How Should We Evaluate Data Deletion in Graph-Based ANN Indexes?

Tomohiro Yamashita
The University of Tokyo
`t_yamashita@hal.t.u-tokyo.ac.jp`

Daichi Amagata
The University of Osaka
`amagata.daichi@ist.osaka-u.ac.jp`

Yusuke Matsui
The University of Tokyo
`matsui@hal.t.u-tokyo.ac.jp`

Abstract

Approximate Nearest Neighbor Search (ANNS) has recently gained significant attention due to its many applications, such as Retrieval-Augmented Generation. Such applications require ANNS algorithms that support dynamic data, so the ANNS problem on dynamic data has attracted considerable interest. However, a comprehensive evaluation methodology for data deletion in ANNS has yet to be established. This study proposes an experimental framework and comprehensive evaluation metrics to assess the efficiency of data deletion for ANNS indexes under practical use cases. Specifically, we categorize data deletion methods in graph-based ANNS into three approaches and formalize them mathematically. The performance is assessed in terms of accuracy, query speed, and other relevant metrics. Finally, we apply the proposed evaluation framework to Hierarchical Navigable Small World, one of the state-of-the-art ANNS methods, to analyze the effects of data deletion, and propose Deletion Control, a method which dynamically selects the appropriate deletion method under a required search accuracy.

1 Introduction

ANNS is an essential building block for applications such as Retrieval-Augmented Generation (RAG) [1] and recommendation systems [2]. In these applications, frequent data updates occur due to the addition of new products and the removal of unavailable items. Consequently, research has been conducted on ANNS algorithms that support data deletion, including IVF-based methods [3], product quantization-based methods [4], and graph-based methods [5, 6, 7].

However, no comprehensive methodology has been established for evaluating data deletions under practical use cases. Evaluating the execution time of data deletion and the search performance after deletion is crucial for selecting ANNS algorithms. Moreover, the experimental settings in existing evaluation are unrealistic, such as re-adding deleted data [5, 6]. The evaluation criteria used in existing studies, furthermore, are not sufficiently comprehensive for assessing deletion performance [3, 7, 8].

To address these limitations, we develop a unified, deployment-oriented evaluation methodology for data deletion in graph-based ANNS. We first formalize three deletion methods (logical deletion, physical deletion, and rebuilding) and implement them within Hierarchical Navigable Small World (HNSW) [9]. We then introduce an experimental protocol that measures deletion latency, search/insert throughput, memory footprint, and post-deletion accuracy under realistic workloads. Based on the experimental results, we propose an algorithm that dynamically switches between deletion methods.

2 Related Work

Ada-IVF [3] and SPFresh [8], which employ the inverted file structure, handle data updates by reassigning vectors to clusters. The following methods in graph-based ANNS algorithms have implemented support for data deletion. First, FreshDiskANN [5] is an algorithm based on DiskANN [10], which utilizes disk storage. It maintains search accuracy by reconnecting edges among neighboring nodes when a node is deleted. Additionally, the MN-RU algorithm [6] addresses the issue of unreachable nodes in HNSW by maintaining a backup graph to preserve search accuracy after data deletions. Furthermore, the IPGM algorithm [7] recalculates all neighboring nodes within one hop of a deleted node and reconnects edges to maintain search accuracy. Existing studies have lacked a practical experimental setup. In this work, we propose a realistic and effective evaluation methodology.

3 Formal Definition of Deletion

We define baseline data deletion methods in graph-based ANNS using pseudocode. Specifically, we categorize these methods into three types: logical deletion, physical deletion, and rebuilding.

Preliminaries Let $\mathcal{P} = \{\mathbf{p}_i\}_{i=1}^{n} \subset \mathbb{R}^d$ be the set of n nodes in the graph, where each node $\mathbf{p}_i \in \mathbb{R}^d$ is a d-dimensional vector. For each node $\mathbf{p}_i$, let $\mathcal{N}_i \subset \{1, 2, \ldots, n\}$ be the set of indices of its neighboring nodes. We define $\mathcal{N} = \{\mathcal{N}_i\}_{i=1}^{n}$ as the collection of all neighborhood sets. Thus, the graph is represented by $\mathcal{P}$ and $\mathcal{N}$.

Algorithm 1: Search in logical deletion

```
Input: $\mathbf{q}, \mathcal{P}, \mathcal{N}, \mathcal{F}$
Output: $\mathcal{R}$
1 $\mathcal{R} \leftarrow \texttt{SEARCH}(\mathbf{q}, \mathcal{P}, \mathcal{N}) \setminus \mathcal{F}$
2 return $\mathcal{R}$
```

We define the standard algorithms for search and graph construction, shared across all deletion approaches. Let $\mathbf{q} \in \mathbb{R}^d$ be a query vector. We define the search algorithm as SEARCH: $(\mathbf{q}, \mathcal{P}, \mathcal{N}) \mapsto \mathcal{R}$, which takes the query vector and the graph as input and returns a set of approximate nearest neighbor $\mathcal{R} \subset \{1, 2, \ldots, n\}$. Similarly, we define the graph construction algorithm as CONSTRUCT: $\mathcal{P} \mapsto \mathcal{N}$, which takes the set of nodes $\mathcal{P}$ as input and returns the neighbor set $\mathcal{N}$. Finally, let $\mathcal{D} \subset \{1, 2, \ldots, n\}$ denote the set of node indices corresponding to deletion queries. We define the deletion algorithm as DELETE: $(\mathcal{D}, \mathcal{P}, \mathcal{N}) \mapsto (\mathcal{P}', \mathcal{N}')$, which takes $\mathcal{D}$, $\mathcal{P}$, and $\mathcal{N}$ as input and outputs the updated node set $\mathcal{P}' \subset \mathcal{P}$ and the updated neighborhood set $\mathcal{N}' \subset \mathcal{N}$ after deletion.

Algorithm 2: Physical deletion

```
Input: $\mathcal{D}, \mathcal{P}, \mathcal{N}$
Output: $\mathcal{P}', \mathcal{N}'$
1 foreach $\mathbf{p}_i \in \mathcal{P}$ do
2     if $i \in \mathcal{D}$ then
3         $\mathcal{P} \leftarrow \mathcal{P} \setminus \{\mathbf{p}_i\}$
4         $\mathcal{N} \leftarrow \{\mathcal{N}_j \in \mathcal{N} \mid j \neq i\}$
5     else
6         $\mathcal{N}_i \leftarrow \mathcal{N}_i \setminus \mathcal{D}$
7 return $\mathcal{P}, \mathcal{N}$
```

Logical Deletion Logical deletion is a method where each deleted node is marked with a flag $\mathcal{F} \subset \{1, 2, \ldots, n\}$ at the time of deletion. Such flags are referenced during the search to exclude flagged nodes from the results. Figure 1a illustrates the mechanism of logical deletion in a graph. The search algorithm is presented in Algorithm 1.

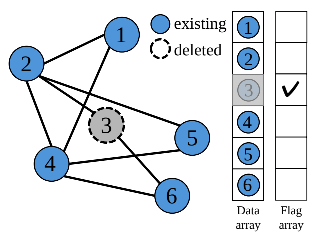
Algorithm 3: Rebuilding

```
Input: $\mathcal{D}, \mathcal{P}$
Output: $\mathcal{P}', \mathcal{N}'$
1 $\mathcal{P} \leftarrow \{\mathbf{p}_i \in \mathcal{P} \mid i \notin \mathcal{D}\}$
2 $\mathcal{N} \leftarrow \texttt{CONSTRUCT}(\mathcal{P})$
3 return $\mathcal{P}, \mathcal{N}$
```

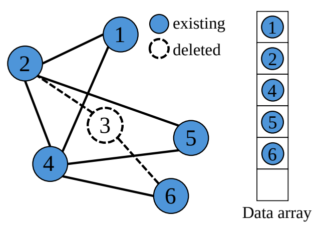
Physical deletion Physical deletion is a method that removes the designated data by deleting all edges connected to the node, as illustrated in Figure 1b. The procedure for this deletion approach is presented in Algorithm 2. The data is also removed from memory.

Rebuilding Rebuilding is a method that removes data both from the graph by reconstructing the graph using all remaining data, as illustrated in Figure 1c. The procedure for this deletion approach is presented in Algorithm 3. Similar to the physical deletion approach, this approach involves removing data from memory during the deletion process.

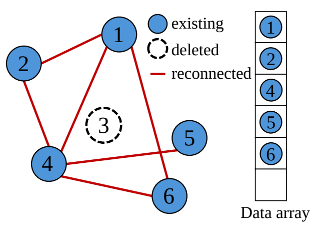
More detailed explanation of each data deletion method are provided in Appendix Sec A.



(a) Logical deletion

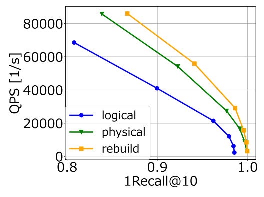


(b) Physical deletion

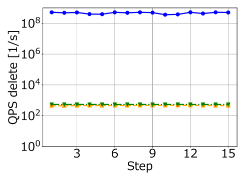


(c) Rebuilding

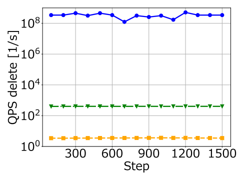
Figure 1: Overview of three data deletion methods: logical deletion, physical deletion, and rebuilding.



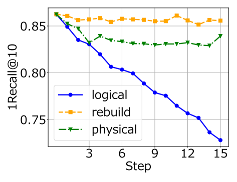
(a) SIFT1M: Comparison of deletion methods at step 5



(b) SIFT1B: QPS of deletion ($b = 10^5$)



(c) SIFT1B: QPS of deletion ($b = 10^3$)



(d) SIFT1B: 1-Recall@10 after deletion ($b = 10^5$)

Figure 2: Evaluation of deletion performance on SIFT1M and SIFT1B datasets. (a) Comparison of search performance after step 5 deletion in SIFT1M. (b, c) QPS for deletion on SIFT1B with varying batch sizes. (d) Impact on 1-Recall@10 after updates on SIFT1B.

4 Experiments

Settings For the experiments, we used SIFT1M [11], GIST1M [11], SIFT1B [12], DEEP1M [13], and Glove100Angular [14]. For SIFT1B, we created a subset of 2×10^6 data points. We evaluated the performance by repeatedly performing insertion and deletion with the same batch size in the database. The detailed experimental settings are described in Appendix Sec B.

Metrics We evaluated search accuracy using 1-Recall@10. To assess query processing speed, we used Queries Per Second (QPS). We measured QPS-search for search speed evaluation, QPS-add for insertion speed evaluation, and QPS-delete for deletion speed evaluation. We adopt the QPS-Recall curve as an evaluation metric by plotting QPS-search against 1-Recall@10.

Experimental Results Figure 2 presents a comparison of search performance before and after data updates for each of the three deletion approaches. Figure 2a shows that post-update search performance is highest with rebuilding, followed by physical deletion, and then logical deletion. From Figure 2b and Figure 2c, it is evident that logical deletion achieves the highest data deletion speed. Additionally, under frequent small-batch deletions, rebuilding is relatively slower than physical deletion. Figure 2d indicates that with repeated data updates, the search accuracy of logical deletion deteriorates. Interestingly, this figure suggests that in physical deletion, search accuracy stabilizes to a constant value after multiple updates. The results for the other metrics are presented in Appendix Sec D.

5 Deletion Control

Problem Statement Based on Sec 4, we discuss how to control data deletion methods in scenarios that require high search accuracy under continuous deletions. Following the experimental setup in Sec 4, we assume a situation where data updates are repeatedly performed with the same batch size b. Here, the input consists of the dataset for retrieval, the accuracy target $\alpha \in (0, 1]$, and a small query training set (query data with known ground truth). In this setting, we repeatedly delete data with batch size b. Our goal is to present a hybrid data deletion strategy such that, even after deletion, the search accuracy remains higher than α.

We consider two approaches. The first applies only physical deletion, while the second employs logical deletion and performs rebuilding once the performance begins to degrade. The first approach is applicable when the requirement for search accuracy is not very strict. In this case, continuing physical deletion does not reduce performance below the required level. The second approach is employed when high search accuracy is required. As shown in Figure 2d, performance can be maintained as long as rebuilding is executed sufficiently often. However, it is computationally expensive. Therefore, logical deletion is applied until just before the performance drops below the required level, at which point rebuilding is performed.

Here, we introduce two parameters, θ and π, necessary for designing the Deletion Control algorithm. First, suppose we repeatedly perform only physical data deletion. Let us consider the minimum 1-Recall@10 achieved in this case and denote it by θ. Next, we analyze the case where we repeatedly delete data logically. We define $R_s \in (0, 1]$ as the 1-Recall@10 after performing s updates by the logical deletion, where R_0 is the 1-Recall@10 before deletion. Here, we define π as the maximum number of steps for which R_s remains above α:

$$\pi = \max_{R_s \geq \alpha} s. \tag{1}$$

Here, θ and π represent the dataset's characteristics, and cannot be measured unless actual query data is available.

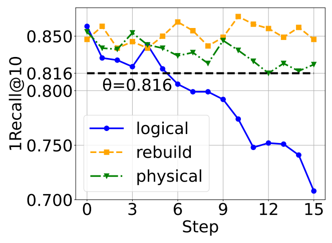


Figure 3: Comparison of 1-Recall@10 on SIFT1B training set (10% queries).

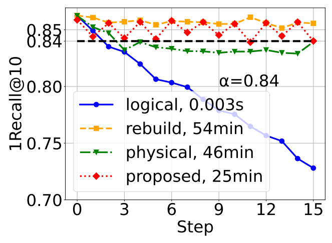


Figure 4: Comparison of 1-Recall@10 on test set (all queries).

Deletion Control Policy We first estimate θ and π through experiments using the training set. As shown in Figure 2d, the search accuracy of physical deletion converges to a stable value. Therefore, we can estimate θ as the lowest value of the measured 1-Recall@10. Also, as shown in Figure 2d, we can approximate R_s as a linear function of the update step s. We therefore define $\Delta = (R_S - R_0)/S$ as the average decrease in 1-Recall@10 per step, where S represents the maximum number of steps for the training set. The π can be estimated as $\pi \approx (\alpha - R_0)/\Delta$.

Then, we select one of the two deletion strategies. First, when $\alpha < \theta$, only physical deletion is repeatedly applied. Since the 1-Recall@10 obtained by physical deletion never falls below θ, the requirement is satisfied as $\alpha < \theta \leq R_s$. Second, when $\alpha \geq \theta$, the procedure alternates between performing logical deletion for π steps and then executing one rebuilding operation. According to Equation 1, the condition $\alpha \leq R_s$ holds for up to π steps. Furthermore, as shown in Figure 2d, rebuilding afterward restores the 1-Recall@10 to R_0. In this manner, the condition $\alpha \leq R_s$ is consistently maintained, thereby satisfying the requirement. The algorithm is shown in Appendix Algorithm 4.

Experiment We conducted an experiment on SIFT1B [12] with $b = 10^5$, setting $\alpha = 0.84$. Figure 3 shows the 1-Recall@10 for the basic deletion methods using 10% of the queries as the training set. From these results, we estimate $\theta = 0.816$ and $\pi = 1$. Since $\alpha > \theta$, we select the deletion strategy that alternates between logical deletion and rebuilding. Figure 4 indicates that the proposed method almost satisfies the required search accuracy. Furthermore, the proposed method has the smallest total deletion time among the deletion strategies that meet the accuracy requirement.

6 Conclusion

This study has three main contributions. First, we formally defined three baseline data deletion approaches for ANNS—logical deletion, physical deletion, and rebuilding—along with their mathematical formulations. Second, we established an experimental setup and evaluation metrics that align with practical use cases. Third, we implemented and empirically evaluated the baseline data deletion approaches on HNSW. We also proposed a deletion control algorithm that selects an appropriate data deletion method based on the required accuracy.

Acknowledgements: This work was supported by JST AIP Acceleration Research JPMJCR23U2, Japan.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[2] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12:331–370, 2002.

[3] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Umar Farooq Minhas, Jeffery Pound, Cedric Renggli, Nima Reyhani, Ihab F Ilyas, Theodoros Rekatsinas, and Shivaram Venkataraman. Incremental ivf index maintenance for streaming vector search. *arXiv preprint arXiv:2411.00970*, 2024.

[4] Cecilia Aguerrebere, Mark Hildebrand, Ishwar Singh Bhati, Theodore Willke, and Mariano Tepper. Locally-adaptive quantization for streaming vector search. *arXiv preprint arXiv:2402.02044*, 2024.

[5] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613*, 2021.

[6] Wentao Xiao, Yueyang Zhan, Rui Xi, Mengshu Hou, and Jianming Liao. Enhancing hnsw index for real-time updates: Addressing unreachable points and performance degradation. *arXiv preprint arXiv:2407.07871*, 2024.

[7] Zhaozhuo Xu, Weijie Zhao, Shulong Tan, Zhixin Zhou, and Ping Li. Proximity graph maintenance for fast online nearest neighbor search. *arXiv preprint arXiv:2206.10839*, 2022. URL `https://arxiv.org/abs/2206.10839`.

[8] Yuming Xu, Hengyu Liang, Jin Li, Shuotao Xu, Qi Chen, Qianxi Zhang, Cheng Li, Ziyue Yang, Fan Yang, Yuqing Yang, et al. Spfresh: Incremental in-place update for billion-scale vector search. In *Symposium on Operating Systems Principles*, pages 545–561, 2023.

[9] 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.

[10] 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. *Advances in Neural Information Processing Systems*, 32, 2019.

[11] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2010.

[12] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864, 2011.

[13] Artem Babenko and Lempitskys Victor. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2055–2063, 2016.

[14] Jeffrey Pennington, Socher Richard, and D. Manning Christopher. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Table 1: Experimental conditions for each dataset

Dataset	d	n_o	n	b
SIFT 1M [11]	128	10^6	5×10^5	10^5
GIST 1M [11]	960	10^6	5×10^5	10^5
SIFT 1B [12]	128	2×10^6	5×10^5	10^3 or 10^5
DEEP 1M [13]	96	10^6	5×10^5	10^5
Glove100Angular [14]	100	10^6	5×10^5	10^5

A Algorithm of three deletion methods

A.1 Logical deletion

Let $\mathcal{F} \subset \{1, 2, \ldots, n\}$ be a set of flags. When performing deletion, the deletion flag set $\mathcal{F}$ is updated for the deletion query set $\mathcal{D}$ as follows:

$$\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{D} \tag{2}$$

During the search process, nearest neighbor candidates are obtained by excluding deleted data from the initial results. The search algorithm incorporating this approach is presented in Algorithm 1. Since logical deletion removes data only from the search results, the effect of data deletion appears at search time.

Since logical deletion only involves updating flags, it can be performed efficiently. However, because the deleted nodes remain in $\mathcal{P}$ and the neighborhood set $\mathcal{N}$, memory consumption accumulates over time. Additionally, if insertions and deletions are repeated, the index has fewer not-deleted vectors. The operation that excludes deleted data from the initial search results may eventually lead to an empty result.

A.2 Physical deletion

During the search for a query vector $\mathbf{q}$, a straightforward search is performed following Equation 3.

$$\mathcal{R} \leftarrow \texttt{SEARCH}(\mathbf{q}, \mathcal{P}, \mathcal{N}) \tag{3}$$

Unlike logical deletion, physical deletion actually removes data and does not require index rebuilding. As a result, it does not need to retain vector data for distance calculations, making it the most memory-efficient method when implemented properly. However, the effectiveness of data removal heavily depends on the implementation of the ANNS algorithm and the memory layout. Furthermore, since deleting edges alters the structure of the graph, it is expected to affect search performance.

A.3 Rebuilding

Since `CONSTRUCT` is invoked whenever a new update batch is given and distance computations are performed, the set of graph nodes $\mathcal{P}$ is required. On the other hand, reconstructing the index ensures that its structure remains optimal, preventing search performance degradation while maintaining appropriate memory consumption.

B Experimental Settings

The datasets in Table 1 were partitioned for insertion and deletion operations, and the ground-truth is recomputed. Let the original dataset's set of base vectors be $\mathcal{P}_o = \{\mathbf{p}_i\}_{i=1}^{n_o}$ and the set of query vectors be $\mathcal{Q} \subset \mathbb{R}^d$. We perform s iterations of data updates, including insertions and deletions. The number of vectors in the index after each update is maintained at $n(\leq n_o)$, regardless of s. A full search is conducted on this index with the query set $\mathcal{Q}$ to obtain the ground-truth set $\mathcal{G}_s \subset \{1, 2, \ldots, n\}$ for each s, where $|\mathcal{G}_s| = |\mathcal{Q}|$. Data deletion and insertion are performed iteratively with an equal number of data points. We define an insertion algorithm, `ADD`: $(\mathcal{P}_{\text{add}}, \mathcal{P}, \mathcal{N}) \mapsto (\mathcal{P}', \mathcal{N}')$, which takes as input the set of vectors to be added $\mathcal{P}_{\text{add}} \subset \mathbb{R}^d$, the existing node set $\mathcal{P} \subset \mathbb{R}^d$, and the neighborhood set $\mathcal{N}$, and outputs the updated node set $\mathcal{P}'$ and the updated neighborhood set $\mathcal{N}'$. Let the batch size for each insertion and deletion operation be $b \in \mathbb{N}$. At step $s = 0$, the index is constructed with the first $n(\leq n_o)$ base vectors as follows.

$$\begin{aligned} \mathcal{P} &\leftarrow \{\mathbf{p}_i\}_{i=1}^{n} \\ \mathcal{N} &\leftarrow \texttt{CONSTRUCT}(\mathcal{P}) \end{aligned} \tag{4}$$

For trials where $s \geq 1$, data deletion is first performed according to Equation 5. Specifically, a set of b consecutive integers is prepared as the deletion index set $\mathcal{D}$, and the vectors corresponding to these indices are removed.

$$\begin{aligned} \mathcal{D} &\leftarrow \{1 + (s-1)b, \dots, sb\} \\ (\mathcal{P}, \mathcal{N}) &\leftarrow \texttt{DELETE}(\mathcal{D}, \mathcal{P}, \mathcal{N}) \end{aligned} \tag{5}$$

Next, b data points are inserted. Specifically, a set of b consecutive vectors, $\mathcal{P}_{\text{add}}$, is prepared and added to the index, as seen below.

$$\begin{aligned} \mathcal{P}_{\text{add}} &\leftarrow \{\mathbf{p}_i \mid i \in \{1 + n + (s-1)b, \dots, n + sb\}\} \\ (\mathcal{P}, \mathcal{N}) &\leftarrow \texttt{ADD}(\mathcal{P}_{\text{add}}, \mathcal{P}, \mathcal{N}) \end{aligned} \tag{6}$$

The above experimental setup implies that when b is small, frequent updates with a small number of data points occur. Conversely, when b is large, a large amount of data is updated in a few iterations. The inserted data points are always new. All experiments were conducted on a single thread using an Intel(R) Core(TM) i7-13700H@2.4GHz processor with 32GB RAM, running Ubuntu 22.04.5.

C Mathematical Representation of Evaluation Metrics

The 1-Recall@k, which represents search accuracy, is defined as follows. Let n_q be the number of queries, and for a given query $\mathbf{q}_i \in \mathbb{R}^d$, let $g_i \in \{1, 2, \dots, n\}$ denote the ground-truth nearest neighbor. Additionally, let $\hat{\mathcal{R}}_i \subset \{1, 2, \dots, n\}$ with $|\hat{\mathcal{R}}_i| = k$ represent the approximate k-nearest neighbors obtained through ANNS. Defining $f(\cdot)$ as a function that returns 1 if the condition is true and 0 otherwise, 1-Recall@k is expressed as shown in Equation 7.

$$\text{1-Recall@}k = \frac{1}{n_q} \sum_{i=1}^{n_q} f(g_i \in \hat{\mathcal{R}}_i), \tag{7}$$

In this study, we set $k = 10$ and use 1-Recall@10 to evaluate search accuracy. A higher recall indicates better search accuracy.

To evaluate the query processing speed, we use Queries Per Second (QPS). When processing n_q queries in t seconds, QPS is defined as shown in Equation 8:

$$\text{QPS} = \frac{n_q}{t} [1/\text{s}] \tag{8}$$

A higher QPS value indicates faster query processing. We measure QPS-search to evaluate search speed, QPS-add to evaluate data insertion speed, and QPS-delete to evaluate data deletion speed. Additionally, we adopt the QPS-Recall curve as an evaluation metric, where the horizontal axis represents 1-Recall@10, and the vertical axis represents QPS-search. This curve is obtained by varying the search parameters of HNSW. A curve positioned toward the upper right of the graph indicates higher search performance.

D All Experimental Results

The experimental results for SIFT1M [11] are shown in Figure 5, those for GIST1M [11] are presented in Figure 6, those for DEEP1M [13] are presented in Figure 9, and those for Glove100Angular [14] are presented in Figure 10. The results for SIFT1B [12] with a batch size of $b = 10^5$ are shown in Figure 7, while those with $b = 10^3$ are given in Figure 8. In the following sections, we discuss the experimental results for each evaluation metric.

D.1 QPS-Recall

From Figure 5a, Figure 6a, Figure 7a, Figure 8a, Figure 9a and Figure 10a, it is evident that rebuilding maintains search performance even after repeated insertions and deletions. In contrast, logical deletion significantly degrades search performance across all datasets as insertions and deletions are repeated. The plotted points in each graph indicate that both search accuracy and search speed deteriorate in this case. Furthermore, the search performance of physical deletion is slightly lower than that of rebuilding across all datasets.

D.2 1-Recall@10

From Figure 5b, Figure 6b, Figure 7b, Figure 8b, Figure 9b and Figure 10b, it can be observed that search accuracy in logical deletion decreases as insertions and deletions are repeated. Additionally, the accuracy of rebuilding is the highest, followed by physical deletion, which exhibits lower accuracy than rebuilding. Furthermore, Figure 7b and Figure 8b show that in physical deletion, search accuracy stabilizes after a certain number of insertion and deletion steps. This indicates that the structural properties of the graph become stable after a sufficient number of operations. Moreover, a larger batch size results in a higher converged accuracy. This suggests that when insertions and deletions are performed repeatedly, a larger batch size facilitates better recovery of the graph structure during the insertion process.

D.3 Memory Usage

From Figure 5c, Figure 6c, Figure 7c, Figure 8c, Figure 9c and Figure 10c, it is evident that memory consumption in logical deletion increases linearly with each step across all datasets. This indicates that in logical deletion, the deleted data remains in memory. In contrast, memory usage remains unchanged for both rebuilding and physical deletion. This confirms that these methods effectively reclaim memory space when data is deleted.

D.4 QPS-add

From Figure 5d, Figure 6d, Figure 7d, Figure 9d and Figure 10d, it can be observed that when data is inserted and deleted in batches of $b = 10^5$, the data insertion speed remains unchanged. However, as shown in Figure 8d, when the batch size is reduced to $b = 10^3$, the data insertion speed in logical deletion exhibits significant variations at each step.

Additionally, when data insertion and deletion are performed in batches of $b = 10^5$, physical deletion exhibits the highest data insertion speed. This is likely because repeated physical deletions gradually make the graph sparser, thereby reducing the number of distance calculations required during data insertion.

D.5 QPS-delete

From Figure 5e, Figure 6e, Figure 7e, Figure 8e, Figure 9e and Figure 10e, it is evident that across all datasets, logical deletion achieves the highest data deletion speed, on the order of approximately $10^9[1/\mathrm{s}]$. In contrast, both rebuilding and physical deletion operate at a significantly lower speed, at most on the order of $10^3[1/\mathrm{s}]$. When data insertion and deletion are performed in batches of $b = 10^5$, logical deletion can be completed in approximately $10^{-4}[\mathrm{s}]$, whereas physical deletion requires up to $10^2[\mathrm{s}]$.

From Figure 5e and Figure 6e, it can be observed that the dimensionality of the inserted and deleted vectors affects only the speed of rebuilding. SIFT1M [11] has a dimensionality of 128, whereas GIST1M [11] has a dimensionality of 960. This difference impacts rebuilding because it requires distance calculations during deletion. As the vector dimensionality increases, the time needed for a single-distance calculation also increases, leading to slower deletion speeds.

From Figure 7e and Figure 8e, it can be observed that the data deletion speed of physical deletion remains almost unchanged regardless of the batch size b. This indicates that physical deletion primarily involves memory operations for the specified deletion queries, leading to a consistent processing speed. Specifically, when the batch size is reduced from $b = 10^5$ to $b = 10^3$, meaning the number of deletions per step is reduced to $1/100$, the deletion speed of physical deletion remains

Algorithm 4: Deletion Control

```
Input: required search accuracy $\alpha$; estimated $\theta$ and $\pi$.
1 if $a \leq \theta$ then
2     foreach update step do
3         DELETE via physical deletion;
4 else
5     $s \leftarrow 0$
6     while for each update step do
7         DELETE via logical deletion;    $s \leftarrow s + 1$
8         if $s = \pi$ then
9             CONSTRUCT to rebuild from current $\mathcal{P}$;    $s \leftarrow 0$
```

nearly constant. In contrast, the speed of rebuilding decreases by approximately a factor of 100. This is because when the dataset size is relatively small, the processing time required for rebuilding remains almost constant, regardless of the number of deleted data points.

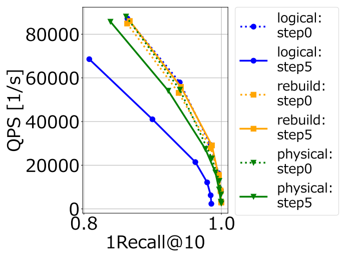
D.6 QPS-search

From Figure 5f, Figure 6f, Figure 7f, Figure 8f and Figure 9f, it is evident that across all datasets, search speed is highest when using physical deletion. As discussed in Sec D.4, this is likely because physical deletion gradually makes the graph sparser, reducing the number of distance calculations required during the search.

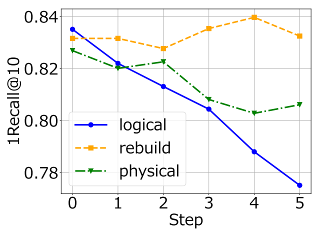
Similarly, across all datasets, logical deletion results in the slowest search speed. This is likely because, in logical deletion, an additional operation is required after the standard search process: the retrieved results must be filtered by referencing a flag array to exclude deleted data.

E Deletion Control

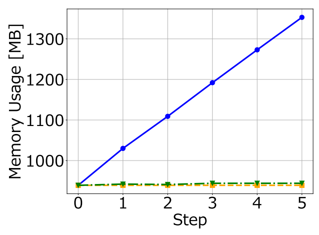
The algorithm of Deletion Control strategy is shown in Algorithm 4.



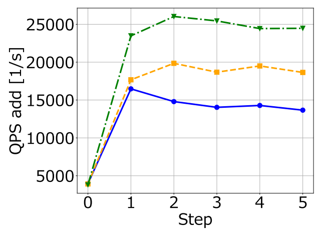
(a) QPS-Recall



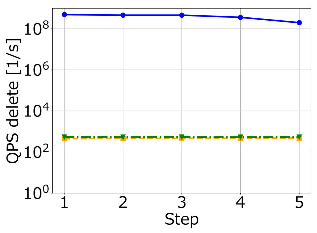
(b) 1-Recall@10



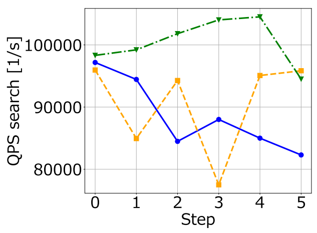
(c) Memory Usage



(d) QPS-add

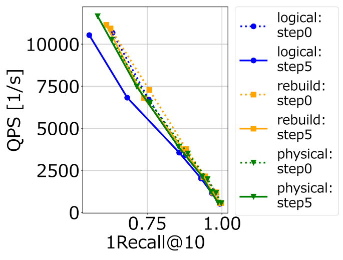


(e) QPS-delete

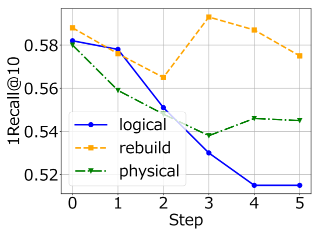


(f) QPS-search

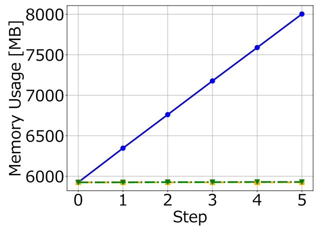
Figure 5: Performance comparison of the three deletion methods at each step on SIFT1M.



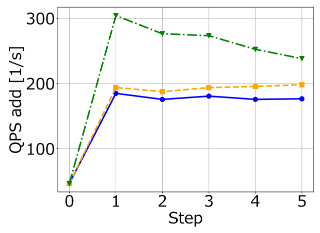
(a) QPS-Recall



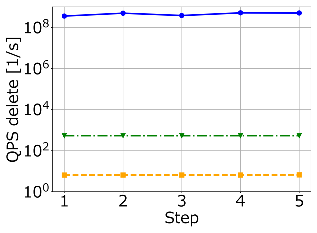
(b) 1-Recall@10



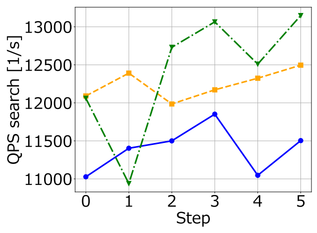
(c) Memory Usage



(d) QPS-add

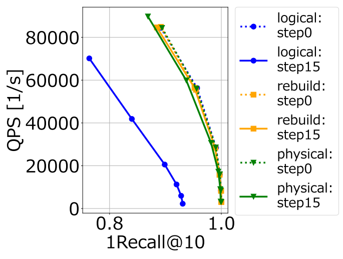


(e) QPS-delete

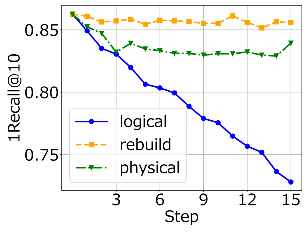


(f) QPS-search

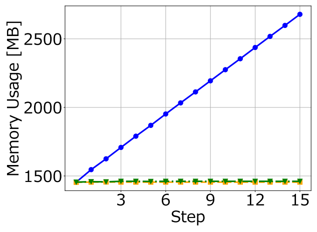
Figure 6: Performance comparison of the three deletion methods at each step on GIST1M.



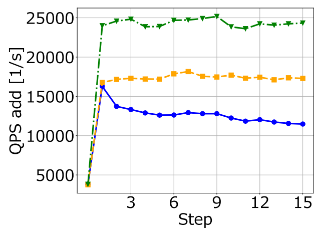
(a) QPS-Recall



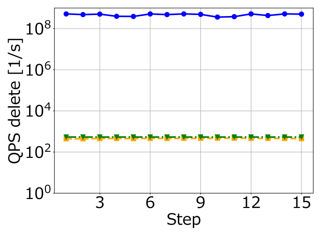
(b) 1-Recall@10



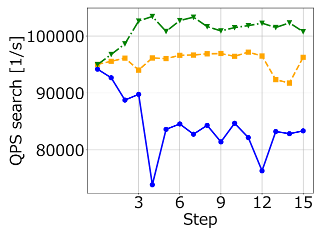
(c) Memory Usage



(d) QPS-add

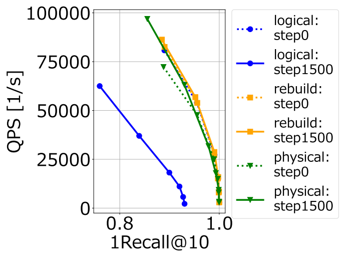


(e) QPS-delete

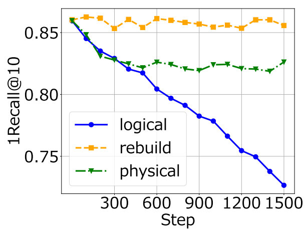


(f) QPS-search

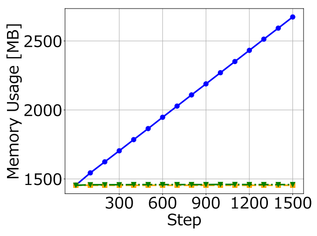
Figure 7: Performance comparison of the three deletion methods at each step on SIFT1B with $b = 10^5$.



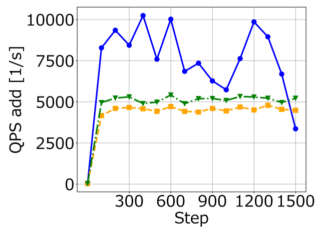
(a) QPS-Recall



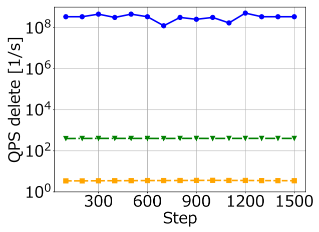
(b) 1-Recall@10



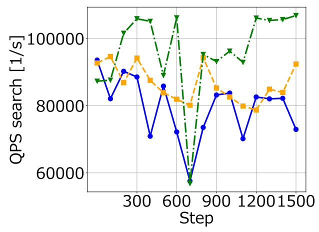
(c) Memory Usage



(d) QPS-add

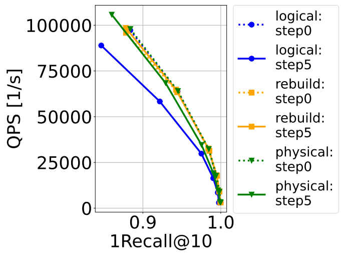


(e) QPS-delete

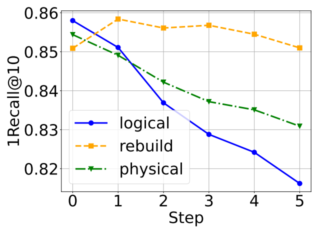


(f) QPS-search

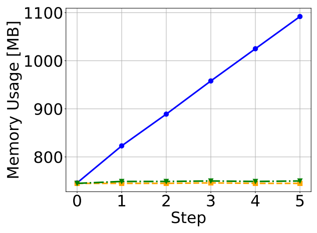
Figure 8: Performance comparison of the three deletion methods at each step on SIFT1B with $b = 10^3$.



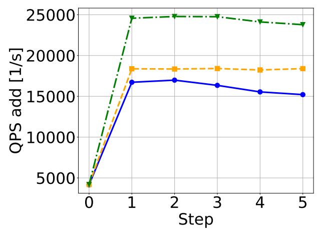
(a) QPS-Recall



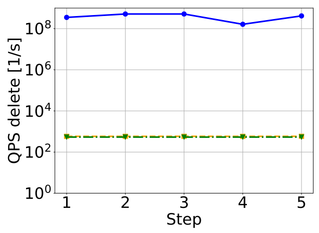
(b) 1-Recall@10



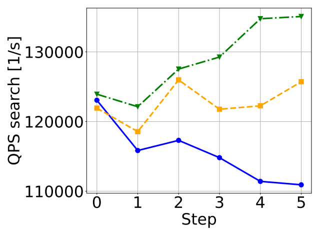
(c) Memory Usage



(d) QPS-add

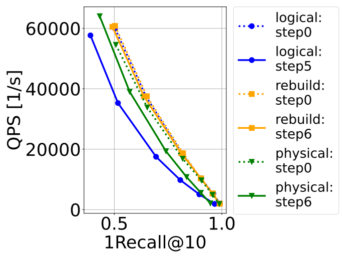


(e) QPS-delete

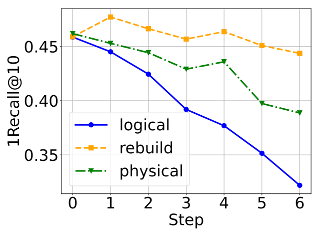


(f) QPS-search

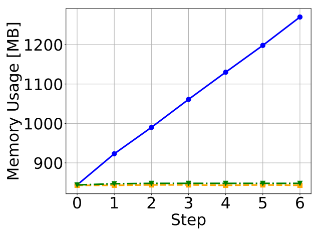
Figure 9: Performance comparison of the three deletion methods at each step on DEEP1M.



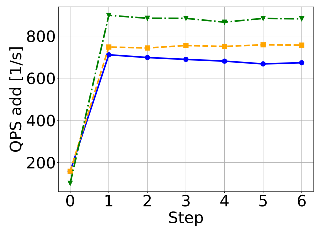
(a) QPS-Recall



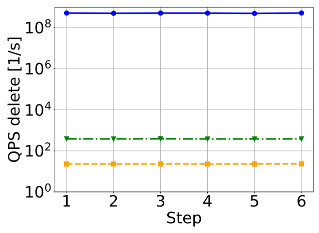
(b) 1-Recall@10



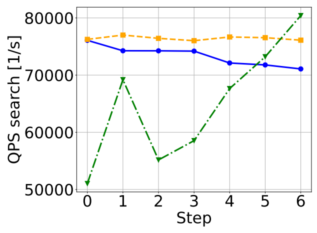
(c) Memory Usage



(d) QPS-add



(e) QPS-delete



(f) QPS-search

Figure 10: Performance comparison of the three deletion methods at each step on GLOVE100.