Short Paper

Learned Accelerator Framework for Angular-Distance-Based High-Dimensional DBSCAN

Yifan Wang
University of Florida
wangyifan@ufl.edu

Daisy Zhe Wang
University of Florida
daisyw@ufl.edu

ABSTRACT

Density-based clustering is a commonly used tool in data science. Today many data science works are utilizing high-dimensional neural embeddings. However, traditional density-based clustering techniques like DBSCAN have a degraded performance on high-dimensional data. In this paper, we propose LAF, a generic learned accelerator framework to speed up the original DBSCAN and the sampling-based variants of DBSCAN on high-dimensional data with angular distance metric. This framework consists of a learned cardinality estimator and a post-processing module. The cardinality estimator can fast predict whether a data point is core or not to skip unnecessary range queries, while the post-processing module detects the false negative predictions and merges the falsely separated clusters. The evaluation shows our LAF-enhanced DBSCAN method outperforms the state-of-the-art efficient DBSCAN variants on both efficiency and quality.

1 INTRODUCTION

Today's data science research benefits significantly from neural embeddings that are high-dimensional vectors generated by deep neural models. As a widely applied technique in data science, clustering has been associated with embeddings, e.g., [22, 28] learn effective passage embeddings with clustering, [19, 23] utilize clustering to accelerate the similarity search over embeddings, etc.

As a representative clustering algorithm, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [7] has a long history of being applied on low-dimensional spatial data (2D or 3D). However, DBSCAN usually has a low efficiency, caused by the compute-intensive range queries and becoming more significant on high-dimensional data due to the curse of dimensionality. Specifically, DBSCAN considers clusters to be high-density areas separated by low-density areas. Based on this, the algorithm repeatedly expands each cluster to its neighboring high-density areas (where the points are called *core points*) until the cluster is completely surrounded by low-density areas (where each point is either a *non-core point* or a noise). For each point, DBSCAN has to do a heavy range search to determine whether it is core or not, which requires intensive computation and limits its application in large-scale high-dimensional data analysis. To improve the efficiency of DBSCAN, previous works propose many variants, e.g., sampling-based DBSCAN variants [5, 11, 12, 14, 21] improve the efficiency by executing the heaviest computation within a small subset instead of the whole dataset, some other works reduce the latency by accelerating the range queries in DBSCAN [15, 26], [1–4, 8, 9] prune unnecessary distance computation during the clustering, etc. But many of them are designed for low to middle dimensional data (mostly less than 100-dimensions). Therefore they are still not suitable for high-dimensional neural embeddings with hundreds to thousands of dimensions (e.g., the 768-dim BERT [6] embeddings).

In this paper, we solve this problem by skipping the range queries for non-core and noise points, given that only the number of neighbors is needed to confirm the point is non-core or noise. And this can be done by cardinality estimation, i.e., the techniques to estimate the number of results before executing a query. In multi-dimensional data management, cardinality estimation is usually used to predict the number of neighbors from a distance-based range query without executing it, and the estimation results can help optimize the critical operations like range search and similarity join. Traditional cardinality estimation for range queries relies on sampling or kernel density estimation [17]. Recent works apply advanced machine learning to it and propose the *learned cardinality estimation* techniques. They are normally based on regression models from non-deep regressors (e.g., XGBoost) to deep regressors (e.g., deep neural networks), whose input is the query point and the distance threshold (i.e., the range), and output is the estimated number of neighbors in that range. By learning the data distribution, learned cardinality estimation makes more accurate prediction than the traditional approaches. The state-of-the-art learned cardinality estimation methods deploy various deep regression models, e.g., Convolutional neural network [18], Recursive Model Index [13, 24], Deep Lattice Network [27], CardNet [24], SelNet [25], etc., which perform effectively on high-dimensional data. And they can achieve high predicting efficiency both theoretically and practically. In theory, when a model structure is fixed, its prediction time complexity is constant with the data scale, while in practice, the models can be significantly accelerated by GPU. The training time is not an issue due to the generalization capability of the neural models, i.e., a trained estimator can be used on any other dataset with similar distribution. With a learned cardinality estimator, whether a point is core or not can be predicted before executing the range query, by which the unnecessary computation on non-core and noise points will be effectively reduced.

Particularly, our approach is designed for clustering based on angular distance, like cosine distance, and in this paper we will not investigate other distance metrics like Euclidean distance, due to two reasons: (1) Angular distance is worth being specifically studied. In neural embedding based applications, cosine and Euclidean distances are the dominant distance metrics for measuring embedding similarity. So focusing on cosine distance is enough to benefit a wide range of applications. (2) Our idea is theoretically more suitable for this metric. Angular distance is usually bounded, e.g., cosine distance is within the range $0 \sim 2$, which makes training of the cardinality estimator more effective than using Euclidean distance. Specifically, a regressor normally makes better predictions when the training set covers more possible input values, which is hard for Euclidean distance whose value range is infinite (i.e., $0 \sim +\infty$), but easier for the bounded cosine distance. For example, in our evaluation we construct the training set using cosine distance thresholds from 0.1 to 0.9, which is enough to cover most cases. Therefore, other distances are out of scope for this paper. However, our method does not have a hard constraint on the distance metric, so we may explore Euclidean distance in future work.

We propose **LAF**, a generic **L**earned **A**ccelerator **F**ramework to speed up DBSCAN and its sampling-based variants based on angular distance. LAF enhances the algorithm efficiency by placing an extra cardinality estimation step before each range query. If a point is predicted as non-core/noise, the range query for it will be skipped to reduce the computation. This approach works not only on DBSCAN, but also on its sampling-based variants, as the same kind of computation waste also exists in processing the sampled subset. LAF also includes a post-processing module to compensate for the clustering quality loss by detecting and merging the wrongly separated clusters caused by the false negative predictions (i.e., predicting core points as non-core/noise). To our best knowledge, we are one of the first studies that improve the efficiency of high-dimensional DBSCAN-like clustering by cardinality estimation. And we have open-sourced the code at https://github.com/wyfunique/LAF-DBSCAN.

Our LAF-enhanced DBSCAN outperforms the state-of-the-art approximate DBSCAN variants in the evaluation. Specifically, LAF-enhanced DBSCAN achieves up to 2.9x speedup for DBSCAN and is 60% ~ 140% faster than the state-of-the-art approximate DBSCAN variants, with high clustering quality on high-dimensional vectors, and the selected sampling-based DBSCAN variant is also accelerated significantly by LAF (i.e., up to 6.7x speedup) with only tiny or no quality loss. The main contributions of this paper are as follows:

(1) We develop LAF, a generic learned accelerator framework to accelerate a wide range of DBSCAN-like clustering algorithms.
(2) We propose a novel efficient high-dimensional DBSCAN algorithm using the framework.
(3) We conduct experiments on popular high-dimensional datasets and show the high performance of our proposed algorithm and the usefulness of LAF.

2 THE APPROACH

2.1 DBSCAN, LAF and enhanced DBSCAN

Algorithm 1 shows the DBSCAN pseudocode in black text (the red text is inserted by LAF, which is introduced below). DBSCAN classifies the data points as core, non-core and noise points, depending on the number of their neighbors within a range. Given a distance function $d(\cdot,\cdot)$ and a distance threshold ϵ, DBSCAN does a range query for each data point P to find its neighbors $\mathcal{N} = \{Q | d(P,Q) < \epsilon\}$. If it has at least τ neighbors, point P is a core point and the current cluster will be expanded to its neighbors; otherwise P is non-core or noise, and the current cluster will not grow from P to its neighbors. When the current cluster cannot grow any more, the next cluster will start from some other un-clustered core points. Such a process is repeated until there are no more unclassified (i.e., *undefined* in Algorithm 1) points. The difference between non-core and noise points is whether it has a core point neighbor. If a point itself is not core but has at least one neighbor that is core point, then it is a non-core point and will be part of the cluster boundary, while a noise point has no core point neighbor and will not be classified into any cluster. For simplification, in the rest of this paper we denote both of the non-core and noise points as *stop points* when it is unnecessary to distinguish between them.

LAF works as a plugin to the target algorithm: (1) The cardinality estimator is placed right before each range query, and the range query will be executed only if the current point is predicted as core. (2) The post-processing module is inserted at the end of the clustering to detect false negative predictions which wrongly estimate core points as not, and merge the clusters separated by such false stop points. This is to compensate for the effectiveness loss caused by the prediction error. Based on LAF, we implement an efficient high-dimensional DBSCAN, called *LAF-enhanced DBSCAN* (a.k.a, LAF-DBSCAN). We show its pseudocode in Algorithm 1 and use red text to highlight the inserted lines by LAF, while the other lines are the same as original DBSCAN. We also develop *LAF-DBSCAN++*, an enhanced version of DBSCAN++ [11] by LAF, to present the capability of LAF for accelerating variants of DBSCAN. The details are discussed in Section 3.

Basically, LAF inserts three critical functions: CardEst, UpdatePartialNeighbors and PostProcessing, as well as a map $\mathcal{E}$ recording all the points which are predicted as stop points (called *predicted stop points*) and their *partial neighbors*. Here we use the term "partial neighbors" because for each predicted stop point, $\mathcal{E}$ does not record all its neighbors, but only a subset generated by UpdatePartialNeighbors. CardEst is simply using cardinality estimator to predict number of the range query results, while the other two functions are both for the post-processing. Note that we do not use the exact value of τ with CardEst to predict whether a point is core or not. Instead, τ is multiplied with a positive factor α to threshold the predicted cardinality, as shown in line 6 and 22 of Algorithm 1. Here α is used to adjust the false positive and false negative rates, such that users can control the prediction error and manipulate the speed-quality trade-off. Specifically, when α increases, false negative rate increases as more predictions become lower than the threshold, resulting in higher speed and lower quality. When α decreases, false positive rate increases, leading to lower speed and higher quality.

If a point is predicted to be stop point (line 6-9, 26-27), the corresponding entry will be added into $\mathcal{E}$, otherwise the range query will be executed and the point will be double checked with the query results. At the same time, $\mathcal{E}$ will be updated by UpdatePartialNeighbors using the query results. Finally the post-processing uses $\mathcal{E}$ to update the clustering results C.

2.2 Post-processing strategy

$\mathcal{E}$ records the *partial neighbors* (i.e., a subset of the true neighbors) for each predicted stop point. It is filled by UpdatePartialNeighbors in such a way (as shown in Algorithm 2): if a predicted stop point P_n is found by another point P as neighbor, then P is also neighbor of P_n and will be added to $\mathcal{E}(P_n)$. Function PostProcessing (Algorithm 3) detects the false predicted stop points and merges the clusters separated by those points. Specifically, a point P in $\mathcal{E}$ is a false negative if it has at least τ partial neighbors (line 2). In such case PostProcessing will randomly select a cluster around it as the destination cluster (line 3-4), and merge the rest wrongly separated clusters to the destination (line 5).

3 EXPERIMENTS

3.1 Experiment settings

Environment: A Lambda Quad workstation with 28 3.30GHz Intel Core i9-9940X CPUs, 4 RTX 2080 Ti GPUs and 128 GB RAM.

Algorithm 1 LAF-enhanced DBSCAN (LAF-DBSCAN)

```
Input: Dataset $\mathcal{D}$, distance function $d(\cdot,\cdot)$, distance threshold $\epsilon$, minimum number of neighbors $\tau$, error factor $\alpha$
Output: the map from points to their cluster IDs $C$
1:  Cluster ID c := 0
2:  Map from predicted stop points to partial neighbors $\mathcal{E}$:=$\emptyset$
3:  for each point P in $\mathcal{D}$ do $C$(P) := undefined
4:  for each point P in $\mathcal{D}$ do
5:      if $C$(P) $\neq$ undefined then continue
6:      if CardEst(P) < $\alpha\tau$ then
7:          $C$(P) := noise
8:          if P not in $\mathcal{E}$ then $\mathcal{E}$(P) := $\emptyset$
9:          continue
10:     Neighbors $\mathcal{N}$ := RangeQuery($\mathcal{D}$, $d$, P, $\epsilon$)
11:     $\mathcal{E}$ := UpdatePartialNeighbors(P, $\mathcal{N}$, $\mathcal{E}$)
12:     if $|\mathcal{N}|$ < $\tau$ then
13:         $C$(P) := noise
14:         continue
15:     c := c + 1
16:     $C$(P) := c
17:     $\mathcal{S}$ := $\mathcal{N}$ - {P}
18:     for each point Q in $\mathcal{S}$ do
19:         if $C$(Q) = noise then $C$(Q) := c
20:         if $C$(Q) $\neq$ undefined then continue
21:         $C$(Q) := c
22:         if CardEst(Q) $\geq$ $\alpha\tau$ then
23:             $\mathcal{N}$ := RangeQuery($\mathcal{D}$, $d$, Q, $\epsilon$)
24:             $\mathcal{E}$ := UpdatePartialNeighbors(Q, $\mathcal{N}$, $\mathcal{E}$)
25:             if $|\mathcal{N}|$ $\geq$ $\tau$ then $\mathcal{S}$ := $\mathcal{S}$ $\cup$ $\mathcal{N}$
26:         else
27:             if Q not in $\mathcal{E}$ then $\mathcal{E}$(Q) := $\emptyset$
28: $C$ := PostProcessing($C$, $\mathcal{E}$, $\tau$)
29: return $C$
```

Algorithm 2 UpdatePartialNeighbors

```
Input: Data point P, its neighbors $\mathcal{N}$, the map $\mathcal{E}$
Output: the updated $\mathcal{E}$
1: for each neighbor $P_n$ in $\mathcal{N}$ do
2:     if $P_n$ is in $\mathcal{E}$ then $\mathcal{E}(P_n)$ := $\mathcal{E}(P_n)$ $\cup$ {P}
3: return $\mathcal{E}$
```

Algorithm 3 PostProcessing

```
Input: the map $C$ from point to cluster, the map $\mathcal{E}$, $\tau$
Output: the updated $C$
1: for each point P in $\mathcal{E}$ do
2:     if $|\mathcal{E}(P)|$ $\geq$ $\tau$ then
3:         Randomly select a non-noise neighbor P' in set $\mathcal{E}$(P)
4:         Destination cluster ID $c'$ := $C$(P')
5:         Merge the clusters of $\mathcal{E}$(P) into the destination cluster.
6: return $C$
```

Datasets: Table 1 provides an overview for our evaluation datasets, reporting their sizes, data dimensions, error factors used in evaluation, and their vector types. We introduce more details here:

(1) NYTimes: 300k bag-of-words vectors of NYTimes news articles. We randomly sample 150k vectors from them, normalize the samples and reduce their dimension to 256 through Gaussian random projection, which is the same way as *ANN-Benchmarks*[1]. The resulting dataset is named *NYT-150k*.
(2) Glove: 1.2M word embeddings (200-dimensional) pre-trained on tweets. We sample 150k vectors from them and name the sampled dataset *Glove-150k*.
(3) MS MARCO [16]: a benchmarking dataset for passage retrieval, including 8.8M passages. We follow a similar way to [23] to process this dataset, i.e., generating a 768-dimensional embedding for each passage using the same deep model as [23], sampling the embeddings into several subsets and naming them as “MS-” followed by the size (e.g., “MS-100k” includes around 100k embeddings). In this paper we sample 3 datasets, MS-50k, MS-100k and MS-150k.

[1]https://github.com/erikbern/ann-benchmarks

In addition, we normalize all the data vectors and split each dataset into training and testing sets by a ratio of 8:2. For each dataset, we first train the learned cardinality estimator on the training set, then evaluate all the methods on the corresponding testing set, i.e., all the reported experiment results are collected on those testing sets.

Dataset	#Points	Dim	α	Type
NYT-150k	150,000	256	1.15	Bag-of-words
Glove-150k	150,000	200	2.0	Word embedding
MS-150k	152,185	768	7.7	Passage embedding
MS-100k	107,400	768	2.0	Passage embedding
MS-50k	53,700	768	1.5	Passage embedding

Table 1: Evaluation dataset information, including the number of points (*#Points*), data dimension (*Dim*), error factor α of LAF-DBSCAN on each of them, and the vector type (*Type*).

Metrics: As discussed in Section 1, the distance metric in the evaluation is cosine distance. For some baselines which support Euclidean distance only, since all data points are normalized, we use Equation 1 to convert cosine distance (d_{cos}) into Euclidean distance (d_{euc}), such that the distances in our methods are equivalent to those in the baselines. For example, by the equation, when $d_{cos} = 0.5$, the equivalent $d_{euc} = 1.0$, so if we set the distance threshold $\epsilon = 0.5$ in our methods, the threshold in the baselines will be set as 1.0.

$$d_{euc}(\vec{u},\vec{v}) = \sqrt{2d_{cos}(\vec{u},\vec{v})} \quad (\text{if } \|\vec{u}\| = \|\vec{v}\| = 1) \tag{1}$$

The evaluation metrics include efficiency and effectiveness metrics. For efficiency, the metric is elapsed time of clustering (including the cardinality estimator prediction time and excluding its training time). For effectiveness, the metrics are (1) adjusted RAND index (ARI) [10] and (2) adjusted mutual information score (AMI) [20], computed against the ground truth. A higher score means a better clustering quality. Here we use the clustering results of original DBSCAN as ground truth.

Our methods: In addition to LAF-DBSCAN, we also use LAF to accelerate a sampling-based DBSCAN variant, DBSCAN++ [11]. The resulting method is named *LAF-DBSCAN++*, whose goal is to present that LAF works not only on DBSCAN but also on its sampling-based variants (as mentioned in Section 1). So it just acts as an auxiliary method and our major method is still LAF-DBSCAN in the evaluation. In both methods, the cardinality estimator model is an RMI [13] with three stages, respectively including 1, 2, 4 fully-connected neural networks from top to bottom stage. Each neural network has 4 hidden layers whose widths are 512, 512, 256, and 128. Such an estimator has been used as a strong baseline in [24], from where we borrow the code directly. On each training set, the cardinality estimator is trained for 200 epochs with batch size 512.

Though there are also other learned cardinality estimators, like CardNet [24] and SelNet [25], we will not explore which estimator is the best for our methods, as it is out of scope for this paper. Specifically, the goal of this paper is to reveal the potential of such a new idea on speeding up DBSCAN, and in our evaluation the RMI has already performed well enough to demonstrate the potential.

Baselines: The baselines are described as follows:

(1) DBSCAN: the original DBSCAN. Its clustering results are used as the ground truth for other methods.
(2) DBSCAN++ [2] [11]: an approximate DBSCAN variant that speeds up DBSCAN by sampling the dataset and limiting the heaviest computation within the samples. Specifically, DBSCAN++ samples a subset of data points, within which the core points are detected w.r.t. the entire dataset. Then the clusters first grow around those core points within the subset, and finally all the unclassified points outside the subset are directly assigned to their closest core points. Our LAF-DBSCAN++ method is built on top of DBSCAN++.
(3) KNN-BLOCK DBSCAN [3] [3]: an approximate DBSCAN variant which improves efficiency by pruning unnecessary distance computation with K-nearest neighbor queries. We denote it as "KNN-BLOCK" in the tables and figures.
(4) BLOCK-DBSCAN [4] [2]: a method similar to KNN-BLOCK DBSCAN, but facilitated by cover tree based range queries.
(5) ρ-approximate DBSCAN [5] [8, 9]: an approximate DBSCAN variant which accelerates DBSCAN by relaxing the density criteria with an approximation factor ρ ($\rho > 0$).

Our methods and the baselines are all implemented mainly in C++.

Parameters: The key parameters in all experiments (except the trade-off evaluation) are set as follows, while their settings in the trade-off are introduced separately in Section 3.4. (1) Distance threshold ϵ and neighbor threshold τ are set dynamically in different experiments, which will be explicitly stated. (2) For LAF-DBSCAN, the error factor α is set in an ad-hoc manner for different datasets, as reported in Table 1. For LAF-DBSCAN++, its α is fixed to be 1.0. (3) For DBSCAN++, the sample fraction p is automatically set based on the ratio of predicted core points. Specifically, we first get the ratio of points that are predicted as core by the cardinality estimator (denoted by R_c), then $p = \delta + R_c$, where δ is a user-determined offset ranging from 0.1 to 0.3. In our evaluation, the final p normally ranges within $0.2 \sim 0.6$. And p of LAF-DBSCAN++ keeps identical to DBSCAN++. (4) For KNN-BLOCK DBSCAN, we control two parameters of the k-means tree for KNN search: branching factor (set as 10) and ratio of leaves to check (set as 0.6). (5) For BLOCK-DBSCAN, we control the basis of the cover tree (set as 2) and the maximum iterations when computing the minimum distance between inner core blocks (i.e., *RNT* in [2], set as 10). (6) For ρ-approximate DBSCAN, we set $\rho = 1.0$.

3.2 Representative (ϵ, τ) and proper α selection

We select the proper ϵ and τ according to the *noise ratio*, i.e., the portion of noise points in each dataset. A proper (ϵ, τ) should lead to (1) a low to middle noise ratio and (2) enough number of clusters, since the clustering makes no sense when too many noises exist or most points are grouped into very few clusters. So we do a grid search to select the values of (ϵ, τ) which make (1) noise ratio smaller than 0.6 and (2) the number of clusters large than 20 in most datasets. Table 2 shows part of the statistics, where each cell includes a pair *(noise ratio, number of clusters)* for the corresponding case. The cells satisfying the conditions are highlighted, for example, (0.55, 5) and (0.6, 5) are both proper (ϵ, τ) since either of them makes at least 2 out of 3 datasets satisfy the conditions; while (0.5, 5) and (0.7, 5) should be avoided. Finally, we choose three (ϵ, τ) values to report throughout this paper: (0.5, 3), (0.55, 5) and (0.6, 5).

(ϵ, τ)	MS-50k	MS-100k	MS-150k
(0.5, 3)	(0.63, 654)	**(0.53, 1071)**	**(0.47, 1225)**
(0.5, 5)	(0.83, 174)	(0.72, 348)	(0.64, 380)
(0.55, 5)	(0.65, 183)	**(0.48, 223)**	**(0.39, 175)**
(0.6, 5)	**(0.38, 92)**	**(0.21, 70)**	**(0.15, 47)**
(0.7, 5)	(0.005, 1)	(0.0007, 1)	(0.0004, 1)

Table 2: Part of the statistics about noise ratio and number of clusters. They are collected by running DBSCAN with different ϵ and τ on each dataset. In the table each cell below the dataset name is a pair *(noise ratio, total number of clusters)*, and the proper value pairs are highlighted by bold text.

In this section we also discuss the proper setting of error factor α. Basically, there is no quantifiable way to predict the best α, as it depends on the dataset. The method we use for this paper is grid search, and our observation can help guide the users: when the vector type is fixed (e.g., dense neural embedding), α should be larger for the larger dataset size or higher data dimension. This can be observed in Table 1 on Glove and the three MS datasets. The reason is probably the bias in training set. For example, according to Table 2, with the increasing data scale, the noise ratio decreases, meaning the fraction of core points increases. Such a bias in training makes the cardinality estimator more likely to predict a larger value. Therefore the α should also increase accordingly.

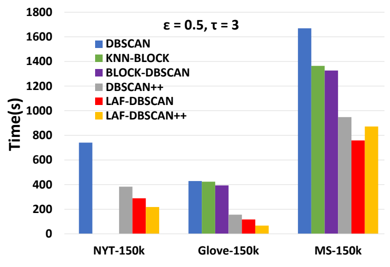
3.3 Efficiency and effectiveness evaluation

We first evaluate the efficiency and effectiveness of each method on the three largest datasets, NYT-150k, Glove-150k and MS-150k. Table 3 reports the clustering quality via ARI and AMI scores

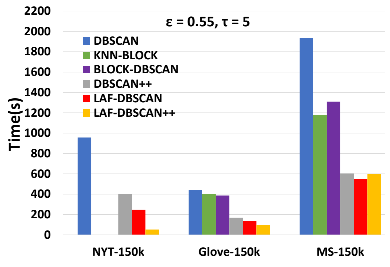
	(ϵ, τ)	**Method**	NYT-150k	Glove-150k	MS-150k
ARI	(0.5,3)	KNN-BLOCK	-	0.8597	**0.6004**
		BLOCK-DBSCAN	-	**0.8825**	0.4953
		DBSCAN++	**0.7933**	0.8129	0.4218
		LAF-DBSCAN	0.7731	0.8660	0.4134
		LAF-DBSCAN++	0.7321	0.7746	0.4113
	(0.55,5)	KNN-BLOCK	-	0.6942	0.1862
		BLOCK-DBSCAN	-	0.8508	0.2283
		DBSCAN++	1.0	0.7869	0.1321
		LAF-DBSCAN	1.0	**0.8520**	**0.2309**
		LAF-DBSCAN++	1.0	0.7444	0.1138
	(0.6,5)	KNN-BLOCK	-	0.2665	-0.0444
		BLOCK-DBSCAN	-	0.6399	0.0046
		DBSCAN++	1.0	0.7801	**0.3687**
		LAF-DBSCAN	1.0	**0.8797**	0.2643
		LAF-DBSCAN++	1.0	0.7653	0.3519
AMI	(0.5,3)	KNN-BLOCK	-	0.4994	**0.4254**
		BLOCK-DBSCAN	-	0.6613	0.3945
		DBSCAN++	0.6872	0.6369	0.3965
		LAF-DBSCAN	**0.7050**	**0.7558**	0.4196
		LAF-DBSCAN++	0.6245	0.5947	0.3879
	(0.55,5)	KNN-BLOCK	-	0.3391	0.1738
		BLOCK-DBSCAN	-	0.6364	0.2626
		DBSCAN++	1.0	0.6578	0.2288
		LAF-DBSCAN	1.0	**0.7554**	**0.3017**
		LAF-DBSCAN++	1.0	0.6068	0.2210
	(0.6,5)	KNN-BLOCK	-	0.1427	0.0390
		BLOCK-DBSCAN	-	0.4988	0.1259
		DBSCAN++	1.0	0.7061	**0.2836**
		LAF-DBSCAN	1.0	**0.8167**	0.2763
		LAF-DBSCAN++	1.0	0.6822	0.2750

Table 3: Clustering quality (AMI and ARI scores) of the approximate methods on the three largest datasets

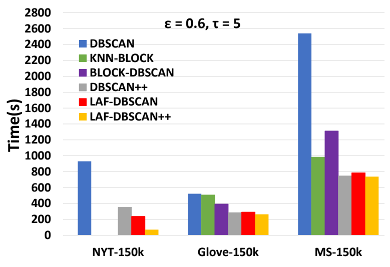
[2]code available at https://github.com/jenniferjang/dbscanpp
[3]code available at https://github.com/XFastDataLab/KNN-BLOCK-DBSCAN
[4]code available at https://github.com/XFastDataLab/BLOCK-DBSCAN
[5]code and binary available at https://sites.google.com/view/approxdbscan, we use the version 2.0



(a) $\epsilon = 0.5$, $\tau = 3$.



(b) $\epsilon = 0.55$, $\tau = 5$.



(c) $\epsilon = 0.6$, $\tau = 5$.

Figure 1: Clustering time of all the methods on the three largest datasets

(ϵ, τ)	MS-50k	MS-100k	MS-150k
(0.5, 3)	864.7s/206.6s	3499.8s/882.7s	6931.1s/1669.5s
(0.55, 5)	854.7s/180.3s	3367.0s/827.6s	6595.1s/1936.6s
(0.6, 5)	753.3s/219.9s	2817.9s/1041.1s	5385.9s/2539.9s

Table 4: Clustering time of ρ-approximate DBSCAN vs. DBSCAN on different dataset scales. Each cell presents a pair of time, "t_1/t_2", where t_1 is the time of ρ-approximate DBSCAN and t_2 is that of DBSCAN given the same (ϵ, τ) and dataset.

(the higher, the better) for all the approximate methods. As the ground truth, DBSCAN is not included in the table. And Figure 1 illustrates the clustering time of those methods. Due to unknown bugs in KNN-BLOCK DBSCAN and BLOCK-DBSCAN, they cannot run on NYT-150k, so their results on NYT-150k are missed in Table 3 and Figure 1. Note that we do not include ρ-approximate DBSCAN in Table 3, Figure 1 or any following experiment, due to its significantly low efficiency on high-dimensional data. Specifically, by [8], a larger ρ makes ρ-approximate DBSCAN more efficient, and ρ ranges from 0.001 to 0.1 in [8]. However, though we have enlarged ρ to 1.0 in our evaluation, the method still presents a low efficiency which is even slower than the naive DBSCAN, as shown in Table 4. This means it suffers much from curse of dimensionality and should not be applied in high-dimensional space. And [2] provides further explanation for this problem of ρ-approximate DBSCAN.

It is observed that (1) LAF-DBSCAN and LAF-DBSCAN++ achieve the highest efficiency in most cases. For example, LAF-DBSCAN makes up to 2.9x acceleration to DBSCAN as well as reaches 1.6x speed over DBSCAN++, 2.2x speed over KNN-BLOCK DBSCAN and 2.4x speed over BLOCK-DBSCAN. (2) LAF-DBSCAN achieves the highest quality in most cases, and in the cases of NYT-150k where the three methods have same scores, LAF-DBSCAN only takes 60% ∼ 70% time of DBSCAN++, as shown in Figure 1. (3) LAF-DBSCAN++ usually has a slightly lower quality than DBSCAN++, but gains much more on the efficiency (i.e., up to 6.7x acceleration to DBSCAN++), which makes it practical with better speed-quality trade-off capability than DBSCAN++. (4) Due to curse of dimensionality, all methods perform worse on MS-150k than the other datasets. Specifically, higher dimension usually means more complex distribution. For LAF, the distribution is harder to fit and more false negative predictions (FN) are made, e.g., when $\epsilon = 0.5, \tau = 3$, the number of FN in NYT/Glove/MS-150k are respectively 5687/2010/7425, which has a negative correlation with the results in Table 3. More complex distribution also makes sampling less representative and neighbor search less effective, which degrades clustering quality of the baselines too.

3.4 Speed-quality trade-off evaluation

We use MS-150k and Glove-150k with the setting $\epsilon = 0.5, \tau = 3$ to present the speed-quality trade-off capabilities of all the approximate methods except ρ-approximate DBSCAN as discussed in Section 3.3. We adjust the performance of DBSCAN++ and LAF-DBSCAN++ by varying the sample fraction p, which is completed by varying the offset δ (mentioned in Section 3.1) within 0.1 ∼ 0.9, while for LAF-DBSCAN the error factor α is varied from 1.1 to 15.0 (which is fixed as 1.0 in LAF-DBSCAN++). For KNN-BLOCK DBSCAN, we vary the branching factor within 3 ∼ 20 and the leaves ratio from 0.001 to 0.3. For BLOCK-DBSCAN, we vary the cover tree basis from 1.1 to 5 while fix the maximum iterations as 10.

As illustrated in Figure 2 and 3, LAF-DBSCAN and LAF-DBSCAN++ have the best speed-quality trade-off capabilities in the high-quality areas (e.g., where AMI > 0.4) on both Glove-150k and MS-150k. Given that the real world normally demands a relatively high clustering quality, we conclude that LAF-DBSCAN and LAF-DBSCAN++ achieve better trade-off capabilities than all the baselines in practice. Furthermore, on both datasets, LAF-DBSCAN++ presents a better trade-off than DBSCAN++, meaning that LAF significantly reduces the clustering time of DBSCAN++ with a relatively tiny quality loss, which further proves the strength and usefulness of LAF for a wide range of DBSCAN variants.

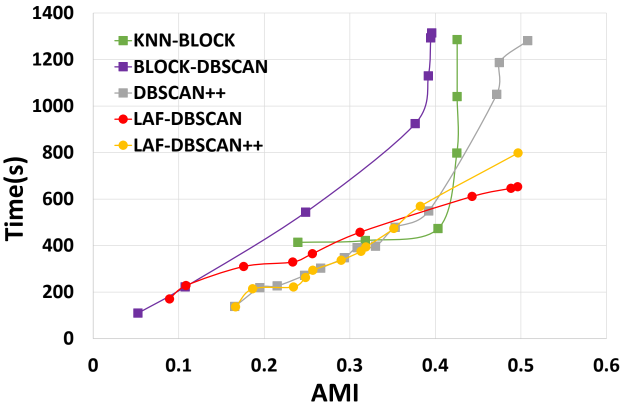


Figure 2: Speed-quality trade-off curves of the approximate methods on dataset MS-150k

3.5 Scalability evaluation

To evaluate the scalability, we run all the methods on the three MS datasets of different scales, and report the results for $\epsilon = 0.55$ and $\tau = 5$, as the results of other (ϵ, τ) are similar. The quality scores are reported in Table 5 and the efficiency results are reported

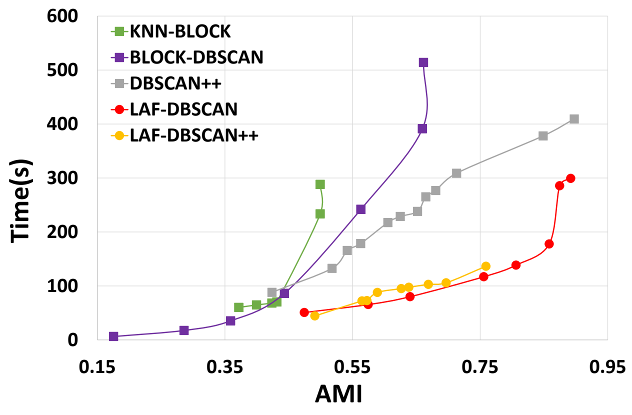


Figure 3: Speed-quality trade-off curves of the approximate methods on dataset Glove-150k

	Method	MS-50k	MS-100k	MS-150k
ARI	KNN-BLOCK	0.7577	0.3828	0.1862
	BLOCK-DBSCAN	**0.7710**	0.4632	0.2283
	DBSCAN++	0.7238	0.4690	0.1321
	LAF-DBSCAN	0.7581	**0.5524**	**0.2309**
	LAF-DBSCAN++	0.6455	0.3077	0.1138
AMI	KNN-BLOCK	0.5708	0.2736	0.1738
	BLOCK-DBSCAN	0.6134	0.3518	0.2626
	DBSCAN++	0.6264	0.4494	0.2288
	LAF-DBSCAN	**0.7043**	**0.5034**	**0.3017**
	LAF-DBSCAN++	0.5328	0.3197	0.2210

Table 5: Clustering quality of all the approximate methods on datasets of different scales (ϵ = 0.55, τ = 5)

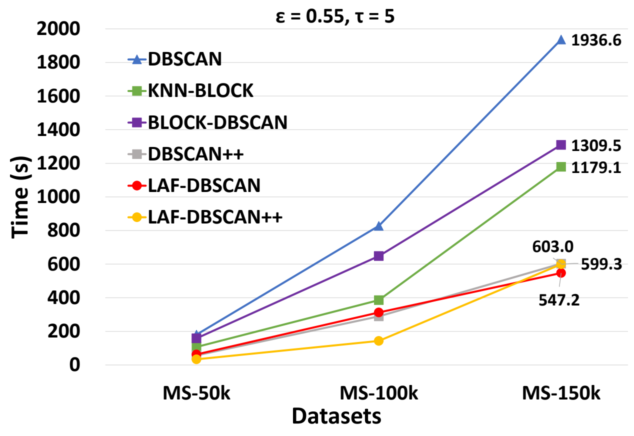


Figure 4: Clustering time of all the methods on datasets of different scales (ϵ = 0.55, τ = 5)

in Figure 4 where we annotate the points of MS-150k with the numbers of the clustering time for a clearer view. They prove that our LAF-enhanced methods are highly effective and scalable, based on these observations: (1) similar to the case of efficiency and effectiveness evaluation, here LAF-DBSCAN still achieves the best quality in most cases, with the highest speed on the largest dataset (whose time is the shortest 547.2s). (2) LAF-DBSCAN has the slowest growth of clustering time when data scale increases, which presents its higher scalability than the baselines. (3) In most cases the quality of LAF-DBSCAN++ is close to DBSCAN++, while in other cases they get closer quickly with the increasing data scale, showing the higher scalability of LAF-DBSCAN++ than DBSCAN++.

(ϵ, τ)	Dataset	MC/TC	MP/TPC	ASMC
(0.5, 3)	NYT-150k	63/92	209/19358	3.32
(0.55, 5)	Glove-150k	39/81	250/7879	6.41
(0.55, 5)	MS-150k	159/176	1107/18384	6.96

Table 6: Statistics for fully missed clusters by LAF-DBSCAN. *MC*, the number of fully Missed Clusters; *TC*, the Total number of groundtruth Clusters; *MP*, the number of Missed data Points; *TPC*, the Total number of data Points belonging to the groundtruth Clusters, i.e., the non-noise points; *ASMC*, Average Size of the fully Missed Clusters.

3.6 Missed cluster analysis

In addition to the wrongly split cluster error discussed in Section 2, a cluster may be fully missed if all its core points are falsely predicted to be non-core or noise. Fortunately, this fully missed cluster error only has a negligible impact on the quality, as it usually occurs in very tiny clusters. We choose the cases where LAF-DBSCAN achieves the lowest quality on each dataset according to Table 3 (i.e., (ϵ,τ) = (0.5,3) on NYT-150k, (0.55,5) on Glove-150k and MS-150k) and report the fully missed cluster information in Table 6. Though in the worst cases, LAF-DBSCAN fully misses more than 50% clusters, it still guarantee the major clusters to be found, since the missed clusters in total include only 1% ∼ 6% of the non-noise points. And the average size of the missed clusters (ASMC) is too tiny (i.e., only including 3 ∼ 7 points on average) to have a non-trivial impact on the overall clustering quality. Therefore, we do not further discuss such an error in this paper.

4 CONCLUSION AND FUTURE WORK

To improve efficiency and scalability of high-dimensional DBSCAN-like clustering for angular distance, we propose LAF, a generic learned accelerator framework using learned cardinality estimation techniques to reduce unnecessary range queries in the clustering, and compensating for the quality loss by detecting the false negative and merging the wrongly separated clusters. Our evaluation shows that the LAF-enhanced methods do have a significantly higher efficiency than the state-of-the-art efficient DBSCAN approaches with also high quality, as well as a better speed-quality trade-off capability than the baselines. The main limitation of this work is the limited range of applicable distance metrics. But since there is no hard constraint on the distance metric, our methods are easy to adapt to other distances, which will be explored in the future work. The future work also includes studying the impact of the cardinality estimator being used, extensively investigating the proper α, etc.

REFERENCES

[1] Yewang Chen, Shengyu Tang, Nizar Bouguila, Cheng Wang, Jixiang Du, and HaiLin Li. 2018. A fast clustering algorithm based on pruning unnecessary distance computations in DBSCAN for high-dimensional data. *Pattern Recognition* 83 (2018), 375–387.
[2] Yewang Chen, Lida Zhou, Nizar Bouguila, Cheng Wang, Yi Chen, and Jixiang Du. 2021. BLOCK-DBSCAN: Fast clustering for large scale data. *Pattern Recognition* 109 (2021), 107624. https://doi.org/10.1016/j.patcog.2020.107624
[3] Yewang Chen, Lida Zhou, Songwen Pei, Zhiwen Yu, Yi Chen, Xin Liu, Jixiang Du, and Naixue Xiong. 2019. KNN-BLOCK DBSCAN: Fast clustering for large-scale data. *IEEE transactions on systems, man, and cybernetics: systems* 51, 6 (2019), 3939–3953.
[4] Difei Cheng, Ruihang Xu, and Bo Zhang. 2021. Fast Density Estimation for Density-based Clustering Methods. *arXiv preprint arXiv:2109.11383* (2021).
[5] Igor de Moura Ventorim, Diego Luchi, Alexandre L. Rodrigues, and Flávio Miguel Varejão. 2021. BIRCHSCAN: A sampling method for applying DBSCAN to large datasets. *Expert Syst. Appl.* 184 (2021), 115518.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *kdd*, Vol. 96. 226–231.

[8] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 519–530. https://doi.org/10.1145/2723372.2737792

[9] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 14 (jul 2017), 45 pages. https://doi.org/10.1145/3083897

[10] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[11] Jennifer Jang and Heinrich Jiang. 2018. DBSCAN++: Towards fast and scalable density clustering. https://doi.org/10.48550/ARXIV.1810.13105

[12] Heinrich Jiang, Jennifer Jang, and Jakub Lacki. 2020. Faster DBSCAN via subsampled similarity queries. *Advances in Neural Information Processing Systems* 33 (2020), 22407–22419.

[13] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*. 489–504.

[14] Diego Luchi, Alexandre Loureiros Rodrigues, and Flávio Miguel Varejão. 2019. Sampling approaches for applying DBSCAN to large datasets. *Pattern Recognition Letters* 117 (2019), 90–96.

[15] Yinghua Lv, Tinghuai Ma, Meili Tang, Jie Cao, Yuan Tian, Abdullah Al-Dhelaan, and Mznah Al-Rodhaan. 2016. An efficient and scalable density-based clustering algorithm for datasets with complex structures. *Neurocomputing* 171 (2016), 9–22.

[16] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.

[17] Jianbin Qin, Wei Wang, Chuan Xiao, Ying Zhang, and Yaoshu Wang. 2021. High-Dimensional Similarity Query Processing for Data Science. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 4062–4063. https://doi.org/10.1145/3447548.3470811

[18] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned cardinality estimation for similarity queries. In *Proceedings of the 2021 International Conference on Management of Data*. 1745–1757.

[19] Yao Tian, Tingyun Yan, Xi Zhao, Kai Huang, and Xiaofang Zhou. 2022. A Learned Index for Exact Similarity Search in Metric Spaces. https://doi.org/10.48550/ARXIV.2204.10028

[20] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research* 11 (2010), 2837–2854.

[21] P Viswanath and V Suresh Babu. 2009. Rough-DBSCAN: A fast hybrid density based clustering method for large data sets. *Pattern Recognition Letters* 30, 16 (2009), 1477–1488.

[22] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. Pseudo-relevance feedback for multiple representation dense retrieval. In *Proceedings of the 2021 ACM SIGIR International Conference on Theory of Information Retrieval*. 297–306.

[23] Yifan Wang, Haodi Ma, and Daisy Zhe Wang. 2022. LIDER: an efficient high-dimensional learned index for large-scale dense passage retrieval. *Proceedings of the VLDB Endowment* 16, 2 (2022), 154–166.

[24] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Xin Cao, Yifang Sun, Wei Wang, and Makoto Onizuka. 2020. Monotonic cardinality estimation of similarity selection: A deep learning approach. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1197–1212.

[25] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, and Yoshiharu Ishikawa. 2021. Consistent and flexible selectivity estimation for high-dimensional data. In *Proceedings of the 2021 International Conference on Management of Data*. 2319–2327.

[26] Shaoyuan Weng, Jin Gou, and Zongwen Fan. 2021. *h*-DBSCAN: A simple fast DBSCAN algorithm for big data. In *Asian Conference on Machine Learning*. PMLR, 81–96.

[27] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta. 2017. Deep lattice networks and partial monotonic functions. *Advances in neural information processing systems* 30 (2017).

[28] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2022. Learning Discrete Representations via Constrained Clustering for Effective and Efficient Dense Retrieval. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22)*. Association for Computing Machinery, New York, NY, USA, 1328–1336. https://doi.org/10.1145/3488560.3498443