

Article

L-rCBF: Learning-Based Key–Value Data Structure for Dynamic Data Processing

Yejee Lee and Hayoung Byun * 

Department of Electronics Engineering, Myongji University, Yongin 17058, Republic of Korea; yejee0821@mju.ac.kr

* Correspondence: hbyun@mju.ac.kr

Abstract: Learning-based data structures, such as a learned Bloom filter and a learned functional Bloom filter (L-FBF), have recently been proposed to replace traditional structures. However, using these structures for dynamic data processing is difficult because a specific element cannot be deleted from a trained model. A counting Bloom filter with return values (rCBF) is a more efficient key–value structure than a functional Bloom filter (FBF) for repetitive insertions and deletions. In this study, we propose a learned rCBF (L-rCBF) comprising a model, a Bloom filter, and an rCBF and the deletion algorithm for the L-rCBF. To delete a specific element from the L-rCBF, two different operations are performed according to four different cases. In the experiments, the proposed L-rCBF is compared with a single rCBF and an L-FBF in terms of undeletables and search failures, and this comparison is conducted using two different models. In addition, we present a theoretical analysis of the rCBF with experimental results to demonstrate that a structure with an rCBF is more suitable for dynamic data than a structure with an FBF.

Keywords: key–value data structure; Bloom filter; learning-based structure; dynamic data; deep learning



Citation: Lee, Y.; Byun, H. L-rCBF: Learning-Based Key–Value Data Structure for Dynamic Data Processing. *Appl. Sci.* **2023**, *13*, 12116. <https://doi.org/10.3390/app132212116>

Academic Editor: Giacomo Fiumara

Received: 5 October 2023

Revised: 28 October 2023

Accepted: 4 November 2023

Published: 7 November 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

A Bloom filter (BF) is a simple probabilistic data structure using hash functions [1]. The BF can be used to identify elements included in a stored set and filter malicious information. The BF may return false positives for non-elements; however, it ensures no false negatives, which means that the BF returns positives for all programmed elements. Because it is space-efficient, numerous variants of BFs have been proposed and used in various applications, such as IP address lookup [2,3], named data networking (NDN) [4–6], packet classification [7,8], distributed systems [9–11], network security [12,13], and cloud computing [14,15]. A counting Bloom filter (CBF) and stable Bloom filter (SBF), both BF variants using counters, were proposed to support deletions for dynamic data due to the limitation of the standard BF, which can be used only for static data [16–18].

A functional Bloom filter (FBF) [19,20], a BF variant, is a key–value data structure capable of returning a value corresponding to a key in a given set, and it can replace tree- and hash-based structures in numerous applications. A standard BF is a bit array, while an FBF is an array of cells composed of multiple bits. In other words, a BF is used for a binary classification task because it can only answer membership queries, while an FBF is used for a multi-class classification task. The FBF provides insertion and search operations, and depending on the type of data (i.e., dynamic or static data), the FBF can be designed to support or not support a deletion operation. The FBF is more suitable for static data than dynamic data because an FBF designed for static data is better than that for dynamic data in terms of search performance. In other words, the FBF for dynamic data has a weak point: *conflict cells*, which are cells with two or more elements inserted, cannot be used for FBF operations; hence, if elements are repeatedly inserted and deleted, the number of conflict cells increases, resulting in a degradation of the search performance of the FBF.

A counting Bloom filter with return values (rCBF) is a key–value data structure suitable for applications that handle dynamic data, such as the pending interest table (PIT) lookup in NDN [5]. Unlike a BF, an rCBF is an array of cells composed of multiple bits and is used for a multi-class classification task. Each cell in the rCBF consists of two fields: *counter* and *value*. The rCBF provides the same functionalities as the FBF and supports dynamic applications better than the FBF because the rCBF uses counters. Because cells with two or more elements inserted can be used for rCBF operations, if insertions and deletions are repeated, replacing the FBF with an rCBF is appropriate.

Several data structures applying machine learning have recently been proposed [20–22]. Learning-based structures can guarantee the same operations and characteristics as traditional data structures and enhance the performance of the structures. A learned Bloom filter (LBF), a learning-based data structure for a binary classification task, provides the same semantic guarantees as a BF and comprises a learned model and a BF [21,22]. A learned FBF (L-FBF), a learning-based key–value structure for a multi-class classification task, provides the same semantic guarantees as an FBF and comprises a learned model, BF, and FBF [20]. The LBF and L-FBF can significantly improve search performance (i.e., search failure rates, including false positive rates) when using the same memory size as the BF and FBF, respectively. However, the LBF and L-FBF cannot be used for dynamic data processing because the elements stored in the structures cannot be deleted. Hence, learning-based BF structures have the following problems.

- Designing a deletion algorithm for a learning-based structure is difficult because once a model is trained, it does not provide a deletion for a specific element; hence, most existing learning-based structures cannot be used for dynamic data processing [21,22].
- To delete specific elements, learning-based structures should utilize auxiliary structures other than the learned model. If a deletion algorithm for a learning-based structure can be designed, the application range in which the structure can be used will widen considerably.
- In the case of the LBF, a deletion operation cannot be provided because the auxiliary structure is a standard BF. For the deletion operation, the standard BF should be replaced with an updatable BF variant for dynamic data. A stable learned Bloom filter (SLBF) is the first LBF variant for dynamic data and consists of a learned model and SBFs [23]. However, the SLBF can be used only for binary classification.
- The L-FBF was presented for static data in [20]; hence, a deletion algorithm of the L-FBF has never been proposed. In other words, because the auxiliary structures of the L-FBF in [20] are a standard BF and FBF for static data, a deletion operation for the L-FBF cannot be provided.

In this study, we focus on key–value data structures using a learned model for dynamic data. We propose a learned rCBF (L-rCBF) to improve the search and deletion performances under the constraint of the same memory size as a single rCBF and propose a deletion algorithm for the L-rCBF to process dynamic data. To the best of our knowledge, the L-rCBF is the first LBF variant for key–value storage that can support deletions. The contributions of this study can be summarized as follows.

- We propose a learning-based key–value structure for dynamic data processing, called L-rCBF, comprising a learned model, BF, and rCBF. The L-rCBF can be used for multi-class classification. The memory requirement of the learned model does not increase in proportion to the size of the data; therefore, as the data size increases, the proposed L-rCBF is more efficient than an rCBF.
- We propose a deletion algorithm for the L-rCBF. Because the rCBF is suitable for dynamic data as an auxiliary structure, the deletion algorithm can be designed even though the other auxiliary structure is a standard BF. To delete an element, the algorithm performs one of two operations according to four cases: deleting the element from the rCBF in the L-rCBF or programming it into the BF in the L-rCBF.

- The proposed L-rCBF is constructed similarly to the L-FBF. Hence, if the FBF for static data is replaced with that for dynamic data in the L-FBF, the deletion algorithm for the L-rCBF can be applied to the L-FBF.
- We theoretically analyzed the probabilities of *undeletables* and *search failures* for an rCBF and demonstrated the superiority of the proposed L-rCBF through simulation results.

The remainder of this paper is organized as follows. Section 2 describes BF variants, including an FBF and rCBF, and network applications using the BF variants. Section 3 describes the proposed L-rCBF and deletion algorithm. Section 4 describes the theoretical analysis, including the undeletable and search failure probabilities of the rCBF. Section 5 compares the performances of the L-rCBF with a single rCBF and L-FBF, and compares the probabilities between the theoretical and experimental results for the rCBF. Finally, Section 6 concludes the paper.

2. Related Work

2.1. Functional Bloom Filter

The functional Bloom filter (FBF) is a data structure that stores values corresponding to keys, and the key–value pairs are included in set $S = \{(x_1, v_1), (x_2, v_2), \dots, (x_n, v_n)\}$. An FBF is an m -cell array, and each cell stores a value. The FBF can perform three operations: insert (program), search (query), and delete. The FBF can be configured to support the deletion operation depending on whether the data are static or dynamic. The insert operation for dynamic data is slightly different from that for static data. If deletions are infrequent, the FBF is constructed for static data because the search performance of the FBF for static data is better than that for dynamic data. Most previous studies described the FBF for static data without considering deletions; however, we explain an FBF that supports deletion for dynamic data.

The operations use k hash functions to access k cells. The optimal k for the BF structures can be calculated as follows [24,25]:

$$k = \frac{m}{n} \ln 2, \quad (1)$$

where n denotes the number of elements in set S . Let L be the number of bits in a single cell for a value. Each cell is initialized to 0 and can represent $2^L - 2$ values because the maximum value $2^L - 1$ is reserved to represent a *conflict* cell, which implies that two or more elements are stored in the cell.

In inserting an element (x_j, v_j) into an FBF for dynamic data, for $1 \leq j \leq n$, all cells with value 0 among the cells pointed to by k hash functions are set to value v_j , and the other cells, which have values different from 0, are set to *conflict* cells with $2^L - 1$. For an FBF for static data, the cells that already have the value v_j retain this value even if the keys already stored in the cells are not equal to x_j .

In searching for an input in set U (i.e., $U = S \cup S^c$), an FBF returns one of three search results: *negative*, *positive*, or *indeterminable* (INDET). If at least one of the k cells has a value of 0 or if two or more cells among the cells have different values, a negative is returned from the FBF. If every cell except for the conflict cells has the same value, the value is returned, which implies a positive; however, positives may be false owing to hash collisions. If all cells have value $2^L - 1$ (i.e., conflict cells), an INDET is returned. False positive (FP) and INDET results are classified as *search failures*.

In deleting an element in an FBF for dynamic data, only cells storing one element can be used among the k cells. That is, among the cells pointed to by the k hash functions, the cells, except for the conflict cells, are changed to 0. Conflict cells with two or more elements inserted are not used for deletion to prevent false negatives (FNs). If all accessed k cells are conflict cells, the element cannot be deleted and is called an *undeletable*. In addition, the FBF for static data does not support deletion operations.

Even though the FBF operations can be adjusted for dynamic data, the FBF is not perfectly suitable for frequent insertions and deletions because conflict cells cannot be used for the three operations.

In addition, in the three operations of BF structures (i.e., FBF and rCBF), the cells pointed to by k hash functions are accessed. Once a BF structure is constructed, the k does not change while repeating insertions and deletions, unless the structure is reconstructed. Hence, the time complexity of the operations is $O(1)$.

2.2. Counting Bloom Filter with Return Values

The counting Bloom filter with return values (rCBF) is a key–value data structure suitable for dynamic data processing [5]. An rCBF is an m -cell array, where each cell consists of *counter* and *value* fields. The rCBF performs three operations using k hash functions: insert (program), search (query), and delete. The optimal k used for the rCBF can be calculated using $\frac{m}{n} \ln 2$ as in (1), where n is the number of elements in set $S = \{(x_1, v_1), (x_2, v_2), \dots, (x_n, v_n)\}$ [24,25].

Let L be the number of bits in the value field and R the number of bits in the counter field. Each value field can represent $2^L - 1$ values because the rCBF does not need to reserve a value to represent a *conflict* cell, unlike the FBF, which can represent $2^L - 2$ values. Each counter field represents the number of inserted elements (c) in each cell, and the maximum count c_{max} is $2^R - 1$.

Figure 1 presents an example of inserting elements into an rCBF with $n = 3$, $m = 7$, and $k = 3$. To insert element (x_3, v_3) into the cells pointed to by $h_1(x_3)$, $h_2(x_3)$, and $h_3(x_3)$, v_3 is XORed with the values in the cells (i.e., $\text{rCBF}[h_i(x_3)].\text{value} = \text{rCBF}[h_i(x_3)].\text{value} \oplus v_3$ for $i = 1, 2, 3$), and all counters in the cells are incremented by 1, except for counters with c_{max} (i.e., $\text{rCBF}[h_i(x_3)].\text{count} = \text{rCBF}[h_i(x_3)].\text{count} + 1$).

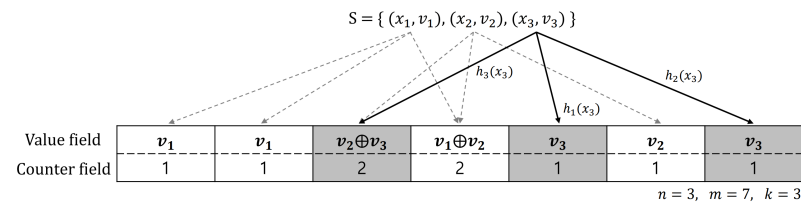


Figure 1. Counting Bloom filter with return values (rCBF).

In the search operation, an rCBF returns one of three search results: *negative*, *positive*, or *indeterminable* (INDET). For a given input in set U (i.e., $U = S \cup S^c$), if at least one counter in the k cells is 0 or if two or more cells among the cells with $c = 1$ have different values, a negative is returned from the rCBF. If the value fields in cells with $c = 1$ are identical, the value in the fields is returned, which implies a positive; however, positives may be false owing to hash collisions. If all counters are greater than 1, an INDET is returned. FP and INDET results are classified as *search failures*.

In the rCBF, even cells with two or more elements inserted can be used to delete elements; hence, elements can be deleted efficiently, improving the deletion performance. Algorithm 1 shows the procedure for deleting an element from the rCBF. To delete element (x, v) in S , v is XORed with the values in the cells pointed to by the k hash functions, and all counters in the cells are decremented by 1, except for counters with 0 or c_{max} . For a cell with c_{max} , the counter in the cell should not be decremented to prevent an FN because the cell may have more than c_{max} elements. If the accessed k cells have counters with c_{max} , the element cannot be deleted and is called an *undeletable*. However, the deletion performance of the rCBF is better than that of the FBF because the rCBF can use cells with hash collisions, except for those with c_{max} .

Algorithm 1 Deletion algorithm of rCBF

```

1: procedure delete_rCBF(rCBF,  $x$ ,  $v$ )
2:   for  $i \leftarrow 1$  to  $k$  do
3:     if rCBF[ $h_i(x)$ ].count  $\neq c_{max}$ 
       && rCBF[ $h_i(x)$ ].count  $\neq 0$  then
4:       rCBF[ $h_i(x)$ ].value = rCBF[ $h_i(x)$ ].value  $\oplus v$ 
5:       rCBF[ $h_i(x)$ ].count = rCBF[ $h_i(x)$ ].count  $- 1$ 

```

2.3. Network Applications Using Bloom Filter Variants

BF variants have been used in various network applications, such as IP address lookup, NDN, packet classification, and distributed systems, because BF structures can filter and search for specific information and the structures are space-efficient. Depending on the purpose of each application, appropriate BF variants can be utilized. For IP address lookup, an efficient prefix-matching algorithm is required to identify the longest prefix. Because a BF can filter out unnecessary accesses, parallel BF and BF-chaining methods were proposed for longest prefix matching [2,3].

In NDN, efficient name lookup algorithms are required in forwarding information base (FIB) and pending interest table (PIT). For FIB lookup, because name-based longest prefix matching should be performed for URL data, an index data structure was proposed with a hash table and a CBF to balance the load [4]. For PIT lookup, an efficient exact-matching algorithm is required, and frequent update operations (i.e., insertions and deletions) are performed. Hence, BF-based PIT architectures were proposed using the FBF and rCBF [5].

Packet classification is a technique to categorize network traffic using multiple header fields to implement quality of service. In software defined networking (SDN), because more fields are used for packet classification, the flow table with all the information is stored in an external memory. Packet classification algorithms using BF variants, such as a standard BF, cuckoo BF, and LBF, were proposed to reduce the number of accesses to the external memory [7,8].

In distributed systems, two tasks exist: reconciliation to obtain the union of the sets of two hosts and data deduplication to eliminate duplicate data between two hosts [9,10,15]. In the tasks, the set difference between two hosts should be efficiently computed, and communication complexity should be minimized. Because of space and access efficiencies, BF structures (i.e., a standard BF and an invertible BF) were used for the tasks [9,11].

3. Proposed Work

This paper proposes a learning-based key–value structure, learned rCBF (L-rCBF), which can provide deletion for dynamic data. Learning-based structures can achieve better performance with the same amount of memory as traditional structures. The basic concept of this structure was briefly introduced in [26] by the same authors.

3.1. Learned rCBF

We aim to propose a learning-based data structure that guarantees the same semantic characteristics as a single rCBF suitable for dynamic data processing. A learned model for multi-class classification can be used to construct a key–value data structure because the structure can be considered a multi-class classification task with Q classes if Q values are available for the key–value pairs. However, the model is not fully representative of the characteristics of an rCBF because the model can return false negatives (FNs) and false-class results (FRs), while the rCBF never produces an FN or FR. Hence, appropriate auxiliary structures should be added to provide the same semantic guarantees as the rCBF.

The proposed L-rCBF comprises a learned model, false-class result BF (FR-BF), and verification rCBF (V-rCBF) to guarantee the same semantic characteristics as a single rCBF. The model has the advantage of maintaining a consistent memory requirement regardless of the size of the positive data; however, it may produce FNs by returning negative class

0 for elements in positive class v (i.e., classified as non-elements included in S^c), and FRs by returning positive class q for elements belonging to class v for $1 \leq v, q \leq Q$ and $v \neq q$. Because the rCBF never produces an FN or FR, a single BF and rCBF should be constructed to prevent FNs and FRs.

Because both the rCBF and FBF are BF structures that can be used as key–value structures, even though the rCBF is better than the FBF in terms of dynamic data processing, the only difference between L-rCBF and L-FBF is whether the rCBF or FBF is used as the verification structure. Hence, because the auxiliary structures of the L-rCBF and L-FBF are similar, the construction and search procedures of the L-rCBF are the same as those of the L-FBF in [20], except for the construction and search of the V-rCBF (i.e., using the V-rCBF instead of the verification FBF (V-FBF)).

In constructing the L-rCBF, the model is first trained for sets S and S_{tr}^c (i.e., S^c for training), and then tested for every element in S to construct the FR-BF and V-rCBF. The model learns the distribution of elements included in each positive class (i.e., elements in class v for $1 \leq v \leq Q$) and that of non-elements (i.e., non-elements in negative class 0) from the training data. Improving the accuracy of the model is crucial because as the accuracy increases, the elements that must be stored in the auxiliary structures decrease, resulting in a reduction in the overall memory requirements of the L-rCBF.

After training the model and before testing it, a threshold (τ_m) in the model should be set to adjust the false positive rate (FPR) of the L-rCBF. Even though the additional FR-BF and V-rCBF can solve the FRs and FNs from the model, the auxiliary structures cannot control the FPR. Furthermore, the elements stored in the FR-BF and V-rCBF vary depending on τ_m . In other words, a large τ_m can reduce the FPR from the model; however, it may cause more FNs from the model, which implies that a greater number of elements should be programmed into the V-rCBF. In the experiment, we first set the desired FPR of the model, and the τ_m was set according to the desired FPR using the validation set S_{vd}^c (i.e., S^c for setting τ_m). After setting τ_m , the model is tested for elements in set S . Elements with FR returned from the model (i.e., S_{FR}) are programmed into the FR-BF; elements with FR or FN returned from the model (i.e., S_{FR} or S_{FN}), as well as those with FP from the FR-BF (i.e., B_{FP}) among those with true positive (TP) from the model (i.e., S_{TP}), are programmed into the V-rCBF (i.e., $S_V = S_{FR} \cup S_{FN} \cup B_{FP}$ and $B_{FP} \subset S_{TP}$).

In the search procedure for the L-rCBF, the model is first tested for a given input in set U (i.e., $U = S \cup S_{ts}^c$ for the test and $S_{ts}^c = S^c - S_{tr}^c - S_{vd}^c$). If a negative is returned, the V-rCBF is queried to prevent FNs and returns the final result (i.e., negative, positive (a value), or *indeterminable* (INDET)). If the model returns a value, the FR-BF is queried to filter out an FR. If a positive is returned from the FR-BF, the V-rCBF is queried to return a true result because a positive from the FR-BF implies that a positive class (value) returned from the model is false (i.e., FR). Hence, the V-rCBF returns the final result. Otherwise, if a negative is returned from the FR-BF, the value from the model is returned as the final result because the value is not an FR.

3.2. Deletion Algorithm of L-rCBF

The proposed L-rCBF should provide a deletion to guarantee the same characteristics as those of a single rCBF. However, once a model in the L-rCBF is trained on a set, specific elements cannot be deleted from the model because it learns the distribution of the set. Therefore, we propose a deletion algorithm for elements inserted using the auxiliary structures (i.e., the FR-BF and V-rCBF). In other words, the novelty of this paper lies in the fact that a key–value structure using a learned model can support deletions. Although L-rCBF is similar to L-FBF in terms of both construction and search procedures, the L-FBF in [20] cannot support deletions because the V-FBF in the L-FBF is for static data without considering deletions.

Algorithm 2 shows the deletion algorithm for the L-rCBF. When searching for an element after deleting it, the L-rCBF should return a negative. Hence, depending on which set the element to be deleted belongs to (i.e., S_{FN} , S_{FR} , B_{FP} , or $S_{TP} - B_{FP}$), one of two

operations is performed: OP1) delete the element from the V-rCBF or OP2) program the element to the FR-BF.

Algorithm 2 Deletion algorithm of L-rCBF

```

1: procedure delete_LrCBF( $x, v$ )
2:    $result = testModel(x)$ 
3:   if  $result == 0$  then                                     ▷ FN from model
4:      $operation = OP1$ 
5:   else if  $result \neq v$  then                                   ▷ FR from model
6:      $operation = OP1$ 
7:   else                                                       ▷ TP from model
8:     if queryBF(FR-BF,  $x$ ) then                               ▷ FP from FR-BF
9:        $operation = OP1$ 
10:    else
11:       $operation = OP2$ 
12:    if  $operation == OP1$  then
13:      delete_rCBF(V-rCBF,  $x, v$ )
14:    else
15:      programBF(FR-BF,  $x$ )
  
```

To delete element (x, v) in S for $1 \leq v \leq Q$, a learned model is first tested (line 2). If the element is included in S_{FN} , the model returns a negative (i.e., $result == 0$), which implies that the element is stored in the V-rCBF; hence, for the V-rCBF to return a negative as a final result, the element is deleted from the V-rCBF (lines 3, 4, 12, and 13). If the element is included in S_{FR} , the model returns a value not equal to v , resulting in a query to the FR-BF. Because every element in S_{FR} is stored in the FR-BF, it returns a positive, and the V-rCBF is then queried. Hence, the element is deleted from the V-rCBF (lines 5, 6, 12, and 13).

Even though the model returns v (i.e., S_{TP}), the element is deleted from the V-rCBF if the FR-BF returns a positive (i.e., B_{FP} , lines 8, 9, 12, and 13). Otherwise, if the FR-BF returns a negative (i.e., $S_{TP} - B_{FP}$), the element is programmed into the FR-BF (lines 11 and 15) to change the final result of the search, from v to negative, after the deletion. In other words, when searching for the element in $S_{TP} - B_{FP}$ after OP2, the FR-BF returns a positive, and the V-rCBF then returns a negative as the final result because the element has never been programmed into the V-rCBF. In addition, the time complexity of the deletion algorithm is $O(1)$ because the model is tested once and one of the auxiliary structures (i.e., V-rCBF or FR-BF) is accessed to perform OP1 or OP2.

After performing OP2, the L-rCBF may return a few FNs because additional FPs from the FR-BF may occur. Before deletions, elements in $S_{TP} - B_{FP}$ are not stored in both the FR-BF and V-rCBF because the model returns true values for the elements. However, if additional FPs from the FR-BF occur after deleting some elements in $S_{TP} - B_{FP}$, the V-rCBF is queried for the elements with FPs (i.e., those in a subset of $S_{TP} - B_{FP}$) and returns negatives, which are false results (i.e., FNs). To prevent such FNs, when the number of deleted elements exceeds a threshold (τ_d), the L-rCBF is reconstructed for the remaining elements. τ_d can be set according to the number of additional FPs from the FR-BF ($n_{fp,b}$) because the FPs from the FR-BF cause the V-rCBF to return the FNs as the final results. Thus, τ_d is set to the number of deleted elements at the point where the theoretically calculated $n_{fp,b}$ is not greater than and close to 1.

The L-rCBF has two cases of *undeletables*: undeletable by FP (UNDEL-FP) and undeletable by conflict cells (UNDEL-C). For UNDEL-FPs, the L-rCBF returns a value for a deleted element because an FP occurs in the V-rCBF. For UNDEL-Cs, when performing OP1, if all accessed cells have maximum counters in the V-rCBF, the element cannot be deleted from the V-rCBF, resulting in an undeletable from the L-rCBF.

Figure 2a,b show the access paths of the four types of elements in the L-rCBF when searching before and after deleting them, respectively. For an element (x_1, v_1) in set S_{FN} , before deleting it, the model returns a negative, and the V-rCBF then returns v_1 as a final

In this section, we theoretically analyze the *undeletable* and *search failure* probabilities of a single rCBF because the performance of the V-rCBF is a crucial determinant that significantly affects the performance of the proposed L-rCBF. Additionally, we briefly analyze the probabilities of a single FBF for dynamic data. Consequently, this section demonstrates that the L-rCBF utilizing an rCBF is more suitable for dynamic data than the L-FBF using an FBF.

4.1. Undeletable Probability of rCBF

In deleting element (x, v) from an rCBF, the *undeletable* of the rCBF means that it cannot be deleted because every cell pointed to by k hash functions has c_{max} . The *undeletable* probability of the rCBF ($P(U_r)$) can be obtained in a similar manner to that of a ternary BF in [27] because both structures do not allow the decrement of counters with c_{max} .

Let $p_{(j)}$ be the probability that a specific cell is selected by j hash indexes among the $k(n-1)$ indexes of $n-1$ elements, except for x to be deleted. Then,

$$p_{(j)} = \binom{k(n-1)}{j} \cdot \left(\frac{1}{m}\right)^j \cdot \left(1 - \frac{1}{m}\right)^{k(n-1)-j}. \quad (2)$$

If the specific cell is the one selected by one of the indexes of x to be deleted, the maximum j to avoid the cell with c_{max} is $c_{max} - 2$. In other words, if $j = c_{max} - 1$, the cell has c_{max} because the cell is selected by one index of x and selected by $c_{max} - 1$ indexes of $n-1$ elements. Hence, the elements stored in the cell cannot be deleted. Therefore, $P(U_r)$ can be calculated as follows:

$$P(U_r) = \left(1 - \sum_{j=0}^{c_{max}-2} p_{(j)}\right)^k, \quad (3)$$

where $\sum_{j=0}^{c_{max}-2} p_{(j)}$ denotes the probability that a specific cell does not have c_{max} ; hence, $1 - \sum_{j=0}^{c_{max}-2} p_{(j)}$ denotes the probability that a specific cell has c_{max} . If at least one of the k cells for x does not have c_{max} , then x can be deleted. Therefore, $P(U_r)$ is the probability that all k cells selected by x have c_{max} .

4.2. Search Failure Probability of rCBF

The search failures of an rCBF include both false positives (FPs) and indeterminables (INDETs). The search failure probability ($P(F_s)$) of the rCBF is formulated as follows, based on [20]:

$$\begin{aligned} P(F_s) &= P(I) + P(F_p) \\ &= \left(P(S)P(I|S) + P(S^c)P(I|S^c)\right) + P(S^c)P(F_p|S^c) \\ &= P(S)P(I|S) + P(S^c)\left(P(I|S^c) + P(F_p|S^c)\right), \end{aligned} \quad (4)$$

where $P(I)$ is the probability of an INDET for input y in set S or S^c , and $P(F_p)$ is the probability of an FP for input y in S^c .

$P(I|S)$ is the probability that every counter in the k cells for y included in S is two or more (i.e., $c \geq 2$). Hence,

$$P(I|S) = (p_{ci})^k, \quad (5)$$

where p_{ci} is the probability that, for a specific cell pointed to by at least one of the k indexes for y in S , the cell is selected by at least one of the $k(n-1)$ indexes. Thus,

$$p_{ci} = 1 - \left(1 - \frac{1}{m}\right)^{k(n-1)}. \quad (6)$$

$P(I|S^c)$ is the probability that every counter in the k cells for y not in S (i.e., in S^c) is two or more (i.e., $c \geq 2$). Hence,

$$P(I|S^c) = (p_{cn})^k, \quad (7)$$

where p_{cn} is the probability that a specific cell is selected by at least two of the kn indexes of the n elements in S . Thus,

$$p_{cn} = 1 - \left(1 - \frac{1}{m}\right)^{kn} - \frac{kn}{m} \cdot \left(1 - \frac{1}{m}\right)^{kn-1}, \quad (8)$$

where $\left(1 - \frac{1}{m}\right)^{kn}$ is the probability that the cell is not selected by any of the kn indexes, and $\frac{kn}{m} \cdot \left(1 - \frac{1}{m}\right)^{kn-1}$ is the probability that the cell is selected by one of them.

$P(F_P|S^c)$ is the probability that the rCBF returns a value for y not in S (i.e., in S^c). Let Q be the number of return values; we assume that the values are uniformly distributed. Then,

$$P(F_P|S^c) = Q \sum_{i=1}^k \binom{k}{i} \cdot (p_{pn})^i \cdot (p_{cn})^{k-i}, \quad (9)$$

where p_{pn} is the probability that a specific cell has one of the Q values with $c = 1$. Hence,

$$p_{pn} = \frac{1}{Q} \cdot \frac{kn}{m} \cdot \left(1 - \frac{1}{m}\right)^{kn-1}. \quad (10)$$

The $P(F_P|S^c)$ of the rCBF in (9) seems to be the same as that of the FBF for static data [20]; however, p_{pn} and p_{cn} in (10) and (8) are not the same as those of the FBF for static data because the FBF does not support the deletion operation. Thus, the $P(F_P|S^c)$ of the rCBF is not the same as that of the FBF for static data.

If an FBF supporting the deletion operation is constructed, the search failure probability of the FBF is the same as that of the rCBF. In addition, the undeletable probability of the FBF ($P(U_f)$) is the same as $P(I|S)$ in (5). When using the same m , n , and k for the rCBF and FBF, $P(U_r) < P(U_f)$ under the constraint $c_{max} \geq 3$ (i.e., $R \geq 2$). Hence, the L-rCBF using an rCBF is more suitable for dynamic data processing than the L-FBF using an FBF.

5. Performance Evaluation

Section 5.1 compares the proposed L-rCBF with a single rCBF and L-FBF, and compares the performance of L-rCBFs composed of different models. Section 5.2 compares the theoretical results for undeletable and search failure probabilities of an rCBF with the experimental results.

5.1. Performance Comparison of rCBF, L-FBF, and L-rCBF

Our simulation was performed for datasets with specific distributions because learning-based data structures use the distribution of the elements in each set. A total of 245,514 URLs were used as a positive set (i.e., S) with six return values (i.e., six classes) [28], and 1,491,178 blacklisted URLs were used as a negative set (i.e., S^c) [29].

We compared five BF structures: a single rCBF, L-FBF₁, L-FBF₂, L-rCBF₁, and L-rCBF₂. Each of the two L-FBFs supports the deletion operation using a V-FBF for dynamic data. This performance comparison used two models with different classification accuracies and memory requirements. One model was included in the L-FBF₁ and L-rCBF₁, and the other model was included in the L-FBF₂ and L-rCBF₂. The memory requirement of a model generally increases with the accuracy of the model in a learning-based structure. However, this does not necessarily imply that the total memory requirement of the structure increases. For a fair comparison, we first constructed the L-rCBF₁, and then constructed the other four structures with the same memory requirement as that of the L-rCBF₁.

To train the two models, character-level pretrained embeddings were utilized with principal component analysis (PCA). The models include a long short-term memory (LSTM) layer, two one-dimensional convolutional neural network (CNN) layers, and three fully connected layers with softmax activation. However, because the hyperparameters of the two models are different, the memory requirements and accuracies of the models are also different. Table 1 compares the number of weights (w) and memory requirements of the

two models. The memory requirement of a model depends on w because it is calculated using w . Model₂ for the L-rCBF₂ and L-FBF₂ requires more memory than Model₁ for the L-rCBF₁ and L-FBF₁; however, Model₂ provides a higher level of accuracy.

Table 1. Comparison of memory requirements of two models.

	Model ₁	Model ₂
Number of weights (w)	6207	48,055
Memory requirements (kB)	24.828	192.22

Because an additional FP from the FR-BF causes an FN from the V-rCBF, an L-rCBF should be reconstructed at threshold τ_d before the FR-BF begins to return an additional FP owing to too many deletions. Threshold τ_d can be set to the number of deleted elements at the point where the theoretically calculated $n_{fp,b}$ is less than and close to 1. Let n_{tp} be $|S_{TP}|$, n_b the number of elements programmed in the FR-BF (i.e., $|S_{FR}|$), m_b the FR-BF size, k_b the number of hash functions of the FR-BF, and n'_b the number of elements to be programmed into the FR-BF for deletion. Therefore, $n_{fp,b}$ can be calculated as follows:

$$n_{fp,b} = (n_{tp} - n'_b) \cdot \left(1 - \left(1 - \frac{1}{m_b}\right)^{k_b(n_b + n'_b)}\right)^{k_b}. \quad (11)$$

Figure 3 shows the theoretical and experimental $n_{fp,b}$ in the L-rCBF₁ with Model₁ according to the elements deleted. When deleting more than 15% of the elements in the L-rCBF₁, $n_{fp,b} \geq 1$. Hence, τ_d can be predefined as 15% of the elements. In this experiment, reconstruction was not considered for a simple performance comparison; hence, to evaluate the deletion performance, 15% of the randomly selected URLs among those in S were deleted. In addition, a search performance experiment was performed for all URLs in U (i.e., $U = S \cup S_{ts}^c$).

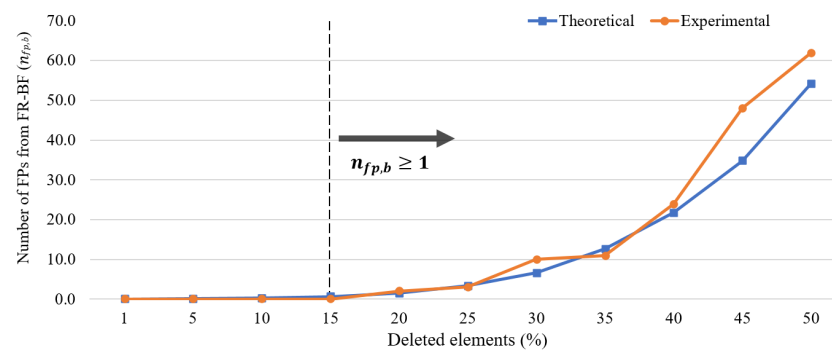


Figure 3. Number of false positives from the FR-BF in the L-rCBF₁ with Model₁ according to the elements deleted.

In the construction procedure, the L-rCBF₁ is constructed first, and then the other four structures are constructed with the same amount of memory as that of the L-rCBF₁. Let n_v be the number of elements programmed into a verification structure (i.e., $|S_v| = |S_{FR} \cup S_{FN} \cup B_{FP}|$), m_v the number of cells in the verification structure, α_v the size factor of the verification structure (i.e., $\alpha_v = \frac{m_v}{n_v}$), and α_b the size factor of the FR-BF (i.e., $\alpha_b = \frac{m_b}{n_b}$). The L-rCBF₁ comprises Model₁, a $32n_b$ -bit FR-BF, and an $8n_v$ -cell V-rCBF (i.e., $\alpha_b = 32$ and $\alpha_v = 8$ in the L-rCBF₁). To allocate the same amount of memory as in the L-rCBF₁, the size factors of the FR-BFs and verification structures should be adjusted in the L-FBF₁, L-FBF₂, and L-rCBF₂.

For the FR-BF, α_b depends on the model accuracy. As the accuracy of the model increases, n_{tp} increases and n_b decreases. To satisfy the condition $n_{fp,b} < 1$ until 15% of the elements are deleted in the learning-based structures, each α_b in the L-rCBF₂ and L-FBF₂

using Model₂ with a higher accuracy than Model₁ should be increased to increase m_b , considering (11). In other words, α_b in the L-FBF₁ is the same as that in the L-rCBF₁ (i.e., 32) because both structures use Model₁. However, each α_b in the L-FBF₂ and L-rCBF₂ is set to 49 to satisfy the condition. Hence, to support deletions, α_b should be increased as the accuracy of the model increases.

A verification structure is constructed with the remaining memory after constructing a model and FR-BF from the total memory. We assume that $R = 2$ and $L = 3$ because six values exist. A single cell in an rCBF has five bits (i.e., $R + L$ bits), whereas a single cell in an FBF has three bits (i.e., L bits). Therefore, if an L-FBF and L-rCBF include the same model, the α_v of the V-FBF in the L-FBF is greater than that of the V-rCBF in the L-rCBF. Therefore, the α_v in the L-FBF₁ is 13.33, 14.03 in the L-FBF₂, and 8.42 in the L-rCBF₂. In addition, the size factor of the single rCBF is 6.19.

Table 2 compares the *undeletable* rates of the rCBF, L-FBF₁, L-FBF₂, L-rCBF₁, and L-rCBF₂ when using the same amount of memory. No UNDEL-FPs from the rCBF and L-FBFs are observed because the structures return more UNDEL-Cs than the L-rCBFs. However, the total number of undeletables from each L-rCBF is less than those from the rCBF and L-FBFs. In terms of the deletion performance, structures using rCBFs outperform those using FBFs because of their counter fields. Even though each V-FBF in the L-FBFs has more cells than each V-rCBF in the L-rCBFs, the undeletable rates of the L-rCBF₁ and L-rCBF₂ improve by 83.67% and 76.67% compared with those of the L-FBF₁ and L-FBF₂, respectively.

Table 2. Comparison of undeletable rates when using same amount of memory (%).

	rCBF	L-FBF ₁	L-FBF ₂	L-rCBF ₁	L-rCBF ₂
UNDEL-FP	0	0	0	0.014	0.014
UNDEL-C	0.030	0.098	0.060	0.003	0
Total	0.030	0.098	0.060	0.016	0.014

Table 3 compares the *search failure* rates of the rCBF, L-FBF₁, L-FBF₂, L-rCBF₁, and L-rCBF₂ when using the same amount of memory. The reduction rate in search failures represents the proportion of reduced search failures observed in a learning-based structure relative to the single rCBF. The four learning-based structures improve the search failure rates compared with the single rCBF. When comparing L-FBFs with L-rCBFs, the search failure rates of the L-FBFs are better than those of the L-rCBFs because the α_v of the V-FBFs (i.e., 13.33 and 14.03) are greater than those of the V-rCBFs (i.e., 8 and 8.42). However, if insertions and deletions are repeated for dynamic data, the performance gap of the search failure rates between the L-rCBF and L-FBF with the same model would be reduced owing to an increase in the number of conflict cells in the V-FBF. Furthermore, because of the significantly superior deletion performance of the L-rCBFs compared with that of the L-FBFs, as shown in Table 2, the L-rCBFs are more appropriate than L-FBFs for dynamic data processing. In addition, when comparing the L-FBF₁ to L-FBF₂ or the L-rCBF₁ to L-rCBF₂, in terms of the search and deletion performances, each structure using Model₂ outperforms each structure using Model₁, despite Model₂ requiring more memory than Model₁, because of the higher accuracy of Model₂.

For static data, because insertions and deletions are infrequent and searches are primarily performed, using the L-FBF with an improved search performance is more efficient than using a single rCBF. Especially if the FBF for dynamic data is replaced with that for static data in the L-FBF, the search performance of the L-FBF improves. Table 4 compares the *search failure* rates of the rCBF, L-FBF₂ with the FBF for dynamic data, and L-FBF₂ with the FBF for static data when using the same amount of memory. Because the number of conflict cells in the FBF for static data is less than that for dynamic data, using the FBF for static data can reduce the number of INDETs included in set S . Hence, the L-FBF for static data is more efficient than a single rCBF and the L-FBF for dynamic data when insertions and deletions are infrequent.

Table 3. Comparison of search failure rates when using same amount of memory (%).

		rCBF	L-FBF ₁	L-FBF ₂	L-rCBF ₁	L-rCBF ₂
S	TP	94.896	99.901	99.938	98.767	99.181
	INDET ($I S$)	5.104	0.099	0.062	1.233	0.819
S ^c	TN	99.318	99.480	99.453	99.388	99.407
	FP ($F_P S^c$)	0.646	0.520	0.547	0.610	0.593
	INDET ($I S^c$)	0.036	0	0	0.002	0
Search failures	FP+INDET	1.307	0.258	0.245	0.998	0.734
Reduction rate in search failures compared with a single rCBF		-	80.26	81.25	23.64	43.84

Table 4. Comparison of search failure rates between L-FBFs for dynamic and static data (%).

		rCBF	L-FBF ₂	
			Dynamic Data	Static Data
S	TP	94.896	99.938	99.982
	INDET ($I S$)	5.104	0.062	0.018
S ^c	TN	99.318	99.453	99.453
	FP ($F_P S^c$)	0.646	0.547	0.547
	INDET ($I S^c$)	0.036	0	0
Search failures	FP+INDET	1.307	0.245	0.218
Reduction rate in search failures compared with a single rCBF		-	81.25	83.32

Additionally, we compare the L-rCBFs (i.e., L-rCBF₁ and L-rCBF₂) to two L-FBFs (i.e., L-FBF₃ and L-FBF₄), which are identical to the L-FBF₁ and L-FBF₂ except for the V-FBFs, and the α_v of the V-FBFs have the same values as those of the V-rCBFs in the L-rCBFs: 8 and 8.42, respectively. Table 5 compares the *undeletable* rates when using a verification structure with the same α_v . The undeletable rates of the L-rCBF₁ and L-rCBF₂ are reduced by 98.73% and 98.20%, respectively, compared with those of the L-FBF₃ and L-FBF₄, respectively. In terms of the search performance, if both an L-FBF and L-rCBF with the same model possess the same α_b and α_v values, the structures have the same search failure rates. Hence, the search failure rates of the L-rCBF₁ and L-FBF₃ are the same, and those of the L-rCBF₂ and L-FBF₄ are also the same. However, if insertions and deletions are repeated, the search failure rates of the L-rCBFs would be better than those of the L-FBFs.

Table 5. Comparison of undeletable rates when using verification structure with same α_v (%).

	L-FBF ₃	L-FBF ₄	L-rCBF ₁	L-rCBF ₂
UNDEL-FP	0.027	0.022	0.014	0.014
UNDEL-C	1.227	0.758	0.003	0
Total	1.255	0.779	0.016	0.014

5.2. Comparison of Probabilities between Theoretical and Experimental Results for rCBF

This section compares the *undeletable* and *search failure* probabilities (i.e., $P(U)$ and $P(F_s)$) between the theoretical and experimental results for the rCBF and FBF supporting the deletion operation. To obtain the results for $P(U)$ and $P(F_s)$, experiments were performed using 2^{15} random URLs for set S and $2 * 2^{15}$ URLs for S^c, and the URLs were obtained from ALEXA [30]. To allocate the same number of bits to a value field in the rCBF and a cell in the FBF, we assumed 254 return values; hence, a cell in the rCBF has a two-bit counter and eight-bit value field, and a cell in the FBF has eight bits to store values. However, the rCBF can store up to 255 values for eight bits (i.e., $2^8 - 1$) because the rCBF does not need to reserve the maximum value $2^8 - 1$ as a conflict value.

Let n be the number of elements stored in a BF structure, α the size factor of the structure, m the number of cells in the structure (i.e., BF size $m = \alpha n$), and M the memory requirement of the structure. Figures 4 and 5 compare the $P(U)$ and $P(F_S)$ between the theoretical and experimental results according to the BF size, respectively. When each α of the FBF is 2, 4, and 8, each α of the rCBF₁ is 1.6, 3.2, and 6.4, respectively, because the rCBF₁ uses the same M as the FBF. The rCBF₂ uses the same α as the FBF. Although the α of the rCBF₁ is smaller than that of the FBF, the $P(U)$ of the rCBF₁ is much smaller than that of the FBF. The $P(F_S)$ of the rCBF₁ is slightly greater than that of the FBF; however, if insertions and deletions are repeated, the $P(F_S)$ of the rCBF would be better than that of the FBF. Hence, with dynamic data, replacing the FBF with the rCBF can improve the performance of the overall structure (i.e., L-rCBF), even though the FBF is better than the rCBF in terms of the $P(F_S)$ when using the same M . In addition, the experimental results validated the theoretical analysis, as shown in Figures 4 and 5.

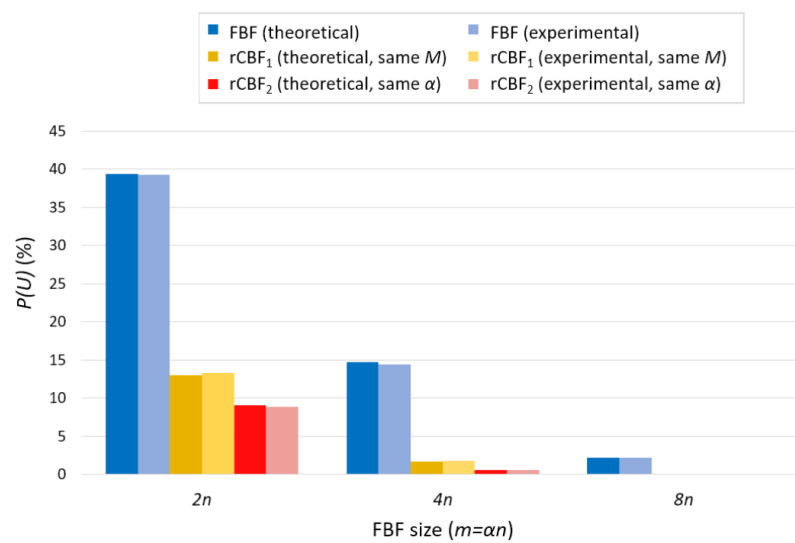


Figure 4. Comparison of undeletable probabilities between theoretical and experimental results according to BF size.

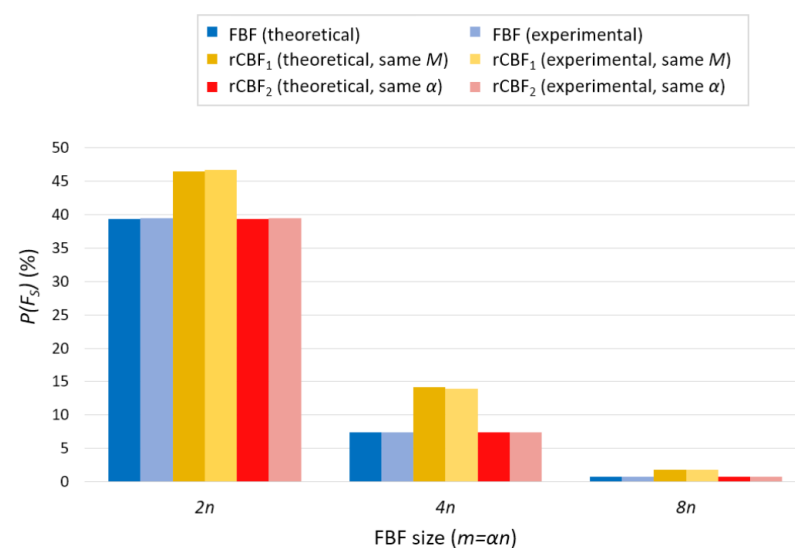


Figure 5. Comparison of search failure probabilities between theoretical and experimental results according to BF size.

6. Conclusions

In this paper, we propose an L-rCBF for dynamic data and design a deletion algorithm for the L-rCBF that can be applied to deletions in an L-FBF. The proposed L-rCBF can be utilized in various applications involving repetitive update operations (i.e., insertions and deletions), such as PIT lookup in NDN and deduplication in distributed systems. As an expandable and deletable key–value structure, the L-rCBF allows efficient operations and can achieve the purpose of each application without other data structures.

Using the same amount of memory, the proposed L-rCBF is better than a single rCBF in terms of *undeletables* and *search failures* because a model in the L-rCBF consumes a relatively small amount of memory, regardless of the amount of data. Hence, as the amount of positive data to be stored increases, the L-rCBF becomes more efficient than a single rCBF.

Using the same amount of memory, an L-rCBF is better than an L-FBF with the same model in terms of *undeletables* because an rCBF can utilize cells with hash collisions for deletion, unlike an FBF. Moreover, as the number of return values increases, the L-rCBF becomes more efficient than the L-FBF. Therefore, for dynamic data processing with frequent insertions and deletions, the proposed L-rCBF is more suitable than the L-FBF, and deletions from the L-rCBF and L-FBF can be implemented using the proposed deletion algorithm. In addition, the theoretical analyses and experiments demonstrated the superiority of structures using an rCBF.

Furthermore, the performance of a learning-based structure can vary depending on the accuracy of the model used. In other words, the higher the accuracy of a model, the better the deletion and search performance of the learning-based structure. Despite the higher memory requirement of a model with high accuracy compared with a model with lower accuracy, the accurate model reduces the sizes of the auxiliary structures (i.e., the FR-BF and verification structure) by reducing the number of elements stored in the structures, thereby enhancing the overall performance. However, owing to the tradeoff between memory requirements and accuracy, the size of a learned model cannot be increased in an unlimited manner to improve accuracy when designing a learning-based structure. In addition, if a complex model is designed to increase accuracy, the training time of the model increases as well as the memory requirement. Therefore, if reconstructions occur, the processing speed of the L-rCBF using the complex model is degraded compared to the L-rCBF using a simpler and smaller model.

Author Contributions: Conceptualization, H.B.; methodology, H.B.; software, Y.L.; validation, Y.L. and H.B.; formal analysis, H.B.; investigation, Y.L. and H.B.; resources, H.B.; data curation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, H.B.; visualization, Y.L.; supervision, H.B.; project administration, H.B.; funding acquisition, H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2021R1F1A1051646).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BF	Bloom filter
FBF	Functional Bloom filter
rCBF	Counting Bloom filter with return values

LBF	Learned Bloom filter
L-FBF	Learned functional Bloom filter
L-rCBF	Learned counting Bloom filter with return values
FR-BF	False-class result Bloom filter
V-FBF	Verification functional Bloom filter
V-rCBF	Verification counting Bloom filter with return values
NDN	Named data networking
FIB	Forwarding information base
PIT	Pending interest table
SDN	Software defined networking
TP	True positive
FP	False positive
FR	False-class result
FN	False negative
INDET	Indeterminable
UNDEL	Undeletable
PCA	Principal component analysis
LSTM	Long short-term memory
CNN	Convolutional neural network

References

1. Bloom, B. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426. [\[CrossRef\]](#)
2. Dharmapurikar, S.; Krishnamurthy, P.; Taylor, D.E. Longest prefix matching using bloom filters. *IEEE/ACM Trans. Netw.* **2006**, *14*, 397–409. [\[CrossRef\]](#)
3. Mun, J.H.; Lim, H. New approach for efficient ip address lookup using a bloom filter in trie-based algorithms. *IEEE Trans. Comput.* **2016**, *65*, 1558–1565. [\[CrossRef\]](#)
4. Dai, H.; Lu, J.; Wang, Y.; Pan, T.; Liu, B. BFAST: High-Speed and Memory-Efficient Approach for NDN Forwarding Engine. *IEEE/ACM Trans. Netw.* **2017**, *25*, 1235–1248. [\[CrossRef\]](#)
5. Jang, S.; Byun, H.; Lim, H. Dynamically Allocated Bloom Filter-Based PIT Architectures. *IEEE Access* **2022**, *10*, 28165–28179. [\[CrossRef\]](#)
6. Wu, Q.; Wang, Q.; Zhang, M.; Zheng, R.; Zhu, J.; Hu, J. Learned bloom-filter for the efficient name lookup in information-centric networking. *J. Netw. Comput. Appl.* **2021**, *186*, 103077. [\[CrossRef\]](#)
7. Reviriego, P.; Martínez, J.; Larrabeiti, D.; Pontarelli, S. Cuckoo filters and bloom filters: Comparison and application to packet classification. *IEEE Trans. Netw. Service Manag.* **2020**, *17*, 2690–2701. [\[CrossRef\]](#)
8. Yang, M.; Gao, D.; Foh, C.H.; Qin, Y.; Leung, V.C.M. A Learned Bloom Filter-Assisted Scheme for Packet Classification in Software-Defined Networking. *IEEE Trans. Netw. Service Manag.* **2022**, *19*, 5064–5077. [\[CrossRef\]](#)
9. Eppstein, D.; Goodrich, M.T.; Uyeda, F.; Varghese, G. What's the difference? efficient set reconciliation without prior context. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 218–229. [\[CrossRef\]](#)
10. Xia, W.; Feng, D.; Jiang, H.; Zhang, Y.; Chang, V.; Zou, X. Accelerating content-defined-chunking based data deduplication by exploiting parallelism. *Future Gener. Comput. Syst.* **2019**, *98*, 406–418. [\[CrossRef\]](#)
11. Cheng, G.; Guo, D.; Luo, L.; Xia, J.; Gu, S. LOFS: A Lightweight Online File Storage Strategy for Effective Data Deduplication at Network Edge. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2263–2276. [\[CrossRef\]](#)
12. Patgiri, R.; Biswas, A.; Nayak, S. deepBF: Malicious URL detection using learned Bloom Filter and evolutionary deep learning. *Comput. Commun.* **2023**, *200*, 30–41. [\[CrossRef\]](#)
13. Reviriego, P.; Hernández, J.A.; Dai, A.; Shrivastava, A. Learned bloom filters in adversarial environments: A malicious URL detection use-case. In Proceedings of the 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR), Paris, France, 7–10 June 2021; pp. 1–6.
14. Xiong, S.; Yao, Y.; Li, S.; Cao, Q.; He, T.; Qi, H.; Tolbert, L.; Liu, Y. kBF: Towards Approximate and Bloom Filter based Key-Value Storage for Cloud Computing Systems. *IEEE Trans. Cloud Comput.* **2017**, *5*, 85–98. [\[CrossRef\]](#)
15. Vijayakumar, P.; Chang, V.; Deborah, L.J.; Kshatriya, B.S.R. Key management and key distribution for secure group communication in mobile and cloud network. *Future Gener. Comput. Syst.* **2018**, *84*, 123–125. [\[CrossRef\]](#)
16. Fan, L.; Cao, P.; Almeida, J.; Broder, A. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.* **2000**, *8*, 281–293. [\[CrossRef\]](#)
17. Nayak, S.; Patgiri, R. countBF: A general-purpose high accuracy and space efficient counting bloom filter. In Proceedings of the 2021 17th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 25–29 October 2021; pp. 355–359.
18. Deng, F.; Rafiei, D. Approximately detecting duplicates for streaming data using stable bloom filters. In Proceedings of the 2006 International Conference on Management of Data (SIGMOD), Chicago, IL, USA, 27–29 June 2006; pp. 25–36.

19. Bonomi, F.; Mitzenmacher, M.; Panigrah, R.; Singh, S.; Varghese, G. Beyond bloom filters: From approximate membership checks to approximate state machines. In Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), Pisa, Italy, 11–15 September 2006; pp. 315–326.
20. Byun, H.; Lim, H. Learned FBF: Learning-Based Functional Bloom Filter for Key–Value Storage. *IEEE Trans. Comput.* **2022**, *71*, 1928–1938. [[CrossRef](#)]
21. Kraska, T.; Beutel, A.; Chi, E.; Dean, J.; Polyzotis, N. The Case for Learned Index Structures. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD), Houston, TX, USA, 10–15 June 2018; pp. 489–504.
22. Mitzenmacher, M. A Model for Learned Bloom Filters, and Optimizing by Sandwiching. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS), Montréal, QC, Canada, 3–8 December 2018; pp. 462–471.
23. Liu, Q.; Zheng, L.; Shen, Y.; Chen, L. Stable learned bloom filters for data streams. *Proc. VLDB Endow.* **2020**, *13*, 2355–2367. [[CrossRef](#)]
24. Tarkoma, S.; Rothenberg, C.E.; Lagerspetz, E. Theory and Practice of Bloom Filters for Distributed Systems. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 131–155. [[CrossRef](#)]
25. Broder, A.; Mitzenmacher, M. Network applications of Bloom filters: A survey. *Internet Math.* **2003**, *1*, 485–509. [[CrossRef](#)]
26. Lee, Y.; Byun, H. Learned Counting Bloom Filter with Return Values for Deletion of Dynamic Data. In Proceedings of the IEIE Summer Conference, Jeju, Republic of Korea, 29 June–1 July 2022. (In Korean)
27. Lim, H.; Lee, J.; Byun, H.; Yim, C. Ternary Bloom Filter Replacing Counting Bloom Filter. *IEEE Commun. Lett.* **2017**, *21*, 278–281. [[CrossRef](#)]
28. Web Directory. Available online: curlie.org/ (accessed on 15 December 2020).
29. Free Online Dataset of Blacklisted URLs. Available online: www.shallalist.de/ (accessed on 15 December 2020).
30. Alexa the Web Information Company. Available online: www.alexa.com/ (accessed on 17 May 2020).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.