lemma 3.8)**.** *Given a query $q \in \mathbb{R}^d$ and $L, R, k \in \mathbb{N}_+$, the* RECOVER *of the data-structure* LSH *runs in (expected) time*

$$O(Lkn^{o(1)} + LR)$$

Proof. The procedure has one for loop with L iterations. In each iteration, the running time consists of two parts

- The evaluation of $\mathcal{H}_l(q)$ takes $O(kn^{o(1)})$ time
- The RETRIEVE operation takes $O(|\mathcal{H}_l(q)|)$ time. By Lemma C.7, $|\mathcal{H}_l(q)| = O(R)$

The running time of one iteration is $O(kn^{o(1)} + R)$

The final running of this procedure is $L \cdot O(kn^{o(1)} + R) = O(Lkn^{o(1)} + LR)$.

Thus, we complete the proof. □

Based on the running time of RECOVER in LSH above, we prove the running time of QUERY procedure in DYNAMICKDE.

Lemma C.9 (Query part of Theorem 1.2, formal version of Lemma 3.9)**.** *Given a query $q \in \mathbb{R}^d$, the* QUERY *of the data-structure* DYNAMICKDE *(Algorithm 7) runs in (expected) time*

$$O(\epsilon^{-2} n^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot f_{\mathsf{KDE}}^{-o(1)} \cdot \mathrm{cost}(K) \log^3 n).$$

Proof. First, the algorithm do a for loop with $K_1 = O(\epsilon^{-2} \log n \cdot f_{\mathsf{KDE}}^{-o(1)})$ iterations.

In each iteration, the running time consists of three parts.

Part 1. The running time from line 5 to line 8, which is a for loop with R iterations. In each iteration, the running time comes from

- By Lemma 3.8, we replace L with $K_2, j = O(\mathrm{cost}(K) \log n)$, $J = O(\log(1/f_{\mathsf{KDE}}))$ and k with $k_j = O(\log n)$. Thus line 6 takes $O(n^{o(1)} \mathrm{cost}(K) \log^2 n)$ time.
- line 7 takes $O(|\mathcal{H}_{a,r}.\mathcal{R}|)$ time. By Lemma B.3, $|\mathcal{H}_{a,r}.\mathcal{R}| = O(1)$. Thus the running time of line 7 is $O(1)$.

Thus the running time of this for loop is $R \cdot O(n^{o(1)} \mathrm{cost}(K) \log^2 n) = O(n^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \log^2 n)$, where we use $R = O(\log(1/f_{\mathsf{KDE}}))$.

Part 2. The running time from line 11 to line 18, which is a forloop with $|\mathcal{S}|$ iterations. In each iteration, the running time is $O(1)$. By Lemma B.3, $|\mathcal{S}| = O(R)$. Thus the running time of this forloop is $O(\log(1/f_{\mathsf{KDE}}))$, where we use $R = O(\log(1/f_{\mathsf{KDE}}))$.

Part 3. The running time of line 9, 10 and 19 is $O(1)$

The final running time of QUERY is

$$\begin{aligned} & K_1 \cdot (\mathbf{Part1} + \mathbf{Part2} + \mathbf{Part3}) \\ = & K_1 \cdot O(n^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot \mathrm{cost}(K) \log^2 n \\ & + O(\log(1/f_{\mathsf{KDE}})) + O(1)) \\ = & O(\epsilon^{-2} n^{o(1)} \log(1/f_{\mathsf{KDE}}) \cdot f_{\mathsf{KDE}}^{-o(1)} \cdot \mathrm{cost}(K) \log^3 n) \end{aligned}$$

where the first step follows directly from the running time of three parts, and the last step follows from $K_1 = O(\epsilon^{-2} \log n \cdot f_{\mathsf{KDE}}^{-o(1)})$

Thus, we complete the proof.

□

C.4 LSH DATA STRUCTURE

In this section, we present the LSH data structures with the following procedures:

Initialize Given a data set $\{x_1, \cdots, x_n\} \in \mathbb{R}^d$ and integral parameters k, L, it first invokes private procedure CHOOSE-HASHFUNC. The idea behind this is to amplify the "sensitivity" of hashing by concatenating k basic hashing functions from the family $\mathcal{H}$(Algorithm 8 line 9) into a new function. Thus we obtain a family of "augmented" hash function $\mathcal{H}_l, l \in [L]$ (Algorithm 1 line 7). We follow by CONSTRUCTHASHTABLE in which we hash each point x_i using the hashing function $\mathcal{H}_l$. Then we obtain L hash tables corresponding to L hash functions which can be updated quickly.

Recover Given a query $q \in \mathbb{R}^d$, it finds the bucket where q is hashed by $\mathcal{H}_{\updownarrow}$ and retrieve all the points in the bucket according to hashtable $\mathcal{T}_l$. This operation applies to all L hashtables.

UpdateHashTable Given a new data point $z \in \mathbb{R}^d$ and index $i \in [n]$, it repeats the following operations for all $l \in [L]$: find bucket $\mathcal{H}_l(z)$ and insert point z; find bucket $\mathcal{H}_l(x_i)$ and delete point x_i.

Next, we provide a private procedure of LSH in Algorithm 1.

Algorithm 8 LSH, members and public procedures

1: **data structure** LSH
2: **members**
3: $d, n \in \mathbb{N}_+$ ▷ d is dimension, n is number of data points
4: $K, L \in \mathbb{N}_+$ ▷ K is amplification factor, L is number of repetition for hashing
5: $p_{\mathrm{near}}, p_{\mathrm{far}} \in (0, 1)$ ▷ Collision probability
6: For $l \in L$, $\mathcal{T}_l := [n]$ ▷ Hashtable recording data points hashed by $\mathcal{H}_l$
7: $\mathcal{R} := [n]$ ▷ retrieved points
8: $\mathcal{H} := \{f \in \mathcal{H} : \mathbb{R}^d \to [M]\}$ ▷ M is number of buckets for hashing family $\mathcal{H}$
9: For $l \in [L]$, $\mathcal{H}_l \in \mathcal{H}^K$ ▷ Family of amplified hash functions with at most M^K non-empty buckets
10: For $b \in [M^K]$, $\mathcal{S}_b$:=AVL tree ▷ Use AVL tree to store points in bucket
11: **end members**
12:
13: **public**
14: **procedure** INITIALIZE($\{x_i\}_{i\in[n]} \subset \mathbb{R}^d, k, L \in \mathbb{N}_+$)
15: CHOOSEHASHFUNC(k, L)
16: CONSTRUCTHASHTABLE($\{x_i\}_{i\in[n]}$)
17: **end procedure**
18:
19: **procedure** RECOVER($q \in \mathbb{R}^d$)
20: $\mathcal{R} \leftarrow 0$
21: **for** $l \in [L]$ **do**
22: $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{T}_l$.RETRIEVE($\mathcal{H}_l(q)$) ▷ Find the bucket $\mathcal{H}_l(q)$ in $\mathcal{T}_l$ and retrieve all points
23: **end for**
24: **end procedure**
25:
26: **procedure** UPDATEHASHTABLE($z \in \mathbb{R}^d, i \in [n]$)
27: **for** $l \in [L]$ **do**
28: $\mathcal{H}_l(z)$.INSERT(z) ▷ $\mathcal{H}_l(z)$ denotes the bucket that z is mapped to
29: $\mathcal{H}_l(x_i)$.DELETE(x_i)
30: **end for**
31: **end procedure**
32: **end data structure**

D CORRECTNESS: A BRIEF SUMMARY

In this section, we briefly discuss the correctness of the algorithm from data independence (Section D.1), unbiasedness (Section D.2) and variance (Section D.3) aspects.

D.1 DATA INDEPENDENCE OF INITIALIZE AND UPDATE.

We will show that procedure INITIALIZE is independent of the input data. Hence, it is convenient for procedure UPDATE to change only a small part of the stored data, without re-initializing the whole data structure. For INITIALIZE, it first samples data points by doing n rounds of Bernoulli trials on each data point (Algorithm 5, line 26). Then it invokes LSH.INITIALIZE to hash the sampled data points into some buckets, which will be stored into an LSH instance $\mathcal{H}_{a,r}$ (Algorithm 5, Line 5). For UPDATE, it finds the bucket where the old data point is hashed (Algorithm 8, Line 26 and Line 28) and replaces it with the new one. Thus procedure UPDATE maintains the initialized structure.

D.2 UNBIASEDNESS OF QUERY

We now present the unbiasedness of the estimator (up to some inverse polynomial error).

Lemma D.1 (unbiasedness of the QUERY, informal version of Lemma E.1)**.** *Given $f^*_{\mathsf{KDE}} \in (0, 1)$, $f_{\mathsf{KDE}} \geq f^*_{\mathsf{KDE}}$, $\epsilon \in (f^{10}_{\mathsf{KDE}}, 1)$ and $q \in \mathbb{R}^d$, we claim that estimator T_a for any $a \in [K_1]$ constructed in line 19 Algorithm 7 satisfies* $(1 -$

$n^{-9})nf^*_{\mathsf{KDE}} \leq \mathbb{E}[T_a] \leq nf^*_{\mathsf{KDE}}$.

D.3 VARIANCE BOUND FOR QUERY

We present the variance bound of our estimator.

Lemma D.2 (Variance bound for QUERY, informal version of Lemma E.2)**.** *Given $f^*_{\mathsf{KDE}} \in (0,1)$, $f_{\mathsf{KDE}}/4 \leq f^*_{\mathsf{KDE}} \leq f_{\mathsf{KDE}}$, $\epsilon \in (f^{10}_{\mathsf{KDE}}, 1)$ and $q \in \mathbb{R}^d$,*

QUERY *can output a $(1 \pm \epsilon)$-factor approximation to f^*_{KDE}.*

E CORRECTNESS: DETAILS

The goal of this section is to prove the correctness of our algorithms. In Section E.1, we provide the proof of unbiasedness for the query. In Section E.2, we provide the proof of variance bound for the query.

E.1 UNBIASEDNESS OF QUERY

In this section, we prove that the estimator returned by QUERY is unbiased.

Lemma E.1 (unbiasedness of the QUERY, formal version of Lemma D.1)**.** *For every $f^*_{\mathsf{KDE}} \in (0,1)$, every $f_{\mathsf{KDE}} \geq f^*_{\mathsf{KDE}}$, every $\epsilon \in (f^{10}_{\mathsf{KDE}}, 1)$, every $q \in \mathbb{R}^d$, estimator $T_a = \sum_{x_i \in \mathcal{S}} \frac{w_i}{p_i}$ for any $a \in [K_1]$ constructed in line 19 Algorithm 7 satisfies the following:*

$$(1 - n^{-9})nf^*_{\mathsf{KDE}} \leq \mathbb{E}[T_a] \leq nf^*_{\mathsf{KDE}}$$

Proof. Let $\mathcal{E}$ be the event that all the points are sampled. Let $T := T_a$ (see line 19 Algorithm 7). By Lemma B.2 and union bound, we have

$$\Pr[\mathcal{E}] \geq 1 - n^{-9}$$

Thus we obtain $\mathbb{E}[T] = \sum_{i=1}^n \frac{\mathbb{E}[\chi_i]}{p_i} w_i$ and $(1 - n^{-9})p_i \leq \mathbb{E}[\chi_i] \leq p_i$, where $\chi_i = 1$ is defined to be the event that point p_i gets sampled and recovered in the phase corresponding to its weight level, and $\chi_i = 0$ is defined to the contrary. Thus

$$(1 - n^{-9})nf^*_{\mathsf{KDE}} \leq \mathbb{E}[T] \leq nf^*_{\mathsf{KDE}}$$

□

E.2 VARIANCE BOUND FOR QUERY

The goal of this section is to prove the variance bound of the estimator.

Lemma E.2 (Variance bound for QUERY, formal version of Lemma D.2)**.** *For every $f^*_{\mathsf{KDE}} \in (0,1)$, every $\epsilon \in (f^{10}_{\mathsf{KDE}}, 1)$, every $q \in \mathbb{R}^d$, using estimators $T_a = \sum_{x_i \in \mathcal{S}} \frac{w_i}{p_i}$, for $a \in [K_1]$ constructed in line 19 Algorithm 7, where $f_{\mathsf{KDE}}/4 \leq f^*_{\mathsf{KDE}} \leq f_{\mathsf{KDE}}$, one can output a $(1 \pm \epsilon)$-factor approximation to f^*_{KDE}.*

Proof. First, we have $T \leq n^2 f^*_{\mathsf{KDE}}$, where equality holds when all the points are sampled and recovered in the phase of their weight levels. By Lemma D.1, we have

$$\mathbb{E}[T \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + n^2 f^*_{\mathsf{KDE}}(1 - \Pr[\mathcal{E}]) \geq \mathbb{E}[T]$$

Also, we have

$$\mathbb{E}[T \mid \mathcal{E}] \leq \frac{\mathbb{E}[T]}{\Pr[\mathcal{E}]} \leq \frac{nf^*_{\mathsf{KDE}}}{\Pr[\mathcal{E}]} = nf^*_{\mathsf{KDE}}(1 + o(1/n^9))$$

Then, we have

$$
\begin{aligned}
\mathbb{E}[T^2] &= \mathbb{E}[(\sum_{p_i \in P} \chi_i \frac{w_i}{p_i})^2] \\
&= \sum_{i \neq j} \mathbb{E}[\chi_i \chi_j \frac{w_i w_j}{p_i p_j}] + \sum_{i \in [n]} \mathbb{E}[\chi_i \frac{w_i^2}{p_i^2}] \\
&\leq \sum_{i \neq j} w_i w_j + \sum_{i \in [n]} \frac{w_i^2}{p_i} \mathbb{I}[p_i = 1] + \sum_{i \in [n]} \frac{w_i^2}{p_i} \mathbb{I}[p_i \neq 1] \\
&\leq (\sum_i w_i)^2 + \max_i \{ \frac{w_i}{p_i} \mathbb{I}[p_i \neq 1] \} \sum_{i \in [n]} w_i \\
&\leq 2n^2 (f_{\mathsf{KDE}}^*)^2 + \max_{j \in [J], p_i \in L_j} \{ w_i 2^{j+1} \} n f_{\mathsf{KDE}} \cdot n f_{\mathsf{KDE}}^* \\
&\leq 4n^2 f_{\mathsf{KDE}}^2
\end{aligned}
$$

where the first step follows definition of T, the second step follows from expanding the square of summation, the third step follows from $\chi_i \chi_j \leq 1$, the fourth step follows from $\frac{w_i^2}{p_i} \mathbb{I}[p_i = 1] \leq w_i^2$ and $(\sum_i w_i)^2 = \sum_{i \neq j} w_i w_j + \sum_{i \in [n]} w_i^2$, the fifth step follows from $n f_{\mathsf{KDE}}^* = (\sum_i w_i)^2$ and $p_j \geq 1/(n \cdot 2^{j+1} f_{\mathsf{KDE}})$ and $f_{\mathsf{KDE}} \geq f_{\mathsf{KDE}}^*$.

$$
\mathbb{E}[Z^2 \mid \mathcal{E}] \leq \frac{\mathbb{E}[Z^2]}{\Pr[\mathcal{E}]} \leq n^2 f_{\mathsf{KDE}}^{2-o(1)} (1 + o(1/n^9))
$$

Now, we repeat this process for $K_1 = \frac{C \log n}{\epsilon^2} \cdot f_{\mathsf{KDE}}^{-o(1)}$ times with constant C. Then, we have $(1 \pm \epsilon)$-factor approximation with higher success probability. We show that if we repeat the procedure m times and take average, denoted as $\bar{T}$, we have:

$$
\begin{aligned}
& \Pr[|\bar{T} - n f_{\mathsf{KDE}}^*| \geq \epsilon n f_{\mathsf{KDE}}^*] \\
\leq\ & \Pr[|\bar{T} - \mathbb{E}[T]| \geq \epsilon n f_{\mathsf{KDE}}^* - |\mathbb{E}[T] - n f_{\mathsf{KDE}}^*|] \\
\leq\ & \Pr[|\bar{T} - \mathbb{E}[T]| \geq (\epsilon - n^{-9}) n f_{\mathsf{KDE}}^*] \\
\leq\ & \frac{\mathbb{E}[\bar{T}^2]}{(\epsilon - n^{-9})^2 (n^2 f_{\mathsf{KDE}}^*)^2} \\
\leq\ & \frac{1}{m} \frac{64 n^2 (f_{\mathsf{KDE}}^*)^2}{(\epsilon - n^{-9})^2 (n^2 f_{\mathsf{KDE}}^*)^2}
\end{aligned}
$$

where the first step follows from $|\bar{T} - n f_{\mathsf{KDE}}^*| \leq |\bar{T} - \mathbb{E}[T]| + |\mathbb{E}[T] - n f_{\mathsf{KDE}}^*|$, the second step follows from $\mathbb{E}[T] \geq (1 - n^{-9} n f_{\mathsf{KDE}}^*)$, the third step follows from Markov inequality and the last step follows from $\mathbb{E}[\bar{T}^2] \leq \mathbb{E}[T^2]/m \leq 4n^2 f_{\mathsf{KDE}}^2$ and $f_{\mathsf{KDE}} \leq 4 f_{\mathsf{KDE}}^*$.

We can repeat $m = O(\frac{1}{\epsilon^2})$ times to upper bound failure probability to δ and then take the median out of $O(\log(1/\delta))$ means, where we assume $\delta = \frac{1}{\mathrm{poly}(n)}$.

□

F ADVERSARY

In this section, we provide the detailed proofs for the lemmas in Section 4.

Starting Point In Section D, we have already obtained a query algorithm with constant success probability for a fixed query point.

Lemma F.1 (Starting with constant probability, restatement of Lemma 4.1)**.** *Given $\epsilon \in (0, 0.1)$, a query point $q \in \mathbb{R}^d$ and a set of data points $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^d$, let $f_{\mathsf{KDE}}^*(q) := \frac{1}{|X|} \sum_{x \in X} f(x, q)$ be an estimator $\mathcal{D}$ can answer the query which satisfies:*

$$(1 - \epsilon) \cdot f_{\mathsf{KDE}}^*(q) \leq \mathcal{D}.\textsc{query}(q, \epsilon) \leq (1 + \epsilon) \cdot f_{\mathsf{KDE}}^*(q)$$

with probability 0.9.

Proof. By Lemma E.1 and Lemma E.2, our QUERY procedure can provide an estimator that answers kernel density estimation correctly with constant probability. □

Boost the constant probability to high probability. Next, we begin to boost the success probability by repeating the query procedure and taking the median output.

Lemma F.2 (Boost the constant probability to high probability, restatement of Lemma 4.2)**.** *Let $\delta_1 \in (0, 0.1)$ denote the failure probability. Let $\epsilon \in (0, 0.1)$ denote the accuracy parameter. Given $L = O(\log(1/\delta_1))$ estimators $\{\mathcal{D}_j\}_{j=1}^L$. For each fixed query point $q \in \mathbb{R}^d$, the median of queries from L estimators satisfies that:*

$$\begin{aligned}(1 - \epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f_{\mathsf{KDE}}^*(q)\end{aligned}$$

with probability $1 - \delta_1$.

Proof. From Lemma 4.1 we know each estimator $\mathcal{D}_j$ can answer the query that satisfies:

$$(1 - \epsilon) \cdot f_{\mathsf{KDE}}^*(q) \leq \mathcal{D}.\textsc{query}(q, \epsilon) \leq (1 + \epsilon) \cdot f_{\mathsf{KDE}}^*(q)$$

with probability 0.9.

From the chernoff bound we know the median of $L = O(\log(1/\delta_1))$ queries from $\{\mathcal{D}_j\}_{j=1}^L$ satisfies:

$$\begin{aligned}(1 - \epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f_{\mathsf{KDE}}^*(q)\end{aligned}$$

with probability $1 - \delta_1$.

Therefore, we complete the proof. □

From each fixed point to all the net points. So far, the success probability of our algorithm is still for a fix point. We will introduce ϵ-net on a unit ball and show the high success probability for all the net points.

Fact F.3. *Let N denote the ϵ_0-net of $\{x \in \mathbb{R}^d \mid \|x\|_2 \leq 1\}$. We use $|N|$ to denote the number of points in N. Then $|N| \leq (10/\epsilon_0)^d$.*

This fact shows that we can bound the size of an ϵ-net with an inverse of ϵ. We use this fact to conclude the number of repetitions we need to obtain the correctness of QUERY on all net points.

Lemma F.4 (From each fixed points to all the net points, restatement of Lemma 4.4)**.** *Let N denote the ϵ_0-net of $\{x \in \mathbb{R}^d \mid \|x\|_2 \leq 1\}$. We use $|N|$ to denote the number of points in N. Given $L = \log(|N|/\delta)$ estimators $\{\mathcal{D}_j\}_{j=1}^L$.*

With probability $1 - \delta$, we have: for all $q \in N$, the median of queries from L estimators satisfies that:

$$\begin{aligned}(1 - \epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\textsc{query}(q, \epsilon)\}_{j=1}^L) \\ &\leq (1 + \epsilon) \cdot f_{\mathsf{KDE}}^*(q).\end{aligned}$$

Proof. There are $|N|$ points on the d dimension ϵ-net when $\|q\|_2 \leq 1$. From Lemma F.2 we know that for each query point q on N, we have :

$$\begin{aligned}(1-\epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\mathrm{QUERY}(q,\epsilon)\}_{j=1}^L) \\ &\leq (1+\epsilon) \cdot f_{\mathsf{KDE}}^*(q)\end{aligned}$$

with probability $1 - \delta/|N|$.

By union bound all $|N|$ points on N, we have:

$$\begin{aligned}\forall \|q\|_2 \leq 1, \quad (1-\epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\mathrm{QUERY}(q,\epsilon)\}_{j=1}^L) \\ &\leq (1+\epsilon) \cdot f_{\mathsf{KDE}}^*(q)\end{aligned}$$

with probability $1 - \delta$. □

From net points to all points. With Lemma F.4, we are ready to extend the correctness for net points to the whole unit ball. We demonstrate that all query points $\|q\|_2 \leq 1$ can be answered approximately with high probability in the following lemma.

Lemma F.5 (From net points to all points, restatement of Lemma 4.5)**.** *Let $\epsilon \in (0, 0.1)$. Let $\mathcal{L} \geq 1$. Let $\delta \in (0, 0.1)$. Let $\tau \in [0, 1]$. Given $L = O(\log((\mathcal{L}/\epsilon\tau)^d/\delta))$ estimators $\{\mathcal{D}_j\}_{j=1}^L$, with probability $1 - \delta$, for all query points $\|p\|_2 \leq 1$, we have the median of queries from L estimators satisfies that:*

$$\begin{aligned}\forall \|p\|_2 \leq 1, \quad (1-\epsilon) \cdot f_{\mathsf{KDE}}^*(p) &\leq \mathrm{Median}(\{\mathcal{D}_j.\mathrm{QUERY}(q,\epsilon)\}_{j=1}^L) \\ &\leq (1+\epsilon) \cdot f_{\mathsf{KDE}}^*(p).\end{aligned}$$

where q is the closest net point of p.

Proof. We define an event ξ to be the following,

$$\begin{aligned}\forall q \in N, \quad (1-\epsilon) \cdot f_{\mathsf{KDE}}^*(q) &\leq \mathrm{Median}(\{\mathcal{D}_j.\mathrm{QUERY}(q,\epsilon)\}_{j=1}^L) \\ &\leq (1+\epsilon) \cdot f_{\mathsf{KDE}}^*(q)\end{aligned}$$

Using Lemma F.4 with $L = \log(|N|/\delta)$, we know that

$$\Pr[\text{ event } \xi \text{ holds }] \geq 1 - \delta$$

Using Fact F.3, we know that

$$\begin{aligned}L &= \log(|N|/\delta) \\ &= \log((10/\epsilon_0)^d/\delta) \\ &= \log((10\mathcal{L}/\epsilon\tau)^d/\delta)\end{aligned}$$

where the last step follows from $\epsilon_0 = \epsilon\tau/\mathcal{L}$.

We condition the above event E to be held. (Then the remaining proof does not depend on any randomness, for each and for all becomes the same.)

For each point $p \notin N$, there exists a $q \in N$ such that

$$\|p - q\|_2 \leq \epsilon_0 \tag{9}$$

For each p, we know

$$\begin{aligned}&|\mathrm{Median}_j \mathcal{D}_j.\mathrm{QUERY}(q,\epsilon) - f_{\mathsf{KDE}}^*(p)| \\ \leq\ &|\mathrm{Median}_j \mathcal{D}_j.\mathrm{QUERY}(q,\epsilon) - f_{\mathsf{KDE}}^*(q)| + |f_{\mathsf{KDE}}^*(q) - f_{\mathsf{KDE}}^*(p)| \\ \leq\ &\epsilon f_{\mathsf{KDE}}^*(q) + \mathcal{L} \cdot \|p - q\|_2\end{aligned}$$

$$\leq \epsilon(f^*_{\mathsf{KDE}}(p) + \mathcal{L}\epsilon_0) + \mathcal{L} \cdot \epsilon_0$$
$$\leq \epsilon(f^*_{\mathsf{KDE}}(p) + 2\tau)$$

where the first step follows from Lipschitz, the second step follows from Eq. (9), the third step follows from $\epsilon_0 \leq \epsilon\tau/\mathcal{L}$.

Using $\forall j \in [L] : \mathcal{D}_j.\mathrm{QUERY}(p, \epsilon) \geq \tau$, we have

$$(1 - 3\epsilon) \cdot f^*_{\mathsf{KDE}}(p) \leq \mathrm{Median}(\{\mathcal{D}_j.\mathrm{QUERY}(q, \epsilon)\}_{j=1}^L)$$
$$\leq (1 + 3\epsilon) \cdot f^*_{\mathsf{KDE}}(p).$$

Rescaling the ϵ completes the proof. □

Thus, we obtain an algorithm that could respond to adversary queries robustly.

G TECHNICAL CLAIMS

In this section, we list some technical claims that are useful for our main results. We start by giving an upper bound on sizes of geometric weight levels (Definition 2.1).

Lemma G.1 (Sizes of geometric weight levels, informal version of Lemma B.1)**.** *Given $r \in [R]$, we have $|L_r| \leq 2^r n f^*_{\mathsf{KDE}} \leq 2^r n f_{\mathsf{KDE}}$.*

Next, we show a lemma for the probability of recovering a point in the query procedure, given that this point is sampled in the preprocessing stage.

Lemma G.2 (Probability for sampled point recovery, informal version of Lemma B.2)**.** *Suppose that we invoke* DYNAMICKDE.INITIALIZE. *Suppose when $a = a^*$ and $r = r^*$, we sample a point $x \in L_{r^*}$. Given a query q, we invoke* DYNAMICKDE.QUERY. *With probability at least $1 - \frac{1}{n^{10}}$, H_{a^*,r^*} recovered x.*

With the above lemma, we can bound the number of recovered points in expectation. We show that there are only $O(1)$ points recovered by LSH in each geometric weight level (Definition 2.1).

Lemma G.3 (Upper bound on number of recovered points in expectation, informal version of Lemma B.3)**.** *Fix a query $q \in \mathbb{R}^d$. We define $R := \lceil \log \frac{1}{f_{\mathsf{KDE}}} \rceil$. Fix $r \in [R]$, we define $p := p_{\mathrm{near},r}$. For each $(i, r) \in [R] \times [R]$, we define $c_{i,r} := \min\{\frac{z_{i-1}}{z_r}, \log^{\frac{1}{7}} n\}$. There exists $k \in \mathbb{N}_+$*

$$k := k_r := \frac{1}{\log(1/p)} \max_{l \in \{r+1, \cdots, R+1\}} \lceil \frac{l - r}{c^2_{l,r}(1 - o(1))} \rceil,$$

such that for any $i > j$ $\mathbb{E}_{h^* \sim \mathcal{H}^k}[|\{x' \in L_i : h^*(x') = h^*(q)\}|] = O(1)$

Finally, we claim that the kernel function is Lipschitz. This is an important property for designing robust algorithms.

Lemma G.4. *Suppose kernel function $f^*_{\mathsf{KDE}} : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ satisfies the following properties:*

- *Radial: there exists a funtion $f : \mathbb{R} \to [0, 1]$ such that $f(p, q) = f(\|p - q\|_2)$, for all $p, q \in \mathbb{R}$.*
- *Decreasing: f is decreasing*
- *Lipschitz: f is $\mathcal{L}$-Lipschitz*

Then KDE function $f^*_{\mathsf{KDE}} : \mathbb{R}^d \to [0, 1]$, $f^*_{\mathsf{KDE}}(q) := \frac{1}{|P|} \sum_{p \in P} f(p, q)$ is $\mathcal{L}$-Lipschitz, i.e.

$$|f^*_{\mathsf{KDE}}(q) - f^*_{\mathsf{KDE}}(q')| \leq \mathcal{L} \cdot \|q - q'\|_2$$

Proof. For any $q, q' \in \mathbb{R}^d$, we have:

$$|f^*_{\mathsf{KDE}}(q) - f^*_{\mathsf{KDE}}(q')|$$

$$
\begin{aligned}
&= |\frac{1}{|P|}\sum_{p\in P} f(p,q) - \frac{1}{|P|}\sum_{p\in P} f(p,q')| \\
&\leq \frac{1}{|P|}\sum_{p\in P} |f(p,q) - f(p,q')| \\
&= \frac{1}{|P|}\sum_{p\in P} |f(\|p-q\|_2) - f(\|p-q'\|_2)| \\
&\leq \frac{1}{|P|}\sum_{p\in P} \mathcal{L}\cdot |\ \|p-q\|_2 - \|p-q'\|_2\ | \\
&\leq \frac{1}{|P|}\sum_{p\in P} \mathcal{L}\cdot \|q-q'\|_2 \\
&= \mathcal{L}\cdot \|q-q'\|_2
\end{aligned}
$$

where the first step follows from the definition of f^*_{KDE}, the second step follows from triangular inequality of absolute value, the third step follows from the property of radial kernel, the fourth step follows from the Lipschitz property of f, the fifth step follows from the triangular property of L_2 norm, and the last step follows from canceling $|P|$.

Thus, we complete the proof. □

H DISCUSSION

Implementation in Deep Neural Networks. Our data structure DYNAMICKDE can be integrated with deep neural networks in tasks that require efficient and adaptive kernel-based similarity or density estimation. One application is in the embedding or attention layers of neural architectures, where kernel similarity computations are needed. For example, in Zandieh et al. [2023], they reduce the softmax matrix in attention computation to a variant of KDE, and implement the efficient KDE solver to approximate the attention computation in sub-quadratic time. Our dynamic maintenance of KDE data structures with robustness to adversarial queries can also be used to study the attention computation problem in future work. To implement this in DNNs, we can follow these steps:

1. **Preprocessing:** Use the INITIALIZE of the data-structure DYNAMICKDE (Algorithm 2) to preprocess the data using the chosen kernel and LSH hash tables. This can be done on feature embeddings generated by earlier layers of the network.
2. **Query Phase:** During inference or training (especially in attention-like mechanisms), use QUERY of DYNAMICKDE (Algorithm 4) to efficiently compute kernel densities for input query embeddings. The approximation guarantees are preserved due to importance sampling and LSH recovery.
3. **Online/Incremental Learning:** When new data points are introduced (e.g., in continual learning or streaming settings), UPDATE of DYNAMICKDE (Algorithm 3) allows for sublinear-time integration of new points, without reinitialization of the entire data structure.

Dynamically Evolving Data Distributions. We believe that our method can handle dynamically evolving data distributions, such as the distribution in Xie et al. [2024]. This is because we focus on dynamically updating kernel density estimates in response to insertions/deletions in the dataset and handling adaptive/adversarial queries. Xie et al. [2024] analyzes evolving domain generalization (EDG), where the data distribution changes over time due to factors like temporal shifts (concept drift, covariate shift). It proposes Mutual Information-Based Sequential Autoencoders (MISTS) that explicitly separate dynamic and invariant features to adapt across evolving domains. Our method does provide robustness to dynamic changes in the dataset, making it applicable in scenarios where the data evolves over time. However, this may require a more careful analysis of the details of both works, so we leave this as a future direction.

Justification of Assumptions. In our proof, we consider the worst-case scenario. Specifically, we apply the ϵ-net technique, where we union bound over all balls in the covering net to ensure robustness against adaptive, potentially adversarial queries. We do not assume queries are i.i.d., unlike many traditional data structures. Instead, our framework is explicitly designed for adaptive queries, which are the essence of adversarial behavior in real-world applications such as interactive machine learning, data poisoning scenarios, and online optimization. The robustness guarantees are built up incrementally: from

single-query success probability, to net points on the unit ball via ϵ-nets, and finally to all query points in the input space. This ensures our results extend beyond static, non-adversarial settings and are well-founded in scenarios where queries are interdependent or chosen in response to previous answers.

Practical Implications. Kernel density estimation (KDE) has a direct connection to efficient attention computation in Transformers, as widely discussed in prior works Tsai et al. [2019], Zandieh et al. [2023], Alman and Song [2023, 2024a,b, 2025a,b]. Recall that for the token sequence $X_\ell \in \mathbb{R}^{n \times d}$ at Transformer layer ℓ and weight matrices $Q, K \in \mathbb{R}^{d \times d}$, the attention weight matrix is given by

$$\mathsf{Attn}(Q, K) := D^{-1} \exp(X_\ell Q K^\top X_\ell^\top),$$

where $D := \mathrm{diag}(\exp(X_\ell Q K^\top X_\ell^\top) \mathbf{1}_n)$ and $\exp(A)_{i,j} = \exp(A_{i,j})$ for all matrices A.

We can set $k_i := (X_\ell K)_{i,*}, q_i := (X_\ell Q)_{i,*}, A_{i,j} = \exp(q_i^\top k_j)$ for all $i \in [n]$. Then, it is evident that A is a kernel matrix whose entries are exponentiated inner products, and the only difference between $\mathsf{Attn}(Q, K) = D^{-1} A$ and kernel computation of A is the normalization matrix D^{-1}.

To make the connection explicit, we can consider the Gaussian kernel

$$f_{\mathrm{Gaussian}}(k, q) := \exp(-0.5 \sigma^{-2} \|k - q\|_2^2).$$

When $\|k\|_2 = 1$ and $\|q\|_2 = 1$, the Gaussian kernel can be simplified as

$$f_{\mathrm{Gaussian}}(k, q) = \exp(\sigma^{-2}(q^\top k - 2)),$$

which corresponds to $\exp(q^\top k)$ and exactly recovers the attention computation $A_{i,j} = \exp(q_i^\top k_j)$.

Thus, the efficient KDE algorithm proposed in this paper may inspire further applications in efficient attention computation, including Transformer architectures Frantar and Alistarh [2023], Liang et al. [2025], Chen et al. [2025b], Gao et al. [2025], graph attention Veličković et al. [2018], Fu et al. [2021], Zhang [2024], Li et al. [2025], and attention-inspired regression Deng et al. [2023a], Gao et al. [2023], Song et al. [2024], Deng et al. [2024]. These architectural advancements can further enhance computational efficiency across various fields, such as diffusion models Hu et al. [2024], Shen et al. [2025], Cao et al. [2025c], Guo et al. [2025] and flow-based generative models Chen et al. [2025a], Cao et al. [2025a,b], Gong et al. [2025].