Received 30 March 2024, accepted 27 April 2024, date of publication 8 May 2024, date of current version 20 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3399062

RESEARCH ARTICLE

FeCBF: A Novel Sub-Optimal Cascaded Bloom Filter Structure Based on Feature Extraction

QUANG-MANH DUONG[1], KHOA-SANG NGUYEN[1], HAI-DUONG NGUYEN[1], XUAN-UOC DAO[1], ANH-TUAN DO[2], (Senior Member, IEEE), AND QUANG-KIEN TRINH[1], (Member, IEEE)
[1]Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi 11917, Vietnam
[2]Institute of Microelectronics (IME), Agency for Science, Technology and Research (A*STAR), Singapore 138634

Corresponding author: Quang-Kien Trinh (kien.trinh@lqdtu.edu.vn)

ABSTRACT This work presents the Feature extraction Cascaded Bloom Filter (FeCBF), a novel probabilistic data structure formed by cascading multiple Bloom filters in an optimum sequence. The FeCBF's distinctive design of alternating positive and negative filter layers effectively suppresses False Positive/Negative Rates (FPR/FNR), enabling exact filtering with reasonable resource cost. Compared to other state-of-the-art designs on the same experimental dataset, FeCBF demonstrates significant memory space savings of 45% to 76% while maintaining the best FPR in its class. The proposed model also includes a closed-form expression for determining the sub-optimal FeCBF configuration based on desired filter performance metrics, offering the potential for automatic design flow. The FeCBF architecture, designed for hardware implementation, holds promise for many applications. It can be readily deployed as an accelerator in various computing problems, including massive content filtering, network traffic filtering, and online malware/virus detection.

INDEX TERMS Bloom filter, feature extraction, pattern matching, big-data filter, probabilistic filter, hardware acceleration.

I. INTRODUCTION

Bloom filter (BF), a well-known hash data structure for approximate filtering with modest memory space, was first proposed by Bloom [1]. Initially, it was described as a compact probabilistic data structure representing words in a dictionary. However, due to the lack of technological availability, there was little interest in using BFs for networking until 1995. After this point, the use of BFs for networking gained widespread interest in academia and industry, marking a significant turning point in the evolution of this field.

In recent years, BFs and their variants have been widely used in applications related to networking (routing [2], Named Data Networking [3], network security [4]), database (duplicate detection [5], content synchronization [6]), and biometric identification [7]. Today, BFs appear in popular browsers like Microsoft Bing and Google Chrome. Microsoft Bing search engine uses multilevel hierarchical BFs for the BitFunnel search index [8], which provides a lower cost than the previous Bing search index. Google Chrome preliminarily identifies malicious URLs using the local BF [9]. Most recently, in the context of the widespread Coronavirus on a global scale, a BF solution was used to detect and warn users about close contact with suspected infected individuals [10], achieving almost zero FPR ($\sim 10^{-15}$) with a small memory cost.

In the previous study [11], we used a co-design platform: a software-based BF using Python and a hardware-based BF using FPGAOur co-design platform works based on extracting features from input data, thereby designing the subsequence filter layers to achieve a lower error rate than the conventional standard BF. However, the BF solutions published in [11] are limited in a few case studies without explicit design methodology and exhibit only moderate error rate improvement. This work significantly extends our study in [11], aiming for a more general and effective filter design methodology to fit a wide range of real-world data filtering problems. The main contributions of this study are summarized as follows.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Quan.

- A novel architecture of a high-precision Bloom-based filter constructed by cascading data-dependent subfilters that is capable of aggressively suppressing the misclassification with significant resource-saving;
- A complete design methodology based on a closed-form analytical model to predict and locate the sub-optimal solutions for the proposed BF design under specific boundary constraints.

The rest of this work is as follows. Section II introduces the background of BF and related works. Section III presents a novel concept of the FeCBF structure and its design flow based on a sub-optimal analytical design model that gives insights into the relationship and trade-off between the major FeCBF design and performance metrics. Section IV practically verifies the FeCBF analytical design model. Section V discusses the efficiency and performance of the sub-optimal FeCBF compared to other prior-art Bloom-based filters. Section VI concludes this study.

II. PRELIMINARY

A. BACKGROUND OF BLOOM FILTER

A BF composes a member set $\{a_1, a_2, \ldots, a_n\}$, a hash function set $\{h_1, h_2, \ldots, h_k\}$, and an array of m-bit elements initialized with all bits '0'. In the BF construction phase, for adding a member a_i to BF, the set of k hash functions is used to calculate k addresses $\left\{h_j\left(a_i\right), j = \overline{1, k}\right\}$ and write bits '1' into corresponding memory cells in the m-bit array. In working mode (querying), an element q_i belongs to the constructed filter only if all memory cells located at hashed addresses correspond to q_i store bits '1'. Examples of adding and querying operations are illustrated in Fig. 1. Assuming the BF size $m = 32$ bits, the number of member elements $n = 3$, and the number of hash functions $k = 3$. The elements a_1, a_2, and a_3 after the adding are represented by the triples of bits '1' at addresses (0, 6, 13), (8, 16, 22), and (16, 25, 31), respectively. In the querying, BF uses three hash functions to determine three corresponding addresses for the query elements q_1, q_2, and q_3 in the m-bit array. Element q_1 is a case of "True Positive," in which all the cells of BF that correspond to this element are bits '1'. Element q_2 corresponds to "True Negative" when at least one cell contains a bit '0'. Element q_3 is "False Positive," where all the cells corresponding to this element are bits '1' but intrinsically q_3 does not match any element in the member set. A standard BF (SBF) of finite size may have a non-zero False Positive Rate (FPR), but does not allow a False Negative Rate (FNR $= 0$), i.e., a query returns either "element is possibly in BF" or "element is truly not in BF" results.

The theoretical FPR (FPR_theo) is calculated by the following equation when the filter size m is large enough.

$$\text{FPR}_\text{theo} \approx \left(1 - e^{-kn/m}\right)^k \tag{1}$$

Given n and m, the optimal number of hash functions k_opt to achieve the minimum value of FPR_theo is determined by

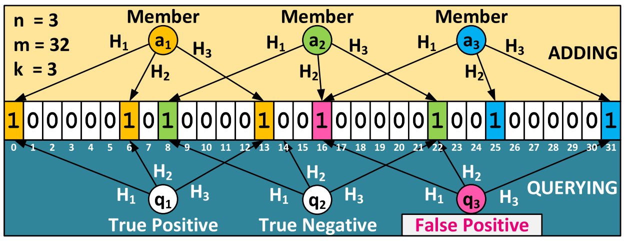

FIGURE 1. An example of the standard BF with two basic operations: adding and querying.

the following equation.

$$k_\text{opt} = \frac{m}{n}\ln 2 \approx 0.6931\frac{m}{n} \tag{2}$$

where the minimum FPR_theo corresponding to the optimal value k_opt in (2) is calculated by the following equation.

$$\text{FPR}_\text{theo} = \left(\frac{1}{2}\right)^{k_\text{opt}} \approx 0.6185^{m/n} \tag{3}$$

The number of bits m of the SBF that responds to a given element number of member set n and a specific target of error rate FPR_target is calculated by the following equation.

$$m = -\frac{n \cdot \ln(\text{FPR}_\text{target})}{(\ln 2)^2} \tag{4}$$

A detailed explanation of (1)–(4) can be found in [3]. In an example illustrating (4), to ensure that the FPR_theo is not more than 1%, using the optimal number of hash functions k_opt, the average number of bits per BF member has to be approximately 9.6. Depending on the problems and priority criteria, the fundamental design parameters (n, m, k, FPR_target) must be appropriately adjusted and accordingly compromised.

B. RELATED WORKS

In the following, we summarize some recent studies on BF design, categorized by optimization method, including hash optimization, applying Machine Learning (ML) techniques, and filter structure modification.

Several interesting hash optimization methods have been proposed in [12], [13], and [14]. Lumetta and Mitzenmacher [12] proposed employing two or more independent groups of hash functions (HF) for adding and querying operations, thus reducing the FPR by 43.5% compared to SBF. Hao et al. [13] proposed a partitioned hashing approach by dividing the member set into disjoint subsets and using different appropriate combinations of HFs for each of them. Unlike the approach in [12], where each query has to use all groups of HFs, in [13], each query incurs one additional hash operation compared to the SBF. By carefully partitioning the member elements and selecting combinations of HFs, the fill factor of the '1' bits in [13] is reduced, thus reducing the FPR by 54.6% compared to that of SBF. However, solutions in [12] and [13] have high computational complexity and lack flexibility in responding to changes in the BF member set. Recently, Xie et al. [14] proposed a BF approach that supports the customization of HFs for positive elements in the two-round membership query with customized HFs stored in an additional hash table. Compared to the solutions proposed in [12] and [13], the solution in [14] achieves a significant improvement in FPR so this model will be selected as the representative of the hash optimization method for further comparison in our work.

Based on the optimization method using ML techniques, Kraska et al. [15] have shown that combining ML techniques can improve index structures and related structures such as BFs. The first proposed ML technique is Learned BF (LBF), which comprises a classifier block of Learning functions serving as the pre-filter and a backup BF playing the role of the post-filter. If the classification score of a query element determined by the pre-filter is higher than a threshold level, the element is treated as a member; otherwise, the element is queried by the post-filter, thereby reducing the number of hash operations. The LBF model proposed by Kraska et al. [15] serves as the foundation design for the subsequent proposed variants of the LBF. Mitzenmacher et al. [16] are the first to propose a reasonably accurate mathematical model for evaluating the effectiveness of the LBF. Furthermore, the authors proposed an enhanced model for LBF, namely "Sandwiched LBF," in which an initial BF is added before the classifier BF and the backup filter. Dai and Shrivastava [17] later proposed the "Adaptive LBF," in which the classifier BF divides the input data into more than two groups, which permits the HFs for each group to be adapted and optimized to specific contexts. In [14], Xie et al. independently evaluated the Sandwiched LBF and Adaptive LBF models. They showed surprising results where the Adaptive LBF model exhibited the worst classification accuracy compared to all other designs, i.e., LBF, Sandwiched LBF, and even SBF. Also, from that study, the Sandwiched structure [16] was proven the best and is selected as a representative LBF for comparison in this work.

Chazelle et al. [18] proposed modifying the SBF structure, which is called the Bloomier filter. Dietzfelbinger and Pagh [19] described a modified Bloomier filter that could respond to approximate membership queries. Graf et al. [20] further proposed an implementation solution of the approach in [19]. They named it the Xor filter and announced that it could be faster than SBF and have more minor memory utilization. Thus, the Xor filter [20] represents a BF variation used for comparison.

All the Bloom-based filter structures classified according to the above design ideas have certain limitations. For example, hash optimization in [12], [13], and [14] or applied ML techniques in [15] and [16] typically leads to higher computational complexity, while the proposed modified structure BF in [19] and [20] only considers the single-layer design. Furthermore, none of these prior works pay attention to the characteristics of the input data to improve Bloom filter performance. The major shortcomings of the f-HABF [14], SLBF [16], and Xor filter [20] motivate the FeCBF filter structure proposed in this study.

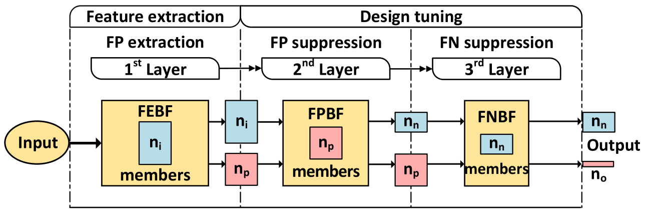

FIGURE 2. The generic structure of FeCBF includes a first layer for Feature extraction (FEBF), a second layer for False Positive suppression (FPBF), and a third layer for False Negative suppression (FNBF).

III. FeCBF GENERIC DESIGN

Our proposed FeCBF is inspired by the "Feature extraction" idea introduced by [11] and further improved in this study. The term "Feature extraction" could be understood as the design of subsequent filter layers considering the characteristics of the previous filter layers. The characteristic extraction requires intensive computing power, but as we will show, it is feasible and can be done at a reasonable time cost using modern computing devices. In particular, based on the design methodology and error prediction model presented in this study, FeCBF could be tuned to achieve the target error rate with sub-optimum cost in resources.

The design of the proposed FeCBF structure includes an Extraction phase and a Tuning phase, as depicted in Fig. 2. The FeCBF consists of three cascaded filter layers: FEBF (Feature extraction BF), FPBF (False Positive BF) and FNBF (False Negative BF), each filter layer is designed reversely to the previous layer. For example, suppose the first FEBF filter layer causes False Positive elements, so the next filter layer should ideally suppress these unexpected elements but, on the other hand, generate False Negative elements. To proceed in such a way, the following filter layer collects (maximumly possible) the erroneously filtered elements of the previous filter layer as its own member set. Thus, the number of input elements and False Positive/False Negative ratio will decrease exponentially as the number of layers increases, and the filter will also rapidly shrink in size at the subsequent layers. To eliminate FeCBF's filtering errors, multiple pairs of small-size cascaded subfilters for False Positive/False Negative elimination might be theoretically required, or a pair of such subfilters with correspondingly large enough sizes might be employed. However, cascading multiple subfilters increases the complexity of the filtering structure and can cause extra lookup latency. Therefore, in the proposed FeCBF structure, we apply cascading of 3 subfilters, i.e., a Feature-extraction subfilter and a pair of subfilters for False Positive and False Negative suppression, with their size ratios optimized based on a mathematical analytical model presented in Section IV-A. The structure of FeCBF, consisting of three functionally specialized cascading subfilters, is sufficient for various filtering scenarios and requirements, as will be experimentally demonstrated in Section IV-D.

In the descriptions below, we use the following notations: I_{BF}, N_{BF} are the input set and the member set of a filter;

$\mathrm{TP_{BF}}$, $\mathrm{FP_{BF}}$, $\mathrm{FN_{BF}}$ are the sets of True Positive, False Positive (if any), and False Negative (if any) elements of the filter, respectively; function $s\,(\mathrm{T})$ returns the number of elements of the set T. In the specific case of FeCBF, assuming that FeCBF has a member set $\mathrm{N_{FeCBF}}$ with $s\,(\mathrm{N_{FeCBF}}) = n_i$ and the input set $\mathrm{I_{FeCBF}}$ that does not include $\mathrm{N_{FeCBF}}$ with $s\,(\mathrm{I_{FeCBF}}) = \chi \cdot n_i$, typically $\chi \gg 1$. The query elements are L bit vectors. For example, if the member set has $n_i = 1$ million elements and the input covers the space of data strings $L = 32$ bits, then $\chi = (2^{32} - 10^6)/10^6 \approx 4294$.

A. EXTRACTION PHASE

As shown in Fig. 2, the Extraction phase uses an SBF with a member set $\mathrm{N_{FEBF}} = \mathrm{N_{FeCBF}}$ and performs query operations of the input space $\mathrm{I_{FEBF}} = \mathrm{I_{FeCBF}}$ to capture the majority of FEBF's False Positive elements in the $\mathrm{FP_{FEBF}}$ set. This set reflects the unique relationship with the member set $\mathrm{N_{FEBF}}$, the input space $\mathrm{I_{FEBF}}$, the filter size m, and the hash type Htype of FEBF. In other words, $\mathrm{FP_{FEBF}}$ could be considered as a "Feature set" of FEBF with the $\mathrm{N_{FEBF}}$ member set in response to the $\mathrm{I_{FeCBF}}$ input set. $\mathrm{FP_{FEBF}}$ hence bears the characteristic of the first filter layer results; any changes to the input and structure of FEBF will immediately reflect on the $\mathrm{FP_{FEBF}}$. Because FEBF is the first filter layer, it can be chosen solely based on available allocated resources (i.e., memory). The theoretical error rates of this filter layer are calculated as follows.

$$\begin{cases} \mathrm{FPR_{1layer}} = \mathrm{FPR_{FEBF}} = \dfrac{s(\mathrm{FP_{FEBF}})}{s(\mathrm{I_{FeCBF}})} = \dfrac{n_p}{\chi \cdot n_i} \\ \mathrm{FNR_{1layer}} = \mathrm{FNR_{FEBF}} = 0 \end{cases} \tag{5}$$

where $n_p = s(\mathrm{FP_{FEBF}})$ is the number of False Positive elements at the first FeCBF's filter layer; $s(\mathrm{I_{FeCBF}})$ is the FeCBF's inputs which are not filter members and χ times larger than the number of filter members n_i, i.e., equal to $\chi \cdot n_i$; hence, χ is called the input-to-member scaling factor.

B. TUNING PHASE

The Tuning phase is performed after the Extraction phase. It will use the information extracted from the input space of FEBF to select appropriate design parameters for subsequent filters FPBF and FNBF to achieve maximum filtering efficiency. At this point, the design of FPBF and FNBF has to consider the results of the previous Extraction phase rather than a random structure for the best optimization. This phase is divided into the following two substeps.

1) FALSE POSITIVE SUPPRESSION STEP

This filtering step is performed by the FPBF, also known as the inverting filter, because it allows FEBF's False Positive elements to pass through but blocks the members of FEBF. The input to this filter includes the elements that pass through FEBF, i.e., the member set $\mathrm{N_{FEBF}}$, and the False Positive element set $\mathrm{FP_{FEBF}}$ of the previous filter layer. In this way, the queried elements that first pass through FEBF and again pass through FPBF have a high probability of being present in the $\mathrm{FP_{FEBF}}$ set. If $\mathrm{FP_{FEBF}}$ is fully defined, and FPBF is an ideal filter, then all False Positive elements of FEBF (so is FeCBF) would be theoretically identified and, hence, be blocked at this layer. On the other hand, the element that passes FEBF but could not pass FPBF, under that ideal assumption, belongs to $\mathrm{N_{FeCBF}}$.

However, the imperfection of FPBF leads to the phenomenon that a small proportion of True Positive elements come from the $\mathrm{N_{FEBF}}$ member set can pass through FPBF as the False Positive elements of this subfilter. These elements are classified as the False Negatives of FeCBF. Additionally, in cases where 100% of the elements in the $\mathrm{FP_{FEBF}}$ set cannot be identified, the possibility of FEBF's False Positive elements still exists, although this possibility is very low. The theoretical error rates at the output of the second filter layer are calculated as follows.

$$\begin{cases} \mathrm{FNR_{2layers}} = \dfrac{s(\mathrm{FP_{FPBF}})}{s(\mathrm{I_{FeCBF}})} = \dfrac{n_n}{\chi \cdot n_i} \\ \mathrm{FPR_{2layers}} \cong \mathrm{FNR_{FPBF}} = 0 \end{cases} \tag{6}$$

where $n_n = s(\mathrm{FP_{FPBF}})$ is the number of FPBF's False Positive elements.

Equation (6) shows that eliminating FeCBF's False Positive elements by the two filter layers, FEBF and FPBF, is feasible at the cost of generating FeCBF's False Negative elements. However, considering that the input set of FPBF has been largely suppressed after the first filter layer, it can be predicted that FPBF will not consume as much memory as FEBF.

2) FALSE NEGATIVE SUPPRESSION STEP

After the second filter layer, False Negative elements are the source of 2-layer FeCBF's filtering error. Although they could be relatively small, they can become undesirable for some specific applications. Therefore, we propose to use a third filter layer called FNBF to remove those.

The input to the third filter layer FNBF consists of the member set $\mathrm{N_{FPBF}}$ of the second filter layer FPBF, where $s\,(\mathrm{N_{FPBF}}) = s\,(\mathrm{FP_{FEBF}}) = n_p$ and the False Positive set $\mathrm{FP_{FPBF}}$ of this second layer FPBF, where $s\,(\mathrm{FP_{FPBF}}) = n_n$. If we use n_n elements of $\mathrm{FP_{FPBF}}$ as the member set of the third layer FNBF, we could, similarly, almost remove those False Negative elements of FeCBF. As a result, the alternate use of False Positive (FPBF) and False Negative (FNBF) subfilters in the FeCBF structure can substantially remove unwanted elements at the final output. However, a similar effect occurs due to the imperfection of FNBF that leads to false identification of some elements belonging to $\mathrm{FP_{FEBF}}$ and causing False Positive elements of FNBF itself. The false identification rate at this step is theoretically very small because the number of FNBF's input and member elements has been significantly reduced compared to the FeCBF's input elements. The error rates at the output of the third filter layer are expressed as

TABLE 1. Summary of FeCBF query result.

Actual presence of q_x in the member set	Subfilter's query results			FeCBF query result	Type of query result
	QR_{FEBF}	QR_{FPBF}	QR_{FNBF}	QR_{FeCBF}	
0	0	x	x	0	True Negative
	1	1	0	0	True Negative
	1	1	1	1	False Positive
1	1	0	x	1	True Positive
	1	1	1	1	True Positive

follows.

$$\begin{cases} FPR_{3layers} = \dfrac{s(FP_{FNBF})}{s(I_{FeCBF})} = \dfrac{n_o}{\chi \cdot n_i} \\ FNR_{3layers} \cong FPR_{FPBF} = 0 \end{cases} \tag{7}$$

where $n_o = s(FP_{FNBF})$ is the number of FNBF's False Positive elements and the FeCBF's output False Positive elements.

In the 3-layer FeCBF structure, it is worth noting that the input and member sets of the last filter layer FNBF are expected to be very small; hence, this filter layer will consume an insignificant amount of resources and could be aggressively optimized for suppressing the final error rate.

C. WORKING MODE

To ensure normal operations, each FeCBF's subfilter is designed with four basic ports: a Q_I input port that receives any query element q_x; an EN (ENABLE) port that allows filters to operate; a QR (Query Result) port to return the query result of the q_x (QR = '0'/'1' indicates q_x not belong/belong to filter member set), and a Q_O output port is designed to forward the query element q_x to the next filter layer in case QR = '1'. The QR signals from the filter layers, specifically QR_{FEBF}, QR_{FPBF}, and QR_{FNBF}, are used as input data for the decision circuit to synthesize the final query result of q_x through the match signal (i.e., QR_{FeCBF}). QR_{FeCBF} equals '0'/'1', which indicates that q_x is not present/present in the N_{FeCBF} member set.

Table 1 lists the results of all possible query cases for the input element q_x by the 3-layer FeCBF filter structure. Column 1 is the ground truth information about the presence of q_x in the N_{FeCBF} member set, and columns 2-5 list the query results of the subfilters and FeCBF, respectively. Column 6 describes whether the query results are True or False about the presence (Positive) or absence (Negative) of q_x based on complete information. FeCBF does not know in advance about the existence of q_x in the member set and can only conclude as accurately as possible based on query information from all subfilters (columns 2-4). For example, FeCBF can cause a False Positive query result (see column 6 - row 3), i.e., q_x is not in the N_{FeCBF} member set, but the match signal QR_{FeCBF} = '1', where the corresponding FPR is calculated according to (7).

The application circuit of FeCBF is shown in Fig. 3, which includes the query circuit, decision circuit, and application

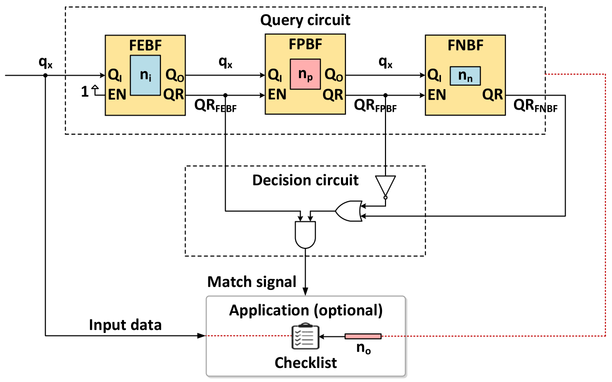

FIGURE 3. Practical application model of the proposed FeCBF structure.

(optional). At the input, a query element q_x is fed to the query circuit and, in another path, fed directly to the application, which will wait for the match signal to make the appropriate decision about FeCBF membership. The decision circuit determines its output match signal, i.e., QR_{FeCBF}, from the input signals QR_{FEBF}, QR_{FPBF}, and QR_{FNBF} shown in Table 1, and this circuit can be synthesized based on the minimization of the QR_{FeCBF} output logic function (e.g., using the Karnaugh map). The query circuit provides the application with an output set n_0 of non-suppressed FeCBF's False Positive elements to use as a checklist for error correction of the decision circuit if the match signal of this circuit is not True. The software handles this optionally if accurate filtering is required.

IV. SUB-OPTIMAL FeCBF DESIGN ANALYTICAL MODEL

Filter design optimization is a process of reducing the FPR/FNR of FeCBF under the limited filter size or minimize the filter size with a specific FPR/FNR target. The FeCBF model analysis and evaluation will be implemented on the software. In addition, all time-consuming pre-processing tasks are performed offline. For example, using a typical PC configuration,[1] the Feature extraction process with the entire 32-bit input space, i.e., 4 billion elements, could be performed in approximately 3 hours.

A. FeCBF MATHEMATICAL MODEL

For the convenience of mathematical expressions, we assume the memory sizes of the FeCBF's subfilters to be m_{FEBF}, m_{FPBF}, and m_{FNBF}, respectively. The size constraint of FEBF, FPBF, and FNBF relative to their total size $\mathcal{M}$ is described as follows.

$$\begin{cases} m_{FEBF} = \alpha \cdot \mathcal{M} \\ m_{FPBF} = \beta \cdot \mathcal{M} \\ m_{FNBF} = (1 - \alpha - \beta) \cdot \mathcal{M} \end{cases} \tag{8}$$

where α and β are, respectively, the size ratios of FEBF and FPBF compared to the total size $\mathcal{M}$ of FeCBF, their values need to satisfy the condition: $0 \leq \alpha + \beta \leq 1$.

[1]Intel(R) Core(TM) i5-12500 3.0 GHz, RAM 16Gb DDR4 2667 MHz, SSD 512Gb NVMe; OS Windows 10 Pro v.22H2; IDE Pycharm Community 2023.2, Python 3.9.

The error rate of each subfilter FEBF, FPBF, and FNBF (according to (3)) is calculated as follows.

$$\begin{cases} \mathrm{FPR_{FEBF}} = \dfrac{n_p}{\chi \cdot n_i} \approx 0.6185^{\frac{m_{\mathrm{FEBF}}}{n_i}} \\ \mathrm{FPR_{FPBF}} = \dfrac{n_n}{n_i} \approx 0.6185^{\frac{m_{\mathrm{FPBF}}}{n_p}} \\ \mathrm{FPR_{FNBF}} = \dfrac{n_o}{n_p} \approx 0.6185^{\frac{m_{\mathrm{FNBF}}}{n_n}} \end{cases} \tag{9}$$

Thus, the number of False Positive elements n_p, n_n, and n_o recorded at the output of each FeCBF layer is as follows.

$$\begin{cases} n_p \approx \chi \cdot n_i \cdot 0.6185^{\frac{\alpha \cdot \mathcal{M}}{n_i}} \\ n_n \approx n_i \cdot 0.6185^{\left(\frac{\beta \cdot \mathcal{M}}{\chi \cdot n_i \cdot 0.6185^{\frac{\alpha \cdot \mathcal{M}}{n_i}}}\right)} \\ n_o \approx n_p \cdot 0.6185^{\frac{(1-\alpha-\beta)\cdot \mathcal{M}}{n_n}} \end{cases} \tag{10}$$

Let $C = 0.6185^{\mathrm{M_{norm}}}$, where $\mathcal{M}_{\mathrm{norm}} = M/n_i$ is the normalized size of FeCBF, meaning the average number of memory bits provided per element of the $\mathrm{N_{FeCBF}}$ member set. Equation (10) can be rewritten as follows.

$$\begin{cases} n_p \approx \chi \cdot n_i \cdot C^{\alpha} \\ n_n \approx n_i \cdot C^{\left(\frac{\beta}{\chi \cdot C^{\alpha}}\right)} \\ n_o \approx \chi \cdot n_i \cdot C^{\alpha} \cdot C^{\left[\frac{(1-\alpha-\beta)}{C^{\left(\frac{\beta}{\chi \cdot C^{\alpha}}\right)}}\right]} \end{cases} \tag{11}$$

Finally, we define the normalized error rate of (3-layer) FeCBF, which can be simplified as follows.

$$\begin{cases} \mathrm{FPR_{FeCBF}} = \dfrac{n_o}{\chi \cdot n_i} \approx C^{\left[\alpha + (1-\alpha-\beta)\cdot C^{-\frac{\beta}{\chi \cdot C^{\alpha}}}\right]} \\ \mathrm{FPR_{norm}} = \dfrac{\mathrm{FPR_{FeCBF}}}{\mathrm{FPR_{EqSBF}}} \approx C^{\left[\alpha + (1-\alpha-\beta)\cdot C^{-\frac{\beta}{\chi \cdot C^{\alpha}}} - 1\right]} \end{cases} \tag{12}$$

where $\mathrm{FPR_{EqSBF}} \approx 0.6185^{\mathrm{M_{norm}}} = C$ is the theoretical FPR(calculated according to (3)) of SBF with the same size $\mathcal{M}$ and input set $\mathrm{N_{FeCBF}}$. The $\mathrm{FPR_{norm}}$ represents the reduction (optimization) level in the FeCBF's error rate compared to that of SBF at iso-memory utilization. The smaller the achievable value of $\mathrm{FPR_{norm}}$, the better the quality of the FeCBF design.

B. FeCBF DESIGN FLOW

The formula to predict the error rate improvement of FeCBF proposed compared to SBF in (12) through the $\mathrm{FPR_{norm}}$ resemble itself a complex multilevel exponential function, hence calculating multilevel derivatives to find the optimum values is feasible in theory but not very practical. Therefore, we use a numerical approach to locate the sub-optimal design configuration for FeCBF under certain input constraints. Based on the FeCBF analytical model in Section IV-A, we developed Python scripts that calculate the sub-optimal

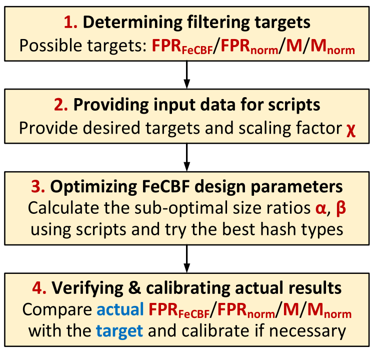
1. Determining filtering targets
Possible targets: FPR_FeCBF/FPR_norm/M/M_norm
2. Providing input data for scripts
Provide desired targets and scaling factor χ
3. Optimizing FeCBF design parameters
Calculate the sub-optimal size ratios α, β
using scripts and try the best hash types
4. Verifying & calibrating actual results
Compare actual FPR_FeCBF/FPR_norm/M/M_norm
with the target and calibrate if necessary

FIGURE 4. Reduced design flow for determining the sub-optimal parameters of the FeCBF structure to meet the target requirements.

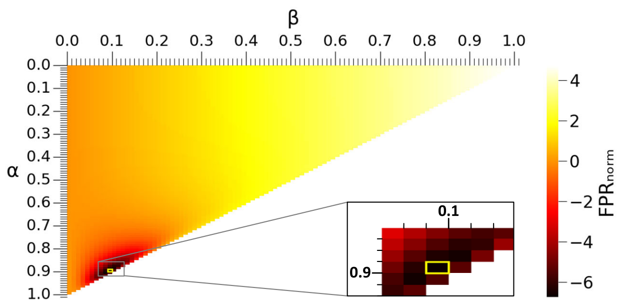

FIGURE 5. The dependence of $\mathrm{FPR_{norm}}$ on the size ratios α, β of FeCBF's subfilters in decimal logarithmic scale with the specific normalized size $\mathrm{M_{norm}} = 23$ and input-to-member scaling factor $\chi = 4294$.

required memory to achieve an expected target error rate (the algorithm can be found in Appendix A) or calculate the best possible error rate under the constraint of the memory (the algorithm can be found in Appendix B). The output data of the scripts are the optimal size ratios of the subfilters FEBF, FPBF, and FNBF, predictably calculated according to α and β parameters. The FeCBF design flow based on a specific member set $\mathrm{N_{FeCBF}}$ and input-to-member scaling factor χ to achieve the target parameters is depicted in Fig. 4.

The heatmap plotted in Fig. 5 illustrates a case study of optimizing the FeCBF's sub-filter design with the input parameters $\mathrm{M_{norm}} = 23$, and $\chi = 4294$. For each $\mathrm{M_{norm}}$, the FEBF, FPBF, and FNBF size ratios to their total size are α, β, and $(1 - \alpha - \beta)$. Since $(\alpha + \beta) \leq 1$, the points representing $\mathrm{FPR_{norm}}$ are populated only in the upper left half of the triangle. In this domain, we could estimate the values of $\mathrm{FPR_{norm}}$ (expressed in decimal logarithmic scale) corresponding to the annotations shown on the color bar of the heatmap. The sub-optimal point of $\mathrm{FPR_{norm}}$ is located in a highlighted area in the heatmap; note that its location can vary depending on the specific values of $\mathrm{M_{norm}}$ and χ. In this example, the region where $\mathrm{FPR_{norm}} < 1$ corresponds to the range $\beta < 0.20$, the region where $10^{-4} < \mathrm{FPR_{norm}} < 10^{-2}$ corresponds to the ranges $0.80 < \alpha < 0.95$ and $0.03 < \beta < 0.20$, the region where $\mathrm{FPR_{norm}} \sim 10^{-6}$ corresponds to the ranges $0.86 < \alpha < 0.92$ and $0.08 < \beta < 0.12$, and the sub-optimal value of $\mathrm{FPR_{norm}}$ in this particular example ($\sim 10^{-6.7}$), reached at the point with coordinates ($\alpha = 0.89$, $\beta =$

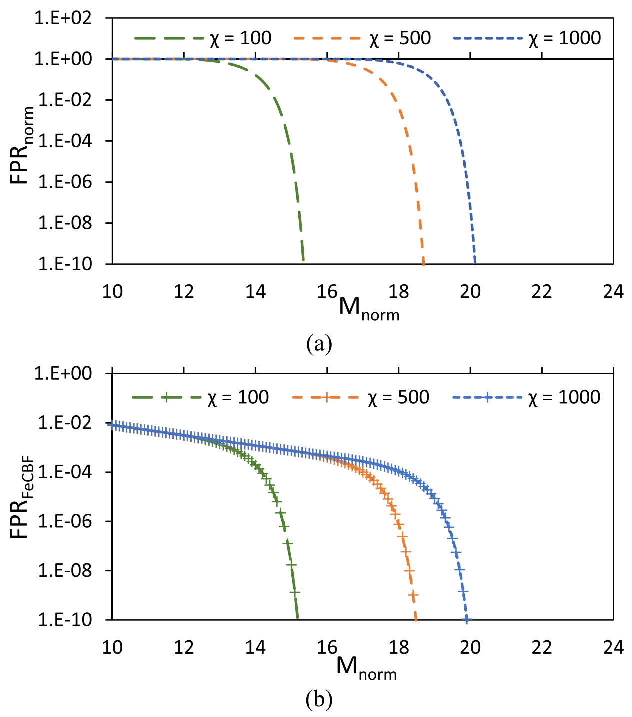

FIGURE 6. The dependence of FPR_{norm} on (a) the absolute error rate FPR_{FeCBF} (b) the normalized filter size M_{norm} at fixed input-to-member scaling factor $\chi = 100$, 500, and 1000, respectively.

0.09) with an accuracy of size ratios α, β is 10^{-2}, which could be adjusted in scripts. These values may change according to the boundary conditions of the design problem (member set, input set, etc.). However, the general trend is very consistent, where the size of the post-filter is typically significantly smaller than the size of the pre-filter. These additional filter layers, however, could bring tremendous improvement in the filtering accuracy, as discussed in the following Subsection.

C. IMPACTS OF FeCBF DESIGN PARAMETERS

1) DEPENDENCE OF ERROR RATE ON THE FILTER SIZE

Based on the design analytical model and the support of the developed Python scripts, we evaluated the dependence of the absolute and normalized FPRs according to the allocated memory space of FeCBF (Fig. 6). The experimental data were taken corresponding to typical values of the input-to-member scaling factor, varying from $\chi = 100$ up to $\chi = 1000$. The graph of FPR_{norm} in Fig. 6(a) shows that for each value of χ, there is a corresponding threshold level of M_{norm}, from which the error reduction effect of FeCBF compared to SBF is becoming significant.

Specifically, at small values of M_{norm}, FeCBF does not give better accuracy than SBF, i.e. $FPR_{norm} \approx 1$. Once M_{norm} starts to surpass a particular threshold value (from $M_{norm} \approx 12$, 16, and 17 corresponding to $\chi = 100, 500$, and 1000), but as soon as it exceeds that threshold, FPR_{norm} decreases rapidly, and the filtering accuracy of FeCBF is getting much superior to that of SBF. To acquire more generic data, we conducted a deeper analysis of how FPR_{norm} is dependent on M_{norm} under a broader range of χ from $10 \div 4294$, i.e., covering the entire 32-bit space. These data are skipped in Fig. 6(a) (for clarity), but the general trend of the FPR_{norm} dependency on M_{norm} is quite similar. Specifically, the threshold levels of M_{norm} increase proportionally from about 7 to 21, corresponding to the change of χ from 10 to 4294.

Fig. 6(b) presents the difference in the absolute error rate of FeCBF compared to the normalized, with a slight decrease in the absolute error rate when the normalized size gradually increases but does not exceed its threshold. In contrast, the normalized error rate (see Fig. 6(a)) remains almost unchanged under the same memory constraints, i.e., the error rate reduction of FeCBF is more rapid than SBF. Fig. 6(a) and Fig. 6(b) show that the error rate of FeCBF could be hundreds of times smaller ($\sim$1E-02) compared to SBF at small filter sizes, i.e., not yet exceeding the threshold.

Overall, without changing the input-to-member scaling factor χ, the error rates of FeCBF are significantly reduced compared to SBF as soon as its normalized size of FeCBF exceeds specific threshold values, which comes with only an insignificant cost of memory resources. For example, in Fig. 6(a), to reduce the FPR of FeCBF compared to SBF from 1%, i.e., $FPR_{norm} = $ 1E-02 to approximately zero, i.e., $FPR_{norm} =$ 1E-10, the additional normalized memories allocated for FeCBF corresponding to $\chi = 100$, 500, and 1000 are almost equal and approximately 0.8.

2) DEPENDENCE OF ERROR RATE ON THE NUMBER OF QUERY ELEMENTS

The graph in Fig. 7(a) represents the dependence of FPR_{FeCBF} and FPR_{norm} on the input-to-member scaling factor χ, varied from $100 \div 4294$. For this analysis, M_{norm} is set to be constants with fixed values 20, 21, and 22, considering those are the moderate ranges for the typical filter size (according to our numerical analysis). As can be seen from the plots, an upper bound of χ exists for each fixed value of the M_{norm}. Below this, the error rate of FeCBF remains much superior to that of SBF at iso-size, i.e., $FPR_{norm} \ll 1$. In addition, in an extreme case with a small input-to-member scaling factor (~ 10), which is not presented on the graph, the value of the M_{norm} needs to be no lower than a threshold level ($\cong$7) to ensure that FeCBF's error rate is better (lower) than SBF's.

A similar trend can be observed in Fig. 7(b), which represents how the absolute error rate FPR_{FeCBF} changes concerning χ. With a large input set, the error rate has an intrinsic upper bound, which is the FPR of the SBF. At small and moderate ranges of χ, we could observe the rapid decrease of FPR by reducing the input set size. Nonetheless, these upper-bound values of χ are typically quite large to meet practical requirements, i.e., thousands of times larger than the member set, and could be tuned at a reasonable cost of resources (by increasing the normalized size).

3) DEPENDENCE OF FILTER SIZE ON THE NUMBER OF QUERY ELEMENTS

Fig. 8(a) shows the dependence of the M_{norm} on the scaling factor χ when fixing the target FPR_{norm} to 10^{-1}, 10^{-5}, and

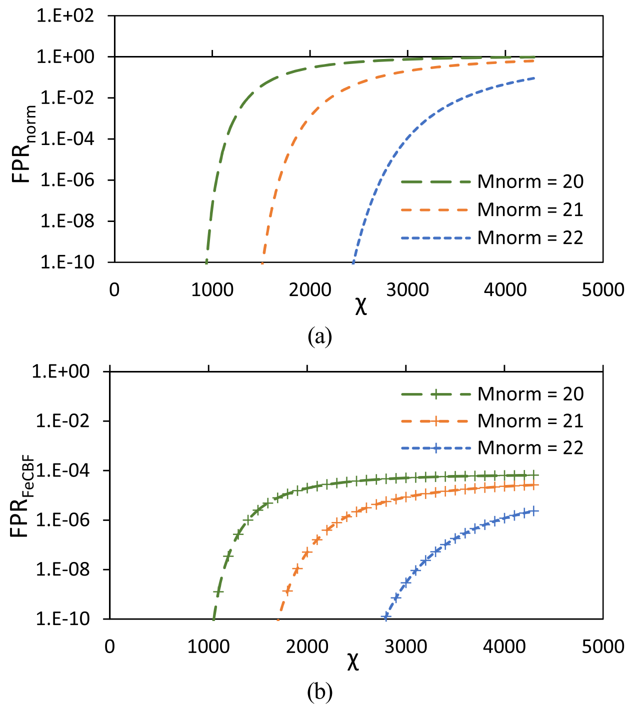

FIGURE 7. The dependence of FPR$_{norm}$ on (a) the absolute error rate FPR$_{FeCBF}$ (b) the input-to-member scaling factor χ at a fixed normalized filter size M$_{norm}$ = 20, 21, and 22, respectively.

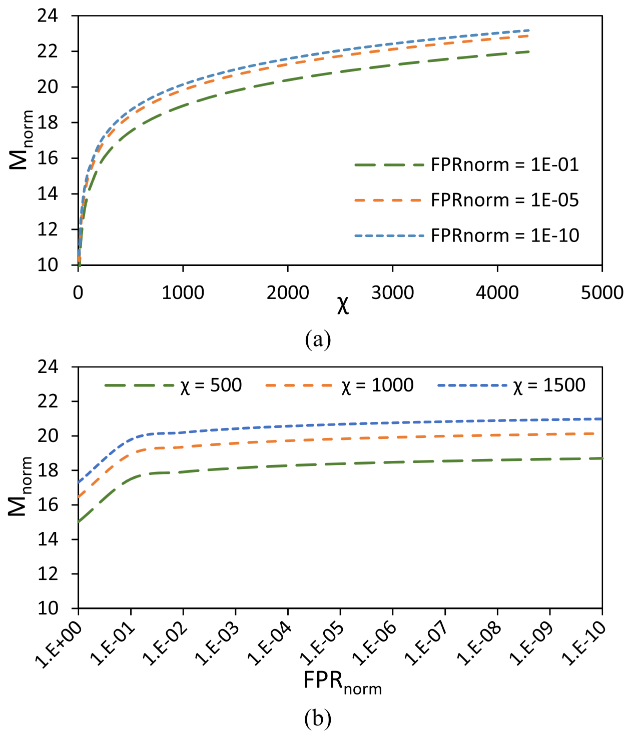

FIGURE 8. The dependence of M$_{norm}$ on (a) the input-to-member scaling factor χ at the fixed normalized error rate FPR$_{norm}$ = 1E-01, 1E-05, and 1E-10 (b) the desired target FPR$_{norm}$ at fixed χ = 500, 1000, and 1500.

10^{-10}. The predictive analytical model shows that M$_{norm}$ increases non-linearly with χ. Still, the difference is insignificant at various fixed values of the target FPR$_{norm}$. Again, this confirms that a reasonable increase of M$_{norm}$ could substantially improve the filtering accuracy. For example, FPR$_{norm}$ reduces up to 1,000 times (10^{-1} to 10^{-5}) when increasing M$_{norm}$ to about $0.887 \div 0.888$ (bit/element), and it is almost independent of χ as long as this factor is large enough (i.e., from ~ 500 and above).

Overall, FeCBF offers a significant FPR reduction compared to SBF at a small cost of resources. This could be achieved by adjusting the filter parameters based on our proposed model in (9)–(12).

4) DEPENDENCE OF FILTER SIZE ON THE DESIRED TARGET ERROR RATE

Fig. 8(b) shows the dependence of M$_{norm}$ on the desired target FPR$_{norm}$, varying from 1E+00 to 1E-10 when fixing χ = 500, 1000, and 1500. From the plot, the FeCBF's normalized sizes must reach minimum approximate values of 15, 16.5, and 17.3 bit/element, respectively, to ensure an absolute error rate lower than or equal to SBF, i.e., FPR$_{norm}$ $\leq$ 1E+00. To reach 10x better, i.e., FPR$_{norm}$ = 1E-01, the additional normalized bit/element allocated for FeCBF needs to be increased by approximately 2.5 and is weakly dependent on χ. From FPR$_{norm}$ = 1E-02 until FPR$_{norm}$ = 1E-10, the additional memory for FeCBF is almost independent of its query element number and incurs only insignificant memory resources. Specifically, to achieve levels of error reduction 1E-02, 1E-03, and 1E-04, the additional normalized memory for FeCBF are approximately 0.41, 0.22, and 0.15, respectively. Until the highest FPR$_{norm}$ = 1E-10, the additional memories for FeCBF are very small, approximately equal to 0.05, which could be considered negligible. This is partially because the error rate has already been reduced to a saturated level.

D. PRACTICAL VERIFICATION OF THE FeCBF MODEL

1) EXPERIMENTAL SETUP

The model presented in the preceding Section is a predictive model based on the closed-form expression for related filter parameters. In this Section, we conduct an actual filter design using some generic random datasets and filter configuration to verify the correctness of the proposed model and design. Specifically, we consider the data set of 32-bit length elements randomly distributed.[2] The Feature extraction of input datasets is the most time-consuming task but is a one-time operation, i.e., performed in offline mode. We practically show that the average time for the Feature extraction of entirely 32-bit input datasets, i.e., the input space of $2^{32} = 4,294,967,296$ elements might take 3 hours, as mentioned in Section III. The sizes of FeCBF's subfilter are pre-determined by the predictive model according to the sub-optimal configuration and have been evaluated with additional HFs selection and optimization (see the following Subsection).

[2]We intentionally choose query elements of 32 bit-length, considering it can fit, for example, the patterns of IPv4 in the network filtering application.

TABLE 2. The HF pairs and the corresponding Htype parameter.

HT [a]	HF 1	HF 2	HT	HF 1	HF 2	H	HF 1	HF 2
1	farm	spooky	5	farm	metro	9	spooky	xxhash
2	farm	mmh3.1	6	farm	FNV	10	spooky	metro
3	farm	mmh3.2	7	spooky	mmh3.1	11	CRC	farm
4	farm	xxhash	8	spooky	mmh3.2	12	CRC	spooky

[a]Hash Type.

This confirms that the proposed approach is feasible for a practical problem (limited input of a few thousand million members). However, a faster computing hardware system or distributed filtering approach may be needed when expanding the input space, but this is not the primary focus of this work.

2) HF SELECTION

For all the subfilters of FeCBF, the number k of HFs is optimally calculated according to (2). Moreover, to reduce the volume of the hash computation without increasing the FPR, we adopted the algorithm proposed by Kirsch and Mitzenmacher [21], in which the set of k HFs is generated from a base HF pair. Each base HF pair (HF1, HF2) corresponds to a hash type and is denoted as Htype, which takes the integer values. The specific base HF pairs and the corresponding values of the Htype are listed in Table 2.

3) EXPERIMENT RESULT

From the FeCBF's design parameters predicted using our developed scripts and represented on graphs in Fig. 6, Fig. 7, and Fig. 8, some representative results are selected for experimental evaluations on real datasets to verify the correctness of the proposed FeCBF design model. Specifically, the verification experiments performed include the dependence of FPR_{norm} on M_{norm} at fixed $\chi = 100$ (Fig. 9(a)), the dependence of FPR_{norm} on χ at fixed $M_{norm} = 20$ (Fig. 9(b)), the dependence of M_{norm} on χ at fixed $FPR_{norm} = 1E\text{-}01$ (Fig. 9(c)), and the dependence of M_{norm} on FPR_{norm} at fixed $\chi = 500$ (Fig. 9(d)).

The member set of FEBF has a standard number of 1 million randomly generated 32-bit elements, from which the member sets of FPBF and FNBF are collected by extracting False Positive elements of the corresponding previous subfilters. As can be seen, the actual filtering results of FeCBF in all test cases relatively match the predicted ones. The detailed numerical data shows that the average deviation between expected and actual FPR_{norm} in Fig. 9(a), Fig. 9(b), Fig. 9(c), and Fig. 9(d) is about 1.6%, 2.3%, 0.1%, and 0.3% respectively, with a tendency for the actual filtering efficiency to be slightly better than predicted. The improvements of FeCBF in practical implementations were achieved by careful selection of the optimized hash type for each subfilter from the entire set of the 12 hash types listed in Table 2, some of which such as Htype 2, Htype 4, and Htype 7 are the most frequently used for minimizing the FPR of subfilters, as shown in Fig. 10. A critical parameter affecting the choice of a hash type is the subfilter's size, where a hash type might be the most efficient for a particular subfilter; however, it is not necessarily the best when the size of this filter changes. Nonetheless, optimizing the HF selections only has a minor extra impact (0.18 ÷ 1.17% on average), which could be considered an optional design step, the complexity of an HF could be a more critical design consideration for practical hardware implementation.

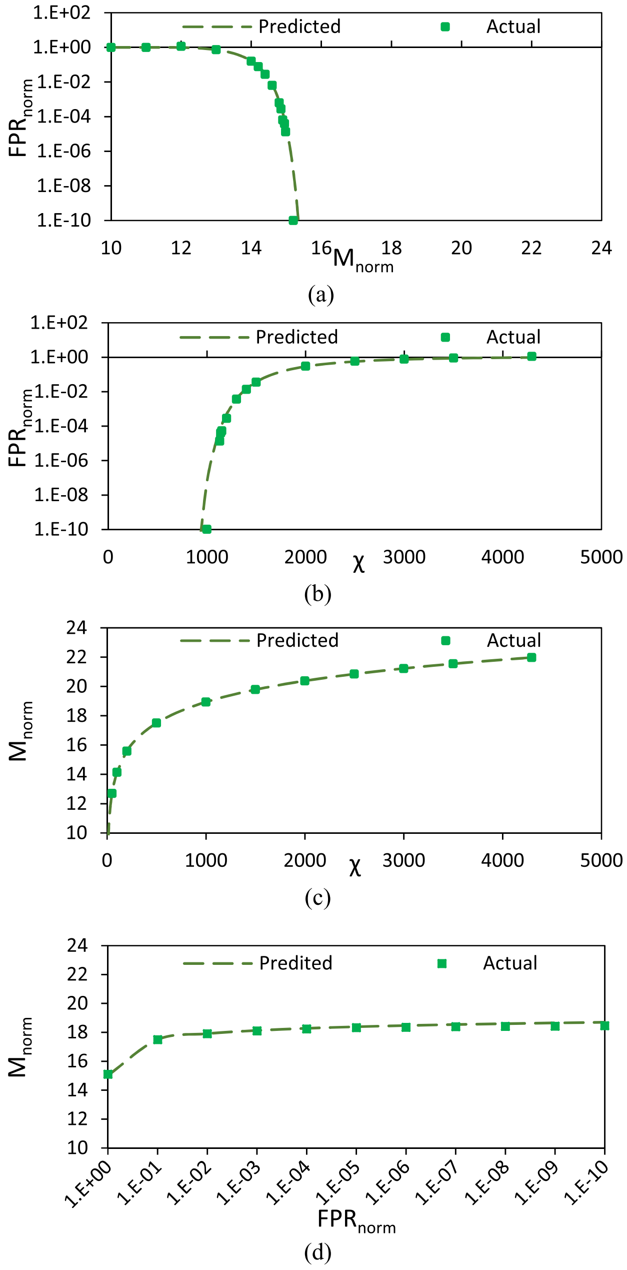

FIGURE 9. Practical verification of FeCBF design model (a) Dependence of error rate on the filter size, (b) Dependence of error rate on the number of query elements, (c) Dependence of filter size on the number of query elements, and (d) Dependence of filter size on the desired target error rate.

V. COMPARISON TO STATE-OF-THE-ART FILTERS

A. EVALUATION DATA AND PARAMETERS

In Section IV-D, we demonstrated the superiority of FeCBF over SBF in classification accuracy. In some case studies, the

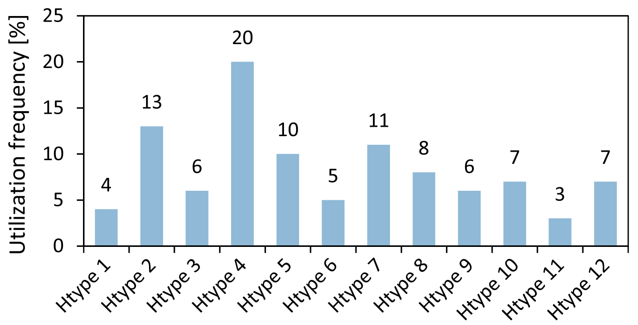

FIGURE 10. **The hash type effectiveness is evaluated based on their utilization frequency in real-world FeCBF tests.**

experimental results show that FeCBF could suppress filtering errors with the nearly empty output sets of False Positive elements, i.e., $n_o \cong 0$. In this Section, we compare FeCBF to three other selected state-of-the-art filters introduced in Section II-B, including f-HABF [14], SLBF [16], and Xor filter [20]. The experimental setup for FeCBF is entirely similar to other representative filters in terms of member sets and input datasets. We use the same dataset generated in [14] by the open-source tool YCSB (Yahoo! Cloud Serving Benchmark [22]) for experiments with f-HABF, SLBF, and Xor filter as the common dataset for evaluations in this work. Thus, the experimental dataset for FeCBF, f-HABF, SLBF, and Xor filter consists of 24, 074, 812 elements of 96-bit, in which 12, 500, 611 elements are *positive* (members of the filter), and the remaining 11, 574, 202 elements are *negative*, i.e., do not belong to the member set; thus $\chi = 11, 574, 202$ / $12, 500, 611 \approx 0, 925891$.

The filter sizes in all the Xor filter, SLBF, and f-HABF experiments in [14] were the same and varied from 100Mb to 260Mb. The lowest FPR was frequently achieved with f-HABF; however, their specific values were reported only in the two extreme cases corresponding to the lower-bound size (100Mb) and upper-bound size (260Mb) of the evaluated filters, equal to 3.46E−03 and 3.63E−06. We take these values as the desired targets of $\text{FPR}_{\text{FeCBF}}$ to determine the sub-optimal FeCBF parameters based on the proposed design flow. The FeCBF's input parameters are as follows: n_i = 12,500,611; $\chi \approx 0.926$; $\text{FPR}_{\text{target1}}$ = 3.46E-03, and $\text{FPR}_{\text{target2}}$ = 3.63E-06. The FeCBF's output parameters are the optimized values of subfilter size ratios α, β, and the FeCBF total size to achieve the smallest memory spaces and meet the input requirements.

B. IMPLEMENTATION RESULTS AND COMPARISON

The optimized design parameters of FeCBF are depicted in Fig. 11, including the sizes of FeCBF's sub-filters and their corresponding numbers of member elements. The actual sub-optimal sizes of FeCBF to achieve the pre-defined upper and lower bounds of target error rate $\text{FPR}_{\text{target1}}$ and $\text{FPR}_{\text{target2}}$ are $\mathcal{M}_1$ = 55.36 Mb (size ratios of FEBF, FPBF, and FNBF are 0.43, 0.41, and 0.16, respectively) and $\mathcal{M}_2$ = 62.22 Mb

TABLE 3. **Comparison of FeCBF with state-of-the-art bloom-based filters.**

	SBF	JEA 2020 [20]	ANIPS 2018 [16]	ICDE 2021 [14]	FeCBF (This work)
Represen-tative filter	Single layer	Xor filter	SLBF	f-HABF	3-layer FeCBF
Key ideas	BF baseline	Bloomier filter	Machine Learning	Hash optimization	Fe [a], Inv [b], DeN [c]
Member set	12,500,611				
Filter size [Mb]	100 ÷ 260				**55.36 ÷ 62.22**
Filter size vs. SBF [x]	1				**0.55 ÷ 0.24**
Actual FPR	1.78E-02 ÷ 2.83E-05	1.57E-02 ÷ 1.59E-05	6.81E-03 ÷ 1.72E-05	**3.46E-03 ÷ 3.63E-06**	
Actual FPR/ vs. SBF [x]	1	0.88 ÷ 0.56	0.38 ÷ 0.61	**0.19 ÷ 0.13**	
Constr. time vs. SBF [x]	1	**2.24**	> 10^3	2.3	4.61 ÷ 3.6
Query time vs. SBF [x]	1	**0.68**	> 500	1.04	2.29 ÷ 1.59
Design model	Yes	No	**Yes**	No	**Yes**

[a]Feature extraction, [b]Inverse filter (FPBF), [c]De-Negative filter (FNBF).

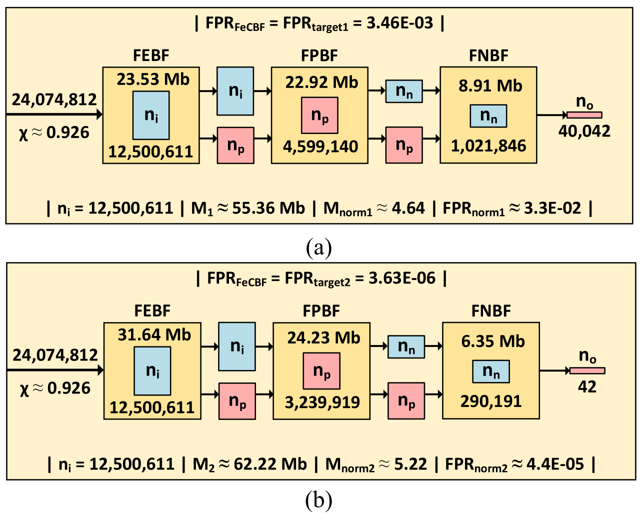
| FPR_FeCBF = FPR_target1 = 3.46E-03 |
FEBF
23.53 Mb
12,500,611
FPBF
22.92 Mb
4,599,140
FNBF
8.91 Mb
1,021,846
24,074,812
χ ≈ 0.926
n_i
n_p
n_n
n_o
40,042
| n_i = 12,500,611 | M_1 ≈ 55.36 Mb | M_norm1 ≈ 4.64 | FPR_norm1 ≈ 3.3E-02 |
(a)
| FPR_FeCBF = FPR_target2 = 3.63E-06 |
FEBF
31.64 Mb
12,500,611
FPBF
24.23 Mb
3,239,919
FNBF
6.35 Mb
290,191
24,074,812
χ ≈ 0.926
n_o
42
| n_i = 12,500,611 | M_2 ≈ 62.22 Mb | M_norm2 ≈ 5.22 | FPR_norm2 ≈ 4.4E-05 |
(b)

FIGURE 11. **The sub-optimal configurations of the FeCBF with the actual dataset of 24,074,812 elements, including 12,500,611 member elements, to achieve the desired target $\text{FPR}_{\text{FeCBF}}$ equals (a) 3.46E-03 and (b) 3.63E-06.**

(size ratios of FEBF, FPBF, and FNBF are 0.51, 0.39, and 0.16, respectively). Compared with SBF of the same size, the corresponding values of FPR_{norm} equal 3.3E−02 and 4.4E−05 in actual FeCBF evaluations.

In the following, we compare FeCBF with other data filter structures proposed in [14], [16], and [20] for some major design and performance metrics, which include the filter size, the filtering error rate, the filter construction time, and the query time (Table 3). The latter is the offline time spent on the designing step, such as Feature extraction/hash optimization and adding the member set. Note that all the time metrics are normalized to that of SBF for fair comparison since the hardware configurations in each work are not the same.

Regarding the filter size, the optimal $\mathcal{M}_1$ and $\mathcal{M}_2$ sizes of FeCBF are 55% and 24% of the f-HABF, SLBF, and Xor

Algorithm 1 Pseudo Code for Calculating the Sub-Optimal Size of FeCBF to Achieve a Desired Target Error Rate

```
Input: target FPR_FeCBF or FPR_norm, scaling factor $\chi$, increment steps $\alpha^{step}$, $\beta^{step}$, $M_{norm}^{step}$ of $\alpha$, $\beta$, and $M_{norm}$.
Output: sub-optimal size ratios $\alpha_{opt}$, $\beta_{opt}$, and sub-minimum normalized size $M_{norm}^{min}$.
1:  $M_{norm} \leftarrow M_{norm}^{init}$ [Initial estimated $M_{norm}$][a]
2:  NumLoopOfAlpha $\leftarrow 1/\alpha^{step}$ [Loop number of $\alpha$]
3:  NumLoopOfBeta $\leftarrow 1/\beta^{step}$ [Loop number of $\beta$]
4:  while FPR[b] is greater than TargetErrorRate do
5:      $M_{norm} \leftarrow M_{norm} + M_{norm}^{step}$
6:      $C \leftarrow 0.6185^{M_{norm}}$ [Re-Calculate the value of $C$]
7:      $\alpha \leftarrow 0$ [Re-Initialize the value of $\alpha$ to 0]
8:      for $i$ from 1 to NumLoopOfAlpha do
9:          $\beta \leftarrow 0$ [Re-Initialize the value of $\beta$ to 0]
10:         for $j$ from 1 to NumLoopOfBeta do
11:             TempVar $\leftarrow FPR_{model}$[c]
12:             if FPR is greater than TempVar do
13:                 FPR $\leftarrow$ TempVar
14:                 $\alpha_{opt} \leftarrow \alpha$
15:                 $\beta_{opt} \leftarrow \beta$
16:             end if
17:             $\beta \leftarrow \beta + \beta^{step}$
18:         end for
19:         $\alpha \leftarrow \alpha + \alpha^{step}$
20:     end for
21: end while
22: $M_{norm}^{min} \leftarrow M_{norm}$
```

[a] The initial value of M_{norm} is estimated based on the FeCBF design analytical model presented in Subsection IV-C.3.
[b] FPR is the current value (haven't reached the desired target) of FeCBF's error rate, which could be FPR_{FeCBF} or FPR_{norm}.
[c] FPR_{model} is the predicted value of FeCBF's error rate, which could be FPR_{FeCBF} or FPR_{norm}, and is calculated according to (12).

Algorithm 2 Pseudo Code for Optimizing the Error Rate of FeCBF With a Fixed Total Filter Size

```
Input: normalized size $M_{norm}$, scaling factor $\chi$, increment steps $\alpha^{step}$, $\beta^{step}$ of $\alpha$, $\beta$.
Output: sub-optimal size ratios $\alpha_{opt}$, $\beta_{opt}$, and sub-optimal error rates $FPR_{FeCBF}^{opt}$ or $FPR_{norm}^{opt}$.
1:  $C \leftarrow 0.6185^{M_{norm}}$ [Calculate the value of $C$]
2:  NumLoopOfAlpha $\leftarrow 1/\alpha^{step}$ [Loop number of $\alpha$]
3:  NumLoopOfBeta $\leftarrow 1/\beta^{step}$ [Loop number of $\beta$]
4:  $\alpha \leftarrow 0$ [Initialize the value of $\alpha$ to 0]
5:  for $i$ from 1 to NumLoopOfAlpha do
6:      $\beta \leftarrow 0$ [Initialize the value of $\beta$ to 0]
7:      for $j$ from 1 to NumLoopOfBeta do
8:          TempVar $\leftarrow FPR_{model}$[a]
9:          if FPR is greater than TempVar do
10:             FPR $\leftarrow$ TempVar[b]
11:             $\alpha_{opt} \leftarrow \alpha$
12:             $\beta_{opt} \leftarrow \beta$
13:         end if
14:         $\beta \leftarrow \beta + \beta^{step}$
15:     end for
16:     $\alpha \leftarrow \alpha + \alpha^{step}$
17: end for
18: end while
19: $FPR^{opt} \leftarrow$ FPR[b]
```

[a] FPR_{model} is the predicted value of FeCBF's error rate, which could be FPR_{FeCBF} or FPR_{norm}, and is calculated according to (12).
[b] FPR is the current value (non-optimized) of FeCBF's error rate, which could be FPR_{FeCBF} or FPR_{norm}.

uniform sizes, respectively, with the same experimental setup. Note that with that sizing, the FPR of FeCBF is achieved in the same as that of f-HABF and is the best in the class for the FPR. This FPR is about one order of magnitude less than those of Xor filter and SLBF. The SLBF based on ML techniques achieves a medium level of FPR, which might be reduced in cases where the input elements have more correlations.

SLBF is much slower than other filter structures regarding filter construction time since building ML models is not best supported on non-GPU machines. FeCBF takes slightly longer for construction time compared to f-HABF and Xor. Note that with a larger input set, FeCBF would take much longer for the Feature extraction task, though this metric is just the offline time (i.e., one-time) at the filter design step, which is not important in practice.

A similar trend could be observed regarding the query time, where SLBF is the worst because of the computational complexity related to ML models. At the same time, the Xor filter exhibits the fastest. In this specific implementation, the query time of the proposed FeCBF is slightly greater than f-HABF and about $3-4x$ of Xor. The query time of FeCBF highly depends on the filtering context and could be lower than that of SBF. Specifically, the query time of FeCBF's experiments in Section V-A. compared to SBF ranges from $0.83x$ to $1.13x$. The variations in query time of FeCBF in different filtering contexts are mainly caused by the variation of α, β, and χ corresponding to FeCBF configurations.

Based on the evaluation results presented in Table 3, FeCBF outperforms the rest in memory saving, especially in resource-utilization efficiency. Finally, it is worth noting that excluding SBF, only SLBF and FeCBF come with explicit predictive models and design procedures to determine the sub-optimal filter parameters. This could be important for deploying the filter in practice, especially in the case of dynamically changing targets and boundary conditions.

VI. CONCLUSION

This paper presents a novel filter structure, FeCBF, and its design methodology, which can achieve high accuracy with low memory requirements. FeCBF uses "Feature extraction" from the input dataset to record potential false-queried elements and cascade interleaved positive/negative subfilters for error correction. We presented a sub-optimal analytical model for the filtering error rate prediction based on fine-tuning the fundamental FeCBF's design parameters. Compared to the representative filters employing ML techniques (SLBF), hash optimization (f-HABF), and improved variation of Bloom filter (Xor filter), FeCBF outperforms these filters in either error rate reduction capability or level of memory saving. Specifically, in experimental evaluations with the same input and member sets, FeCBF and f-HABF show the best reductions of FPR targets from 81% to 87%, compared to SBF. Moreover, FeCBF achieved the smallest memory space, only about 24% to 55%, compared to the uniform size of SBF, SLBF, Xor filter, and f-HABF. The evaluated effectiveness of FeCBF is the basis for implementing this filter model on hardware accelerators to enhance accuracy and minimize resource utilization for big-data filtering applications.

APPENDIX A
SCRIPT FOR OPTIMIZING FeCBF'S MEMORY SIZE

See Algorithm 1.

APPENDIX B
SCRIPT FOR OPTIMIZING FeCBF'S ERROR RATE

See Algorithm 2.

REFERENCES

[1] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[2] G. Cheng, D. Guo, L. Luo, and Y. Qin, "Optimization of multicast source-routing based on Bloom filter," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 700–703, Apr. 2018.

[3] S. Nayak, R. Patgiri, and A. Borah, "A survey on the roles of Bloom filter in implementation of the named data networking," *Comput. Netw.*, vol. 196, Sep. 2021, Art. no. 108232.

[4] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," *Comput. Netw.*, vol. 57, no. 18, pp. 4047–4064, Dec. 2013.

[5] A. Kapoor and V. Arora, "Application of Bloom filter for duplicate URL detection in a web crawler," in *Proc. IEEE 2nd Int. Conf. Collaboration Internet Comput. (CIC)*, Nov. 2016, pp. 246–255.

[6] W. Fu, H. B. Abraham, and P. Crowley, "Synchronizing namespaces with invertible Bloom filters," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, May 2015, pp. 123–134.

[7] M. Gomez-Barrero, C. Rathgeb, G. Li, R. Ramachandra, J. Galbally, and C. Busch, "Multi-biometric template protection based on Bloom filters," *Inf. Fusion*, vol. 42, pp. 37–50, Jul. 2018.

[8] B. Goodwin, M. Hopcroft, D. Luu, A. Clemmer, M. Curmei, S. Elnikety, and Y. He, "BitFunnel: Revisiting signatures for search," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 2017, pp. 605–614.

[9] (Jul. 3, 2014). *Issue 10896048: Transition Safe Browsing From BF to Prefix Set—Code Review*. [Online]. Available: http://Chromiumcodereview.appspot.com

[10] R. Canetti, A. Trachtenberg, and M. Varia, "Anonymous collocation discovery: Harnessing privacy to tame the coronavirus," 2020, *arXiv:2003.13670*.

[11] Q. M. Duong, X. U. Dao, H. D. Nguyen, N. H. T. Tran, N.-H. Le, and Q.-K. Trinh, "An analysis of the effectiveness of cascaded and CAM-assisted Bloom filters for data filtering," in *Proc. Int. Conf. Intell. Syst. Netw.* Singapore: Springer, 2023, pp. 409–417.

[12] S. Lumetta and M. Mitzenmacher, "Using the power of two choices to improve Bloom filters," *Internet Math.*, vol. 4, no. 1, pp. 17–33, Jan. 2007.

[13] F. Hao, M. Kodialam, and T. V. Lakshman, "Building high accuracy Bloom filters using partitioned hashing," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, Jun. 2007, pp. 277–288.

[14] R. Xie, M. Li, Z. Miao, R. Gu, H. Huang, H. Dai, and G. Chen, "Hash adaptive Bloom filter," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, Chania, Greece, Apr. 2021, pp. 636–647, doi: 10.1109/ICDE51399.2021.00061.

[15] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 489–504.

[16] M. Mitzenmacher and T. Kraska, "A model for learned BFs and optimizing by sandwiching," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–12.

[17] Z. Dai and A. Shrivastava, "Adaptive learned Bloom filter (Ada-BF): Efficient utilization of the classifier with application to real-time information filtering on the web," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, no. 2020, pp. 11700–11710.

[18] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The Bloomier filter: An efficient data structure for static support lookup tables," in *Proc. 15th Annu. ACM-SIAM Symp. Discrete Algorithm*, 2004, pp. 30–39.

[19] D. Martin and R. Pagh, "Succinct data structures for retrieval and approximate membership," in *Proc. Int. Colloq. Automata, Lang., Program.* Berlin, Heidelberg: Springer, 2008, pp. 385–396.

[20] T. M. Graf and D. Lemire, "Xor filters: Faster and smaller than Bloom and cuckoo filters," *J. Exp. Algorithmics*, vol. 25, pp. 1–16, Jun. 2020.

[21] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," in *Proc. ESA*, Zurich, Switzerland. Berlin, Germany: Springer, 2006, pp. 1–12.

[22] M. Barata, J. Bernardino, and P. Furtado, "YCSB and TPC-H: Big data and decision support benchmarks," in *Proc. IEEE Int. Congr. Big Data*, Anchorage, AK, USA, Jun. 2014, pp. 800–801.

QUANG-MANH DUONG received the B.E. degree in radio electronics from Moscow Aviation Institute, Moscow, Russia, in 2006, and the M.S. degree in electronic engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2012, where he is currently pursuing the Ph.D. degree in electronic engineering.

From 2007 to 2019, he was a Lecturer and a Researcher with the Department of Microprocessor Engineering, Faculty of Radio-Electronic Engineering, Le Quy Don Technical University. His research interests include low-power integrated circuit design, energy-efficient signal processing, hardware security, in-memory computing, and emerging memory technologies.

KHOA-SANG NGUYEN received the B.E. degree in electronic and information engineering and the master's degree in information and communication engineering from Nanjing University of Science and Technology, Nanjing, China, in 2012 and 2015, respectively.

Since 2016, he has been a Lecturer and a Researcher with the Department of Microprocessor Engineering, Faculty of Radio-Electronic Engineering, Le Quy Don Technical University. His research interests include low-power integrated circuit design, embedded systems, and digital signal processing.

HAI-DUONG NGUYEN received the Ph.D. degree in electronic engineering from Bauman Moscow State Technical University, Moscow, Russia, in 2007.

Since 1995, he has been teaching computer engineering with Le Quy Don Technical University, Hanoi, Vietnam. He is currently a Senior Researcher and the Head of the Department of Microprocessor Engineering, Faculty of Radio Electronic Engineering, Le Quy Don Technical University. His research interests include computing architecture, embedded systems, digital signal processing, and hardware security.

XUAN-UOC DAO received the B.E. degree in electronic engineering from Bauman Moscow State Technical University, Moscow, Russia, in 2017, and the M.S. degree in electronic engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2022.

His research interests include low-power integrated circuit design, digital signal processing, and hardware security.

ANH-TUAN DO (Senior Member, IEEE) received the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2010.

He was a Research Fellow with VIRTUS, IC Design Centre of Excellence, NTU, from 2010 to 2015. He joined the Digital IC Design Group, Institute of Microelectronics (IME), A*STAR, Singapore, in 2015. He has authored or coauthored more than 100 IEEE journals and conference papers. His research interests include CMOS design, neuromorphic computing, AI hardware, energy-efficient computing architecture, and cryogenic circuits for quantum technologies. He is a TPC Member of A-SSCC. He received the Best Paper Award from ISLPED 2023, ISOCC 2022, and SOCC 2012. He serves as an Associate Editor for IEEE Transactions on Very Large Scale Integration (VLSI) Systems. He serves as a Reviewer for several IEEE journals and conferences, including IEEE Transactions on Circuits and Systems—I: Regular Papers, IEEE Transactions on Circuits and Systems—II: Express Briefs, and IEEE Transactions on Very Large Scale Integration (VLSI) Systems.

QUANG-KIEN TRINH (Member, IEEE) received the Ph.D. degree in computer engineering from the National University of Singapore, in 2018. He is currently an Associate Professor and the Deputy Head of the Department of Microprocessor Engineering, Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Vietnam. He is the author or coauthor of over 50 publications, three patents, and two books. His research interests include low-power integrated circuit design, emerging memory technologies, hardware security, edge-AI circuits, and architecture. He served as a TPC of conferences, such as McSoC, ATC, NICS, and INISCOM. He was a Reviewer of several journals and transactions, including IEEE Transactions on Circuits and Systems—II: Express Briefs and IEEE Transactions on Very Large Scale Integration (VLSI) Systems.