ON STORAGE NEURAL NETWORK AUGMENTED APPROXIMATE NEAREST NEIGHBOR SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale approximate nearest neighbor search (ANN) has been gaining attention along with the latest machine learning researches employing ANNs. If the data is too large to fit in memory, it is necessary to search for the most similar vectors to a given query vector from the data stored in storage devices, not from that in memory. The storage device such as NAND flash memory has larger capacity than the memory device such as DRAM, but they also have larger latency to read data. Therefore, ANN methods for storage require completely different approaches from conventional in-memory ANN methods. Since the approximation that the time required for search is determined only by the amount of data fetched from storage holds under reasonable assumptions, our goal is to minimize it while maximizing recall. For partitioning-based ANNs, vectors are partitioned into clusters in the index building phase. In the search phase, some of the clusters are chosen, the vectors in the chosen clusters are fetched from storage, and the nearest vector is retrieved from the fetched vectors. Thus, the key point is to accurately select the clusters containing the ground truth nearest neighbor vectors. We accomplish this by proposing a method to predict the correct clusters by means of a neural network that is gradually refined by alternating supervised learning and duplicated cluster assignment. Compared to state-of-the-art SPANN and an exhaustive method using k-means clustering and linear search, the proposed method achieves 90% recall on SIFT1M with 80% and 58% less data fetched from storage, respectively.

1 INTRODUCTION

Large-scale Approximate Nearest Neighbor searches (ANNs) for high-dimensional data are receiving growing attentions because of their appearance in emerging directions of deep learning research. For example, in natural language processing, methods leveraging relevant documents retrieval by similar dense vector search have significantly improved the scores of open-domain question-answering tasks (Karpukhin et al., 2020). Also for language modeling tasks, Borgeaud et al. (2021) showed a model augmented by retrieval from 2 trillion tokens performs as well as 25 times larger models. In computer vision, Nakata et al. (2022) showed that image classification using ANNs has potential to alleviate catastrophic forgetting and improves accuracy in continual learning scenarios. In reinforcement learning, exploiting past experiences stored in external memory for an agent to make better decisions has been explored (Blundell et al., 2016; Pritzel et al., 2017). Recently, Goyal et al. (2022) and Humphreys et al. (2022) attempted to scale up the capacity of memory with the help of ANN and showed promising results.

ANNs are algorithms to find one or k key vectors that are the nearest to a given query vector among a large number of key vectors. Strict search is not required but higher recall with lower latency is demanded. In order to achieve this, an index is generally build by data-dependent preprocessing. Thus, an ANN method consists of the index building phase and the search phase. As the number of key vectors increases, storing all of them in memory (e.g. DRAM) becomes very expensive, and it is forced to store the vectors in storage devices such as NAND flash memory. In general, a storage device has much larger capacity per cost, but also its latency is much larger than memory. When all the key vectors are stored in memory, as seen in the most papers regarding ANN, it is effective to reduce the number of calculating distance between vectors by employing graph-based (Malkov & Yashunin, 2018; Fu et al., 2019) or partitioning-based methods (Dong et al., 2019) and/or to

reduce the time for each distance calculation by employing quantization techniques such as Product Quantization (Jégou et al., 2011a). On the other hand, when the key vectors are stored in storage rather than memory, the latency for fetching data from the storage becomes the dominant contributor in the total search latency. Therefore, the ANN method for the latter case (ANN method for storage) needs a completely different approach from the ANN method for the former case (all-in-memory ANN method). In this paper, we identify the most fundamental challenges of the ANN method for storage, and explore ways to solve them.

Since the latency for fetching data is approximately proportional to the amount of fetched data, it should be good strategy to reduce the number of vectors to be fetched during search as much as possible, while achieving high recall. Although SPANN (Chen et al., 2021), which is the state-of-the-art ANN method for storage, is also designed with the same strategy, our investigation reveals that the characteristics of its index are still suboptimal from the viewpoint of the number of fetched vectors under a given recall. Moreover, perhaps surprisingly, an exhaustive method combining simple k-means clustering and linear search can perform better than SPANN on some dataset in terms of this metrics.

Another consideration worth noting is the exploitation of the efficient and high-throughput computation of GPUs. It is reasonable to assume that an ANN algorithm used in deep learning application runs on the same system as the deep learning algorithm runs on, and most deep learning algorithms are designed to be run on the system equipped with GPUs. Therefore, the ANN algorithm will be also run on the system with GPU.

In partitioning-based ANN methods, key vectors are partitioned into clusters, which are referred to as posting lists in SPANN paper, usually based on their proximity in the index building phase. First, in the search phase, some clusters that would contain the desired vector with high probability are chosen according to the distance between the query vector and the representative vector of each cluster. Second, the vectors in the chosen clusters are fetched from storage. Since the page size of storage devices is relatively large (e.g. 4KB), it is efficient to make a page contain vectors in the same cluster, as other ANN methods for storage (Chen et al., 2021; Jayaram Subramanya et al., 2019) also adopt. Third, by computing distances between the query and the fetched vectors, k closest vectors are identified and output. In order to achieve high recall with low latency, we need to increase the accuracy to choose the correct cluster containing the ground truth nearest vector at the first step of the search phase.

If clustering is made by k-means algorithm and a query vector is picked from the existing key vectors, the cluster whose centroid vector is the closest to the query always contains the nearest neighbor key vector by the definition of k-means algorithm. However, when a query is not exactly same as any one of key vectors, which is common case for ANN, often the cluster whose representative vector is the closest to the query does not contain the nearest neighbor vector, and this limits recall. To the best of our knowledge, this problem has not been explicitly addressed in the literatures. Our intuition to tackle with this problem is that the border lines that determine which cluster contains the nearest neighbor vector of a query are different from and more complicated than the border lines that determine the assignment of clusters to key vectors. We employ neural networks trained on given clustered key vectors to predict the correct cluster among the clusters defined by the complicated border lines. Assigning multiple clusters to a key vector is a conventional method to improve the accuracy to choose the correct cluster. Combining this duplication technique with our method could provide the additional effect of relaxing demands on the neural network by simplifying border lines. We demonstrate how our method works with visualization using 2-D toy data, and then empirically show that it is effective for realistic data as well. Our contributions include:

- We clarify that we need to reduce the amount of data fetched from storage when key vectors are sit in storage devices, because the latency for fetching data is dominant part of the search latency.
- We first explicitly point out and address the problem that, in partitioning-based ANN method, often the nearest cluster does not contain the nearest neighbor vector of a query vector and this limits the recall-latency performance.
- We propose a new ANN method combining cluster prediction with neural networks and duplicated cluster assignment, and show empirically that the proposed method improves the performance on two realistic million-scale datasets.

2 RELATED WORKS

ANN for storage. DiskANN (Jayaram Subramanya et al., 2019) is a graph-based ANN method for storage. The information of connections defining graph structure and the full precision vectors are stored on storage and the vectors compressed by Product Quantization (Jégou et al., 2011a) are stored in memory. The algorithm traverses the graph by reading the connection information only on the path from storage and computes distances between a query and the compressed vectors in memory. Although they compensate the deterioration of recall due to lossy compression by combining reranking using full precision vector data, the recall-latency performance is inferior to SPANN (Chen et al., 2021). SPANN is another method dedicated to ANN for storage and exhibits state-of-the-art performance. It employs a partitioning-based approach. By increasing the number of clusters as much as possible, it achieved to reduce the number of vectors fetched from storage under a given recall. In order to reduce the latency to choose clusters during search even when the number of clusters are large, they employ SPTAG algorithm that combines tree-based and graph-based ANNs. They also proposed an efficient duplication method aiming at increasing probability that a chosen cluster contains the ground truth key vector. However, our investigation in Section 3.2.2 shows that its performance can be worse than a naive exhaustive method on some dataset.

ANN with GPU. FAISS (Johnson et al., 2019) supports a lot of ANN algorithms accelerated by using GPU's massively parallel computing. On-storage search is also discussed in their project page. SONG (Zhao et al., 2020) optimized the graph-based ANN algorithm for GPU. They modified the algorithm so that distance computations can be parallelized as much as possible, and showed significant speedup. However, they assume only all-in-memory scenarios.

ANN with neural networks. DSI (Tay et al., 2022) predicts the indices of the nearest neighbor key vectors directly from the query vectors with a neural network. We explore to use neural networks to predict the clusters containing the nearest neighbor vector rather than the vector indices themselves. DSI is also targeted for all-in-memory ANN. BLISS (Gupta et al., 2022) and NeuralLSH (Dong et al., 2019) are methods to improve the partitioning rule using neural networks. They apply the same rule to a query for choosing clusters as well. As depicted in Section 4.1, when the rule for partitioning keys is employed to choose clusters, often the chosen clusters don't contain the ground truth key vector. Our method where a neural network is trained to predict the correct cluster for a given query is orthogonal and can be combined with these methods.

3 PRELIMINARIES

3.1 SYSTEM ENVIRONMENT

In this paper, we assume that the system on which our ANN algorithm runs has GPUs and storage devices in addition to CPUs and memories. The GPU provides high-throughput computing through massively parallel processing. The storage can store larger amount of data at lower cost, but has larger read latency than memory devices. Data that are commonly used for all queries, e.g., all the representative vectors of clusters, are loaded in advance on the memories from which CPU or GPU can read data with low latency and is always there during the search. On the other hand, all the key vectors are stored in the storage devices, and for simplicity, we assume that data fetched from storage for computations for a query is not cached on memory to be reused for computations for another query.

3.2 METRICS

3.2.1 THE NUMBER OF FETCHED VECTORS AS A PROXY METRICS OF LATENCY

Our goal is to minimize the average search time per query for nearest neighbor search, which we refer to as mean latency, and simultaneously to maximize the recall. Without loss of generality, the mean latency T in the systems described in the previous subsection is expressed by the following equation,

$$T = T_a + T_b + T_c,$$

where T_a is the latency for computations using data that are always sit in memory, T_b is the latency required for fetching data from storage, and T_c is the latency for computations using data fetched

from storage for each query. For example, in a partitioning-based ANN method such as SPANN, T_a is the latency for the process to determine the clusters (called as the posting lists in SPANN) to be fetched from storage, T_b is the latency for fetching the vectors in the chosen clusters from storage, and T_c is the latency for the computations to find the nearest neighbor vectors in the fetched vectors. In this paper, for simplicity, assuming that $T_a \ll T_b$ and $T_c \ll T_b$, we employ the following approximation,

$$T \approx T_b.$$

Then, since T_b is roughly proportional to the number of fetched vectors, *the number of fetched vectors* is an effective metrics to evaluate the mean latency.

The above assumptions are reasonable in a realistic setting. For T_c, the computing performances of CPUs equipped with vector arithmetic units and GPUs capable of massively parallel operations range from several hundred GFLOPS to more than TFLOPS. On the other hand, read bandwidth of storage devices is at best a few GB/s even when high-speed NVMe is used. This means the fetched data in T_b can be processed in less than $1/100$ of T_b. Note that if the most of the process for T_c is executed in parallel with the process for T_b, for example, by performing distance calculation in the background of asynchronous storage access, the effective T_c becomes almost zero. For T_a, when an exhaustive linear search is used to choose the clusters, i.e., calculating the distance between the query and the representative vectors of all clusters in order to find the closest clusters, a 10-TFLOPS GPU can process 10 million representative vectors of 100 dimension each within a much shorter time than $T_b = 1$ ms. Also in the SPANN case without GPUs, since a fast algorithm that combines tree-based method and graph-based method is applied to choose the clusters, T_a is quite short even when the number of posting lists is as large as a few hundred million. As a typical example, Table 1 shows the measured T_a, T_b, and T_c on SIFT1M dataset.

	#clusters		
Method	**1K**	**10K**	**100K**
GPU	<0.0002	<0.0004	<0.004
SPANN	<0.24	<0.27	<0.29

	#vectors fetched from storage		
	1k	**10K**	**100K**
T_b	>1	>10	>100
T_c, CPU	<0.0005	<0.003	<0.03
T_c, GPU	<0.0002	<0.0004	<0.004

Table 1: Left: T_a in millisecond. For GPU, T_a is measured using `FlatIndex` of FAISS. Right: T_b and T_c in millisecond. T_b is measured using SPANN implementation. T_c on CPU and GPU are both measured using `FlatIndex` of FAISS.

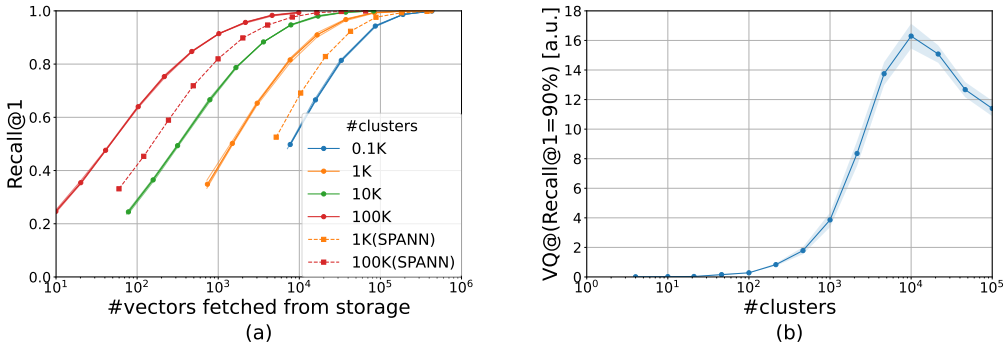

Figure 1: (a) Recall@1 vs the number of vectors fetched from storage. It greatly depends on the number of clusters. (b) VQ under recall@1=90% vs the number of clusters. The line and error bar shows the average and standard deviation of 10 measurements. In these experiments, we use one million SIFT1M base data as key vectors and ten thousand SIFT1B query data as a query vectors.

3.2.2 Memory usage

Since memory usage greatly affects the latency of in-memory ANNs, VQ (Vector-Query), which is a measure of throughput normalized by the memory usage, is introduced in GRIP (Zhang & He,

2019) to compare algorithms with different memory usage as fairly as possible, and is also utilized in SPANN (Chen et al., 2021). SPANN claims superior capacity in large vector search scenarios because this VQ value is greater than that of other algorithms. Here, we consider whether VQ is really fair metrics for comparing the methods with different memory usage. In a partitioning-based ANN method for storage, memory capacity limits the number of clusters since the representative vectors of all the clusters must be kept in memory during search. Then, we investigate how the number of clusters affects the recall-latency and recall-VQ curves. Figure 1(a) shows the dependency of recall versus the number of fetched vectors when the simplest k-means and linear search method (`IVFFlatIndex` of FAISS) are utilized and it is clear that the number of fetched vectors significantly decreases as the number of clusters increases. As shown in Figure 1(b), even when we use VQ metrics, the VQ value under recall@1=90% greatly varies depending on the number of clusters. This indicates that VQ is not a suitable metrics for comparison between algorithms with different memory usage. Based on the above discussion, this paper evaluates ANN methods for storage using *the number of fetched vectors under a given recall and a given number of clusters*.

Another finding by this investigation is that the number of fetched vectors under a given recall of SPANN is not always better than that of the exhaustive method as shown in Figure 1(a).

4 Proposed Method

4.1 Visualization with 2-dimensional toy data

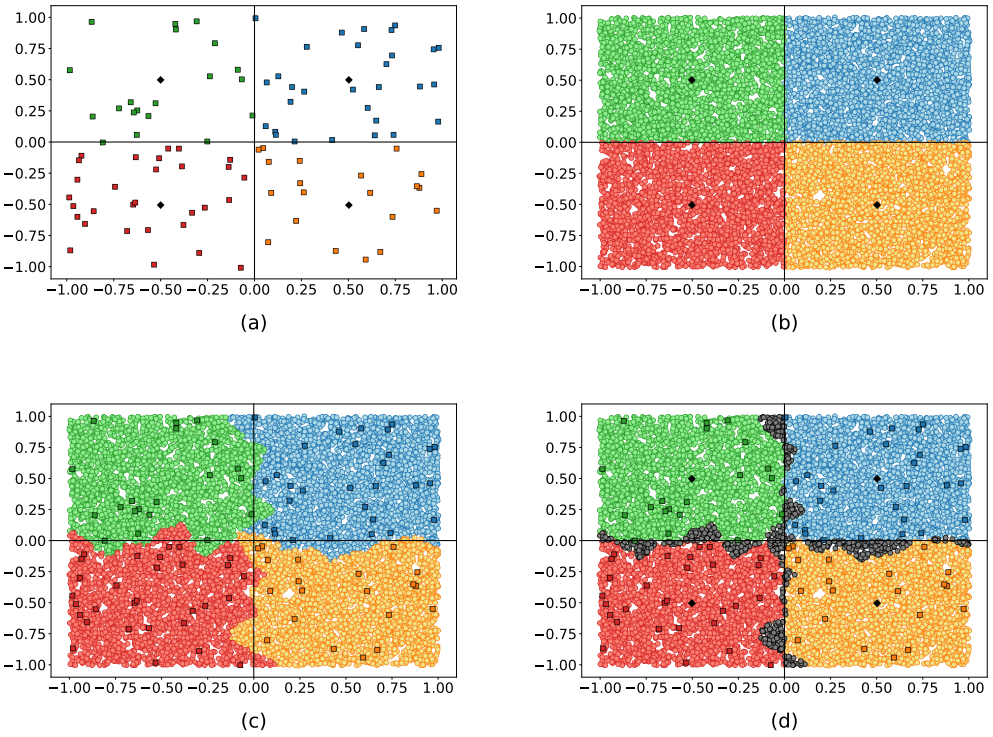

Figure 2: Visualization with 2-dimensional toy data. (a) *Key vectors* are partitioned into four clusters. The cluster assignment is expressed by color. (b) *Query vectors* colored by the *chosen* cluster in the search phase by the conventional method. The query vectors are shown in light-colored circle. (c) *Query vectors* colored by the *correct* cluster that contains the nearest key vector to each query vector. The query vectors are shown in light-colored circle and the key vectors are shown in dark-colored rectangle. (d) Wrong choices are shown in gray.

In this section, we explain the intuition behind our proposed method and visually demonstrate how it improves the accuracy to choose the correct cluster. One hundred key vectors are uniformly sampled from 2-dimensional space between -1 to 1. In the index building phase, we divide the

key vectors into four clusters. The representative vectors of the clusters are placed at $(x, y) = (1/2, 1/2), (-1/2, 1/2), (-1/2, -1/2), (1/2, -1/2)$. Then we assign one cluster to each key vector according to the Euclidean distance, i.e., the distance of a key vector to the representative vector of the assigned cluster is smaller than that of other clusters. Figure 2(a) shows the key vectors colored by the assigned cluster. In a conventional method, when a query is given in the search phase, we choose one cluster whose representative vector is the closest to the query among the four clusters, which is the same manner as that used for assigning clusters to key vectors in the index building phase. Figure 2(b) shows 10000 queries colored by the cluster chosen for each query, and the clusters are clearly divided into four quadrants as expected. On the other hand, Figure 2(c) shows the queries colored by the correct cluster containing the nearest neighbor key vector of each query. We can see that the nearest neighbor key vector of a query vector in the first quadrant, $x > 0, y > 0$, can be contained in the cluster in green whose representative vector is in the second quadrant, $x < 0, y > 0$. As a result, the true border lines of the clusters for query are quite complex. In Figure 2(d), queries where wrong clusters are chosen are shown in gray. When the chosen cluster does not contain the nearest neighbor vector, we need to fetch vectors from another cluster to increase the recall. This phenomenon leads to the increase in the number of fetched vectors under a given recall, and the deterioration of recall-latency tradeoff.

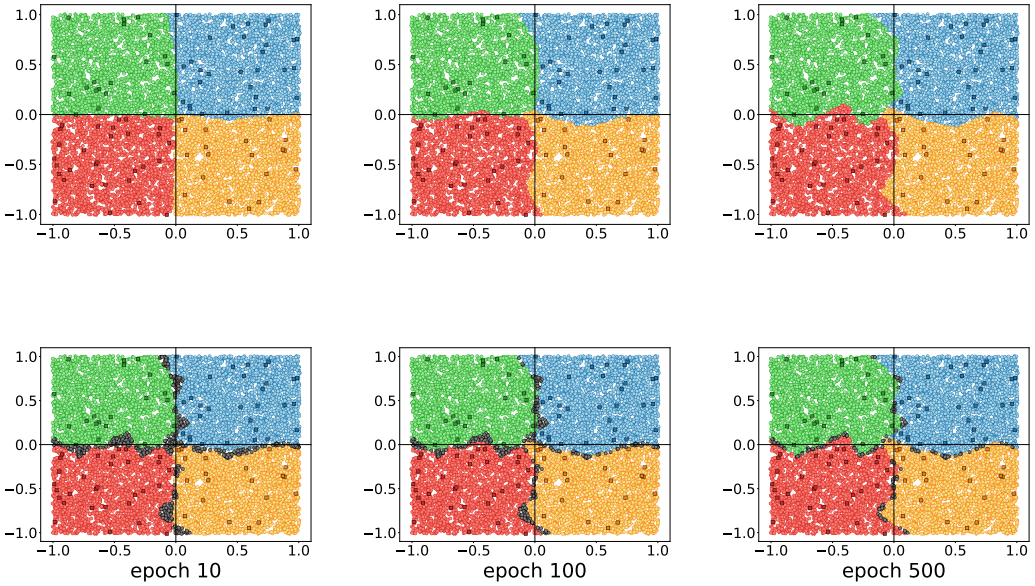

Figure 3: Effect of our proposed method. The upper figures show the *query vectors* colored by the *predicted* cluster by the neural network. The bottom figures show the wrong cluster choices in gray. From left to right, border lines that the neural network predicts are fitting to the ground truth as the training progresses, and the number of wrong cluster choices decreases.

Therefore, to improve the accuracy to choose correct cluster is the fundamental challenge. We attempt to accurately predict the complex border lines by using a neural network that is trained with the objective to choose the correct cluster. We use simple three layer MLP. Input dimension is equal to the dimension of query and key vectors, and output dimension is equal to the number of clusters, and the dimension of hidden layer is set to 128 in this experiment. The query vectors for training are sampled independently every epoch from the same distribution. The ground truth cluster is searched by exhaustive search for each training query. Using those pair samples of query and ground truth cluster as training data, we train the neural network with cross-entropy loss in supervised manner. Note that this is a data-dependent method because we look into the clustered key vectors for generating the ground truth labels. Figure 3 shows that prediction by the neural network approaches the correct border lines of clusters as training proceeds. This results indicate that we can improve the accuracy to choose correct cluster by employing a data-dependently trained neural network.

4.2 EXPERIMENT RESULTS

In this section, we describe the experiment using SIFT (Jégou et al., 2011a;b) and CLIP (Radford et al., 2021) data for demonstrating that our proposed method is useful for realistic data.

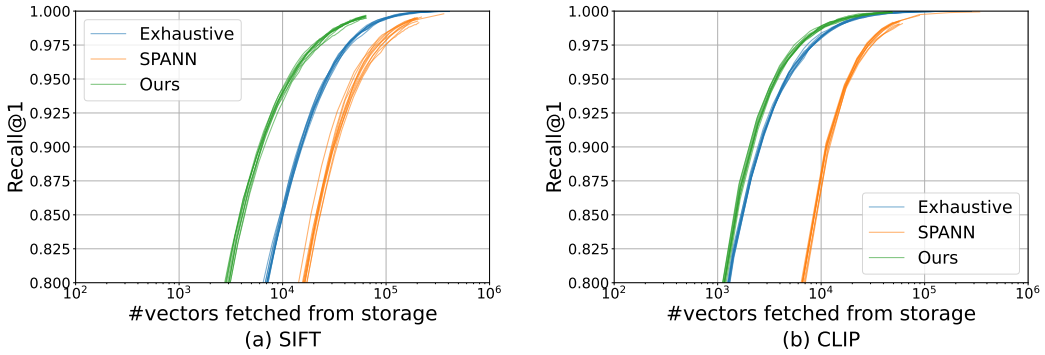

Figure 4: Recall@1 vs the number of vectors fetched from storage.

Method	Recall@1=90%	Recall@1=95%	Recall@1=99%
Exhaustive	14900± 392	26683± 788	72258± 3158
SPANN	30729±1749	52568±3691	151419±16055
Ours	**6165± 223**	**11932± 525**	**39077± 2154**

Table 2: Comparison of the number of fetched vectors under given recall values on SIFT data.

Method	Recall@1=90%	Recall@1=95%	Recall@1=99%
Exhaustive	2547± 47	4655±127	15067± 470
SPANN	11397±337	18531±437	49689±4442
Ours	**2114± 62**	**3625±110**	**11466± 654**

Table 3: Comparison of the number of fetched vectors under given recall values on CLIP data.

Dataset. For SIFT1M, we use one million 128-dimensional SIFT1M base data as key vectors. Another one million data are sampled from SIFT1B base data and used as query vectors for training. SIFT1B query data are used as query vectors for test. Euclidean distance is employed as metrics. For CLIP, we extracted feature vectors from 1.28 million ImageNet (Deng et al., 2009) training data with ViT B/16 model (Dosovitskiy et al., 2021). Although the dimension of the feature vector of the model is 512, we use the first 128 dimension for our experiment. We split it into 0.63 million, 0.64 million, and 0.01 million for key vectors, training query vectors, and test query vectors, respectively. Cosine similarity is employed as metrics.

Comparison with conventional methods. We compare our method with two conventional methods. The first one is the exhaustive method where key vectors are partitioned by k-means in the index building phase, and the distances of a query to the representative vectors of all the clusters partitioned by k-means are calculated and the cluster corresponding to the closest representative vector is chosen in the search phase. The second one is SPANN (Chen et al., 2021). For SPANN, we build the index by the algorithm implemented in SPANN, which includes partitioning process. Since SPANN proposes multiple cluster assignment for improving recall, we set the `ReplicaCount` to its default value 8, which means one key vector is contained by at most 8 clusters. As described in Section 3.2.2, we set the number of clusters for all the methods including our proposed method to 1000 for fair comparison.

Neural network structure. We employ the three-layer MLP. For fair comparison, we carefully design the neural network so as not to require more memory usage than the conventional methods.

As described in Section 3.2.2, if we employed more memory budget, we could significantly improve the recall just by increasing the number of clusters using that memory budget. Concretely, we set the size of hidden layer to 128, which is the same as the vector dimension. Then, the number of parameters of the output layer becomes the dominant among three layers and it is 128×1000, where 1000 is the number of clusters. The memory usage for this output layer is the same as that of the representative vectors of all the clusters which is required for the conventional methods.
Regarding T_c for the neural network inference, the computing in the largest last layer is very close to that of distance calculations between query vectors and all the representative vectors, and the latency of this computation is much smaller than the latency for fetching vectors as shown in Table 1. So still the T_b is dominant even employing the neural network for search.

Training overview. We train the neural network for 150 epochs with AdamW (Loshchilov & Hutter, 2019) optimizer. The batch size is set to 1000. In order to avoid overfitting, we add some noise sampled from normal distribution every iteration to training data. Every 50 epochs, some key vectors are added to another existing cluster. We call this process as duplication.

Detail of duplication process. For a training query vector, if any of top-k_d clusters predicted by the neural network does not contain the ground truth key vector, the pair of top-1 cluster and the ground truth key vector is marked as a candidate pair for duplication. After checking all the training query vectors, we additionally put the key vector into the cluster for the most frequently marked $r_d\%$ pairs. k_d and r_d are hyperparameters and set to 4 and 20 in default, respectively.

Loss function. We compare three loss functions. The first one is naive cross-entropy loss (CE). Although multiple clusters can contain the nearest key vector after duplication, we need to pick only one cluster as a ground truth because CE can not handle multiple positive labels. We use initial ground truth cluster as only one positive all the time. The second one is the modified version of cross-entropy loss (MCE) that picks one cluster where the neural network gives the largest score among the positive clusters. The third one is binary cross-entropy (BCE) loss that can handle multiple positive labels.

Results. The results are shown in Figure 4, Table 2, and Table 3. The figure includes the results of 10 trials each. The tables show the average and the standard deviation values of the 10 trials. Our proposed method achieves the smallest number of vectors fetched from storage under any recall value, which means that our method will provide the smallest mean latency when the latency for storage access is dominant. Under 90% recall on SIFT data, the number of vectors read from storage is 58% and 80% smaller than that of the exhaustive method and SPANN, respectively. Also for CLIP data, steady improvement is obtained.

4.3 Ablation Study

4.3.1 Effect of each ingredient

	Training (loss fn.)	Duplication k_d=4	Duplication iter.	Search by NN	#vectors fetched from storage R@1=0.9	R@1=0.95	R@1=0.99
a. Ours	✓(CE)	✓	✓	✓	**6165± 223**	**11932± 525**	39077±2154
b. Ours	✓(MCE)	✓	✓	✓	7724± 588	13530± 686	**38528±1981**
c. Ours	✓(BCE)	✓	✓	✓	10504±1117	19967±1297	55734±3031
d.	**No train**	**No dup.**		**linear**	14900± 392	26683± 788	72258±3158
e.	✓(CE)	**No dup.**		✓	7305± 210	13922± 350	43177±1234
f.	**No train**	✓	✓	**linear**	7222± 275	13635± 520	41969±1466
g.	✓(CE)	k_d=1	✓	✓	6803± 214	14008± 299	44635±2373
h.	✓(CE)	✓	-	✓	7223± 254	13956± 718	40894±2445
i.	✓(CE)	✓	✓	**linear**	8112± 310	15134± 686	45768±2396

Table 4: Effect of each ingredient.

Table 4 shows how much each ingredient of our proposed method improves the metrics. We report the average and standard deviation values of the number of fetched vectors across 10 trials for each condition. (a) to (c) compare the loss functions for training. The both MCE and CE show

good performance but CE is better in R@1$\leq$0.95 and MCE is better in R@1=0.99. BCE loss deteriorates the performance and the increase in the variation is observed. (d) is the conventional exhaustive method using linear search for choosing clusters and employs neither neural networks nor duplication. (e) employs only neural network and (f) employs only duplication. By comparing (a,d,e,f), the both neural network and duplication contribute to improving performance. (g) shows the effect of the hyperparameter k_d explained in Section 4.2. By increasing k_d, the number of fetched vectors decreases. In (h), we execute duplication only once after 150-epoch training is completed. The result is worse than that in the default setting where duplication is executed every 50 epochs. This indicates that executing duplication process between training can relax the complexity of the border lines of clusters and help the neural network to fit them. (i) shows the result when we use the clustering information obtained after 150 epoch training and duplication in (a) setting, but choose the cluster to be fetched by using linear search across the updated centroid vectors of all the clusters. This shows executing search with the neural network inference is advantageous. From this experiment, we can see that although even only each ingredient of our proposed method can significantly reduce the number of fetched vectors under a given recall compared to the conventional method, further improvement is obtained by combining them.

4.3.2 BUILDING INDEX BY SPANN

In our experiment, we use *k*-means for partitioning, but it is an exhaustive method and can take too much time for larger dataset. In order to confirm that *k*-means is not a necessary component of our method, we apply the algorithm in SPANN to execute partitioning. As a result, the number of fetched vectors under 90% recall significantly improves from 30729$\pm$1749 to 7372$\pm$162. This indicates that we may utilize a fast algorithm such as SPANN for clustering instead of exhaustive k-means when our proposed method is applied to larger dataset.

5 LIMITATIONS AND FUTURE WORKS

Since the discussion in this paper assumes that the condition that the mean latency for search is determined by storage access time holds true, the discussion in this paper may be invalid if this condition is not satisfied.

For CLIP data, the improvement over the exhaustive method is steady but marginal as shown in Figure 4. The difference of the amount of improvement between SIFT and CLIP may come from the difference of how well the training data reproduce the query distribution. This means the effectiveness of the proposed method could be limited under the condition where the query distribution is close to uniform and not predictable. Although this is a common issue in almost all of the ANN methods, it remains future work to address such difficult use case.

Our proposed method has a couple of hyperparameters. Although we show some of their effect in Section 4.3, thorough optimization is a future work. It may dependent on data distribution and required recall value. However, it is not difficult to find the acceptable values for hyperparameters that provide at least better performance than the exhaustive method.

Another apparent remaining future work is to apply the proposed method to larger datasets such as billion-scale or trillion-scale ones. However, we believe that foundings and direction we reveal in this paper will be also useful for them.

6 CONCLUSION

We investigated the requirement to improve the recall and latency tradeoff of large scale approximate nearest neighbor search under the condition where the key vectors are stored in storage devices with large capacity and large read latency. We pointed out that in order to achieve it, we need to reduce the number of vectors fetched from storage devices during search. Then, it is required to choose correct clusters containing the nearest neighbor key vector to a given query vector with high accuracy. We proposed to use neural networks to predict the correct cluster. By employing our proposed method, we achieved to reduce the number of vectors to read from storage by more than 58% under 90% recall on SIFT1M data compared to the conventional methods.

REFERENCES

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control, 2016. URL https://arxiv.org/abs/1606.04460.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. *CoRR*, abs/2112.04426, 2021. URL https://arxiv.org/abs/2112.04426.

Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5199–5212, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning sublinear-time indexing for nearest neighbor search. *arXiv preprint*, arXiv:1901.08544, 2019. URL http://arxiv.org/abs/1901.08544.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graphs. *PVLDB*, 12(5):461 – 474, 2019. doi: 10.14778/3303753.3303754. URL http://www.vldb.org/pvldb/vol12/p461-fu.pdf.

Anirudh Goyal, Abram Friesen, Andrea Banino, Theophane Weber, Nan Rosemary Ke, Adrià Puigdomènech Badia, Arthur Guez, Mehdi Mirza, Peter C Humphreys, Ksenia Konyushova, Michal Valko, Simon Osindero, Timothy Lillicrap, Nicolas Heess, and Charles Blundell. Retrieval-augmented reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7740–7765. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/goyal22a.html.

Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. Bliss: A billion scale index using iterative re-partitioning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, pp. 486495, 2022.

Peter C. Humphreys, Arthur Guez, Olivier Tieleman, Laurent Sifre, Thophane Weber, and Timothy Lillicrap. Large-scale retrieval for reinforcement learning, 2022. URL https://arxiv.org/abs/2206.05314.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011a. doi: 10.1109/TPAMI.2010.57.

Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 861–864, 2011b. doi: 10.1109/ICASSP.2011.5946540.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

Kengo Nakata, Youyang Ng, Daisuke Miyashita, Asuka Maki, Yu-Chieh Lin, and Jun Deguchi. Revisiting a knn-based image classification system with high-capacity storage, 2022. URL https://arxiv.org/abs/2204.01186.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2827–2836. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/pritzel17a.html.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL http://proceedings.mlr.press/v139/radford21a.html.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. Transformer memory as a differentiable search index. *arXiv preprint*, 2022. URL https://arxiv.org/abs/2202.06991.

Minjia Zhang and Yuxiong He. Grip: Multi-store capacity-optimized high-performance nearest neighbor search for vector search engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pp. 16731682, 2019.

Weijie Zhao, Shulong Tan, and Ping Li. Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1033–1044, 2020. doi: 10.1109/ICDE48307.2020.00094.