Efficient Transformer for Point Cloud Data in Geometric Deep Learning

Siqi Miao

Ph.D. Student

ML @ Georgia Tech

Joint work with Zhiyuan Lu (BUPT), Mia Liu (Purdue), Javier Duarte (UCSD), Pan Li (Gatech)

This work has also benefited from insightful discussions with Gage DeZoort (Princeton), Yongbin Feng (Fermilab), Kilian Lieret (Princeton)

Georgia Tech

Miao, Siqi, et al. "Locality-Sensitive Hashing-Based Efficient Point Transformer with Applications in High-Energy Physics." *arXiv preprint arXiv:2402.12535* (2024).

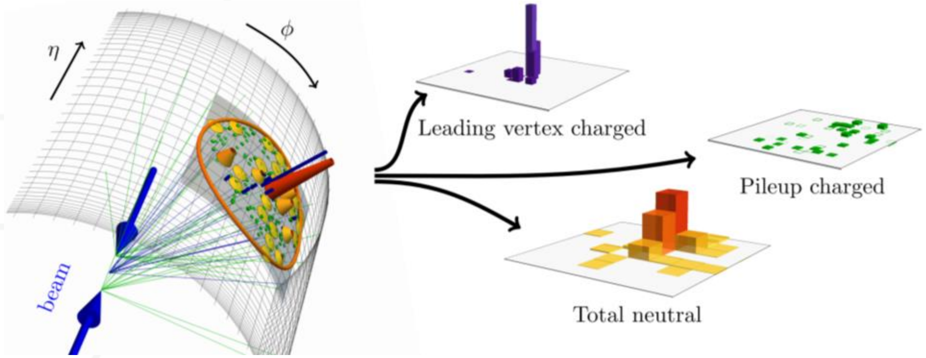
Content

- Background & Motivation
- (Efficient) Transformers
- HEPT: LSH-based Efficient Point Transformer
- Conclusion

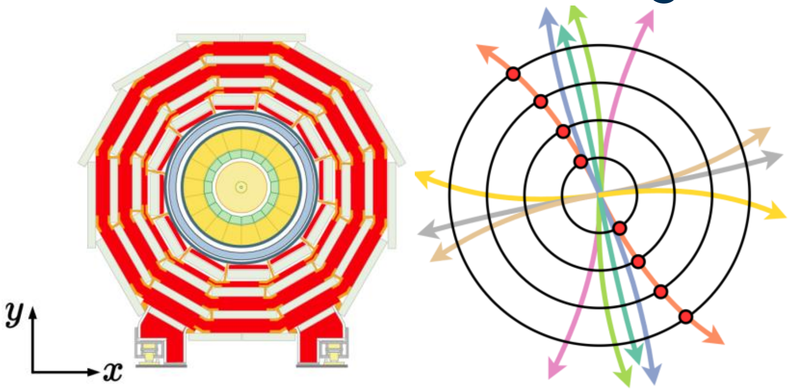
Background & Motivation

Point Clouds in High-Energy Physics

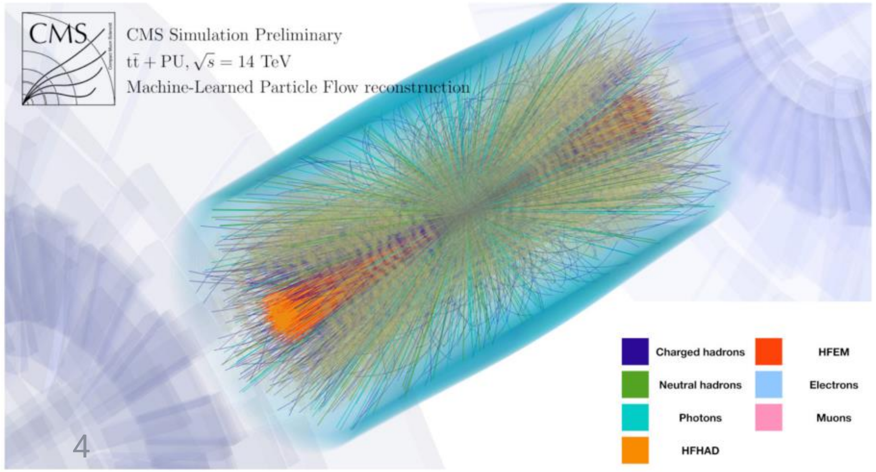
- Pileup Mitigation



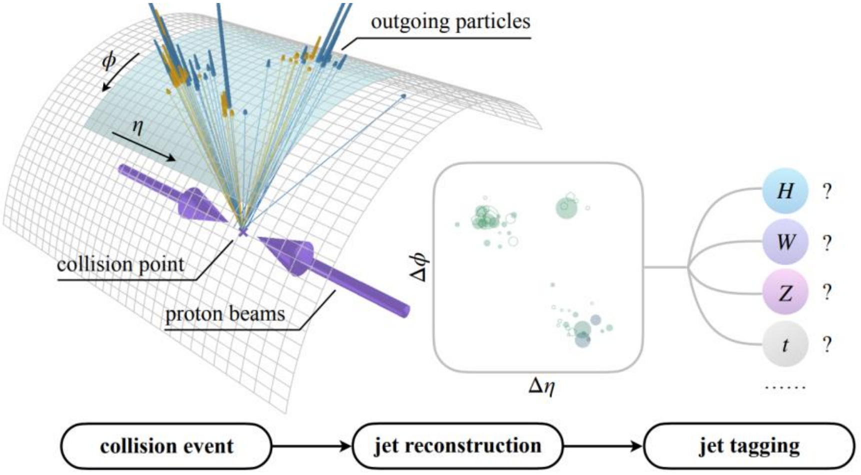
- Particle Tracking

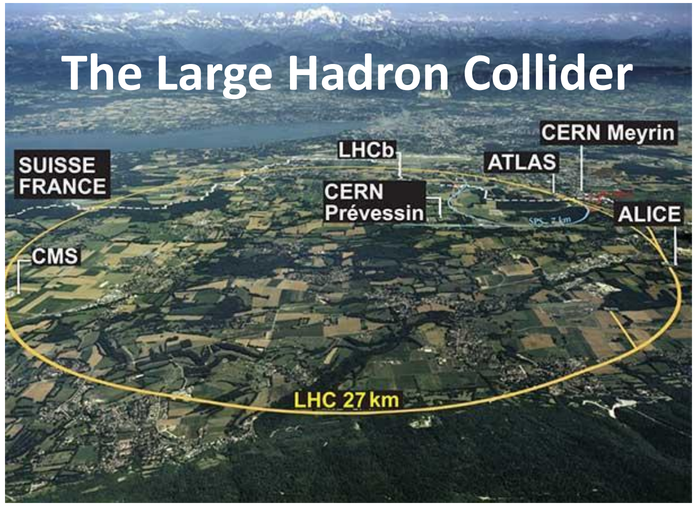


- Particle-flow Reconstruction

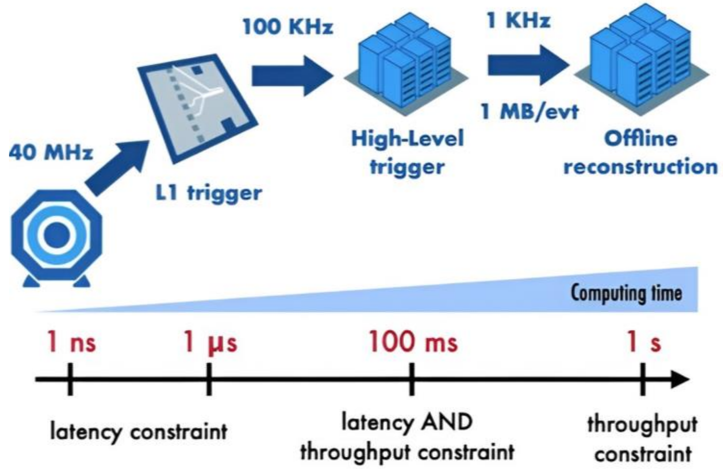


- Jet Tagging



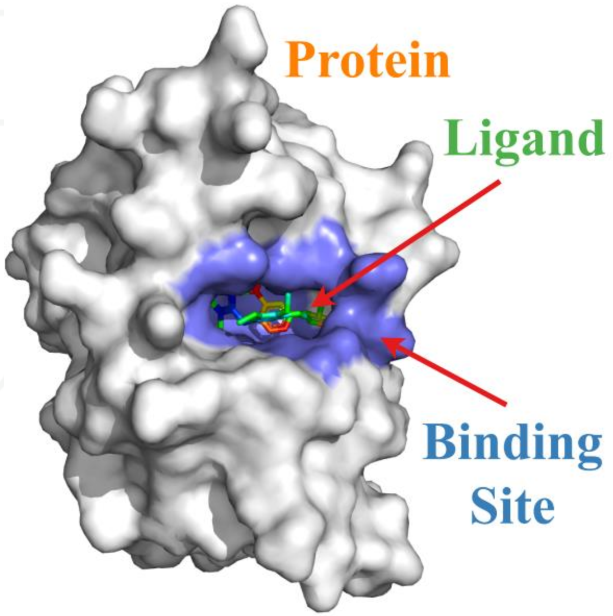


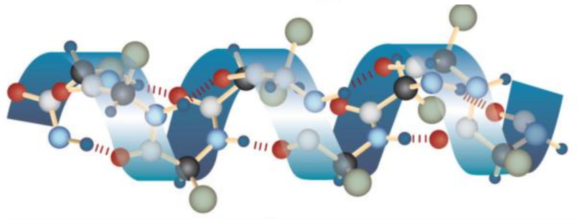
Latency & Throughput



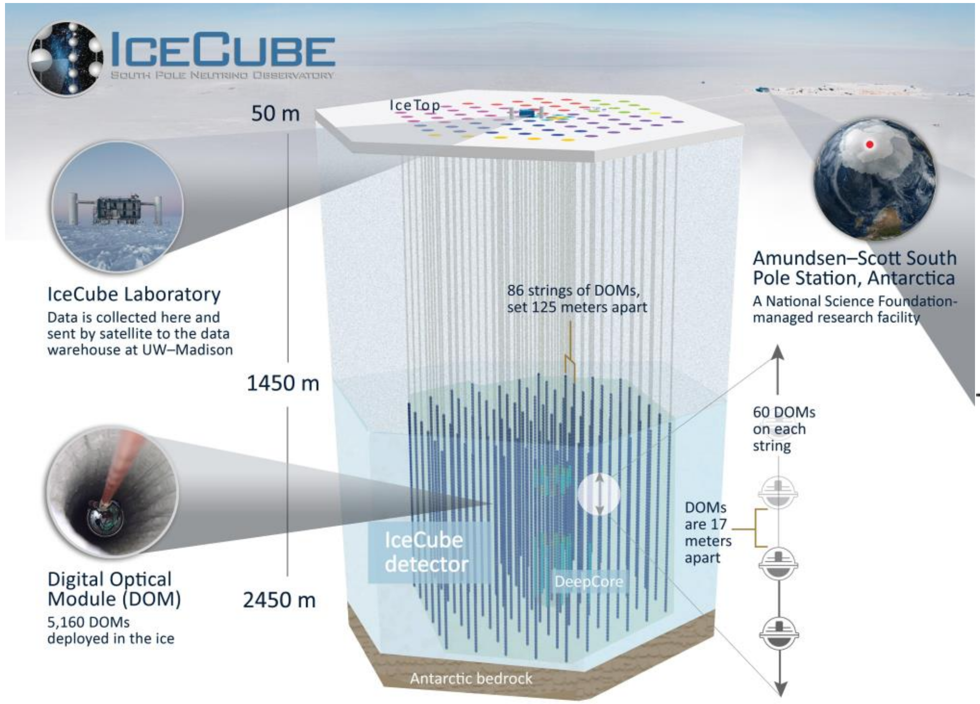
Geometric Deep Learning with Point Cloud Data

- Drug discovery

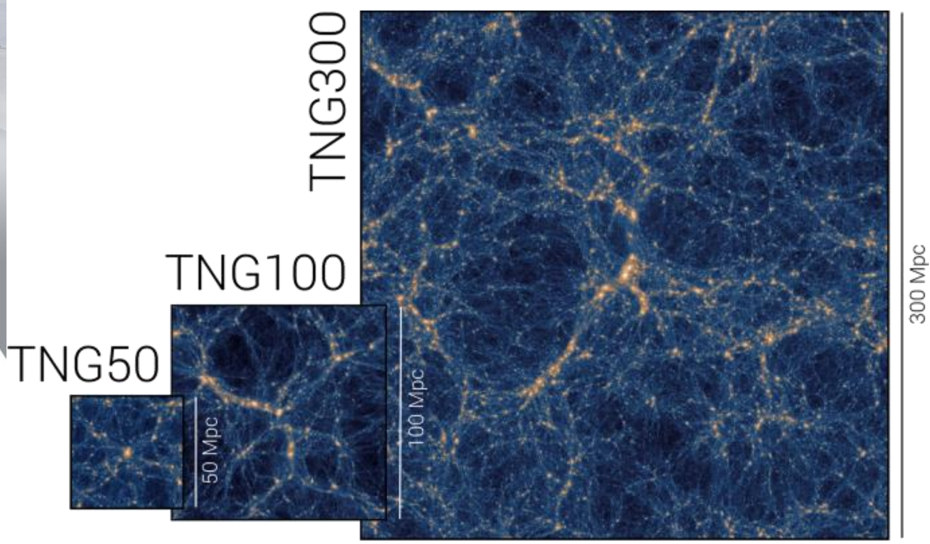




- Neutrino Detection



- Galaxy Evolution

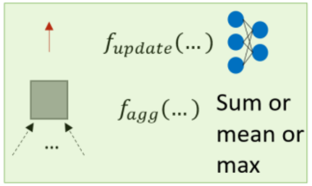


They all require efficient computational methods!

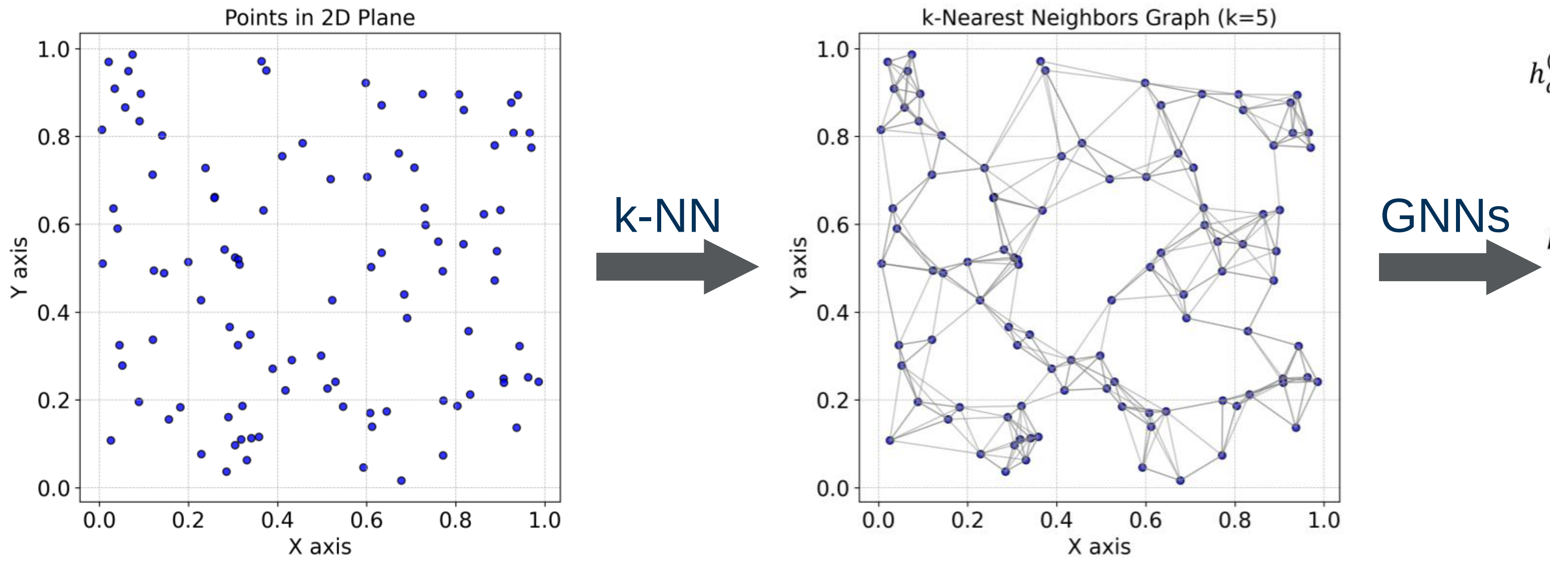
Georgia Tech

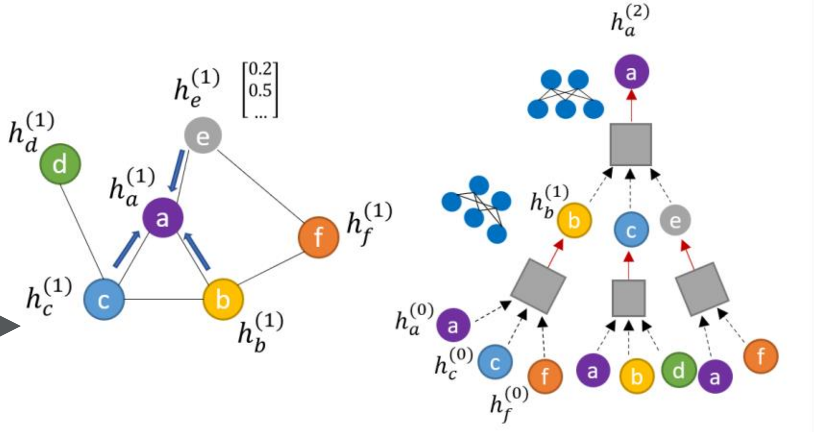
Current Approach

- These point clouds are irregular, but they all hold **local inductive bias**
  - **i.e., a point would primarily interact with its local neighbors**
- Graph neural networks (GNNs) are used,
  - by constructing, e.g., **k-NN graphs**, from point clouds



Graph neural network: one layer





$$h_v^{(t+1)} = f_{update}\left(h_v^{(t)}, f_{agg}\left(\left\{h_u^{(t)} \mid u \in N_v\right\}\right)\right)$$

$$h_G = \text{POOL}\left(\left\{h_v^{(L)} \mid v \in V\right\}\right)$$

k-NN graph construction

- may have $\mathcal{O}(n^2)$ complexity

A sample can easily have over 5k or 50k points!

GNNs involve

- irregular computation & random memory access

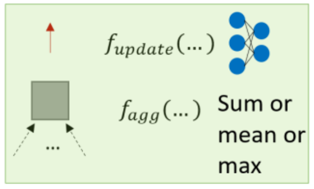
They are not hardware-friendly!

GNNs Are Slow!

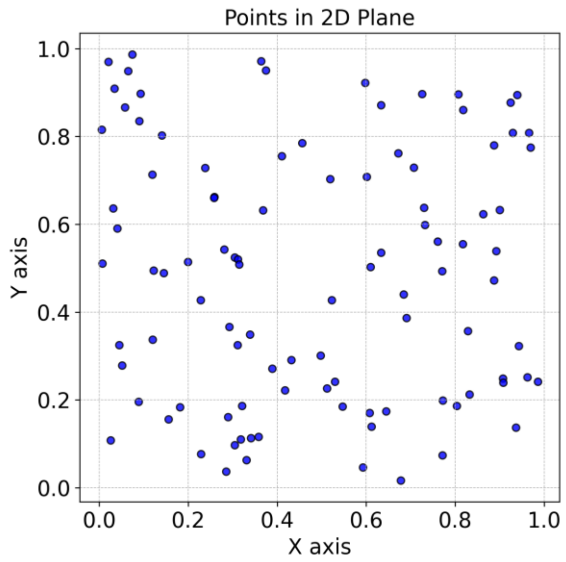
Georgia Tech

Current Approach

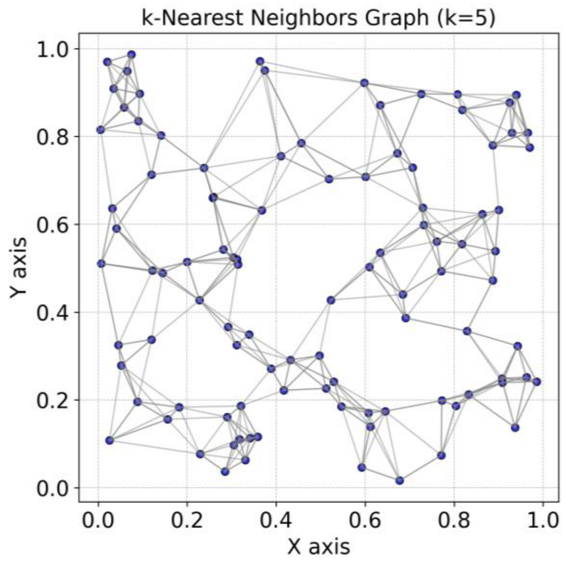
- These point clouds are irregular, but they all hold **local inductive bias**
  - **i.e., a point would primarily interact with its local neighbors**
- Graph neural networks (GNNs) are used,
  - by constructing, e.g., **k-NN graphs**, from point clouds



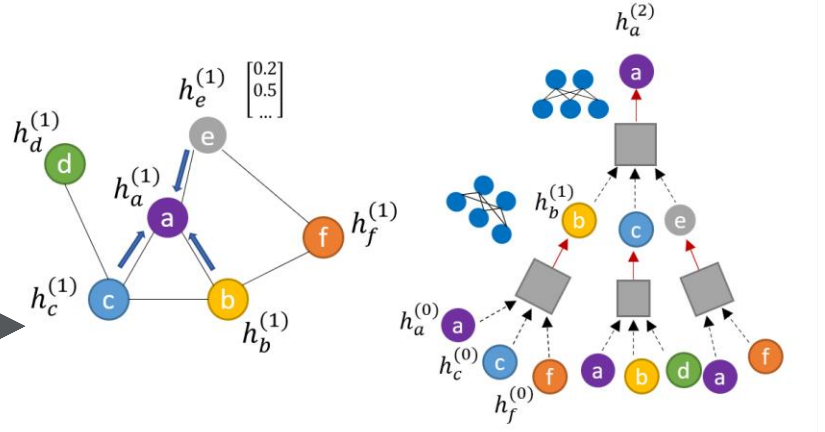
Graph neural network: one layer



k-NN



GNNs



$$h_v^{(t+1)} = f_{update}\left(h_v^{(t)}, f_{agg}\left(\left\{h_u^{(t)} \mid u \in N_v\right\}\right)\right)$$

$$h_G = \mathrm{POOL}\left(\left\{h_v^{(L)} \mid v \in V\right\}\right)$$

k-NN graph construction
- may have $\mathcal{O}(n^2)$ complexity

A sample can easily have over 5k or 50k points!

GNNs involve
- irregular computation & random memory access

They are not hardware-friendly!

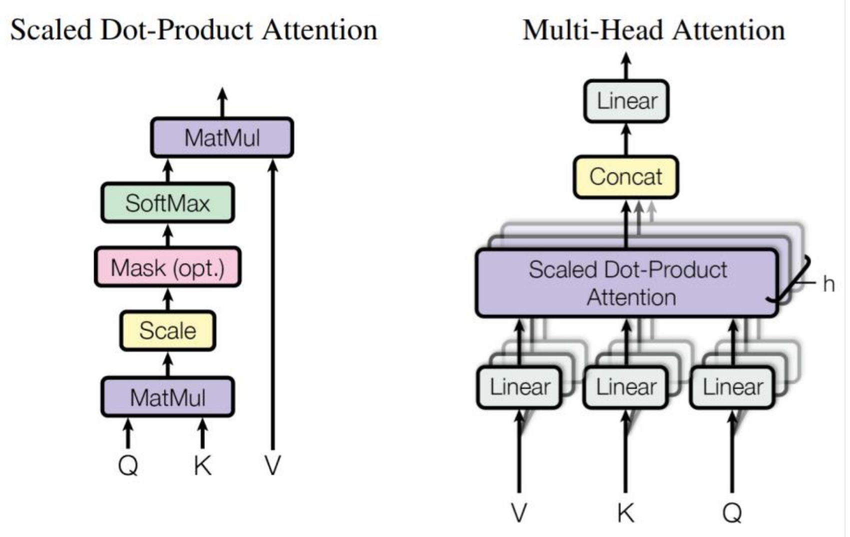
Can we have an **accurate** & **hardware-friendly** model w/ (almost) **linear complexity**?

(Efficient) Transformers

Georgia Tech

Vanilla Transformer

- Self-attention mechanism
  - A token or a point u has three vectors $\boldsymbol{q}_u, \boldsymbol{k}_u, \boldsymbol{v}_u \in \mathbb{R}^d$
  - Stacking them for n points yields three matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times d}$
  - Let's ignore normalization terms for simplicity
    - $\operatorname{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \exp(\boldsymbol{Q}\boldsymbol{K}^\top)\boldsymbol{V}$
    - Capture all pairwise relations
- Why is this **good** for computation?
  - All operations are **regular** matrix multiplication
- Why is this **bad** for computation?
  - The complexity of $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$ is $\mathcal{O}(n^2)$
- We are particularly interested in **efficient** transformers
  - These variants try to decrease the complexity to $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$

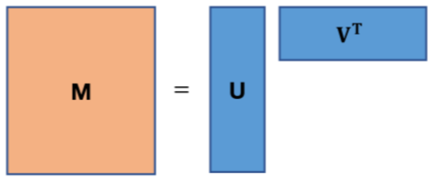


Efficient Transformers via Kernel Approximation

- Viewing $\exp(\boldsymbol{q}_u{}^\top \boldsymbol{k}_u)$ as a kernel
  - Let’s not compute it exactly
  - Instead, use efficient methods to **approximate** it accurately…
  - Ideally, we may achieve
    - **Hardware-friendly** model with only regular computation
    - **No expensive graph construction**
    - **Almost linear complexity**, but may (approximately) capture pair-wise interactions
- There is no free lunch!
  - Studies along this line must **assume some properties** of the attention matrix $\exp(\boldsymbol{Q}\boldsymbol{K}^\top)$
    - for efficient and accurate approximation
  - Two typical **assumptions** (and techniques to use)

Low-rank approximation

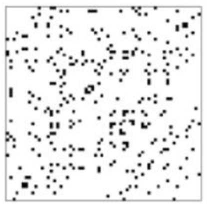
- Random Fourier Features (RFF)



Low rank

Sparse approximation

- Locality-Sensitive Hashing (LSH)



Sparse

Which one is better for GDL tasks?

Efficient Transformers via Kernel Approximation

Low-rank Approximation

- Random Fourier Features (RFFs)
  - For any properly normalized positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $k(\mathbf{0}) = 1$
  - RFFs can approximate such kernels by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi: \mathbb{R}^d \to \mathbb{R}^D$
  - A most common example of such ψ is
    - $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\left(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \dots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x})\right)^\top$
    - $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$, and k^* is the Fourier transform of k
  - Consider RBF kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
    - Then k^* is a standard Gaussian
  - So, one can have
    - $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \hat{\psi}(\boldsymbol{x})^\top \hat{\psi}(\boldsymbol{y})$ with $\hat{\psi}(\boldsymbol{x}) = \exp\left(\frac{\|\boldsymbol{x}\|^2}{2}\right) \psi(\boldsymbol{x})$

$\boxed{\mathcal{O}(n) \text{ complexity!}}$

If $D \ll n$

Sparse Approximation

- Locality-Sensitive Hashing (LSH)

Efficient Transformers via Kernel Approximation

Low-rank Approximation

- Random Fourier Features (RFFs)
  - For any properly normalized positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $k(\mathbf{0}) = 1$
  - RFFs can approximate such kernels by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi: \mathbb{R}^d \to \mathbb{R}^D$
  - A most common example of such ψ is
    - $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \left(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \dots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}) \right)^\top$
    - $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$, and k^* is the Fourier transform of k
  - Consider RBF kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2} \|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
    - Then k^* is a standard Gaussian
  - So, one can have
    - $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \hat{\psi}(\boldsymbol{x})^\top \hat{\psi}(\boldsymbol{y})$ with $\hat{\psi}(\boldsymbol{x}) = \exp\left(\frac{\|\boldsymbol{x}\|^2}{2}\right) \psi(\boldsymbol{x})$

$\boxed{\mathcal{O}(n) \text{ complexity!}}$

If $D \ll n$

Sparse Approximation

- Locality-Sensitive Hashing (LSH)
  - With **high probability**, LSH hashes **close** data points into the **same bucket**, e.g., via $h(\boldsymbol{x}): \mathbb{R}^d \to \mathbb{R}$
  - E.g., by setting $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = \mathbf{1}$, angular distance-based LSH can be used to approx. $\exp(\boldsymbol{x}^\top \boldsymbol{y})$
    - Pairs with close angular distance have large attn
    - So, with high prob. in the same bucket
    - **Compute full attn** for pairs in **the same bucket**

$\boxed{\mathcal{O}(n \log n) \text{ complexity!}}$

If bucket size $\ll n$

Georgia Tech

Efficient Transformers via Kernel Approximation

Low-rank Approximation

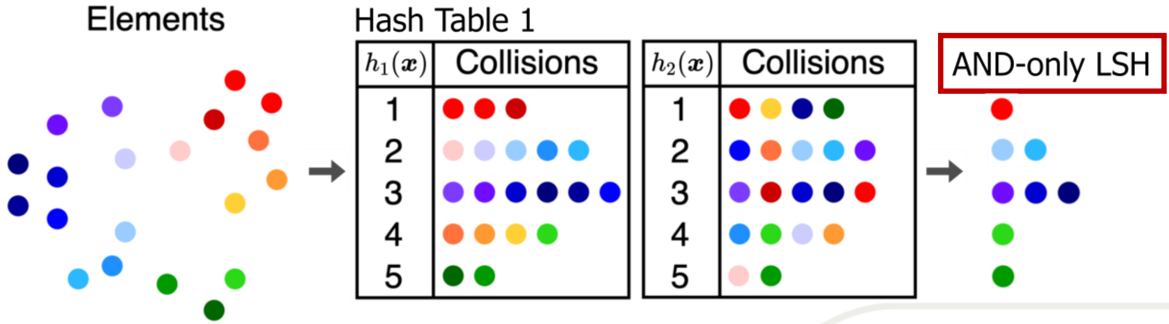
- Random Fourier Features (RFFs)
  - For any properly normalized positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $k(\boldsymbol{0}) = 1$
  - RFFs can approximate such kernels by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi: \mathbb{R}^d \to \mathbb{R}^D$
  - A most common example of such ψ is
    - $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\left(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x})\right)^\top$
    - $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$, and k^* is the Fourier transform of k
  - Consider RBF kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
    - Then k^* is a standard Gaussian
  - So, one can have
    - $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \hat{\psi}(\boldsymbol{x})^\top \hat{\psi}(\boldsymbol{y})$ with $\hat{\psi}(\boldsymbol{x}) = \exp\left(\frac{\|\boldsymbol{x}\|^2}{2}\right)\psi(\boldsymbol{x})$

$\boxed{\mathcal{O}(n) \text{ complexity!}}$

If $D \ll n$

Sparse Approximation

- Locality-Sensitive Hashing (LSH)
  - With **high probability**, LSH hashes **close** data points into the **same bucket**, e.g., via $h(\boldsymbol{x}): \mathbb{R}^d \to \mathbb{R}$
  - E.g., by setting $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = \boldsymbol{1}$, angular distance-based LSH can be used to approx. $\exp(\boldsymbol{x}^\top \boldsymbol{y})$
    - Pairs with close angular distance have large attn
    - So, with high prob. in the same bucket
    - **Compute full attn** for pairs in **the same bucket**
  - There are **three ways** to construct LSH tables…



Efficient Transformers via Kernel Approximation

Low-rank Approximation

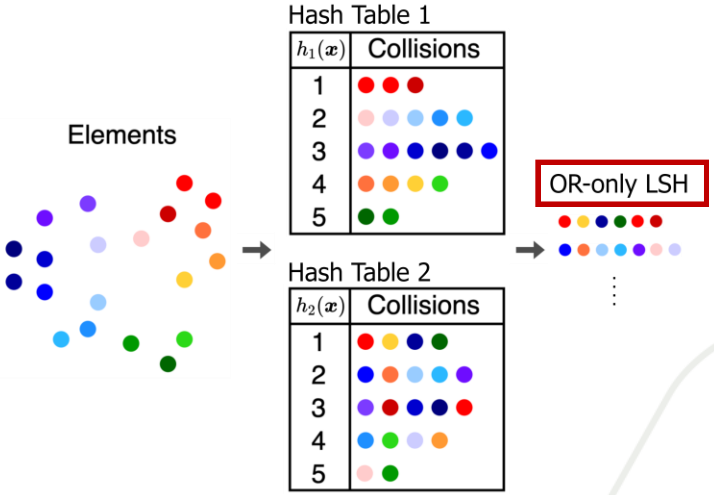
- Random Fourier Features (RFFs)
  - For any properly normalized positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $k(\boldsymbol{0}) = 1$
  - RFFs can approximate such kernels by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi: \mathbb{R}^d \to \mathbb{R}^D$
  - A most common example of such ψ is
    - $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \left(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \dots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}) \right)^\top$
    - $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$, and k^* is the Fourier transform of k
  - Consider RBF kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
    - Then k^* is a standard Gaussian
  - So, one can have
    - $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \hat{\psi}(\boldsymbol{x})^\top \hat{\psi}(\boldsymbol{y})$ with $\hat{\psi}(\boldsymbol{x}) = \exp\left(\frac{\|\boldsymbol{x}\|^2}{2}\right) \psi(\boldsymbol{x})$

$\boxed{\mathcal{O}(n) \text{ complexity!}}$

If $D \ll n$

Sparse Approximation

- Locality-Sensitive Hashing (LSH)
  - With **high probability**, LSH hashes **close** data points into the **same bucket**, e.g., via $h(\boldsymbol{x}): \mathbb{R}^d \to \mathbb{R}$
  - E.g., by setting $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = \boldsymbol{1}$, angular distance-based LSH can be used to approx. $\exp(\boldsymbol{x}^\top \boldsymbol{y})$
    - Pairs with close angular distance have large attn
    - So, with high prob. in the same bucket
    - **Compute full attn** for pairs in **the same bucket**
  - There are **three ways** to construct LSH tables…



Efficient Transformers via Kernel Approximation

Low-rank Approximation

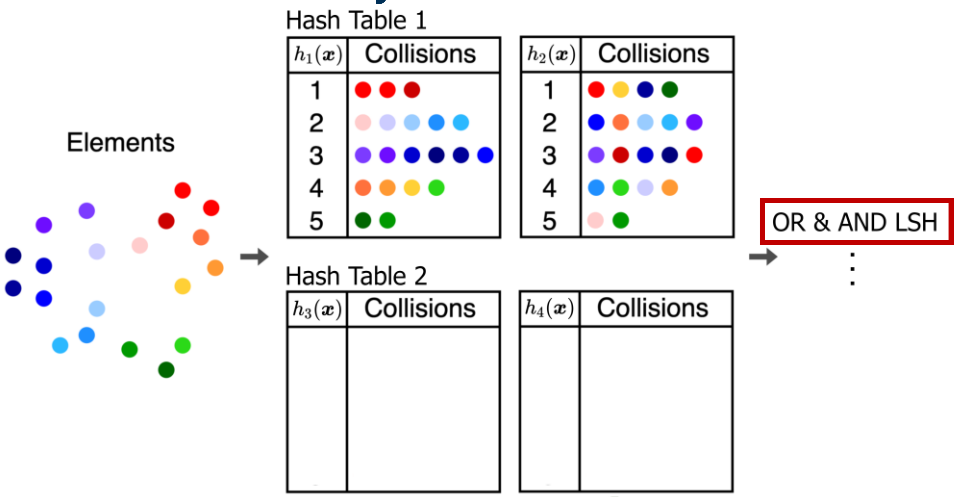
- Random Fourier Features (RFFs)
  - For any properly normalized positive definite shift-invariant kernel $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y})$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $k(\mathbf{0}) = 1$
  - RFFs can approximate such kernels by $k(\boldsymbol{x}, \boldsymbol{y}) \approx \psi(\boldsymbol{x})^\top \psi(\boldsymbol{y})$ with $\psi: \mathbb{R}^d \to \mathbb{R}^D$
  - A most common example of such ψ is
    - $\psi(\boldsymbol{x}) = \sqrt{\frac{2}{D}}\left(\sin(\boldsymbol{w}_1^\top \boldsymbol{x}), \cos(\boldsymbol{w}_1^\top \boldsymbol{x}), \ldots, \sin(\boldsymbol{w}_{D/2}^\top \boldsymbol{x}), \cos(\boldsymbol{w}_{D/2}^\top \boldsymbol{x})\right)^\top$
    - $\boldsymbol{w}_i \overset{iid}{\sim} k^*(\boldsymbol{w})$, and k^* is the Fourier transform of k
  - Consider RBF kernels $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
    - Then k^* is a standard Gaussian
  - So, one can have
    - $\exp(\boldsymbol{x}^\top \boldsymbol{y}) \approx \hat{\psi}(\boldsymbol{x})^\top \hat{\psi}(\boldsymbol{y})$ with $\hat{\psi}(\boldsymbol{x}) = \exp\left(\frac{\|\boldsymbol{x}\|^2}{2}\right)\psi(\boldsymbol{x})$

$\mathcal{O}(n)$ complexity!

If $D \ll n$

Sparse Approximation

- Locality-Sensitive Hashing (LSH)
  - With **high probability**, LSH hashes **close** data points into the **same bucket**, e.g., via $h(\boldsymbol{x}): \mathbb{R}^d \to \mathbb{R}$
  - E.g., by setting $\|\boldsymbol{x}\| = \|\boldsymbol{y}\| = \mathbf{1}$, angular distance-based LSH can be used to approx. $\exp(\boldsymbol{x}^\top \boldsymbol{y})$
    - Pairs with close angular distance have large attn
    - So, with high prob. in the same bucket
    - **Compute full attn** for pairs in **the same bucket**
  - There are **three ways** to construct LSH tables…



Efficient Transformers via Kernel Approximation

Low-rank Approximation

- Random Fourier Features (RFFs)

$\mathcal{O}(n)$ complexity!

Sparse Approximation

- Locality-Sensitive Hashing (LSH)

$\mathcal{O}(n \log n)$ complexity!

Which one is better for GDL tasks?

What properties can we utilize in GDL?

Local Inductive Bias!

Georgia Tech

HEPT: LSH-based Efficient Point Transformer

Low-Rank v.s. Sparse Approx. Under Local Inductive Bias

- To compare RFF-based and LSH-based methods
  - We want to compare their approximation accuracy **under the same computational budget**
- So, for each method we analyze the **tradeoff** between
  - Approximation error (ϵ)
  - Computational complexity (F)

As F changes, how would ϵ change?

- If we assume for the tasks considered
  - A point primarily interacts with its local neighbors
  - And the size of such neighborhood is $\mathcal{O}(\text{polylog}(n))$

Notation: $\tilde{\mathcal{O}}, \tilde{\Theta}$, 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.

1. 😭 *RFF results in an error $\epsilon = \tilde{\Theta}\left(\frac{n}{F}\right)$, which is consistently worse than LSH under subquadratic complexity, i.e., when $F = \tilde{o}(n^2)$*

2. ☹️ *LSH is better for tasks with local inductive bias, yielding $\epsilon = \tilde{\Theta}\left(\frac{1}{n}\right)$ 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 $\epsilon = \tilde{\mathcal{O}}\left(\exp\left(-\frac{F}{n\text{polylog}(n)}\right)\frac{1}{n}\right)$ which means that ϵ can be further exponentially reduced by almost linear complexity $F = \tilde{\mathcal{O}}(n)$.

HEPT: LSH-based Efficient Point Transformer

- So, we aim at utilizing both OR & AND LSH to build an efficient transformer
- Next, we introduce HEPT in detail

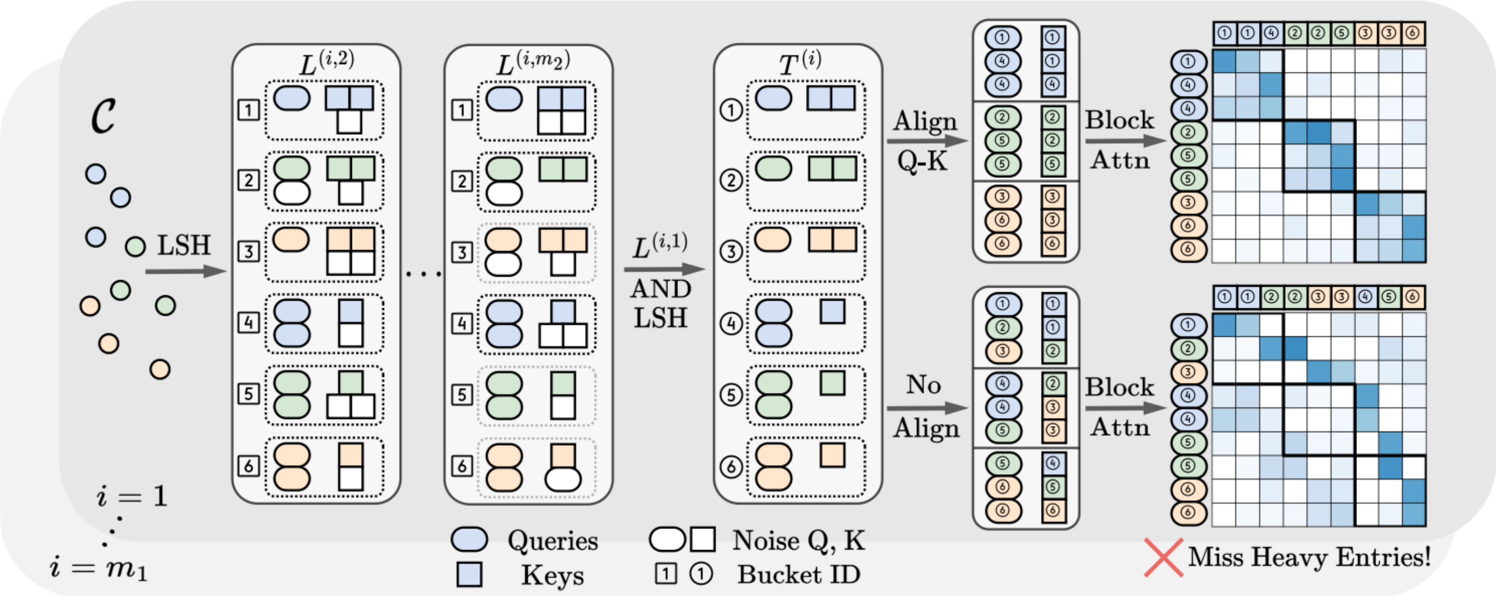


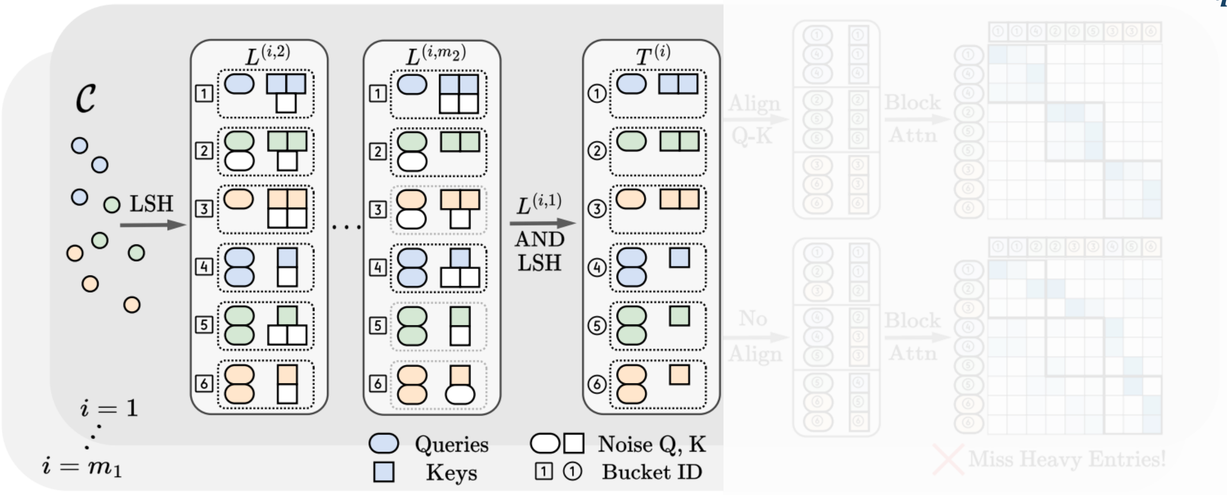
Figure 1: Pipeline of HEPT. Elements that share the same color represent points from the same local neighborhood. HEPT employs OR & AND LSH to minimize noise caused by individual hash functions. HEPT also integrates point coordinates as extra AND LSH codes for query-key alignment, maintaining computational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

HEPT: LSH-based Efficient Point Transformer

- We first introduce a new attn kernel w/ explicit local inductive bias
  - $k(\boldsymbol{q}_u, \boldsymbol{k}_v) = \exp\left(-\frac{1}{2}\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2\right)$
  - $\boldsymbol{q}_u = [\tilde{\boldsymbol{q}}_u \parallel \sqrt{2\omega}\boldsymbol{\rho}_u], \boldsymbol{k}_v = [\tilde{\boldsymbol{k}}_v \parallel \sqrt{2\omega}\boldsymbol{\rho}_v]$
  - $\tilde{\boldsymbol{q}}_u, \tilde{\boldsymbol{k}}_v \in \mathbb{R}^d$ are the **original query/key vectors** from vanilla transformer
  - $\boldsymbol{\rho}_u, \boldsymbol{\rho}_v \in \mathbb{R}^k$ are **point coordinates**
  - $\omega \in \mathbb{R}^+$ is **learnable parameter** to adjust attn scores
- This kernel
  - Enables the **use of E2LSH** in a principled way
    - i.e., an LSH method in Euclidian space
    - If $\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2$ is small (thus high attn), with high prob. they will have similar hash values
  - Exhibits **explicit local inductive bias**
    - i.e., the attention score $k(\boldsymbol{q}_u, \boldsymbol{k}_v) \to 0$ as $\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2$ increases

HEPT: LSH-based Efficient Point Transformer

- A kernel with **explicit local inductive bias**
  - $k(\boldsymbol{q}_u, \boldsymbol{k}_v) = \exp\left(-\frac{1}{2}\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2\right)$
- Then, we approximate this kernel via **OR & AND E2LSH**
  - We construct m_1 hash tables (OR LSH), each with m_2 hash functions (AND LSH)
    - Apply each hash function $h_{\boldsymbol{a}}(\boldsymbol{x}) = \boldsymbol{a} \cdot \boldsymbol{x}$, $\boldsymbol{a} \sim \mathcal{N}(0, \boldsymbol{I})$ for all queries/keys
  - Each query/key yields $m_1 \times m_2$ raw hash values
    - Denoted as $L_{\boldsymbol{q}_u}^{(ij)}, L_{\boldsymbol{k}_v}^{(ij)} \in \mathbb{R}$ for $i \in [m_1]$ and $j \in [m_2]$
    - If $\boldsymbol{q}_u$ and $\boldsymbol{k}_v$ hold **small** $\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2$(thus large attn), they are **likely** to have **close hash values** $L_{\boldsymbol{q}_u}^{(ij)}$ and $L_{\boldsymbol{k}_v}^{(ij)}$
    - For **each** of the m_1 **hash tables**, we **combine** all m_2 **hash values** to yield AND hash code $T_{\boldsymbol{q}_u}^{(i)}, T_{\boldsymbol{k}_v}^{(i)} \in \mathbb{R}$



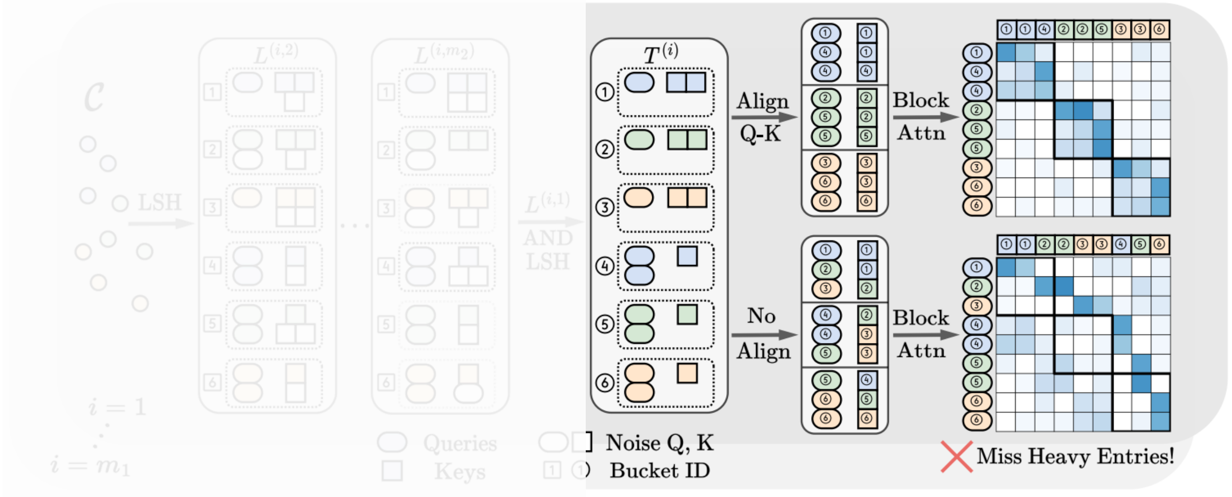
HEPT: LSH-based Efficient Point Transformer

- A kernel with explicit **local inductive bias**
  - $k(\boldsymbol{q}_u, \boldsymbol{k}_v) = \exp\left(-\frac{1}{2}\|\boldsymbol{q}_u - \boldsymbol{k}_v\|^2\right)$
- Then, we approximate this kernel via **OR & AND E2LSH**
  - If follow previous work for **computational regularity** by
    - sorting the AND hash code of queries $T_{\boldsymbol{q}_u}^{(i)}$ and keys $L_{\boldsymbol{k}_v}^{(ij)}$, separately
    - and truncating the buckets to be equal-sized
  - We find a **misalignment** issue…
    - We integrate **point coordinates** in the AND hash code to align Q-K
    - Then sort & truncate buckets and compute pairwise attn in each bucket

Guaranteed low approx. error!

$\mathcal{O}(n \log n)$ complexity!

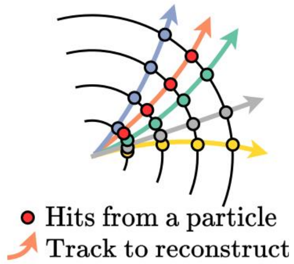
Hardware-friendly!



Georgia Tech

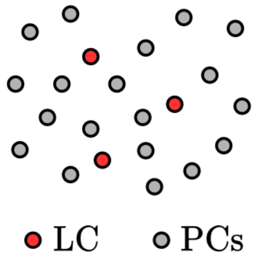
HEPT: LSH-based Efficient Point Transformer

- Datasets



(a) Charged Particle Tracking

- The datasets are derived from the TrackML challenge
- The task is formulated as a **representation learning** problem
  - i.e., learn close embeddings for points originating from the same particle

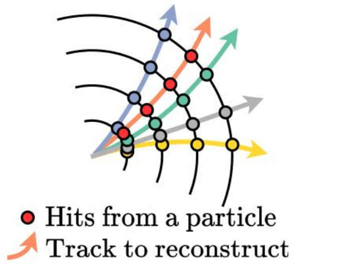


(b) Pileup Mitigation

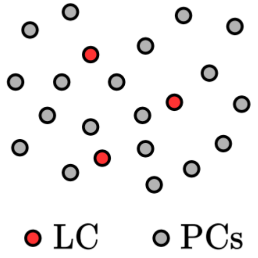
- The dataset is generated with 200PU
- The task is formulated as a **binary point classification** problem
  - i.e., predict if a neutral particle is from pileup collisions or not

HEPT: LSH-based Efficient Point Transformer

- Experiments



(a) Charged Particle Tracking



(b) Pileup Mitigation

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
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**†

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
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**	**12.23**
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**†

Better accuracy than SOTA GNNs & up to **100x speedup***!

*Implemented purely PyTorch, may be further accelerated by applying quantization, FlashAttn, hardware-aware co-design, etc.

Conclusion

Conclusion

- We analyze the error-computation tradeoff of RFF and LSH
  - ✓ We highlight the superiority of LSH-based methods in GDL tasks
  - ✓ LSH with both OR & AND construction yields the best performance
- We propose a novel efficient point transformer HEPT
  - ✓ SOTA accuracy
  - ✓ Up to 100x faster on GPUs (NVIDIA Quatro RXT 6000) compared to SOTA GNNs
- Our code is released
  - ✓ https://github.com/Graph-COM/HEPT
- Our paper is online
  - ✓ https://arxiv.org/abs/2402.12535

Thank you!

Georgia Tech

Questions?