Locality-Sensitive Hashing-Based Efficient Point Transformer with Applications in High-Energy Physics

Siqi Miao [1] **Zhiyuan Lu** [2] **Mia Liu** [3] **Javier Duarte** [4] **Pan Li** [1]

Abstract

This study introduces a novel transformer model optimized for large-scale point cloud processing in scientific domains such as high-energy physics (HEP) and astrophysics. Addressing the limitations of graph neural networks and standard transformers, our model integrates local inductive bias and achieves near-linear complexity with hardware-friendly regular operations. One contribution of this work is the quantitative analysis of the error-complexity tradeoff of various sparsification techniques for building efficient transformers. Our findings highlight the superiority of using locality-sensitive hashing (LSH), especially OR & AND-construction LSH, in kernel approximation for large-scale point cloud data with local inductive bias. Based on this finding, we propose LSH-based Efficient Point Transformer (**HEPT**), which combines E^{2}LSH with OR & AND constructions and is built upon regular computations. HEPT demonstrates remarkable performance on two critical yet time-consuming HEP tasks, significantly outperforming existing GNNs and transformers in accuracy and computational speed, marking a significant advancement in geometric deep learning and large-scale scientific data processing. Our code is available at https://github.com/Graph-COM/HEPT.

1. Introduction

Many scientific applications require the processing of complex research objects, often represented as large-scale point clouds — a set of points within a geometric space — in real time. For instance, in high-energy physics (HEP) (Radovic et al., 2018), to search for new physics beyond the standard model, e.g., new particles predicted by supersymmetric theories (Oerter, 2006; Wess & Bagger, 1992), the CERN Large Hadron Collider (LHC) produces 1 billion particle collisions per second, forming point clouds of detector measurements with tens of thousands of points (Gaillard, 2017), necessitating real-time analysis due to storage limitations. Similarly, in astrophysics (Halzen & Klein, 2010), the IceCube Neutrino Observatory records 3,000 events per second using over 5,000 sensors (Aartsen et al., 2017), and simulations of galaxy formation and evolution need to run billions of particles (Nelson et al., 2019). In drug discovery applications, large-scale real-time computation is crucial for screening billions of protein-antibody pairs, each requiring molecular dynamics simulations of systems with thousands of atoms (Durrant & McCammon, 2011). Facing these extensive computational demands, machine learning, in particular geometric deep learning (GDL), has emerged as a revolutionary tool, offering to replace the most resource-intensive parts of these processes (Bronstein et al., 2017; 2021).

In these scientific applications, inference tasks often exhibit local inductive bias, meaning that the labels to be predicted are primarily determined by aggregating information from local regions within the ambient space. Leveraging local inductive bias can significantly reduce computational complexity. Consequently, graph neural networks (GNNs) have gained widespread use due to their proficiency in exploiting such sparse data patterns (Jing et al., 2021; Kansal et al., 2021; Satorras et al., 2021; DeZoort et al., 2021; Abbasi et al., 2022; Li et al., 2023). However, GNNs still face two computational challenges that hinder their application in real-time scenarios. *First, the procedure of graph construction is time-consuming:* GNNs often use k-NN or other relational rules to construct graphs (Stärk et al., 2022; Lieret et al., 2023). Creating these graphs from n points using brute-force methods involves $\mathcal{O}(n^2)$ complexity. While algorithms like KD-trees theoretically offer $\mathcal{O}(n \log n)$ complexity, their limited parallelizability makes them impractical for real-time data processing pipelines (Wieschollek et al., 2016). *Second, The irregular structure of graphs and the neighborhood aggregation process in GNNs lead to irregular computations and random memory access.* These factors, coupled with dynamic computation graphs for different inputs, pose significant computation challenges

[1]Georgia Institute of Technology [2]Beijing University of Posts and Telecommunications [3]Purdue University [4]University of California San Diego. Correspondence to: Siqi Miao <siqi.miao@gatech.edu>, Pan Li <panli@gatech.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

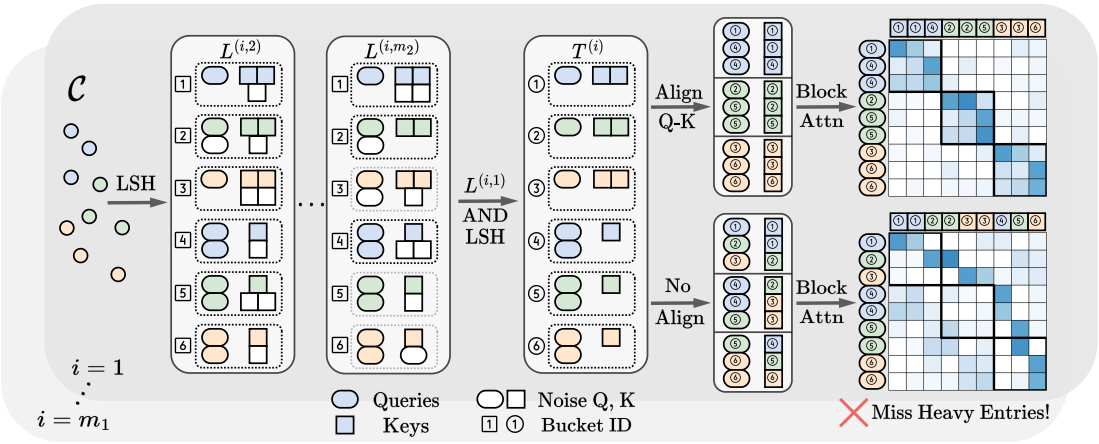


Figure 1: Pipeline of HEPT. Elements that share the same color represent points from the same local neighborhood. HEPT employs OR & AND LSH to minimize noise caused by individual hash functions. HEPT also integrates point coordinates as extra AND LSH codes for query-key alignment, maintaining computational regularity without compromising accuracy.

for conventional hardware architectures (Jang et al., 2010; Hashemi et al., 2018; Abadal et al., 2021). These issues render GNNs less suitable for large-scale real-time point cloud analysis. Therefore, there is a growing interest in exploring alternative approaches to address the above challenges.

Recently, transformer architectures have demonstrated impressive capabilities across various domains (Vaswani et al., 2017; Brown et al., 2020). Unlike GNNs, transformers are noted for their ability to model long-range dependencies and their compatibility with hardware due to regular computation patterns. However, a major limitation of standard transformers is their quadratic complexity to input size, which poses challenges for processing large-scale point cloud data. In this study, we aim to integrate the strengths of both transformers and GNNs by developing an efficient transformer model for point cloud processing. This model incorporates local inductive bias and may achieve near-linear complexity, balancing high accuracy with hardware efficiency through regular and parallelizable computations.

Several studies have been conducted on efficient transformers. However, efficient transformers are not yet fully embraced by scientific domains dealing with geometric data (Kansal et al., 2023; Pata et al., 2023). The primary issue is that existing methods, which use low-rank (Wang et al., 2020) or sparse (Kitaev et al., 2020) approximations of the attention matrix, often overlook approximation errors in method design (Kitaev et al., 2020; Daras et al., 2020) or fail to adequately consider local inductive bias (Choromanski et al., 2021; Peng et al., 2021), leading to undesired model performance. Some methods may compromise approximation accuracy for computational regularity (Kitaev et al., 2020; Daras et al., 2020; Zandieh et al., 2023; Han et al., 2024). Moreover, a systematic understanding of the tradeoff between approximation errors and computational complexity among the different methods is missing, making it difficult to select the most effective method for GDL tasks.

In this work, we close the gap by conducting a quantitative analysis of the error-complexity tradeoff, focusing on two widely used techniques for building efficient transformers: random Fourier features (RFF) (Rahimi & Recht, 2007) and locality-sensitive hashing (LSH) (Indyk & Motwani, 1998). Our analysis indicates that for tasks with local inductive bias, RFF consistently exhibits higher approximation error compared to LSH under subquadratic complexity. We also discover that relying solely on OR-construction LSH results in suboptimal performance, and combining OR & AND-construction LSH (Leskovec et al., 2020), often ignored in prior research, is essential to minimize errors for point clouds in large multidimensional spaces.

Inspired by the analysis, we propose an LS**H**-based **E**fficient **P**oint **T**ransformer (**HEPT**), designed to support highly regular computations with near-linear complexity and provably low approximation errors for tasks with local inductive bias. HEPT leverages a kernel that explicitly embeds local inductive bias for attention calculation, and adopts E^2LSH combined with both OR & AND LSH to effectively minimize approximation errors. To ensure computational regularity without compromising accuracy, HEPT partitions queries and keys into regular buckets based on their LSH codes, and computes only blockwise attention weights. To address the issue of the misalignment of query-key buckets (Sec. 4.3), HEPT proposes to integrate point coordinates as extra AND LSH codes. The pipeline of HEPT is shown in Fig. 1.

To validate the effectiveness of HEPT, we evaluate it on two critical computationally intensive HEP tasks: charged particle tracking (Amrouche et al., 2020) and pileup mitigation (Martínez et al., 2019), with their significance elaborated in Appendix A. HEPT is benchmarked against five GNNs and seven efficient transformers adapted from both NLP and CV domains under a unified framework on three datasets (with one of them contributed by us). HEPT significantly outperforms all baselines, achieving state-of-the-art

(SOTA) accuracy with up to 203× speedup on GPUs. Our experiments also show that existing RFF-based methods fail to deliver competitive performance. LSH-based baselines achieve acceptable accuracy for medium-sized point clouds, however, they struggle to scale to larger datasets with tens of thousands of points, and only HEPT can process them efficiently and achieve SOTA prediction accuracy.

2. Preliminaries

Geometric Deep Learning Tasks. We focus on tasks with each sample represented as a point cloud $\mathcal{C} = (\mathcal{V}, \boldsymbol{X}, \boldsymbol{\rho})$, where $\mathcal{V} = \{u_1, \cdots, u_n\}$ is a set of n points, $\boldsymbol{X} \in \mathbb{R}^{n \times k_1}$ includes k_1-dimensional features for each point, and $\boldsymbol{\rho} \in \mathbb{R}^{n \times k_2}$ specifies the coordinates of these points in a k_2-dimensional space. We consider GDL tasks that require learning meaningful H-dimensional latent representations for each point via a neural network $f : \mathcal{C} \to \mathbb{R}^{n \times H}$. Depending on the specific task, these representations are either used for direct point-wise label prediction or point-pair-wise relationship analysis, or whole point cloud prediction via aggregating (e.g., averaging) these representations.

Random Fourier Features. Consider any positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ that is properly normalized, i.e., $k(\boldsymbol{0}) = 1$. Bochner's Theorem (Rudin, 1991) guarantees that its Fourier transform $k^*(\boldsymbol{w})$ is a probability distribution. Thus, Rahimi & Recht (2007) propose RFFs to approximate such a kernel by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi : \mathbb{R}^d \to \mathbb{R}^D$, where $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \Big(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}) \Big)^\top$, $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$.

Locality-Sensitive Hashing. LSH (Indyk & Motwani, 1998) was proposed for efficient nearest-neighbor search. With high probability, it hashes close data points into the same bucket and distant ones into different buckets. E^{2}LSH (Datar et al., 2004) is an LSH variant for Euclidean distances with a hash family $\mathcal{H}$ and hash functions $h_{\boldsymbol{a},b}(\boldsymbol{x}) \in \mathcal{H}$, where, for a point $\boldsymbol{x} \in \mathbb{R}^d$, $h_{\boldsymbol{a},b}(\boldsymbol{x}) = \lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{x} + b}{r} \rfloor$, $\boldsymbol{a} \sim \mathcal{N}(0, \boldsymbol{I})$, $b \sim \mathcal{U}(0, r)$, and $r > 0$ is a hyperparameter to control bucket sizes. There are also variants for angular distances (Andoni et al., 2015) and inner products (Shrivastava & Li, 2014). To amplify LSH's performance, AND LSH, OR LSH, or a hybrid of both can be utilized. AND LSH concatenates multiple (say m_2) hash functions $h_j \in \mathcal{H}$ to form a new hash family $\mathcal{G}$, where for $g \in \mathcal{G}$, $g(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \ldots, h_{m_2}(\boldsymbol{x})]$, and two points are deemed neighbors if they match across all m_2 hash functions in g. OR LSH, on the other hand, forms multiple (say m_1) hash tables from $\mathcal{G}$, i.e., $g_1(\boldsymbol{x}), \ldots, g_{m_1}(\boldsymbol{x})$ with each $g_i(\boldsymbol{x}) = [h_{i,1}(\boldsymbol{x}), \ldots, h_{i,m_2}(\boldsymbol{x})]$, and two points are neighbors if they match in any one of these m_1 tables. When $m_2 = 1$ (one hash function per table), it becomes OR-only LSH, and when $m_2 \geq 2$, it is a hybrid of OR & AND LSH.

Efficient Transformers as Kernel Approximation. The quadratic complexity of the original transformer (Vaswani et al., 2017) comes from the computation of self-attention. That is, with $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d}$, where each token or point u in the point cloud is associated with a row $\boldsymbol{q}_u, \boldsymbol{k}_u, \boldsymbol{v}_u$ in these matrices, and $\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \exp(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$. Here, $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ is of size $n \times n$, and we omit the normalization terms for simplicity. Viewing the attention as a kernel $\exp(\boldsymbol{x}^\top \boldsymbol{y})$, several methods have been proposed to approximate it for efficiency. Many of these methods are RFF-based (Peng et al., 2021; Choromanski et al., 2021; Luo et al., 2021; Choromanski et al., 2023) or LSH-based (Kitaev et al., 2020; Daras et al., 2020; Zandieh et al., 2023; Han et al., 2024). For example, RFFs can be utilized to approximate $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \widehat{\psi}(\boldsymbol{x})^\top \widehat{\psi}(\boldsymbol{y})$, with, e.g., $\widehat{\psi}(\boldsymbol{x}) = \exp(\frac{\|\boldsymbol{x}\|^2}{2}) \psi(\boldsymbol{x})$ (Peng et al., 2021), reducing the complexity to $\mathcal{O}(n)$. As for LSH-based methods, e.g., Reformer (Kitaev et al., 2020) equalizes query and key vectors and sets their norms to be 1, enabling the use of angular distance-based LSH (Andoni et al., 2015) to efficiently find large entries in the attention matrix $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ as its approximation, resulting in $\mathcal{O}(n \log n)$ complexity.

Notation. Later, we use $\tilde{\mathcal{O}}$, $\tilde{\Theta}$, and $\tilde{o}$ denote soft-$\mathcal{O}$, soft-Θ, and soft-o , respectively. They are variants of Big-$\mathcal{O}$, Big-Θ, and Little-o that suppress polylogarithmic factors.

3. Error-Computation Analysis for RFF/LSH

One of the key steps of designing efficient transformers relies on effective kernel approximation. So, this section aims to analyze the tradeoff between the approximation error (ϵ) and computational complexity (F) of both RFF- and LSH-based methods in point cloud systems. Our goal is to enable direct comparisons between RFF-based and LSH-based methods for GDL tasks, where local inductive bias holds, seeking to provide theoretical guidance for the design of efficient transformers to be discussed in Sec. 4. To summarize, we achieve the following insights: Let ϵ denote the squared error of the attention weight approximation averaged over all point pairs in a system and F denote the total number of floating point operations (FLOPs).

1. RFF results in an error $\epsilon = \tilde{\Theta}(\frac{n}{F})$, which is consistently worse than LSH under subquadratic complexity, i.e., when $F = \tilde{o}(n^2)$.
2. LSH is better suited for tasks with local inductive bias, yielding $\epsilon = \tilde{\Theta}(\frac{1}{n})$ via OR-only LSH. However, OR-only LSH finds it hard to further reduce such error if F is set to be almost linear, i.e., $F = \tilde{\mathcal{O}}(n)$.
3. Utilizing both OR & AND LSH significantly improves performance. The error $\epsilon = \tilde{\mathcal{O}}(\exp(-\frac{F}{n\mathrm{polylog}(n)})\frac{1}{n})$, which means that ϵ can be further exponentially re-

duced by almost linear complexity $F = \tilde{\mathcal{O}}(n)$.

Practitioners primarily interested in the architecture implementation of HEPT may choose to skip the rest of this section, and check Sec. 4 directly.

3.1. Characterizing Local Inductive Bias

The following notions aim to formally characterize the local inductive bias of a point cloud system of interest.

Definition 3.1 (Bounded-Support Kernels)**.** Consider a properly normalized shift-invariant kernel defined as $k_s(\boldsymbol{x}, \boldsymbol{y}) = k_s(\boldsymbol{x} - \boldsymbol{y})$, where $k_s(\boldsymbol{x}, \boldsymbol{y}) \in [0, 1]$, $s > 0$ and $k_s(\boldsymbol{0}) = 1$. This kernel exhibits bounded support, i.e., $k_s(\boldsymbol{x} - \boldsymbol{y}) = 0$ for $\|\boldsymbol{x} - \boldsymbol{y}\|_2 > s$. For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the computational complexity of $k_s(\boldsymbol{x}, \boldsymbol{y})$ is linear in d.

Assumption 3.2 (Local Inductive Bias)**.** Consider a bounded point cloud system $\mathcal{C}$ with n points located at $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ in a d-dim unit ball, i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\|\boldsymbol{x}_i\|_2 \leq 1$. Denote the empirical distribution of the point-pair distances as $\phi(z) = \frac{1}{n(n-1)} \sum_{i,j \in [n], i \neq j} \delta_{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2}(z)$ where $\delta_a(\cdot)$ is 1-dim dirac delta function. $\mathcal{C}$ is said to hold local inductive bias if the ground-truth function for the learning task over $\mathcal{C}$ can be approximated by a transformer with full attention matrices whose attention weights can be represented as a bounded-support kernel k_s between point locations $\boldsymbol{x}_i$'s, where the bound s satisfies $\int_0^s \phi(z)dz \sim \tilde{\mathcal{O}}(\frac{1}{n})$.

Intuitively, local inductive bias assumes that in a point cloud system, a point primarily interacts with its local neighborhood, where the number of points each point interacts with is on average at most $\mathcal{O}(\text{polylog}(n))$. This assumption means that the optimal full attention matrix has at most $\mathcal{O}(n \cdot \text{polylog}(n))$ non-zeros, which gives the foundation to build efficient transformers with almost linear complexity. The challenge lies in how to identify those non-zeros using near-linear complexity and regular operations.

Note that the above-assumed kernel k_s for characterizing local inductive bias can be viewed as an inherent property of the point cloud system and the learning task, which may not necessarily follow the common implementation of attention kernel such as $\exp(F(\boldsymbol{x})^\top G(\boldsymbol{y}))$ with some parameterized functions F, G. Although the conventional kernel $\exp(\boldsymbol{x}^\top \boldsymbol{y})$ is not strictly with bounded support, with the functions F, G, practical attention weights $\exp(F(\boldsymbol{x})^\top G(\boldsymbol{y}))$ still hold the potential of approximating a bounded support kernel and yield reasonable performance. That having been said, as shown in our experiments, an attention kernel that directly models local inductive bias (see Sec. 4.1) often yields better performance for the tasks where local inductive bias indeed exists.

How large could s be in practice? Suppose points are almost uniformly allocated in the d-dim unit ball, and then, $\int_0^s \phi(z)dz = \Theta(s^d)$. In this case, local inductive bias means the point pairs within $s = \tilde{\mathcal{O}}(\frac{1}{n^{1/d}})$ distance hold positive attention weights.

3.2. Error-Computation Tradeoff

RFF. We instantiate our analysis of RFF based on a widely used feature map $\psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ as defined in Sec. 2, where the complexity F is proportional to the feature dimension D. The following theorem indicates that RFF can hardly reduce the error to $\frac{1}{n}$ when F is sub-quadratic in n.

Theorem 3.3 ($\epsilon - F$ Tradeoff of RFF)**.** *Assume $k_s(\boldsymbol{x}, \boldsymbol{y})$ is positive definite. If approximating it by RFF $\psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ in point cloud systems described in Assumption 3.2, the error $\epsilon = \Theta(\frac{nd}{F})$.*

OR-only LSH. Since many previous works use OR-only LSH, we are to first analyze the approximation error in such a setting. Note that F is proportional to the number of hash tables m_1 in this setting and the latter OR & AND LSH setting. We base our analysis on E^{2}LSH with r as the bucket size defined in Sec. 2, while the analysis can be similarly extended to other types of hash functions. To achieve the next theorem, we need a further assumption that is satisfied as long as the point allocation $\phi(z)$ is not concentrated at $z = a$ for some particular $a \in [0, 2)$.

Theorem 3.4 ($\epsilon - F$ Tradeoff of OR-only E^{2}LSH)**.** *Assume there exists r such that $\int_0^r \phi(z)dz \leq c_1 r$ and $\int_r^\infty \frac{1}{z}\phi(z)dz \leq c_2$ for some positive constants c_1 and c_2. The OR-only E^{2}LSH may achieve $\epsilon = \tilde{\Theta}(\exp\left(-\frac{c_3 F}{dn^2 s}\right)\frac{1}{n})$ where c_3 is a positive constant depending on c_1 and c_2.*

Putting Theorem 3.3 and 3.4 together, clearly, OR-only LSH can outperform RFF when $F = \tilde{o}(n^2)$, indicating that LSH is always preferable for subquadratic complexity given point cloud systems with local inductive bias. This is attributed to the fact that LSH tends to zero out kernel values with a high probability for distant pairs.

OR & AND LSH. OR-only LSH's error dependency on $\exp(-\frac{c_3 F}{dn^2 s})$ shows that to further effectively reduce the error $\tilde{\Theta}(\frac{1}{n})$, F has to be in the order of $dn^2 s$. However, as s could be much larger than n^{-1} in practice (see the discussion in Sec. 3.1), this asks for F being super-linear in n. The issue, due to our analysis, is caused by many distant point pairs being mapped to the same hash bucket if one uses OR-only LSH, which motivates us to inspect the use of OR & AND LSH.

Theorem 3.5 ($\epsilon - F$ Tradeoff of OR & AND E^{2}LSH)**.** *Suppose each hash table contains m hash functions. Assume there exists m such that $\int_0^r \phi(z)dz = \tilde{\mathcal{O}}(\frac{1}{n})$ and $\int_r^\infty \phi(z)\frac{r^m}{z^m}dz \leq \int_0^r (\sqrt{2\pi} - \frac{z}{r})^m \phi(z)dz$, where $r = ms$. By choosing such r as the bucket size, the OR & AND E^{2}LSH may achieve $\epsilon = \tilde{\mathcal{O}}(\exp(-\frac{c_4 F}{dn(\text{polylog}(n)+m)})\frac{1}{n})$.*

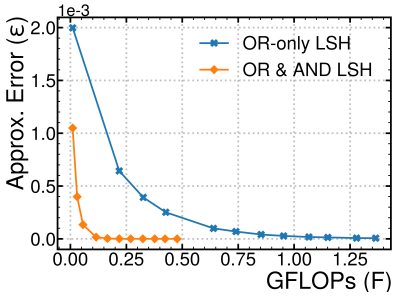


Figure 2: The error-computation tradeoff from numerical experiments. OR & AND LSH decreases the error exponentially with near-linear complexity, validating our analysis.

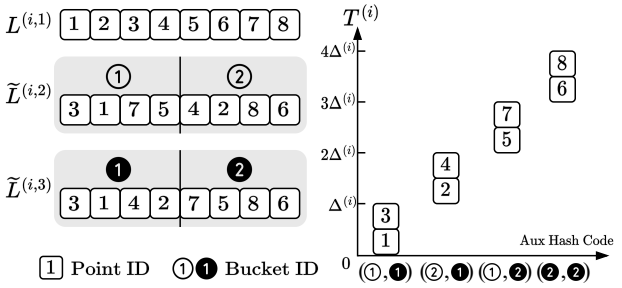


Figure 3: The above shows how to obtain AND hash code $T^{(i)}$ with $m_2 = 3, B_{ij} = 2$. Points are assumed to be pre-sorted based on their raw hash values with $\min(L^{(i,1)}) = 0$.

Note that if we consider systems with almost uniformly allocated points, there exists $m \leq d$ satisfying the assumption. This theorem shows that with OR & AND LSH combined, $F \sim nd(\text{polylog}(n) + d)$ is sufficient to reduce the error exponentially, necessitating the use of OR & AND LSH.

3.3. Numerical Experiments

To further validate the effectiveness of OR & AND LSH as proved in our theoretical analysis, we conducted additional numerical experiments, and the results are depicted in Fig. 2.

In this numerical study, we generate $n = 30,000$ points uniformly distributed across a 2D square with a side length of 10. To model local inductive bias, each point interacts only with its 64 nearest neighbors, approximating a ground-truth kernel value of $\exp\left(-\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|^2\right)$ (our theoretical results are not limited to this kernel). Points beyond this neighborhood have a kernel value of 0. Additional details are provided in Appendix C.3. In this study, since n^2 is roughly of the same magnitude as 1 GFLOP ($1e9$ FLOPs), Fig. 2 reveals that OR-only LSH can only effectively reduce the error when the computational budget is on the order of n^2. Conversely, OR & AND LSH achieves exponential error reduction with substantially fewer FLOPs, demonstrating its superior efficiency and accuracy.

4. HEPT Architecture

Motivated by our theoretical insights, we propose HEPT in this section, which is illustrated in Fig 1. We will first introduce the attention kernel considered, and then describe an approach for approximating it with OR & AND LSH. Lastly, we present a way to ensure computational regularity without compromising approximation accuracy.

4.1. Kernel with Explicit Local Inductive Bias

Given a query-key pair $(\boldsymbol{q}_u, \boldsymbol{k}_v)$, we propose to use the following kernel for attention computation: $k(\boldsymbol{q}_u, \boldsymbol{k}_v) = \exp(-\frac{1}{2}\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2)$, where $\boldsymbol{q}_u = [\widetilde{\boldsymbol{q}}_u \| \sqrt{2\omega}\boldsymbol{\rho}_u]$ and $\boldsymbol{k}_v = [\widetilde{\boldsymbol{k}}_v \| \sqrt{2\omega}\boldsymbol{\rho}_v]$ are concatenated from the original transformer's queries and keys $\widetilde{\boldsymbol{q}}_u, \widetilde{\boldsymbol{k}}_v \in \mathbb{R}^d$ with point coordinates $\boldsymbol{\rho}_u, \boldsymbol{\rho}_v \in \mathbb{R}^{k_2}$ and learnable parameters $\omega \in \mathbb{R}^+$. The full attention mechanism is then $\text{Attn}(\boldsymbol{A}, \boldsymbol{V}) = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V}$, with $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ comprising elements $\boldsymbol{A}_{uv} = k(\boldsymbol{q}_u, \boldsymbol{k}_v)$, and $\boldsymbol{D} = \text{diag}(\boldsymbol{A}\boldsymbol{1})$ for normalization, where $\boldsymbol{1}$ represents an all-one vector. This kernel enables the use of E^{2}LSH (or RFF) for approximation and allows for explicit modeling of local inductive bias: the attention score $k(\boldsymbol{q}_u, \boldsymbol{k}_v) \to 0$ as $\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2$ increases.

Note that HEPT can also support efficient computation of the conventional attention kernel $\exp(\boldsymbol{q}_u^T\boldsymbol{k}_v)$ by transforming it into $\exp(-\frac{1}{2}\|F(\boldsymbol{q}_u) - G(\boldsymbol{k}_v)\|^2)$ for some functions F, G (Shrivastava & Li, 2014; Daras et al., 2020). However, this kernel does not work well for the HEP tasks in this work, due to its failure in explicitly modeling local inductive bias.

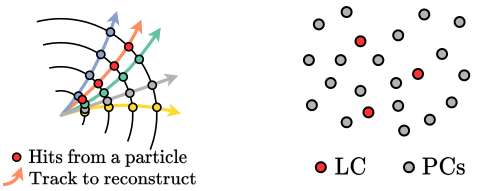
4.2. OR & AND LSH for Attention Computation

As shown in Sec. 3, to effectively approximate kernels with local inductive bias, it is critical to utilize OR & AND LSH for near-linear complexity with guaranteed low approximation errors. Therefore, we propose an architecture that integrates OR & AND LSH for attention computation.

Specifically, we first construct m_1 hash tables (OR LSH), each with m_2 hash functions (AND LSH) for each query and key. Each hash function is E^{2}LSH without bucketization, i.e., $h_{\boldsymbol{a}}(\boldsymbol{x}) = \boldsymbol{a}\cdot\boldsymbol{x}$. Consequently, each query $\boldsymbol{q}_u$ or key $\boldsymbol{k}_v$ yields $m_1 \times m_2$ raw hash values, denoted as $L^{(ij)}_{\boldsymbol{q}_u}, L^{(ij)}_{\boldsymbol{k}_v} \in \mathbb{R}$ for $i \in [m_1]$ and $j \in [m_2]$, respectively. Due to the property of E^{2}LSH, if $\boldsymbol{q}_u$ and $\boldsymbol{k}_v$ hold small $\|\boldsymbol{q}_u - \boldsymbol{k}_v\|_2$, they are likely to have close hash values $L^{(ij)}_{\boldsymbol{q}_u}$ and $L^{(ij)}_{\boldsymbol{k}_v}$.

For each query/key, uniformly denoted as $\boldsymbol{z}$, in the i^{th} hash table, our goal is to combine the m_2 raw hash values $L^{(ij)}_{\boldsymbol{z}}$ into a single ***AND hash code*** $T^{(i)}_{\boldsymbol{z}} \in \mathbb{R}$ such that query/key pairs with close $|T^{(i)}_{\boldsymbol{q}_u} - T^{(i)}_{\boldsymbol{k}_v}|$ must have close $|L^{(ij)}_{\boldsymbol{q}_u} - L^{(ij)}_{\boldsymbol{k}_v}|$ for all $j \in [m_2]$, i.e., an AND operation. Then, attention can be computed using the resulting AND hash codes $T^{(i)}_{\boldsymbol{z}}$'s from those m_1 hash tables. The details are as follows.

Obtaining AND Hash Code $T^{(i)}_{\boldsymbol{z}}$. For each query/key $\boldsymbol{z}$,



(a) Charged Particle Tracking (b) Pileup Mitigation

Figure 4: Illustrations of the two HEP tasks.

we keep $L_{\boldsymbol{z}}^{(i1)}$ as it is as a real-valued 1D ***base hash code***, and bucketize each $L_{\boldsymbol{z}}^{(ij)}$ for $j \geq 2$, which leads to an $(m_2 - 1)$-sized tuple of integer bucket indices, named as ***aux hash code***. As illustrated in Fig 3, we compute the AND hash code by 1) allocating $\boldsymbol{z}$'s only with the same aux hash code to a unique range in $\mathbb{R}$, and 2) within each allocated range, positioning different $\boldsymbol{z}$'s according to their real-valued base hash codes. Specifically, to obtain the aux hash code, given i, j and the number of desired buckets B_{ij}, sort $L_{\boldsymbol{z}}^{(ij)}$ for all queries and keys (thus $2n$ in total). Then, starting from 1, every $\lfloor \frac{2n}{B_{ij}} \rfloor$ consecutive points receives a bucket index denoted as $\widetilde{L}_{\boldsymbol{z}}^{(ij)} \in \{1, ..., B_{ij}\}$. We use $\widetilde{L}_{\boldsymbol{z}}^{(ij)}$ as the aux hash code. Then, the AND hash code $T_{\boldsymbol{z}}^{(i)}$ can be computed as follows: Let $\Delta^{(i)} = \max(L^{(i1)}) - \min(L^{(i1)})$,

$$T_{\boldsymbol{z}}^{(i)} = L_{\boldsymbol{z}}^{(i1)} + \Delta^{(i)} \sum_{j=2}^{m_2} \left[(\widetilde{L}_{\boldsymbol{z}}^{(ij)} - 1) \prod_{j'=2}^{j-1} B_{ij'} \right].$$

This way of computation makes sure that points with the same aux hash codes are assigned to adjacent ranges. Note that to avoid the boundary effect of hash bucketing (two close points being put into two consecutive buckets consistently), we may shift the number of desired buckets B_{ij} for different i, j while guaranteeing the total number of buckets $\prod_{j=2}^{m_2} B_{ij}$ unchanged, which effectively shifts the bucket boundaries, similar to the random shifts in original E^{2}LSH.

Merging m_1 OR LSH Results. By repeating the above steps m_1 times, we yield $T_{\boldsymbol{q}_u}^{(i)}, T_{\boldsymbol{k}_v}^{(i)}$ for every query and key with $i \in [m_1]$, which are used to compute m_1 many sparse attention matrices $\boldsymbol{A}^{(i)}$ with regular non-zero blocks (this process will be elaborated in the next Sec. 4.3). The final embedding can be computed as $\boldsymbol{E} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{n \times d}$, where $\boldsymbol{A}\boldsymbol{V}$ is computed via the sum of m_1 regular block matrix multiplications $\sum_i \boldsymbol{A}^{(i)}\boldsymbol{V}$, and $\boldsymbol{D}$ is computed similarly via $\mathrm{diag}(\sum_i \boldsymbol{A}^{(i)}\mathbf{1})$.

4.3. Regular Computation with Query-Key Alignment

Here, we elaborate the way to compute sparse attention matrix $\boldsymbol{A}^{(i)}$ with regular non-zero blocks via regular operations. A naive way to compute $\boldsymbol{A}^{(i)}$ is to grouping queries and keys with similar $T_{\boldsymbol{q}_u}^{(i)}, T_{\boldsymbol{k}_v}^{(i)}$ into buckets and compute their attention. However, this is inefficient because of the potentially non-uniform allocation between queries' and keys' hash codes. The numbers of queries and the numbers of keys may be very different in the same buckets and may shift significantly across the buckets, of which the attention computation needs significantly irregular computations.

To introduce regularity, we separately process queries and keys. We partition queries into equal-sized buckets by truncating their AND hash codes $T_{\boldsymbol{q}_u}^{(i)}$ for $u \in [n]$, and partition the keys in the same way. Then, we compute the attention between queries and keys with the same bucket index, which essentially computes a block-diagonal attention matrix and is highly regular and hardware-friendly.

However, in practice, we observe that the above way to bucketize queries and keys separately may introduce a misalignment issue between queries and keys and miss those query-key pairs with large attention values, since it may include queries and keys with rather different hash codes in the same bucket, as illustrated with an example in Fig. 1. Reformer (Kitaev et al., 2020) circumvents this issue by tying queries and keys, i.e., $\boldsymbol{q}_u = \boldsymbol{k}_u$, but this limits its modeling capacity. We address this challenge by integrating point coordinates as extra AND hash codes, detailed as follows.

Point Coordinates as Extra AND Hash Codes. In GDL tasks with point cloud data, we may leverage spatial proximity to align query buckets and key buckets. Specifically, we can obtain d' additional AND hash values based on the coordinates of point u, $\boldsymbol{\rho}_u \in \mathbb{R}^{k_2}$ (typically $d' \leq k_2$) for aux hash code computation. These hash values are shared by both queries and keys, i.e., $L_{\boldsymbol{q}_u}^{(i(m_2+\ell))} = L_{\boldsymbol{k}_u}^{(i(m_2+\ell))} = h_{\boldsymbol{a}_\ell}(\boldsymbol{\rho}_u)(= \boldsymbol{a}_\ell \cdot \boldsymbol{\rho}_u) \in \mathbb{R}$, $\ell = 1, 2, ..., d'$. Subsequently, these hash values go through the same procedure to be combined into the AND hash codes as discussed in Sec. 4.2. This process is essentially equivalent to partitioning the input space into various distinct, non-overlapping regions randomly, and guarantees that even if the query and the key hash codes are processed separately, only the attention between point pairs u, v that are close in the geometric space is computed, which well addresses the misalignment issue.

5. Related Work

In this section, we review the most relevant work on efficient transformers and discuss their existing issues. More related work can be found in Appendix B.

Neglecting Error-Computation Tradeoff. FLT (Choromanski et al., 2023) models local inductive bias utilizing RFF for GDL tasks with tens of points and overlooks the bad error-computation tradeoff of RFF. Those works using LSH, Reformer (Kitaev et al., 2020) and Smyrf (Daras et al., 2020) consider OR-only LSH and neglect AND LSH, rendering non-neglectable error for large n; KDEformer (Zandieh et al., 2023) and HyperAttention (Han et al., 2024) em-

ploy AND-only LSH, which often does not work well in practice. Furthermore, to employ angular-distance-based LSH functions, Reformer normalizes their queries and keys, limiting the model's expressiveness. KDEformer and HyperAttention avoid using normalized inputs, but using angular-distance-based LSH for maximum inner product search requires strong assumptions on the alignment between query-key angles and inner products.

Compromised Accuracy for Computational Regularity. Reformer sorts and truncates hash buckets evenly for computational regularity, which does not guarantee that consecutive buckets in a hash table correspond to geometrically neighboring areas. To mitigate this issue, Smyrf, KDEformer, and HyperAttention utilize either E^2LSH (Datar et al., 2004) or hyperplane LSH (Charikar, 2002), ensuring that query/key pairs located in adjacent buckets are geometrically close. However, these methods truncate queries and keys into equal-sized buckets separately, and neglect the query-key misalignment issue, as explained in Sec. 4.3. Both Flatformer (Liu et al., 2023) and DSVT (Wang et al., 2023) from CV propose to project 3D points onto the x and y axes rather than using LSH functions, and attention is computed by grouping points into equal-sized blocks along each axis. Such fixed projection directions may not be suitable for scientific problems with complex geometry. Moreover, some of these methods rely on domain-specific techniques, such as voxelization (Wang et al., 2023), which presents challenges in applications to general point-cloud data.

6. Experiments

We evaluate HEPT for both predictive accuracy and computational performance against a variety of efficient transformers and GNNs on two critical tasks in HEP. In the following, we introduce our datasets, baselines, and experiment settings, and more details can be found in Appendix A and C.

6.1. Datasets

Tracking-6k & Tracking-60k. We use two datasets derived from the TrackML Particle Tracking Dataset (Amrouche et al., 2020) designed for evaluating algorithms that reconstruct charged particle tracks, a crucial task in HEP that requires real-time processing. During collision events, as charged particles pass through tracking detectors, they leave a trail of hits, each recorded with geometric coordinates and additional properties (e.g., momentum). The hits from a single collision event collectively form an attributed point cloud, as illustrated in Fig. 4a. The task is to identify which hits are left by the same particle and group them accordingly for track reconstruction. The current pipeline for this task is time-consuming, representing about 45% of the total collider data reconstruction time (CMS Group, 2022). Thus, accurate and efficient methods for this task are in great demand. ML methods can be used to learn hit (point) embeddings such that the hits originating from the same particle are nearby in the embedding space for downstream clustering and track identification. Differing in scale, Tracking-6k comprises point clouds with about 6,000 points each, while Tracking-60k presents a more challenging scenario with each cloud containing about 60,000 points.

Pileup-10k. This dataset, similar to that in Martínez et al. (2019); Li et al. (2023), is for the task of pileup mitigation, a critical data-denoising step in HEP. Each point cloud within the dataset represents an event resulting from multiple simultaneous proton-proton collisions at the LHC. The individual points in these clouds correspond to the particles generated from the collisions, either from the leading collision (LC) or simultaneous pileup collisions (PCs). The goal of this task is to classify whether each particle originates from the LC or PCs, as illustrated in Fig. 4b, which is a point classification task. There are 1000 point clouds in this dataset, each with about 10,000 points.

6.2. Baselines and Setup

Efficient Transformer Baselines. We evaluate eight efficient transformers from both NLP and CV domains as our baselines. These include the LSH-based Reformer (Kitaev et al., 2020), Smyrf (Daras et al., 2020), and HyperAttn (Han et al., 2024); RFF-based Performer (Choromanski et al., 2021) and FLT (Choromanski et al., 2023); and ScatterBrain (Chen et al., 2021), which integrates both RFF and LSH approaches. From CV, we include Point Transformer (Zhao et al., 2021) and FlatFormer (Liu et al., 2023).

GNN Baselines. Besides collecting results from current SOTA GNNs for the two tasks (Lieret et al., 2023; Li et al., 2023), we use GCN (Kipf & Welling, 2017) as a baseline and further benchmark three GNNs that have been widely used in scientific applications, including GatedGNN (Li et al., 2016; 2023), DGCNN (Wang et al., 2019; Qu & Gouskos, 2020), and GravNet (Qasim et al., 2019).

Random Baselines. The random baselines for the Tracking datasets are implemented by randomly initializing HEPT models without any training. For the Pileup dataset, the random baseline is obtained by randomly assigning the output class probability for each point.

Metrics. For the tracking datasets, the quality of the learned point embeddings is assessed by evaluating how closely the embeddings of hits from the same particle cluster together. Specifically, we use the metric $\mathrm{AP@}\,k = \frac{1}{n}\sum_{u=1}^{n} \mathrm{Prec@}\,k_u$, where k_u represents the number of hits originating from the same particle as hit u. $\mathrm{Prec@}\,k_u$ calculates the precision by retrieving the closest k_u neighbors of hit u in the embedding space. For the pileup dataset, the area under the precision-recall curve (AUC) is employed for

Table 1: Predictive performance on the three datasets. The **Bold**†, **Bold**‡, and **Bold** highlight the first, second, and third best results, respectively. Underline indicates the best transformer baselines.

	Tracking-6k (AP@k)	Tracking-60k (AP@k)	Pileup-10k (AUC)
Random	5.88	5.71	4.22
SOTA GNNs	**91.00**‡	**90.89**‡	**40.26**
Reformer	72.37	72.47	36.70
SMYRF	72.98	71.18	25.20
HyperAttn	71.49	70.22	25.31
Performer	73.17	72.07	28.36
FLT	72.55	71.45	25.26
ScatterBrain	73.35	72.06	30.95
PointTrans	72.33	70.81	40.26
FlatFormer	74.22	70.23	38.61
GCN	79.61	75.38	40.10
DGCNN	**90.74**	**88.66**	33.75
GravNet	90.11	87.99	40.10
GatedGNN	80.98	78.42	40.26
Performer-k_{HEPT}	71.97	69.20	32.81
SMYRF-k_{HEPT}	83.19	71.04	**40.31**‡
FlatFormer-k_{HEPT}	88.18	85.06	39.99
HEPT	**92.66**†	**91.93**†	**40.39**†

this imbalanced binary classification task.

Setup. The transformer baselines are implemented using well-established codebases (Idiap, 2023; Wang, 2023b; Dao & Chen, 2023) or author-provided code (Liu et al., 2023), while GNNs use the implementation from PyG (Fey & Lenssen, 2019). For the results collected from SOTA GNNs, provided model checkpoints (Lieret et al., 2023) are used for evaluation on the Tracking datasets; for the Pileup dataset, the SOTA GNN is trained from scratch using available open-source code in (Li et al., 2023). If not specified, point coordinates are used as the positional encoding for the transformer baselines following Liu et al. (2023); Wang et al. (2023). All models are ensured to have a similar number of trainable parameters and then the FLOPs used are aligned if possible. All models are trained and evaluated with the same seed to ensure reproducibility, using a server with NVIDIA Quadro RTX 6000 GPUs and Intel Xeon Gold 6248R CPUs. Note that all computations were performed on the GPUs, including the construction of k-nn graphs required by some baselines, where the API from PyG (Fey & Lenssen, 2019) was used with k being 64 similar to previous works (Lieret et al., 2023; Li et al., 2023).

Hyperparameter Tuning. The hyperparameters for the baselines and HEPT are tuned with similar budgets, based on performance in the validation set of each dataset. For HEPT, we adopt $m_1 = 3$ hash tables, each with $m_2 = 3$ hash functions for the three datasets. The block size of attention computation is set to 100, and we use only point coordinates without point hidden representations as the AND hash inputs, i.e., $L_{\boldsymbol{q}_u}^{(i(1+\ell))} = L_{\boldsymbol{k}_u}^{(i(1+\ell))} = h_{\boldsymbol{a}_\ell}(\boldsymbol{\rho}_u)(= \boldsymbol{a}_\ell \cdot \boldsymbol{\rho}_u)$, for $\ell = 1, 2$, where note that in HEP, the points are in a 2-d $\eta - \phi$ space (Thais et al., 2022), as detailed in Appendix A.2. We set a fixed total number of buckets $\prod_{j=2}^{m_2} B_{ij}$ and generate different bucket sizes $\{B_{ij}\}$ randomly to mitigate the boundary effect. See Appendix C for detailed settings.

Table 2: Training and test time (ms) per sample. Each entry is the median from at least 100 measurements evaluated on an NVIDIA Quatro RTX 6000. Numbers in $(\cdot)$ are the time used to pre-construct input graphs that may be saved during training if pre-processing is allowed. Note that real-time inference requires building graphs on the fly. The **Bold**† highlights the best results, and **Bold** and Underline indicate the best transformer and GNN baselines, respectively.

	Tracking-6k		Tracking-60k		Pilup-10k	
	Train	Test	Train	Test	Train	Test
SOTA GNNs	559	221	OOM	5781	432(322)	362
Reformer	355	23.1	2570	251	83.3	23.4
SMYRF	348	8.7	2343	69.6	58.6	12.4
HyperAttn	352	8.4	2320	62.1	44.4	12.5
Performer	343	**8.3**	2407	68.7	52.7	12.8
FLT	341	8.4	2369	71.6	55.9	12.7
ScatterBrain	357	13.1	2562	129	109	34.6
PointTrans	476(130)	144	7361(5017)	5143	372(323)	348
FlatFormer	**338**†	**8.3**	**2261**†	**58.7**	**53.7**	**12.2**
GCN	471(129)	138	7332(5009)	5123	376(322)	342
DGCNN	563	287	14098	11779	325	294
GravNet	593	251	13597	11684	312	278
GatedGNN	512(131)	158	7476(5013)	5263	432(328)	362
HEPT	**338**†	**7.0**†	**2312**	**57.9**†	**40.3**†	**10.7**†

Table 3: Ablation studies of HEPT.

	Tracking-60k
HEPT w/o k_{HEPT}	72.28
OR-only LSH	71.42
OR-only LSH*	78.22
OR & AND LSH	70.98
OR & AND LSH*	88.54

6.3. Result Analysis

Predictive Performance. As shown in Table 1, GNNs are suitable for GDL tasks with local inductive bias and have indeed achieved good prediction accuracy. However, HEPT still largely outperforms GNNs (with much lower computational complexity). When compared with other transformers, HEPT's performance gain is even more significant (up to 22%). To inspect the benefits of our proposed kernel k_{HEPT}, we also incorporate it into some transformer baselines when possible. These baselines also yield substantial improvements, validating the necessity of modeling local inductive bias explicitly for GDL tasks in HEP. Moreover, we observe that RFF-based methods Performer and FLT consistently exhibit unsatisfactory performance even with k_{HEPT}, which aligns with our analysis. As for LSH-based methods, SMYRF shows promise with k_{HEPT}, but it is unable to well generalize to the larger dataset Tracking-60k due to its OR-only LSH-based design and the neglect of query-key alignment. Similarly, FlatFormer also achieves good results when paired with our kernel. However, it still falls short of matching the SOTA GNNs and HEPT.

Computational Complexity. Table 2 compares both the

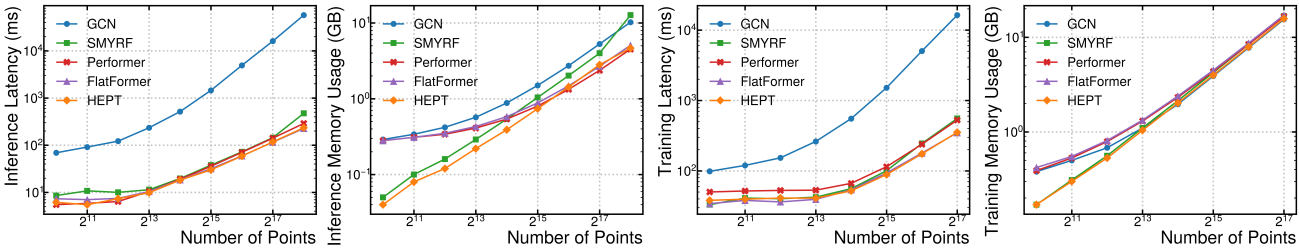


Figure 5: Inference and training costs per point cloud.

training (forward + backward pass) and inference (forward pass) time per sample for all models yielded from Table 1, and their FLOPs and GPU memory usage are reported in Table 7 in the appendix. Clearly, HEPT is among the most efficient transformers, and the gain in computational speed compared with GNNs is tremendous, especially for large point clouds. The speedup can be less significant in training for the Tracking datasets since the loss computation in this task dominates the running time in training (see Table 9). Specifically, in Tracking-60k, HEPT achieves an 89-203$\times$ speedup in inference and a 3-6$\times$ speedup in training, and in Pileup-10k, HEPT is 26-34$\times$ faster for inference and 8-11$\times$ faster for training, all while maintaining SOTA predictive accuracy. The slower performance of GNNs is due to the need for constructing graphs from point clouds and their irregular computation, while efficient transformers such as HEPT avoid graph construction and adopt only efficient regular computation. Moreover, HEPT can be further accelerated by applying such as Float16 computation and FlashAttention (Dao, 2024), which we leave as future studies.

Ablation Studies. We conduct ablation studies whose results are shown in Table 3. *First*, we evaluate the importance of our proposed kernel, where we replace our kernel with the kernel from the original transformer with absolute positional encoding. However, using the traditional kernel significantly reduces the performance of the model. *Second*, to show the effectiveness of OR & AND LSH in a general setting, we remove the use of point coordinates when obtaining aux hash codes. These codes are now computed only based on the query/key vectors. Since query-key alignment is critical, without using point coordinates, we follow (Kitaev et al., 2020) by tying query and key vectors, i.e., $\boldsymbol{q}_u = \boldsymbol{k}_u$ for alignment. The models with such query-key alignment are highlighted with *. As Table 3 shows, query-key alignment is important. With query-key alignment, the advantage of OR & AND LSH over OR-only LSH is obvious. HEPT by using OR & AND LSH and point coordinates for query-key alignment achieves the best performance.

6.4. Scalability Analysis

The considered tasks cover point clouds with 6k, 10k, and 60k points, offering a preliminary view of HEPT's scalability. To further examine scalability across a broader range of input sizes, we evaluate HEPT on point clouds ranging from 1k (2^{10}) to 262k (2^{18}) points. The results are presented in Fig. 5, where we also include a comparison with GCN and the three most efficient transformer baselines as indicated by Table 2, and the same settings used in Table 2 and Table 7 for the pileup mitigation task are employed. The input point clouds are generated randomly to meet the required number of points, and all models are closely matched in terms of FLOPs and trainable parameters (see Table 7). Fig. 5 indicates that HEPT is among the most scalable efficient transformers in terms of both latency and memory usage, even with input sizes extending from 2^{10} to 2^{18}.

Table 4: Performance of HEPT on Tracking-60k with different configurations of hash tables and bucket sizes. Results are reported as AP@k (GFLOPs), i.e., the numbers in parentheses represent the GFLOPs for each configuration.

# Hash Tables	Block Size		
	50	100	150
1	73.57 (24.2)	78.60 (29.8)	80.91 (35.4)
3	87.47 (35.6)	91.93 (52.2)	92.22 (68.9)
5	91.89 (46.9)	92.27 (74.7)	92.34 (102.6)

6.5. Sensitivity Analysis

Table 4 evaluates HEPT on Tracking-60k by varying the number of hash tables and block sizes, keeping other settings the same as those used in Table 1. Notably, configurations using a single hash table correspond to AND-only LSH, which generally performs poorly in practice and our results further verify this claim. For other configurations, HEPT demonstrates robustness, with increased computational budgets generally improving performance.

7. Conclusion

This work introduces HEPT, a new efficient transformer architecture for fast and accurate large-scale point cloud learning in scientific domains. Quantitative analysis on error-computation tradeoff shows the inherent limitations of RFF and the necessity of using OR & AND LSH to design efficient transformers for applications with local inductive bias. Two tasks in HEP have been used for evaluation, where HEPT greatly boosts computational speed and predictive accuracy against existing GNNs and transformers.

Acknowledgement

The authors thank Kilian Lieret, Gage DeZoort, and Yongbin Feng for their helpful discussions. S. Miao, M. Liu, and P. Li are partially supported by the National Science Foundation (NSF) award PHY-2117997 and IIS-2239565. J. Duarte is also supported by the NSF award PHY-2117997 and by the Department of Energy (DOE) award DE-SC0021187.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Aartsen, M., Ackermann, M., Adams, J., Aguilar, J., Ahlers, M., Ahrens, M., Altmann, D., Andeen, K., Anderson, T., Ansseau, I., et al. The icecube realtime alert system. *Astroparticle Physics*, 2017.

Abadal, S., Jain, A., Guirado, R., López-Alonso, J., and Alarcón, E. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys*, 2021.

Abbasi, R., Ackermann, M., Adams, J., Aggarwal, N., Aguilar, J., Ahlers, M., Ahrens, M., Alameddine, J., Alves, A., Amin, N., et al. Graph neural networks for low-energy event classification & reconstruction in icecube. *Journal of Instrumentation*, 2022.

Amrouche, S., Basara, L., Calafiura, P., Estrade, V., Farrell, S., Ferreira, D. R., Finnie, L., Finnie, N., Germain, C., Gligorov, V. V., et al. *The tracking machine learning challenge: accuracy phase*. Springer, 2020.

Amrouche, S., Basara, L., Calafiura, P., Emeliyanov, D., Estrade, V., Farrell, S., Germain, C., Gligorov, V. V., Golling, T., Gorbunov, S., et al. The tracking machine learning challenge: throughput phase. *Computing and Software for Big Science*, 2023.

Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. Practical and optimal lsh for angular distance. *Advances in Neural Information Processing Systems*, 2015.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 2017.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

Charikar, M. S. Similarity estimation techniques from rounding algorithms. *Symposium on Theory of Computing*, 2002.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 2021.

Chen, J., Gao, K., Li, G., and He, K. Nagphormer: A tokenized graph transformer for node classification in large graphs. *International Conference on Learning Representations*, 2022.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *International Conference on Learning Representations*, 2021.

Choromanski, K. M., Li, S., Likhosherstov, V., Dubey, K. A., Luo, S., He, D., Yang, Y., Sarlos, T., Weingarten, T., and Weller, A. Learning a fourier transform for linear relative positional encodings in transformers. *arXiv preprint arXiv:2302.01925*, 2023.

CMS Group. Detector Drawings. Technical report, 2012. URL https://cds.cern.ch/record/1433717.

CMS Group. CMS Phase-2 computing model: Update document by cms offline software and computing group. Technical report, 2022. URL https://cds.cern.ch/record/2815292.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *International Conference on Learning Representations*, 2024.

Dao, T. and Chen, B. Fly. https://github.com/HazyResearch/fly, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 2022.

Daras, G., Kitaev, N., Odena, A., and Dimakis, A. G. Smyrf-efficient attention using asymmetric clustering. *Advances in Neural Information Processing Systems*, 2020.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. *Symposium on Computational Geometry*, 2004.

De Favereau, J., Delaere, C., Demin, P., Giammanco, A., Lemaitre, V., Mertens, A., and Selvaggi, M. Delphes 3: a modular framework for fast simulation of a generic collider experiment. *Journal of High Energy Physics*, 2014.

DeZoort, G., Thais, S., Duarte, J., Razavimaleki, V., Atkinson, M., Ojalvo, I., Neubauer, M., and Elmer, P. Charged particle tracking via edge-classifying interaction networks. *Computing and Software for Big Science*, 2021.

Durrant, J. D. and McCammon, J. A. Molecular dynamics simulations and drug discovery. *BMC Biology*, 2011.

Fan, L., Pang, Z., Zhang, T., Wang, Y.-X., Zhao, H., Wang, F., Wang, N., and Zhang, Z. Embracing single stride 3d object detector with sparse transformer. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Fey, M. and Lenssen, J. E. Fast Graph Representation Learning with PyTorch Geometric. https://github.com/pyg-team/pytorch_geometric, 2019.

Gaillard, M. Cern data centre passes the 200-petabyte milestone. 2017.

Halzen, F. and Klein, S. R. Invited review article: Icecube: an instrument for neutrino astronomy. *Review of Scientific Instruments*, 2010.

Han, I., Jayaram, R., Karbasi, A., Mirrokni, V., Woodruff, D. P., and Zandieh, A. Hyperattention: Long-context attention in near-linear time. *International Conference on Learning Representations*, 2024.

Hashemi, M., Swersky, K., Smith, J., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., and Ranganathan, P. Learning memory access patterns. In *International Conference on Machine Learning*, 2018.

Idiap. Fast transformers. https://github.com/idiap/fast-transformers, 2023.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. *Symposium on Theory of Computing*, 1998.

Jang, B., Schaa, D., Mistry, P., and Kaeli, D. Exploiting memory access patterns to improve memory performance in data-parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 2010.

Jing, B., Eismann, S., Suriana, P., Townshend, R. J., and Dror, R. Learning from protein structure with geometric vector perceptrons. *International Conference on Learning Representations*, 2021.

Kansal, R., Duarte, J., Su, H., Orzari, B., Tomei, T., Pierini, M., Touranakou, M., Gunopulos, D., et al. Particle cloud generation with message passing generative adversarial networks. *Advances in Neural Information Processing Systems*, 2021.

Kansal, R., Li, A., Duarte, J., Chernyavskaya, N., Pierini, M., Orzari, B., and Tomei, T. Evaluating generative models in high energy physics. *Physical Review D*, 2023.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. *International Conference on Machine Learning*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *International Conference on Learning Representations*, 2020.

Langacker, P. *The standard model and beyond*. Taylor & Francis, 2017.

Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of massive data sets*. Cambridge University Press, 2020.

Li, T., Liu, S., Feng, Y., Paspalaki, G., Tran, N. V., Liu, M., and Li, P. Semi-supervised graph neural networks for pileup noise removal. *The European Physical Journal C*, 2023.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.

Lieret, K., DeZoort, G., Chatterjee, D., Park, J., Miao, S., and Li, P. High pileup particle tracking with object condensation. *International Connecting The Dots Workshop*, 2023.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. *IEEE/CVF International Conference on Computer Vision*, 2017.

Liu, Z., Yang, X., Tang, H., Yang, S., and Han, S. Flatformer: Flattened window attention for efficient point cloud transformer. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Luo, S., Li, S., Cai, T., He, D., Peng, D., Zheng, S., Ke, G., Wang, L., and Liu, T.-Y. Stable, fast and accurate: Kernelized attention with relative positional encoding. *Advances in Neural Information Processing Systems*, 2021.

Mao, J., Xue, Y., Niu, M., Bai, H., Feng, J., Liang, X., Xu, H., and Xu, C. Voxel transformer for 3d object detection. *IEEE/CVF International Conference on Computer Vision*, 2021.

Martínez, J. A., Cerri, O., Spiropulu, M., Vlimant, J., and Pierini, M. Pileup mitigation at the large hadron collider with graph neural networks. *The European Physical Journal Plus*, 2019.

Nelson, D., Springel, V., Pillepich, A., Rodriguez-Gomez, V., Torrey, P., Genel, S., Vogelsberger, M., Pakmor, R., Marinacci, F., Weinberger, R., et al. The illustristng simulations: public data release. *Computational Astrophysics and Cosmology*, 2019.

Oerter, R. *The theory of almost everything: The standard model, the unsung triumph of modern physics*. Penguin, 2006.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Pata, J., Wulff, E., Mokhtar, F., Southwick, D., Zhang, M., Girone, M., and Duarte, J. Improved particle-flow event reconstruction with scalable neural networks for current and future particle detectors. *arXiv e-prints*, 2023.

Peng, H., Pappas, N., Yogatama, D., Schwartz, R., Smith, N. A., and Kong, L. Random feature attention. *International Conference on Learning Representations*, 2021.

Qasim, S. R., Kieseler, J., Iiyama, Y., and Pierini, M. Learning representations of irregular particle-detector geometry with distance-weighted graph networks. *The European Physical Journal C*, 2019.

Qu, H. and Gouskos, L. Jet tagging via particle clouds. *Physical Review D*, 2020.

Qu, H., Li, C., and Qian, S. Particle transformer for jet tagging. *International Conference on Machine Learning*, 2022.

Radovic, A., Williams, M., Rousseau, D., Kagan, M., Bonacorsi, D., Himmel, A., Aurisano, A., Terao, K., and Wongjirad, T. Machine learning at the energy and intensity frontiers of particle physics. *Nature*, 2018.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 2007.

Rudin, W. *Fourier Analysis on Groups*. Wiley, 1991.

Satorras, V. G., Hoogeboom, E., and Welling, M. E (n) equivariant graph neural networks. *International Conference on Machine Learning*, 2021.

Seeger, M. Gaussian processes for machine learning. *International journal of neural systems*, 2004.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. *North American Chapter of the Association for Computational Linguistics*, 2018.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. Exphormer: Sparse transformers for graphs. *International Conference on Machine Learning*, 2023.

Shrivastava, A. and Li, P. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *Advances in Neural Information Processing Systems*, 2014.

Stärk, H., Ganea, O., Pattanaik, L., Barzilay, R., and Jaakkola, T. Equibind: Geometric deep learning for drug binding structure prediction. In *International Conference on Machine Learning*, 2022.

Strandlie, A. and Frühwirth, R. Track and vertex reconstruction: From classical to adaptive methods. *Reviews of Modern Physics*, 2010.

Sun, P., Tan, M., Wang, W., Liu, C., Xia, F., Leng, Z., and Anguelov, D. Swformer: Sparse window transformer for 3d object detection in point clouds. *European Conference on Computer Vision*, 2022.

Sun, Y., Dong, L., Huang, S., Ma, S., Xia, Y., Xue, J., Wang, J., and Wei, F. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Sutherland, D. J. and Schneider, J. On the error of random fourier features. *Uncertainty in Artificial Intelligence*, 2015.

Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. Sparse sinkhorn attention. *International Conference on Machine Learning*, 2020.

Thais, S. and Murnane, D. Equivariance is not all you need: Characterizing the utility of equivariant graph neural networks for particle physics tasks. *Knowledge and Logical Reasoning in the Era of Data-driven Learning Workshop at International Conference on Machine Learning*, 2023.

Thais, S., Calafiura, P., Chachamis, G., DeZoort, G., Duarte, J., Ganguly, S., Kagan, M., Murnane, D., Neubauer, M. S., and Terao, K. Graph neural networks in particle physics: Implementations, innovations, and challenges. *US Community Study on the Future of Particle Physics*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Wang, H., Shi, C., Shi, S., Lei, M., Wang, S., He, D., Schiele, B., and Wang, L. Dsvt: Dynamic sparse voxel transformer with rotated sets. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Wang, P. Performer-pytorch. https://github.com/lucidrains/performer-pytorch, 2023a.

Wang, P. Reformer-pytorch. https://github.com/lucidrains/reformer-pytorch, 2023b.

Wang, P.-S. Octformer: Octree-based transformers for 3D point clouds. *ACM Transactions on Graphics*, 2023c.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics*, 2019.

Wess, J. and Bagger, J. *Supersymmetry and supergravity*. Princeton University Press, 1992.

Wieschollek, P., Wang, O., Sorkine-Hornung, A., and Lensch, H. Efficient large-scale approximate nearest neighbor search on the gpu. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016.

Wu, K., Peng, H., Chen, M., Fu, J., and Chao, H. Rethinking and improving relative position encoding for vision transformer. *IEEE/CVF International Conference on Computer Vision*, 2021.

Wu, Q., Zhao, W., Li, Z., Wipf, D. P., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 2022.

Wu, X., Jiang, L., Wang, P.-S., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., and Zhao, H. Point transformer v3: Simpler, faster, stronger. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

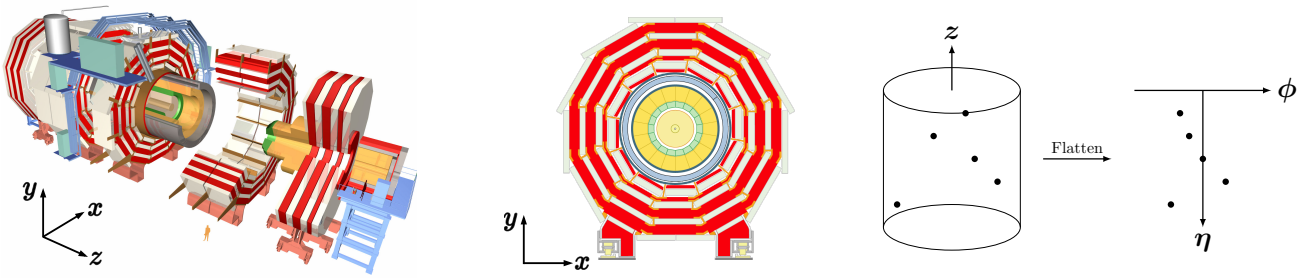
Xiong, Y., Zeng, Z., Chakraborty, R., Tan, M., Fung, G., Li, Y., and Singh, V. Nyströmformer: A nyström-based algorithm for approximating self-attention. *AAAI Conference on Artificial Intelligence*, 2021.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 2020.

Zandieh, A., Han, I., Daliri, M., and Karbasi, A. Kdeformer: Accelerating transformers via kernel density estimation. *International Conference on Machine Learning*, 2023.

Zhang, Z., Liu, Q., Hu, Q., and Lee, C.-K. Hierarchical graph transformer with adaptive node sampling. *Advances in Neural Information Processing Systems*, 2022.

Zhao, H., Jiang, L., Jia, J., Torr, P. H., and Koltun, V. Point transformer. *IEEE/CVF International Conference on Computer Vision*, 2021.



(a) 3D view of CMS detector (b) Transverse view of CMS detector (c) Converting to $\eta - \phi$ space

Figure 6: Visualizations of CMS detector and coordinate systems in HEP analysis, adapted from CMS Group (2012).

Table 5: Statistics of the three datasets.

	# Point Clouds	# Features in $\boldsymbol{X}$	# Dimensions in $\boldsymbol{\rho}$	Avg. # Points per Cloud	Avg. # Labeled Pairs per Cloud	Class Ratio (Pos./Neg.)
Tracking-6k	500	16	2	6.8k	3.7M	N/A
Tracking-60k	50	16	2	56.7k	75.5M	N/A
Pileup-10k	1000	8	2	10.3k	N/A	0.039

A. Details of the Datasets

A.1. Background

Tracking-6k & Tracking-60k. These two datasets are for charged particle tracking at the CERN LHC, which is a crucial task in HEP as it enables precise identification and reconstruction of charged particles' paths, facilitating the determination of their momentum, energy, etc. This capability is crucial for identifying particle types and reconstructing collision events, laying the foundation for precise measurements of particle properties such as mass and charge. These measurements are vital for testing Standard Model predictions and probing for new physics (Langacker, 2017). Additionally, accurate tracking is instrumental in suppressing background noise, distinguishing between signal events and the more common processes, thereby enhancing the detection of rare phenomena and contributing significantly to our understanding of fundamental particles and their interactions at high energies. The tracking process utilizes sophisticated detector systems, such as the inner detector of the ATLAS and CMS experiments, to reconstruct particle trajectories from collision events. However, challenges arise from the vast volume of data generated, background noise, and experimental complexities, necessitating robust yet efficient algorithms, e.g., the LHC operates at an extremely high collision rate, with millions of proton-proton collisions occurring every second to be analyzed. Traditional combinatorial-Kalman-filter-based track reconstruction (Strandlie & Frühwirth, 2010) cannot easily scale up to future LHC data and is difficult to parallelize on heterogeneous computing platforms. And the inherent complexity of GNNs renders it hard for their efficient deployment at the LHC. This study is performed using the TrackML dataset (Amrouche et al., 2023), which simulates the worst-case future LHC pileup conditions (200 interactions per proton bunch crossing) in a generic tracking detector geometry.

Pileup-10k. This dataset focuses on pileup mitigation, a critical challenge in analyzing data from the LHC, where multiple proton-proton collisions occur simultaneously within the same or nearby bunch crossings. These overlapping interactions, known as pileup collisions (PCs), complicate the extraction of meaningful data from the primary collision of interest. Effective pileup mitigation is essential for maintaining the physics sensitivity of LHC experiments, as it involves distinguishing and removing the contributions of noisy particles from PCs to isolate signals from the leading collision (LC), which is associated with the primary vertex having the highest sum of particle momentum. During the 2016 to 2018 LHC runs, the average pileup level was around 40, but this figure is anticipated to rise to as much as 200 in future runs (i.e., more noise and larger input sizes), significantly increasing the complexity of data analysis. The reconstruction of particles from LHC collisions relies on tracking detector hits and calorimeter energy deposits. While the SOTA tracking systems allow for tracking and vertexing of charged particles, enabling the straightforward identification and removal of those associated with PCs, the main challenge lies in dealing with neutral particles, such as photons and neutral hadrons, which do not leave tracks. To address these challenges, simulation samples based on the `DELPHES` framework (De Favereau et al., 2014) are utilized, generating both charged and neutral particles from selected physics processes alongside detector resolution effects, to develop and test pileup mitigation strategies.

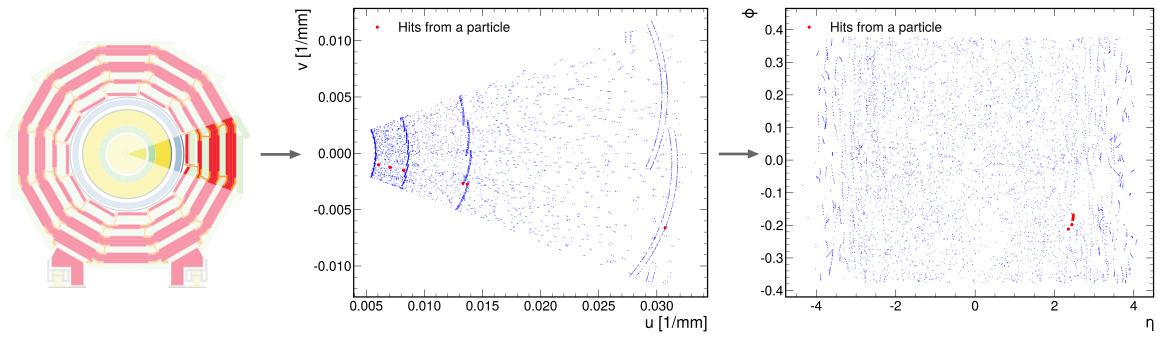


Figure 7: Visualization of a sample from the Tracking datasets, showcasing only the points collected from the detectors in the highlighted region for better illustration. The leftmost part of this figure is adapted from CMS Group (2012).

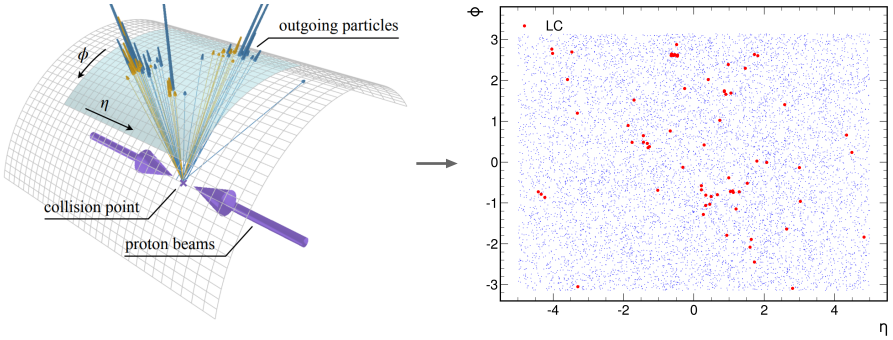


Figure 8: Visualization of a sample from the Pileup dataset. The left part of this figure is adapted from Qu et al. (2022).

A.2. Data Format

The statistics of the three datasets used are shown in Table 5, and Fig. 7 and Fig. 8 provide visualizations of the data samples. Below we introduce the features and labels included in the datasets.

Point Coordinates $\boldsymbol{\rho}$. Following the standard pipeline in HEP analysis, each point in the three datasets is associated with a 2-d coordinate in $\eta - \phi$ space (Thais et al., 2022). Fig. 6 visualizes the CMS detector at the LHC with its 3D and transverse view, where collisions occur at the center, surrounded by multiple layers of cylindrical detectors, and particles flying out from the center will hit the detectors at various locations. Imagine cutting the cylindrical detector along its length and flattening it out into a 2D plane, as illustrated in Fig. 6c. The vertical axis of this plane can represent the pseudorapidity (η), which indicates how far up or down (along the beam axis) the particle hit the detector. The horizontal axis represents the azimuthal angle (ϕ), indicating the particle's direction around the beam axis.

Point Features $\boldsymbol{X}$. Table 6 lists the variables included as point features in the two tasks and their definitions. Note that geometric features, e.g., η and ϕ, are also included as point features for model learning, which is a common practice in HEP and results in non-equivariant models with respect to these geometric features. We follow this practice in our implementation similar to previous works (Lieret et al., 2023; Li et al., 2023) and whether equivariant models are useful in HEP has not reached a consistent conclusion (Thais & Murnane, 2023).

Ground-Truth Labels. For the Tracking datasets, any pairs of hits (points) from the same particle are labeled as positive samples to be learned with similar embeddings, while for each hit its neighboring 256 hits from other particles are labeled as negative pairs. For the Pileup dataset, a particle (point) is labeled positive if it is from LC, and otherwise, it is labeled negative. Note that this task is highly imbalanced, and only about 3.9% of points are labeled positive.

Table 6: Point Features in the two tasks.

Task	Variable	Definition
Tracking	r	Radial distance from the beam axis in cylindrical coordinates.
	ϕ	Azimuthal angle around the beam axis in cylindrical coordinates.
	z	Longitudinal position along the beam axis.
	η	Pseudorapidity, measuring the angle of particle trajectory relative to the beam axis.
	u	Local coordinate axis in a detector plane, orthogonal to v.
	v	Local coordinate axis in a detector plane, orthogonal to u.
	charge_frac	Fraction of the charge collected by a sensor, indicating the quality of a hit.
	ℓ_η	Local pseudorapidity, calculated within a specific sub-detector region.
	ℓ_ϕ	Local azimuthal angle, measured within a specific sub-detector region.
	ℓ_x	Local x coordinate, representing position within a sub-detector.
	ℓ_y	Local y coordinate, representing position within a sub-detector.
	ℓ_z	Local z coordinate, representing position along the beam axis within a sub-detector.
	g_η	Global pseudorapidity, calculated with respect to the overall detector geometry.
	g_ϕ	Global azimuthal angle, measured with respect to the overall detector geometry.
Pileup	η	Pseudorapidity, a measure of the angle relative to the beam axis.
	ϕ	Azimuthal angle around the beam axis in cylindrical coordinates.
	p_x	Momentum component of the particle in the x direction.
	p_y	Momentum component of the particle in the y direction.
	p_t	Transverse momentum, calculated from the x and y momentum components.
	Rapidity	A measure of the particle's velocity in the direction of the beam.
	E	Energy of the particle.
	PID	Particle ID, indicating the type of the particle, e.g., muon, electron, etc.

A.3. Task Formulation

Below we describe how they are formulated as ML tasks.

Tracking-6k & Tracking-60k. To learn clustered embeddings for hits originating from the same particle, we adopt contrastive learning with InfoNCE loss (Oord et al., 2018). For any pairs of hits from the same particle, they are labeled as positive samples to be learned with similar embeddings, while for each hit its neighboring 256 hits from other particles are labeled as negative pairs. Therefore, with learned embeddings for a point u, denoted as $\boldsymbol{h}_u$, the loss is computed as

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp(\text{sim}(\boldsymbol{h}_u, \boldsymbol{h}_v^+))}{\exp(\text{sim}(\boldsymbol{h}_u, \boldsymbol{h}_v^+)) + \sum_{\boldsymbol{h}_v^- \in \mathcal{N}} \exp(\text{sim}(\boldsymbol{h}_u, \boldsymbol{h}_v^-))},$$

where sim is some similarity metric, $\boldsymbol{h}_v^+$ is the embeddings of a point v that is a positive pair with point u, and $\mathcal{N}$ is a set of negative pairs for point u, whose embeddings are denoted as $\boldsymbol{h}_v^-$. In our experiments, we adopt sim as $\exp(-\|\boldsymbol{h}_u - \boldsymbol{h}_v\|^2/\tau)$, where τ is a positive hyperparamter. We also experimented with angular distances and dot product as the similarity metric, and they can hardly work even with GNNs that need no approximation in computation. The dataset split is done in a cloud-wise way, i.e., $80\%/10\%/10\%$ of point clouds are used to train/validate/test models, respectively.

Pileup-10k. Each point cloud consists of both charged and neutral particles (points), and models are only trained to predict the class of each neutral particle, i.e., from LC or PCs. Since this is an imbalanced binary classification task, meaning more points are labeled from PCs, the Focal loss (Lin et al., 2017) is adopted, i.e., a variant of cross-entropy loss for imbalanced classification, for better performance. The dataset split is also done in a cloud-wise way, i.e., $80\%/10\%/10\%$ of point clouds are used to train/validate/test models, respectively.

B. Extended Related Work

Efficient Transformers in NLP. Leveraging local inductive biases in NLP, several works introduce local attention patterns (Child et al., 2019; Tay et al., 2020; Beltagy et al., 2020; Zaheer et al., 2020). Other approaches focus on exploiting the inherent properties of the attention matrix, employing techniques such as RFF or Nystrom for low-rank approximations (Wang et al., 2020; Xiong et al., 2021; Peng et al., 2021; Choromanski et al., 2021), leveraging the sparsity of

the attention matrix with various LSH-based methods (Kitaev et al., 2020; Daras et al., 2020; Zandieh et al., 2023; Han et al., 2024), or combining both properties (Chen et al., 2021). Research has also been conducted on optimizing attention computation on hardware like GPUs (Dao et al., 2022; Dao, 2024) and in designing new linear transformers that replace the original Softmax attention (Katharopoulos et al., 2020; Sun et al., 2023).

Other Efficient Transformers. In CV, domain-specific knowledge has led to the adoption of local neighborhood attention (Zhao et al., 2021; Mao et al., 2021) and the techniques that partition 2D spaces into grids for parallel processing (Fan et al., 2022; Sun et al., 2022). To further improve computational regularity, some methods (Liu et al., 2023; Wang et al., 2023) project points/voxels onto axes, forming equal-sized point sets along each axis. In scalable graph transformers, where the input graphs are assumed to be given, approaches vary from sampling-based techniques for large graphs (Chen et al., 2022; Zhang et al., 2022; Wu et al., 2022) to those that approximate the spectral properties of input graphs (Shirzad et al., 2023).

Attention Kernels & Relevant Positional Encoding. The original transformer (Vaswani et al., 2017) utilizes the attention kernel $\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j)$ with absolute positional encoding (PE) added to the query/key vectors to capture positional information between tokens in sentences. Following this idea, many works, such as those for 3D detection from CV (Liu et al., 2023; Wang et al., 2023) adopt PE similar to this. Instead of absolute PE, there are also studies utilizing relative positional encoding (RPE), and one way to formulate it is $\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + b_{j-i})$, where b_{j-i} could be some learnable parameters for each relative position (Shaw et al., 2018; Wu et al., 2021). Recently, FLT (Choromanski et al., 2023) also considers generalizing the idea of RPE to GDL tasks with point coordinates, and they adopt RFF-based methods with a kernel, e.g., $\exp(\boldsymbol{q}_i^\top \boldsymbol{k}_j + \omega \exp(-\|\boldsymbol{\rho}_i - \boldsymbol{\rho}_j\|^2/\sigma^2))$, where $\boldsymbol{\rho}$ is point coordinates. However, as demonstrated by our analysis and empirically in Table 1, such RFF-based RPE implementation does not work well for large-scale point clouds with local inductive bias (and there may not be an easy way to adopt LSH-based methods to approximate this kernel). Our proposed kernel can also be viewed as a type of RPE regarding the way to leverage the point coordinates, as it effectively incorporates distance information between points in attention computation. However, no efficient transformers (with near-linear complexity) have been developed to enable the approximation of our type of RPE. HEPT is the first work that enables effective approximations of it via LSH and makes the obtained efficient transformer better suited for GDL tasks with local inductive bias.

Point Cloud Serialization for Efficient Point Cloud Processing. Recently, Point Transformer V3 (Wu et al., 2024) summarizes a series of works in CV as point cloud serialization (PCS) techniques, which project irregular point cloud data into regular sequences with locality in the original 3D space being preserved to some degree for efficient computation (Liu et al., 2023; Wang et al., 2023; Wang, 2023c). These studies typically employ fixed serialization patterns, such as those induced from Hilbert or Z-order curves (Wu et al., 2024). Actually, HEPT can also be seen as a PCS technique, but it uses *randomized* serialization patterns through LSH to project point clouds into regular sequences for computing block-diagonal attention. Since those fixed serialization patterns would overlook specific locality patterns (Wu et al., 2024) and only work for low-dimensional data, the *randomized* approach in HEPT offers an effective alternative or complementary PCS method, due to its guaranteed and analyzable ability to capture locality in the data, even when it is high-dimensional.

C. Implementation Details

C.1. Implementation of HEPT

We follow the standard architecture of the transformer (Vaswani et al., 2017), i.e.,

$$\begin{aligned} \boldsymbol{H} &= \mathrm{LN}(\boldsymbol{H}^\ell) \\ \boldsymbol{Q} = [\boldsymbol{H}\boldsymbol{W}_Q \| \sqrt{2\omega}\boldsymbol{\rho}], \quad \boldsymbol{K} &= [\boldsymbol{H}\boldsymbol{W}_K \| \sqrt{2\omega}\boldsymbol{\rho}], \quad \boldsymbol{V} = \boldsymbol{H}\boldsymbol{W}_V \\ \boldsymbol{H}' &= \boldsymbol{H} + \mathrm{MHSA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \\ \boldsymbol{H}^{\ell+1} &= \boldsymbol{H}' + \mathrm{FFN}(\mathrm{LN}(\boldsymbol{H}')), \end{aligned}$$

where $\boldsymbol{H}^\ell \in \mathbb{R}^{n\times h}$ is the learned point embeddings at the ℓ^{th} layer, LN is the layer normalization (Ba et al., 2016), $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$ are learnable projection matrices, $\boldsymbol{\rho} \in \mathbb{R}^{n\times k_2}$ is point coordinate matrix, $\omega \in \mathbb{R}^+$ are positive learnable parameters, $\|$ concatenates two matrices along the column dimension, and FFN denotes a feed-forward layer. MHSA is the multi-head self-attention mechanism, and in our case, the unnormalized attention scores between query-key pairs will be computed via our kernel, and the full attention matrix is approximated by our LSH-based methods illustrated in Fig. 1.

Table 7: FLOPs (G) and GPU memory usage (GB). The **Bold**†, **Bold**‡, and **Bold** highlight the first, second, and third best results, respectively. Note that SOTA GNNs for the Tracking datasets employ rather large models and involve complex operations along edges, leading to a significant amount of FLOPs. Besides these two models, other models for the same dataset are ensured to have the same number of trainable parameters and similar FLOPs if possible. GNNs, despite with fewer FLOPs, may have high training/test time due to (dynamic) graph construction and irregular computations.

	Tracking-6k			Tracking-60k			Pilup-10k		
	FLOPs	Train Mem.	Test Mem.	FLOPs	Train Mem.	Test Mem.	FLOPs	Train Mem.	Test Mem.
SOTA GNNs	316.1	4.7	1.8	3717.8	OOM	16.9	3.9	3.3	1.5
Reformer	6.8	1.8	0.7	101.0	19.0	9.4	8.0	**1.4**	0.6
SMYRF	5.3	1.4	**0.6**	54.6	20.9	3.6	8.1	1.6	**0.5**
HyperAttn	5.2	**1.2**‡	0.52	54.3	19.4	3.3	8.7	1.5	0.6
Performer	5.6	**1.3**	**0.6**	58.6	20.0	3.7	9.7	1.5	0.6
FLT	6.4	**1.2**‡	**0.5**‡	66.9	19.2	3.5	9.9	**1.3**‡	0.6
ScatterBrain	6.4	2.1	0.7	58.8	21.0	5.2	9.7	2.6	0.8
PointTrans	2.9	**1.2**‡	**0.5**‡	30.6	**18.3**‡	4.4	4.9	**1.3**‡	0.7
FlatFormer	5.7	**1.3**	**0.5**‡	58.7	19.8	**2.5**‡	9.6	1.6	**0.4**‡
GCN	2.0	**1.2**‡	0.6	20.9	**18.6**	4.1	3.4	**1.3**‡	**0.4**‡
DGCNN	11.9	1.6	0.6	124.0	22.3	4.1	15.5	1.9	0.6
GravNet	2.1	**0.8**†	**0.2**†	21.9	**14.5**†	**2.4**†	4.3	**0.7**†	**0.4**‡
GatedGNN	2.4	2.5	1.2	24.8	23.8	10.6	3.9	3.3	1.5
HEPT	5.0	**1.3**	**0.5**‡	52.2	19.9	**2.9**	8.5	**1.3**†	**0.2**†

In our implementation for the three datasets, HEPT uses 4 layers and 24 hidden dimensions with 8 attention heads in each layer. In addition, we adopt $m_1 = 3$ hash tables, each with $m_2 = 3$ hash functions for the three datasets. The block size of attention computation is set to 100, and we use only point coordinates without point hidden representations as the AND hash inputs, i.e., $L_{\boldsymbol{q}_u}^{(i(1+\ell))} = L_{\boldsymbol{k}_u}^{(i(1+\ell))} = h_{\boldsymbol{a}_\ell}(\boldsymbol{\rho}_u)(= \boldsymbol{a}_\ell \cdot \boldsymbol{\rho}_u)$, for $\ell = 1, 2$, where in HEP, the points are in a 2-d $\eta - \phi$ space (Thais et al., 2022), as detailed in Appendix A.2. The total number of buckets $\prod_{j=2}^{m_2} B_{ij}$ is tuned for different datasets.

For the Tracking datasets, the resulting 24-dimensional point embeddings are first projected into 12 dimensions to ensure a fair comparison with SOTA GNNs, which output 12-dimensional final point embeddings. Then, the embeddings are fed into the InfoNCE loss as described in Sec. A.3 to learn similar point embeddings for points from the same particle and dissimilar embeddings for those from different particles. For the Pileup dataset, the resulting point embeddings are projected into 1 dimension with a $\mathrm{sigmoid}$ layer for the computation of Focal loss.

C.2. Implementation of Baselines & Hyperparameter Tuning

All baseline transformers are implemented following the same standard transformer architecture as above with the full self-attention module replaced with the corresponding proposed efficient attention modules, and GNNs are realized using the implementation from PyTorch Geometric (Fey & Lenssen, 2019).

For baseline transformers, as the number of trainable parameters and the architecture is fixed, we only need to tune method-specific hyperparameters, and we are to tune these hyperparameters in a (small) range of FLOPs that would not deviate too much (e.g., $\pm 10\%$ GFLOPs) if possible such that all baseline transformers are ensured to be with similar FLOPs for a fair comparison. For GNN baselines, we mainly follow the implementation from the authors' code and change the hidden dimensions to align the number of trainable parameters. In the following, we describe in detail how each baseline is implemented and tuned, and Table 2 and Table 7 benchmark the computational speed, FLOPs, and GPU memory usage for the tuned baseline models.

Basic Settings. For all datasets and baselines, Adam optimizer (Kingma & Ba, 2015) is used. For the two Tracking datasets, the learning rate is tuned from $\{1e^{-2}, 1e^{-3}\}$, and is multiplied by a factor of 0.5 every 500 epochs. Any model will be early-stopped if there is no improvement in the validation set over 200 consecutive epochs, and models can be trained for up to 2000 epochs to ensure convergence. Models for these two datasets are set with 0.33M trainable parameters for efficiency. For the Pileup dataset, the learning rate is tuned from $\{1e^{-3}, 1e^{-4}\}$, and is multiplied by a factor of 0.5 if there is no improvement in the validation set for 20 epochs. For this dataset, models can be trained for up to 200 epochs for

Table 8: Computational cost of each module in HEPT for inference latency (ms).

Module	Tracking-6k	Tracking-60k	Pileup-10k
Attn	5.8 (83%)	52.8 (91%)	9.2 (86%)
Other	1.2 (17%)	5.1 (9%)	1.5 (14%)
Total	7.0 (100%)	57.9 (100%)	10.7 (100%)

Table 9: Computational cost of each module in HEPT for training latency (ms).

Module	Tracking-6k	Tracking-60k	Pileup-10k
Loss+Backward	330 (97.6%)	2248 (97.2%)	28.2 (70%)
Attn	6.6 (2.0%)	57.1 (2.5%)	10.4 (26%)
Other	1.3 (0.4%)	6.2 (0.3%)	1.7 (4%)
Total	338 (100%)	2312 (100%)	40.3 (100%)

convergence, and they are set with 0.31M trainable parameters.

HEPT. Denote G the total number of desired buckets (i.e., the number of unique aux hash codes allowed) when obtaining AND hash codes, which is tuned from $\{10, 15, 20\}$ for Tracking-6k, from $\{100, 150, 200\}$ for Tracking-60k, from $\{100, 120, 140\}$ for Pileup-10k. And B_{ij}'s are generated randomly such that $\prod_{j=2}^{m_2} B_{ij} = G$. Note that B_{ij}'s do not have to be integers.

Reformer (Kitaev et al., 2020) is implemented via (Wang, 2023b). Its hyperparameters are tuned from {(Block Size : 150, # Hash Tables : 3), (Block Size : 100, # Hash Tables : 3), (Block Size : 100, # Hash Tables : 2)}.

SMYRF (Daras et al., 2020) is implemented via (Dao & Chen, 2023). Its hyperparameters are tuned from {(Block Size : 150, # Hash Tables : 3), (Block Size : 100, # Hash Tables : 3), (Block Size : 100, # Hash Tables : 2)}.

HyperAttn (Han et al., 2024) is implemented via the author-provided code. Its hyperparameters are tuned from {(Block Size : 100, Sample Size : 300), (Block Size : 150, Sample Size : 200), (Block Size : 100, Sample Size : 200)}.

Performer (Choromanski et al., 2021) is implemented via (Wang, 2023a). Its number of feature map dimensions is tuned from {150, 200, 250}.

FLT (Choromanski et al., 2023) is implemented via (Wang, 2023a; Idiap, 2023). Its hyperparameters are tuned from {(# Feature Maps for RPE : 10, # Feature Maps for Attn : 150), (# Feature Maps for RPE : 10, # Feature Maps for Attn : 100), (# Feature Maps for RPE : 20, # Feature Maps for Attn : 100)}.

ScatterBrain (Chen et al., 2021) is implemented via (Dao & Chen, 2023). Its hyperparameters are tuned from {(Block Size : 100, # Hash Tables : 2, # Feature Maps : 100), (Block Size : 100, # Hash Tables : 3, # Feature Maps : 50), (Block Size : 50, # Hash Tables : 2, # Feature Maps : 150)}.

FlatFormer (Liu et al., 2023) is implemented via the author-provided code to adapt to general point-cloud data. We follow its proposed architecture, which is a bit different from the standard transformer. We tune its "window shape" by projecting points into each axis and partitioning each axis equally into N parts. This N is tuned from {20, 30, 40} for Tracking-6k, {100, 150, 200} for Tracking-60k, from {30, 40, 50} for Pileup-10k.

Other Baselines. For Point Transformer (Zhao et al., 2021), GCN (Kipf & Welling, 2017), and GravNet (Qasim et al., 2019), we directly adopt the implementation from PyTorch Geometric. For DGCNN (Qu & Gouskos, 2020), we follow the description in the paper and modify the implementation from PyTorch Geometric accordingly. For GatedGNN (Li et al., 2023), we use the author-provided code, which is also based on PyTroch Geometric. These methods do not have extra hyperparameters to tune, and we change their hidden dimensions accordingly to align the number of trainable parameters.

C.3. Implementation of Numerical Experiments

The numerical experiments conducted in Sec. 3.3 generate $n = 30,000$ points uniformly distributed across a 2D square with a side length of 10. To model local inductive bias, each point interacts only with its 64 nearest neighbors, approximating a ground-truth kernel value of $\exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$ (we select this kernel because it is the well-known Gaussian kernel (Seeger, 2004) and aligns with our proposed attention kernel, but our theoretical results are not limited to this kernel). Points beyond this neighborhood have a kernel value of 0.

Then, E^{2}LSH is utilized for approximation. Given a budget of FLOPs F, OR-only LSH approximates the kernel values by setting the number of hash functions per table to 1 and searching the bucket size and the number of hash tables to find its optimized approximation error ϵ in this point cloud system; OR & AND LSH searches the bucket size, the number of hash tables, and the number of hash functions per table, to obtain the optimized error for a given number of FLOPs.

The bucket size in E^{2}LSH is determined by adjusting the quantization term r (see Sec. 2), which is searched from 0.01 to 5 with a step size of 0.05. If searched, both the number of hash tables and the number of hash functions per table are searched from 1 to 20, with a step size of 1.

D. Latency Breakdown

Table 8 and Table 9 evaluate the computational cost of each module in HEPT using the same checkpoints from Table 2. We can see that the majority of time is spent on attention computation during inference. On the other hand, during training, the computation of loss and gradient backpropagation dominates the total running time for the tasks considered in this work.

E. Theoretical Results

In this section, we provide the proof omitted in the main text. Recall the following settings for our analysis:

Definition 3.1 (Bounded-Support Kernels)**.** Consider a properly normalized shift-invariant kernel defined as $k_s(\boldsymbol{x}, \boldsymbol{y}) = k_s(\boldsymbol{x} - \boldsymbol{y})$, where $k_s(\boldsymbol{x}, \boldsymbol{y}) \in [0, 1]$, $s > 0$ and $k_s(\boldsymbol{0}) = 1$. This kernel exhibits bounded support, i.e., $k_s(\boldsymbol{x} - \boldsymbol{y}) = 0$ for $\|\boldsymbol{x} - \boldsymbol{y}\|_2 > s$. For any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, the computational complexity of $k_s(\boldsymbol{x}, \boldsymbol{y})$ is linear in d.

Assumption 3.2 (Local Inductive Bias)**.** Consider a bounded point cloud system $\mathcal{C}$ with n points located at $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ in a d-dim unit ball, i.e., $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\|\boldsymbol{x}_i\|_2 \leq 1$. Denote the empirical distribution of the point-pair distances as $\phi(z) = \frac{1}{n(n-1)} \sum_{i,j \in [n], i \neq j} \delta_{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2}(z)$ where $\delta_a(\cdot)$ is 1-dim dirac delta function. $\mathcal{C}$ is said to hold local inductive bias if the ground-truth function for the learning task over $\mathcal{C}$ can be approximated by a transformer with full attention matrices whose attention weights can be represented as a bounded-support kernel k_s between point locations $\boldsymbol{x}_i$'s, where the bound s satisfies $\int_0^s \phi(z) dz \sim \tilde{\mathcal{O}}(\frac{1}{n})$.

Note that the point cloud system $\mathcal{C}$ may be a given deterministic or sample from a distribution $\mathcal{C} \sim \mathbb{P}$. In the latter case, we slightly abuse the notation by still using ϕ to denote the distance density function defined in Assumption 3.2 while after the expectation over $\mathbb{P}$, $\mathbb{E}_{\mathbb{P}}(\phi)$.

To simplify our notation, we denote $k_s(z) = \mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}}[k_s(\boldsymbol{x} - \boldsymbol{y}) \mid \|\boldsymbol{x} - \boldsymbol{y}\|_2 = z]$ and $k_s^2(z) = \mathbb{E}_{\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}}[k_s^2(\boldsymbol{x} - \boldsymbol{y}) \mid \|\boldsymbol{x} - \boldsymbol{y}\|_2 = z]$ in the following subsections.

E.1. Proof of Theorem 3.3

Theorem 3.3 ($\epsilon - F$ Tradeoff of RFF)**.** *Assume $k_s(\boldsymbol{x}, \boldsymbol{y})$ is positive definite. If approximating it by RFF $\psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ in point cloud systems described in Assumption 3.2, the error $\epsilon = \Theta(\frac{nd}{F})$.*

Proof. Denote $\widehat{k}_s(\boldsymbol{x}, \boldsymbol{y}) = \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$. Since $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \Big[\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}) \Big]^\top$ with $\boldsymbol{w}_i \overset{iid}{\sim} k_s^*(\boldsymbol{w})$, its expected squared error w.r.t. $\boldsymbol{w}$ (Sutherland & Schneider, 2015) is

$$\mathrm{MSE}_{k_s}\left(\widehat{k}_s(\boldsymbol{x}, \boldsymbol{y})\right) = \mathbb{E}_{\boldsymbol{w}}\left[\left(\widehat{k}_s(\boldsymbol{x}, \boldsymbol{y}) - k_s(\boldsymbol{x}, \boldsymbol{y})\right)^2\right] = \frac{1}{D}(1 + k_s(2(\boldsymbol{x} - \boldsymbol{y})) - 2k_s(\boldsymbol{x} - \boldsymbol{y})^2).$$

Therefore, the squared error averaged over all point pairs in the system is

$$\epsilon = \mathbb{E}_{z \sim \phi(z)}\left[\frac{1}{D}\left(1 + k_s(2z) - 2k_s(z)^2\right)\right] = \frac{1}{D}\left(1 + \mathbb{E}_{z \sim \phi(z)}\left[k_s(2z)\right] - 2\mathbb{E}_{z \sim \phi(z)}\left[k_s^2(z)\right]\right).$$

Since $\int_0^s \phi(z) dz \sim \tilde{\mathcal{O}}(\frac{1}{n})$ (Assumption 3.2) and $k_s(z) \in [0, 1]$, we have $\epsilon = \Theta(\frac{1}{D})$.

To use this RFF to approximate the attention mechanism $\boldsymbol{AV}$, where $\boldsymbol{V} \in \mathbb{R}^{n \times d}$ is the value matrix and $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is the unnormalized attention matrix with each entry $\boldsymbol{A}_{\boldsymbol{x}, \boldsymbol{y}} = k_s(\boldsymbol{x}, \boldsymbol{y})$, we first obtain $\boldsymbol{X'}, \boldsymbol{Y'} \in \mathbb{R}^{n \times D}$ with each row given

by $\psi(\boldsymbol{x})$ and $\psi(\boldsymbol{y})$, respectively, which requires $nD(2d-1)+nd=\Theta(nDd)$ FLOPs. Then, $\boldsymbol{AV}\approx\boldsymbol{X}'(\boldsymbol{Y}'^{\top}\boldsymbol{V})$ needs $Dd(2n-1)+nd(2D-1)=\Theta(nDd)$ FLOPs. Note that when using RFF, it is important to first compute $\boldsymbol{Y}'^{\top}\boldsymbol{V}$ to avoid the complexity of n^2 for computing $\boldsymbol{X}'\boldsymbol{Y}'^{\top}$. Thus, the total FLOPs required are $F=\Theta(ndD)$.

Therefore, with $\epsilon=\Theta(\frac{1}{D})$ and $F=\Theta(ndD)$, we have $\epsilon=\Theta\left(\frac{nd}{F}\right)$. □

E.2. Proof of Theorem 3.4

Lemma E.1. *Consider the collision probability $p_r(z)$ in E^{2}LSH, employing the hash function $h_{\boldsymbol{a},b}(\boldsymbol{x})=\lfloor\frac{\boldsymbol{a}\cdot\boldsymbol{x}+b}{r}\rfloor$, for two points $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^d$ with distance $z=\|\boldsymbol{x}-\boldsymbol{y}\|_2$. This probability can be bounded as follows:*

For $z<r$,

$$1-\sqrt{\frac{2}{\pi}}\frac{z}{r}\le p_r(z)\le 1-\sqrt{\frac{1}{2\pi}}\frac{z}{r}.$$

For $z\ge r$,

$$\frac{\sqrt{2}}{3\sqrt{\pi}}\frac{r}{z}\le p_r(z)\le\frac{1}{\sqrt{2\pi}}\frac{r}{z}.$$

Consequently, the expected collision probability for pairs of points in the point cloud systems described in Assumption 3.2 can be bounded as:

$$\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)\right]\le\int_0^r\left(1-\sqrt{\frac{1}{2\pi}}\frac{z}{r}\right)\phi(z)dz+\sqrt{\frac{1}{2\pi}}r\int_r^{\infty}\frac{1}{z}\phi(z)dz.$$

Proof. With hash functions from E^{2}LSH, the collision probability for two distinct points $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^d$ with distance $z=\|\boldsymbol{x}-\boldsymbol{y}\|_2$ is (Datar et al., 2004):

$$p_r(z)=\mathrm{P}\left[h_{\boldsymbol{a},b}(\boldsymbol{x})=h_{\boldsymbol{a},b}(\boldsymbol{y})\right]=\int_0^r\frac{1}{z}f_2\left(\frac{t}{z}\right)\left(1-\frac{t}{r}\right)dt,$$

where $f_2(\cdot)$ denotes the PDF of the absolute value of the 2-stable distribution. Thus,

$$p_r(z)=\mathrm{Erf}\left(\frac{r}{\sqrt{2}z}\right)-\sqrt{\frac{2}{\pi}}\frac{z}{r}\left(1-\exp\left(-\frac{r^2}{2z^2}\right)\right),$$

where $\mathrm{Erf}(u)=\frac{2}{\sqrt{\pi}}\int_0^u e^{-t^2}\,\mathrm{d}t$.

Let $u=\frac{r}{\sqrt{2}z}$, $p_r(z)=f(u)=\mathrm{Erf}(u)-\frac{1}{u\sqrt{\pi}}\left(1-\exp\left(-u^2\right)\right)$. We are to bound $f(u)$ for $u>\frac{1}{\sqrt{2}}$ and $0<u\le\frac{1}{\sqrt{2}}$.

Bounding $f(u)$ for $u>\frac{1}{\sqrt{2}}$. Consider $g(u)=f(u)-(1-\frac{1}{u\sqrt{\pi}})$. Since $g'(u)=\frac{-\exp(-u^2)}{\sqrt{\pi}u^2}<0$, $g(\frac{1}{\sqrt{2}})>0$, and $\lim_{u\to\infty}g(u)=0$, we have $f(u)\ge1-\frac{1}{u\sqrt{\pi}}$. Now, consider $g(u)=f(u)-(1-\sqrt{\frac{1}{\pi}}\frac{1}{2u})$. Since $g'(u)=\frac{-2+\exp(u^2)}{2\exp(u^2)\sqrt{\pi}u^2}$, it has a local minima at $u=\sqrt{\ln 2}$. Because $g'(u)<0$ when $\frac{1}{\sqrt{2}}<u<\sqrt{\ln 2}$, $g'(u)>0$ when $u>\sqrt{\ln 2}$, $g(\frac{1}{\sqrt{2}})<0$, and $\lim_{u\to\infty}f(u)=0$, we have $f(u)\le1-\sqrt{\frac{1}{\pi}}\frac{1}{2u}$.

Bounding $f(u)$ for $0<u\le\frac{1}{\sqrt{2}}$. Consider $g(u)=f(u)-\frac{2}{3\sqrt{\pi}}u$. Since $g'(u)=\frac{3-(2u^2+3/\exp(u^2))}{3\sqrt{\pi}u^2}>0$ when $0<u\le\frac{1}{\sqrt{2}}$, $f(\frac{1}{\sqrt{2}})>0$, and $\lim_{u\to0}f(u)=0$, we have $f(u)\ge\frac{2}{3\sqrt{\pi}}u$. Now, consider $g(u)=f(u)-\frac{1}{\sqrt{\pi}}u$. Since $g'(u)=\frac{1-(u^2+\exp(-u^2))}{\sqrt{\pi}u^2}<0$ when $0<u\le\frac{1}{\sqrt{2}}$, $f(\frac{1}{\sqrt{2}})<0$, and $\lim_{u\to0}f(u)=0$, we have $f(u)\le\frac{1}{\sqrt{\pi}}u$.

Therefore, substituting the u back, we yield the above bounds for $p_r(z)$, which directly gives the bound for $\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)\right]$. □

Lemma E.2. *Consider using E^{2}LSH to approximate the attention mechanism $\boldsymbol{AV}$, where $\boldsymbol{V}\in\mathbb{R}^{n\times d}$ is the value matrix and $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ is the unnormalized attention matrix with each entry $\boldsymbol{A}_{\boldsymbol{x},\boldsymbol{y}}=k_s(\boldsymbol{x},\boldsymbol{y})$. If performing OR-only LSH with m_1 hash functions for the approximation, the required FLOPs are $F=\Theta\left(m_1dn^2\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)\right]+m_1nd\right)$. If constructing m_1 hash tables (OR LSH), with each having m hash functions (AND LSH) for the approximation, the required FLOPs are $F=\Theta\left(m_1dn^2\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)^m\right]+m_1ndm\right)$.*

Proof. First, consider performing OR-only LSH with m_1 hash functions. Let $\mathbb{E}_{z\sim\phi(z)}[p_r(z)]$ denote the expected collision probability for pairs of points in the point cloud system described in Assumption 3.2.

1. Obtaining Hash Codes & Computing Kernel Values. With n points in the system, it needs $n(2d-1)$ FLOPs to obtain hash code from each hash function. Then, $\binom{n}{2}$ point pairs result in $\binom{n}{2}\mathbb{E}_{z\sim\phi(z)}[p_r(z)]$ expected number of collisions (pairs of points with the same hash code) from each of the m_1 hash functions. To compute k_s for all collided pairs, it requires $\Theta(d\binom{n}{2}\mathbb{E}_{z\sim\phi(z)}[p_r(z)])$ FLOPs.

2. Approximating the Attention Mechanism. If one hash function results in B buckets and in each bucket b there are o_b collisions (pairs of points with the same hash code) and n_b unique points, then $\sum_{b\in[B]} o_b = \binom{n}{2}\mathbb{E}_{z\sim\phi(z)}[p_r(z)]$, $\binom{n_b}{2} = o_b$, and $\sum_{b\in[B]} n_b = n$. Computing $\widehat{\boldsymbol{A}}\widehat{\boldsymbol{V}}$ for each bucket b needs $dn_b(2n_b-1)$ FLOPs, since $\widehat{\boldsymbol{A}}$ is $n_b\times n_b$ and $\widehat{\boldsymbol{V}}$ is $n_b\times d$. So, it needs $4d\binom{n}{2}\mathbb{E}_{z\sim\phi(z)}[p_r(z)]+nd$ FLOPs for all buckets since $\sum_{b\in[B]} dn_b(2n_b-1) = \sum_{b\in[B]} 2(n_b^2-n_b)d + n_bd = \sum_{b\in[B]} 4o_bd + n_bd$.

3. Combing m_1 Hash Results. The above process is repeated for m_1 times, and $(m_1-1)nd$ extra FLOPs are needed to combine the m_1 hash results. Therefore, the total FLOPs required is $F = \Theta(m_1dn^2\mathbb{E}_{z\sim\phi(z)}[p_r(z)] + m_1nd)$ if OR-only LSH is performed with m_1 hash functions.

Now, consider constructing m_1 hash tables (OR LSH), each with m hash functions (AND LSH). This results in collision probability $p_r(z)^m$ and now needs $\Theta(m_1ndm)$ FLOPs to obtain all hash codes and combine all results. Then, the total FLOPs required are $F = \Theta\left(m_1dn^2\mathbb{E}_{z\sim\phi(z)}[p_r(z)^m] + m_1ndm\right)$. □

Theorem 3.4 ($\epsilon - F$ Tradeoff of OR-only E^{2}LSH)**.** *Assume there exists r such that $\int_0^r \phi(z)dz \le c_1 r$ and $\int_r^\infty \frac{1}{z}\phi(z)dz \le c_2$ for some positive constants c_1 and c_2. The OR-only E^{2}LSH may achieve $\epsilon = \tilde{\Theta}(\exp\left(-\frac{c_3F}{dn^2s}\right)\frac{1}{n})$ where c_3 is a positive constant depending on c_1 and c_2.*

Proof. Consider performing OR-only E^{2}LSH with m_1 hash functions to approximate k_s in the point cloud system described in Assumption 3.2. The resulting squared error averaged over all point pairs in the system is then $\epsilon = \mathbb{E}_{z\sim\phi(z)}\left[(1-p_r(z))^{m_1}k_s^2(z)\right]$, where $p_r(z)$ is the collision probability of E^{2}LSH.

First, the complexity F has a natural lower bound: Due to Lemma E.2 and the bound of $p_r(z)$ in Lemma E.1,

$$F = \Theta(dm_1n^2\mathbb{E}_{z\sim\phi(z)}[p_r(z)] + m_1nd) \ge \Theta\left(dm_1n^2\left(\int_0^r \phi(z)(1-\sqrt{\frac{2}{\pi}}\frac{z}{r})dz + \int_r^1 \phi(z)\frac{\sqrt{2}}{3\sqrt{\pi}}\frac{r}{z}dz\right)\right)$$

$$\ge \Theta\left(dm_1n^2\left(\int_0^r \phi(z)dz + r\int_r^1 \phi(z)dz\right)\right) \ge \Theta\left(dm_1n^2r\right).$$

Upper bound: Here, we first show that for some positive c_3,

$$\epsilon = \tilde{\mathcal{O}}(\exp\left(-\frac{c_3F}{dn^2s}\right)\frac{1}{n}).$$

We are only interested in the regime with limited complexity where $F = O(dn^2s)$. Otherwise, the above error is almost 0 and the complexity is already super linear because $s \gg \frac{1}{n}$ in general (see the discussion in Sec.3.1). Since for OR-only E^{2}LSH, $F \ge \Theta\left(dm_1n^2r\right)$, this means, in practice, to satisfy $F = O(dn^2s)$, we will set $r \le s$. To better understand this point, if we set $r > s$, intuitively, one single hash function is sufficient to put points within distance s into the same hash bucket with high probability, which is able to compute the attention weights accurately. However, in this case, there will be $n\sqrt{s}$ many points mapped into the same bucket, which gives complexity as much as $\Omega\left(dm_1n^2s\right)$. This can be understood from the lower bound $F \ge \Theta\left(dm_1n^2r\right)$:x when $r > s$, the above bound of F implies $F \ge \Theta\left(dm_1n^2r\right) \ge \Theta\left(dm_1n^2s\right)$.

Let us next suppose $r \le s$. With Lemma E.1, we have

$$\epsilon \le (1-c''\frac{r}{s})^{m_1}\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right] \le \exp\left(-\frac{c''m_1r}{s}\right)\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right],$$

where $c'' = \frac{\sqrt{2}}{3\sqrt{\pi}}$.

By assumption, there exists r such that $\int_0^r \phi(z)dz \le c_1 r$ and $\int_r^\infty \frac{1}{z}\phi(z)dz \le c_2$ for some positive constants c_1 and c_2, we have $\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)\right] \le c'r$, where $c' = c_1 + c_2\sqrt{\frac{1}{2\pi}}$. Then, since $F = \Theta(dm_1n^2\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)\right] + m_1nd)$ (Lemma E.2), we have $F = \mathcal{O}(dm_1n^2c'r)$. Thus,

$$\epsilon \le \exp\left(-\frac{c_3F}{dn^2s}\right)\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right],$$

where c_3 is a positive constant depending on c_1 and c_2.

Since $\int_0^s \phi(z)dz \sim \tilde{\mathcal{O}}\left(\frac{1}{n}\right)$ (Assumption 3.2) and $k_s(z) \in [0,1]$, we have

$$\epsilon = \tilde{\mathcal{O}}(\exp\left(-\frac{c_3F}{dn^2s}\right)\frac{1}{n}).$$

Now, we lower bound ϵ.

Lower bound: Next, we show that there exists a point cloud system $\mathcal{C}$ and the kernel k_s that satisfy the assumption, while for some positive c_5,

$$\epsilon = \Omega(\exp\left(-\frac{c_5F}{dn^2s}\right)\frac{1}{n}).$$

We consider a very common case when $k_s(z) = 1$ when $z \in [0, s]$ and the point cloud is uniformly allocated in the unit ball. In this case, $\phi(z) \propto z^{d-1}$, and $s \sim \frac{1}{n^{1/d}}$ as shown in Sec. 3.1. It is easy to verify that if r satisfies $r \le s$, the conditions in the theorem statement are true: This is because $\int_0^r \phi(z)dz \lesssim r^d < r$ and $\int_r^1 \frac{1}{z}\phi(z)dz \le \frac{d}{d-1}$. Again, we only focus on the regime $r \le s$. It is not hard to show that when $r \gg s$, the lower bound is even higher than above.

When $r \le s$, the above lower bound of F already gives $F = \Omega(dm_1n^2r)$. With Lemma E.1 and $k_s(z) = 1$ for $z \le s$, we have

$$\epsilon = \int_0^s (1-p_r(z))^{m_1}\phi(z)dz \ge \int_0^r (1-p_r(z))^{m_1}\phi(z)dz + \int_r^s (1-p_r(z))^{m_1}\phi(z)dz$$

$$\ge \Theta\left(\int_0^r \left(\frac{1}{\sqrt{2\pi}}\frac{z}{r}\right)^{m_1} z^{d-1}dz + \int_r^s \left(1-\frac{1}{\sqrt{2\pi}}\frac{r}{z}\right)^{m_1} z^{d-1}dz\right) \ge \Theta(r^d(\frac{1}{\sqrt{2\pi}})^{m_1} + (1-\frac{1}{\sqrt{2\pi}})^{m_1}(s^d - r^d))$$

$$= \Theta(\exp(-c_5m_1)\cdot s^d) = \Omega(\exp\left(-\frac{c_5F}{dn^2r}\right)\cdot\frac{1}{n}), \quad \text{where } c_5 \text{ is a positive constant.}$$

This completes the proof. □

E.3. Proof of Theorem 3.5

Theorem 3.5 ($\epsilon - F$ Tradeoff of OR & AND E^{2}LSH)**.** *Suppose each hash table contains m hash functions. Assume there exists m such that $\int_0^r \phi(z)dz = \tilde{\mathcal{O}}(\frac{1}{n})$ and $\int_r^\infty \phi(z)\frac{r^m}{z^m}dz \le \int_0^r(\sqrt{2\pi} - \frac{z}{r})^m\phi(z)dz$, where $r = ms$. By choosing such r as the bucket size, the OR & AND E^{2}LSH may achieve $\epsilon = \tilde{\mathcal{O}}(\exp(-\frac{c_4F}{dn(\text{polylog}(n)+m)})\frac{1}{n})$.*

Proof. Consider using E^{2}LSH to construct m_1 hash tables (OR LSH), each with m hash functions (AND LSH). To approximate k_s in the point cloud systems described in Assumption 3.2, the resulting squared error averaged over all point pairs in the system is $\epsilon = \mathbb{E}_{z\sim\phi(z)}\left[(1-p_r(z)^m)^{m_1}k_s^2(z)\right]$ and the FLOPs required are $F = \Theta(m_1dn^2\mathbb{E}_{z\sim\phi(z)}\left[p_r(z)^m\right] + m_1ndm)$ (Lemma E.2).

Pick the smallest m that satisfies $\int_0^r \phi(z)dz = \tilde{\mathcal{O}}(\frac{1}{n})$ and $\int_r^\infty \phi(z)\frac{r^m}{z^m}dz \le \int_0^r(\sqrt{2\pi} - \frac{z}{r})^m\phi(z)dz$, where $r = ms$, combined with Lemma E.1, then we have

$$F = \mathcal{O}\left(m_1dn^2\int_0^{ms}\left(1-\frac{1}{\sqrt{2\pi}}\frac{z}{ms}\right)^m\phi(z)dz + m_1ndm\right) = \mathcal{O}\left(m_1dn^2\int_0^{ms}\phi(z)dz + m_1ndm\right).$$

Similarly, with $r = ms$ and Lemma E.1, we have

$$\epsilon \le \left(1-\left(1-c'\frac{s}{ms}\right)^m\right)^{m_1}\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right] \le \exp(-c''m_1)\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right],$$

where $c' = \sqrt{\frac{2}{\pi}}$ and $c'' = 1 - c'$.

Therefore,

$$\epsilon = \mathcal{O}\left(\exp\left(-\frac{c''F}{dn^2\int_0^{ms}\phi(z)dz + ndm}\right)\mathbb{E}_{z\sim\phi(z)}\left[k_s^2(z)\right]\right).$$

Since $\int_0^r \phi(z)dz = \tilde{\mathcal{O}}\left(\frac{1}{n}\right)$, $\int_0^s \phi(z)dz \sim \tilde{\mathcal{O}}\left(\frac{1}{n}\right)$, and $k_s(z) \in [0,1]$, we yield

$$\epsilon = \tilde{\mathcal{O}}\left(\exp\left(-\frac{c_4F}{dn(\mathrm{polylog}(n)+m)}\right)\frac{1}{n}\right),$$

where c_4 is some positive constant. □