Fully Dynamic Algorithms for Chamfer Distance

Gramoz Goranci[1], Shaofeng Jiang[2], Peter Kiss[1], Eva Szilagyi[1], **Qiaoyuan Yang**[2]

[1]University of Vienna [2]Peking University

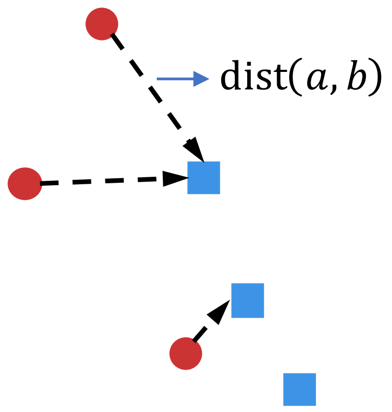
Chamfer Distance

Input: two sets of points $A, B \subset \mathbb{R}^d$

Chamfer Distance:

$$\text{dist}_{\text{CH}}(A, B) = \sum_{a \in A} \text{dist}(a, B),$$

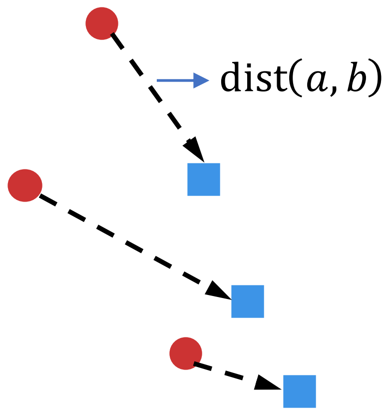
where $\text{dist}(a, B) = \min_{b \in B} \text{dist}(a, b)$ and $\text{dist}(a, b) = \|a - b\|_p$



Chamfer distance

Motivation:

- efficient alternative to Earth-Mover Distance (EMD)
  - EMD requires 1 to 1 mapping, while Chamfer distance is relaxed
  - Chamfer distance is easier to compute than EMD
- used for deep-learning loss, 3D reconstruction and medical imaging



EMD

Fully Dynamic Setting

Static setting: $(1+\epsilon)$-approximation in $\tilde{O}(nd \cdot \epsilon^{-2})$ time [BIJ+23, FI25] near-linear

Our goal: maintain an approximation to Chamfer distance under **dynamic updates**

Update operations: insert/delete a point in A or B

Query: return an approximation to $\mathrm{dist}_{\mathrm{CH}}(A, B)$

Naïve solution: recompute from scratch after each update, with $\tilde{O}(nd \cdot \epsilon^{-2})$ time

Challenge: no existing method breaks the linear update time barrier with constant approximation (even in 2D)

Our Result

breaks the linear update time barrier

- **Low dimension:** $(1+\epsilon)$-approximation in $\tilde{O}(\epsilon^{-d})$ update time
- **High dimension:** $\mathrm{poly}(1/\epsilon)$-approximation in $\tilde{O}(dn^{\epsilon})$ update time

General Theorem. Assume there is a $\big(1+\Theta(\alpha)\big)$-approximate nearest neighbor (NN) oracle on B with query/update time τ. Then there is a dynamic algorithm,

- update: supports insertion/deletion in A and B in $\tilde{O}(\tau)$ time;
- query: in $\tilde{O}((\tau+d)\epsilon^{-2}\max\{1,\alpha^2\})$ time , returns a $(1+\alpha+\epsilon)$-approximation to $\mathrm{dist}_{\mathrm{CH}}(A,B)$.

Technique

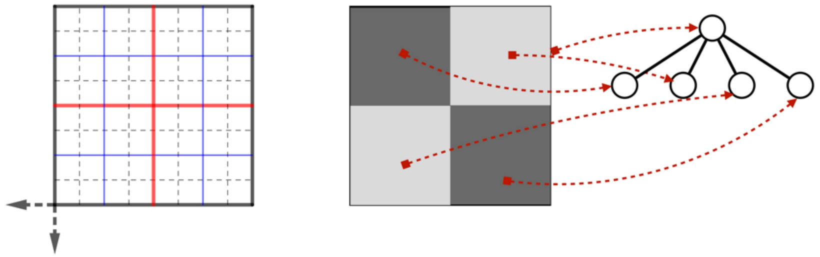
- **Recall:** $\mathrm{dist}_{\mathrm{CH}}(A, B) = \sum_{a \in A} \mathrm{dist}(a, B)$ a sum of nearest-neighbor distances
- We estimate this sum of distance via **importance sampling**
  - [BIJ+23] employed importance sampling to estimate Chamfer distance in static setting
- Importance sampling framework
  - **Sampler:** (1) for each $a \in A$, compute $\widehat{d(a)}$, an $O(\mathrm{polylog}\, n)$-approximation to $\mathrm{dist}(a, B)$

    (2) take $\tilde{O}(1)$ samples S from A with probability proportional to $\widehat{d(a)}$
  - **Estimator:** let $\widehat{D} = \sum_{a \in A} \widehat{d(a)}$, estimate $\mathrm{dist}_{\mathrm{CH}}(A, B)$ by $\frac{1}{|S|} \sum_{a \in S} \frac{\widehat{D}}{\widehat{d(a)}} \mathrm{dist}(a, B)$
  - **Guarantee:** yields a $(1 + \epsilon)$-approximation (w.h.p.)
- We need a **dynamic importance sampler** and a **dynamic estimator**

Dynamic Sampler

Idea: use an efficient tree embedding and do sampling on the tree metric

- An efficient realization is a **randomly shifted quadtree**
  - **Distortion:** $O(\log^2 n)$
  - **Update time:** $\tilde{O}(d)$



- **Recall:** for each $a \in A$, we need $\widehat{d(a)}$, an $O(\text{polylog}\, n)$-approximation to $\text{dist}(a, B)$
- It suffices to use the distance $\text{dist}_T(a, B)$ on the tree as $\widehat{d(a)}$
- Let c_a be the smallest quadtree cell containing a and any point of B, then

$$\text{dist}_T(a, B) = O(c_a\text{'s side length})$$

c_a's side length can be maintained dynamically

Dynamic Estimator

Recall (static estimator):

- $\frac{1}{|S|}\sum_{a \in S} \frac{\widehat{D}}{\widehat{d(a)}} \operatorname{dist}(a, B)$ is a $(1+\epsilon)$-approximation to $\operatorname{dist}_{\mathrm{CH}}(A, B)$ (w.h.p.)

Our estimator (dynamic setting):

- $\widehat{d(a)}$ is maintained in the dynamic sampler; it remains to compute $\operatorname{dist}(a, B)$
- approximate $\operatorname{dist}(a, B)$ by a $(1+\Theta(\alpha))$-approximate NN oracle

Guarantee: the result is a $(1+\alpha+\epsilon)$-approximation to $\operatorname{dist}_{\mathrm{CH}}(A, B)$ (w.h.p.)

Experiments

Updates are simulated via a sliding window over B (insertions/deletions)

Dataset: we use four real-world datasets; default setting: static A, dynamic B

Table 1: Specifications of datasets and experiment parameters.

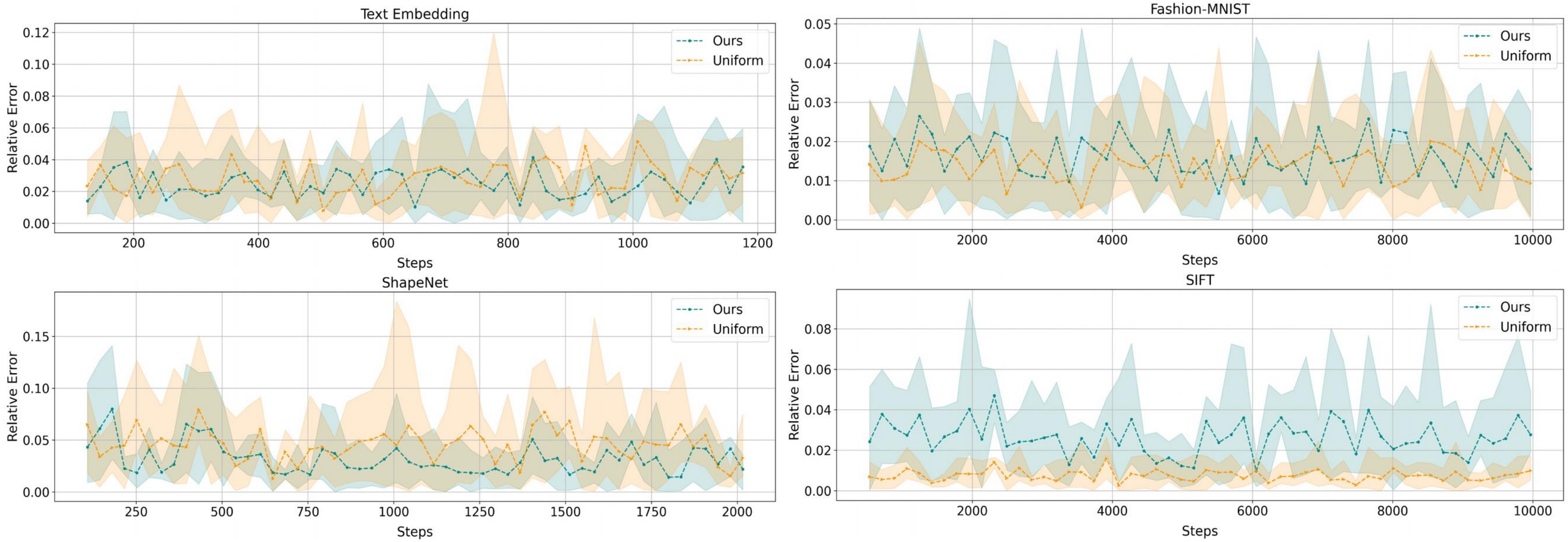
dataset	dimension d	$\|A\|$	$\|B\|$	window size	sample size
Text Embedding	300	~1.9k	~1.2k	100	150
ShapeNet	3	~2k	~2k	100	150
Fashion-MNIST	784	60k	10k	500	200
SIFT	128	1000k	10k	500	300

Baselines:

- **Benchmark** (exact, brute-force) — for each update of B, recompute $\mathrm{dist}_{\mathrm{CH}}(A, B)$ in $O(d|A|)$ time, since there are at most $|A|$ points are affected
- **Uniform** — replace our important sampling with a uniform sampling

Relative Error Curves

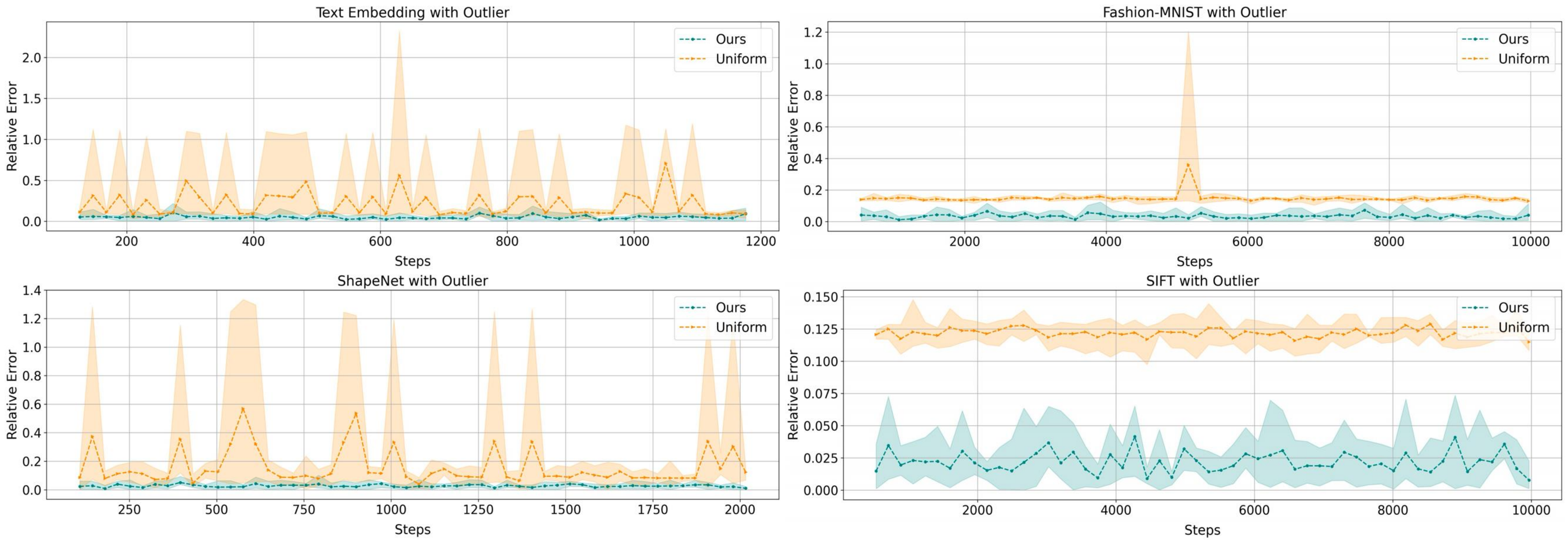
- **Compared with Benchmark:** ≤10% error with 0.03%-5% points as samples
- **Compared with Uniform:** comparable error and variance



Relative error curves for datasets; these are independently run for 5 times, and report the average (the dot), max and min value (the shaded area)

Relative Error Curves: With Planted Outlier

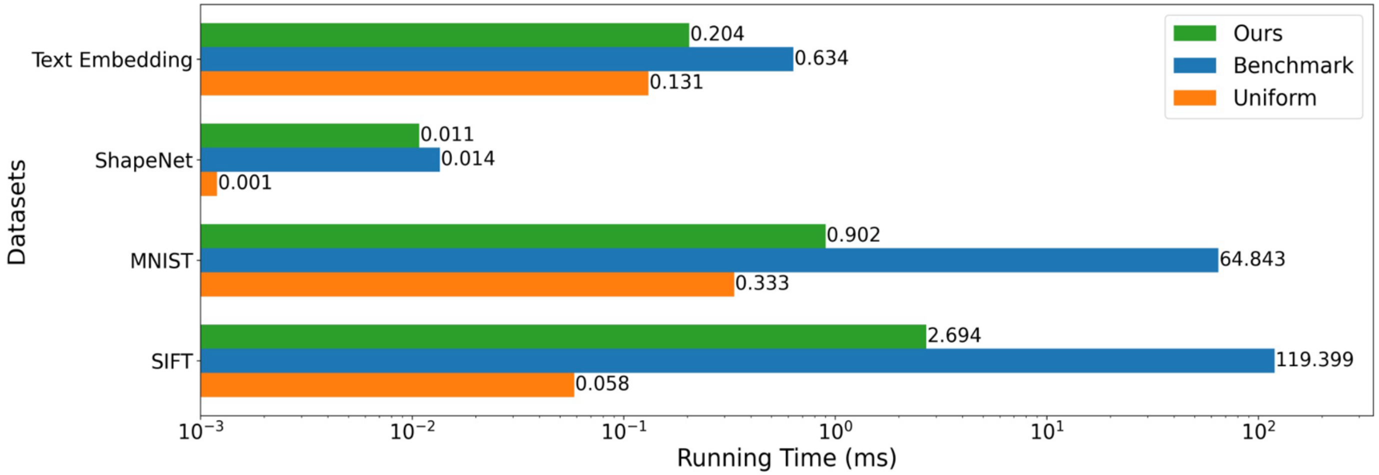
- **Compared with Benchmark:** still ≤10% error with 0.03%-5% points as samples
- **Compared with Uniform:** a clear advantage in both error and variance



Relative error curves for datasets with planted outliers

Running Time

- Orders of magnitude faster than Benchmark on large datasets



Average running time per window update for all algorithms on datasets

Thanks!