Breaking the Scalability Barrier of Content Addressable Memories: A Probabilistic Alternative for Large-Key Associative Search

Sateesan, Arish; Vliegen, Jo; Mentens, Nele

Published in:
ACM Transactions on Reconfigurable Technology and Systems

DOI (link to publication from Publisher):
10.1145/3805708

Publication date:
2026

Document Version
Accepted author manuscript, peer reviewed version

Breaking the Scalability Barrier of Content Addressable Memories: A Probabilistic Alternative for Large-Key Associative Search

ARISH SATEESAN*, Department of Electronic Systems, Aalborg University, Denmark
JO VLIEGEN, ES&S-COSIC, ESAT, KU Leuven, Belgium
NELE MENTENS, ES&S-COSIC, ESAT, KU Leuven, Belgium and LIACS, Leiden University, The Netherlands

Content Addressable Memories (CAMs) offer high-speed, deterministic lookups but face significant scalability challenges with large input keys (>100 bits), leading to excessive power, silicon area, and memory costs. This paper introduces Probabilistic Content Addressable Memory (P-CAM), a novel architecture designed to overcome these limitations by trading strict determinism for memory efficiency and scalability. P-CAM compresses high-dimensional inputs into fixed-size fingerprints using hashing, making memory requirements independent of key length. P-CAM preserves the constant-time lookup advantage of CAMs, while supporting applications with large keys, such as networking, bioinformatics, and machine learning, where conventional CAMs are impractical. FPGA implementation on Xilinx UltraScale+ devices shows that P-CAM maintains constant query latency and delivers 15× improvement in resource efficiency when handling 384-bit keys, compared to state-of-the-art deterministic CAMs designed for narrower inputs. Although P-CAM's probabilistic nature introduces a small, controllable false-positive rate, it can be configured for fully deterministic operation under specific constraints. To the best of our knowledge, P-CAM is the first CAM architecture to employ a fingerprint-based probabilistic data structure as the primary storage mechanism for associative lookup, distinguishing it from prior probabilistic approaches that are limited to set membership checks, offering a robust and scalable alternative for modern data-intensive systems.

CCS Concepts: • **Hardware** → **Memory and dense storage**; **Reconfigurable logic and FPGAs**; **Emerging architectures**; • **Networks** → Network algorithms; • **Theory of computation** → **Probabilistic computation**; **Data structures design and analysis**.

Additional Key Words and Phrases: Content Addressable Memory (CAM), Probabilistic CAM (P-CAM), Associative Memory, FPGA; Approximate Computing, Large-Key Associative Search, Probabilistic Data Structures

ACM Reference Format:
Arish Sateesan, Jo Vliegen, and Nele Mentens. 2026. Breaking the Scalability Barrier of Content Addressable Memories: A Probabilistic Alternative for Large-Key Associative Search. *ACM Trans. Reconfig. Technol. Syst.* 1, 1 (March 2026), 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Part of this work was conducted at KU Leuven.

Authors' Contact Information: Arish Sateesan, arishs@es.aau.dk, Department of Electronic Systems, Aalborg University, Copenhagen, Denmark; Jo Vliegen, jo.vliegen@kuleuven.be, ES&S-COSIC, ESAT, KU Leuven, Leuven, Belgium; Nele Mentens, nele.mentens@kuleuven.be, ES&S-COSIC, ESAT, KU Leuven, Leuven, Belgium and LIACS, Leiden University, Leiden, The Netherlands.

ACM 1936-7414/2026/3-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Content Addressable Memory (CAM) is a specialized associative memory used for high-speed membership queries and memory lookups, particularly used in hardware. Unlike conventional Random Access Memory (RAM), which retrieves data using specific memory addresses, CAM identifies the address of a stored data word based on the content itself. CAM searches its entire memory array in parallel to find the data word and return the stored address, achieving constant lookup time complexity, $O(1)$ [38]. This parallel search capability makes CAM invaluable in networking applications demanding deterministic, low-latency response times, including packet classification, IP routing, firewall filtering, access control lists (ACLs), and Quality-of-Service (QoS) enforcement in networking [27, 28, 38, 54]. Beyond networking, CAMs are also employed in database indexing, cache management, pattern recognition, and high-throughput applications in computational biology, such as DNA/RNA sequence alignment [22, 25].

Historically, CAMs emerged in the context of networking equipment, particularly routers and switches, where line-rate packet processing requires rapid matching of headers against routing or access control tables. Their single-cycle performance, deterministic behavior, and suitability for hardware implementation make them an attractive option for real-time systems. However, these benefits are offset by several challenges. CAMs are known for high dynamic and static power consumption, substantial silicon area requirements, hardware complexity, large memory footprint, and limited scalability [38]. These challenges are particularly severe in high-speed networking environments such as Terabit Ethernet and large-scale deployments like data center switches. They become even more critical on power- and resource-constrained platforms like field-programmable gate arrays (FPGAs), which are increasingly favored for hardware acceleration and in-network computing [40, 56].

CAM architectures are broadly classified into two types based on their matching capabilities. The fundamental variant is the Binary CAM (BCAM), which stores data using only 0s and 1s and performs exact matching. To support more complex search patterns, Ternary CAM (TCAM) extends the traditional BCAM architecture by introducing *don't care* states for partial matches, enabling support for prefix matching and wildcard-based searches. Figure 1 presents a comparison of the operational behavior of RAM, BCAM, and TCAM. The ternary feature of TCAM is vital for applications such as subnet matching in IP routing, packet classification, and ACL evaluations. However, TCAM aggravates the shortcomings of BCAMs as the introduction of ternary logic triples the storage complexity, increases power consumption, and further limits scalability due to additional circuitry for matching and mask storage [9]. As BCAM and TCAM store full entries in fixed-length bit vectors, their scalability deteriorates as key widths and dataset sizes increase [43].

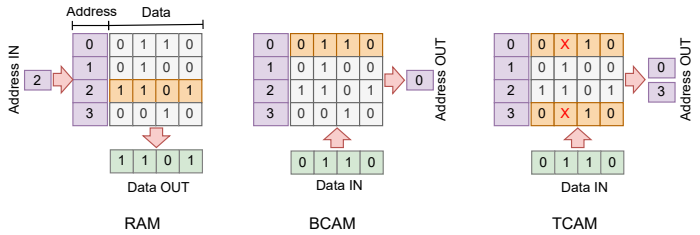

Fig. 1. Operational differences in RAM, BCAM, and TCAM [42]. RAM returns the data stored at the input address. BCAM returns the address where the input data exactly matches the stored data. TCAM stores (data, mask) pairs and returns all addresses where the input data matches the stored data, considering the mask (wildcard matching).

This leads to increased latency due to longer routing delays, lower operating frequencies, and greater difficulty achieving timing closure on modern high-speed hardware platforms [21].

Application Domains and Emerging Requirements

In addition to traditional networking roles, CAMs are being reconsidered in emerging domains. In network security, for instance, CAMs are used in identifying large flows [3], detecting intrusion patterns [19, 29], or performing deep packet inspection [58]. These applications often requires matching on a 5-tuple of fields totalling 104 bits for IPv4 or even larger for IPv6 [53]. In bioinformatics, large-scale sequence alignment tasks frequently require searching for gene patterns in DNA/RNA sequences. A 64-base DNA string encoded with 2 bits per base requires 128 bits, while a 64-residue amino acid sequence, using 5-bit encoding, would need 320 bits [10]. These requirements far exceed the typical input sizes for which CAMs are optimized. Larger patterns further increase complexity, as they require segmentation and multiple CAM operations to match the full pattern. Furthermore, applications in machine learning hardware, graph analytics, and edge AI are increasingly exploring associative memory primitives for fast nearest-neighbor search, input pattern matching, and feature vector comparisons, extending the relevance of CAM-like structures to data-intensive, pattern-based computation [24].

These application domains can broadly be categorized based on their tolerance to errors:
Deterministic Use-Cases: Tasks such as IP routing, firewall filtering, and ACLs demand strict correctness guarantees, where both false positives and false negatives are unacceptable. Traditional BCAM and TCAM architectures are well-suited for such use-cases due to their deterministic behavior.
Probabilistic and Statistical Use-Cases: Other domains, such as large-scale network traffic monitoring, approximate set membership, bioinformatics sequence matching, and machine learning pattern matching, can often tolerate bounded errors in favor of substantial gains in scalability and resource efficiency.

While effective for deterministic workloads, BCAM and TCAM architectures remain constrained by their rigidity and face scalability and implementation challenges as input widths grow. They are ill-suited for modern use cases that demand low-power, dynamic key-value associations, or the ability to map multiple keys to the same label/class, a necessity in tasks such as classification, anomaly detection, and flow aggregation.

Contributions

To address the limitations of traditional CAMs, this work introduces a Probabilistic CAM (P-CAM) architecture that offers a scalable, resource-efficient architecture, optimized for error-tolerant applications. Here, 'traditional CAM' refers to conventional BCAM designs; however, the same scalability challenges exist for TCAMs as well, due to their even higher memory cost. P-CAM also provide mechanisms to support deterministic use-cases under specific operational constraints or static workloads, thus extending its applicability beyond purely probabilistic domains.

The key contributions of this work are as follows:

- The proposal of P-CAM, a novel CAM architecture based on probabilistic associative memory that stores only compact fingerprints instead of full keys, overcoming the key-width scalability limitations of CAMs by trading strict determinism. This allows massive memory savings while preserving constant-time lookup performance with support for modern, dynamic workloads, through simultaneous update/query operations and multi-key-to-class associations.
- A constant-footprint design, achieved through hashing-based fingerprinting, that decouples the memory requirement from the input key width, enabling scalable and efficient hardware-accelerated search for applications with very large input keys where CAMs are infeasible.

- A flexible architecture that provides a controllable trade-off between accuracy and resource usage, with tunable parameters for application-specific tolerance. It can also be configured for deterministic operation under constrained settings, extending its applicability to mission-critical tasks.
- A comprehensive FPGA-based implementation and evaluation of P-CAM, demonstrating its practical viability, constant-time latency, and resource efficiency compared to state-of-the-art CAM designs, particularly for wide inputs.

2 Background and Motivation

2.1 CAM Architectures and Limitations

CAMs, owing to their single-cycle latency and inherent hardware compatibility, have been the focus of optimization efforts for decades. While most existing research focuses on transistor-level improvements and energy-efficient circuit designs, these approaches overlook fundamental challenges such as scalability and adaptability to new use cases [8, 18, 25, 30, 38, 41, 44]. Software alternatives like trie-based methods and hash tables for networking tasks such as prefix matching and IP lookups offer memory-efficient solutions, but are constrained by sequential memory access times and are unsuitable for high-speed applications [46]. Hybrid approaches, such as combining trie-based methods with hash-based membership queries [4, 31], reduce off-chip memory accesses by restricting these accesses to cases where the hash-based query yields a positive result. Despite this, the dependence on off-chip memory introduces high latency, thereby limiting their applicability in latency-sensitive operations on resource-constrained devices. Hierarchical CAMs [37] and partitioned architectures attempt to localise searches to smaller blocks, reducing latency for partial lookups, but scalability remains a bottleneck as datasets grow.

Several memory-optimised CAM architectures on FPGAs, such as DSCAM+ [55], HLSCAM [1] and SplitBucket [47] effectively reduce memory usage by trading off Look-Up Tables (LUTs) on FPGA. While SplitBucket and HLSCAM eliminate the use of memory entirely, DSCAM+ employs a mixed approach to balance LUT and memory usage, proving effective for low-bitwidth inputs. SplitBucket and DSCAM+ target prefix matching for IPv4 routing tables where the prefix size is limited to 32 bits. However, its performance tends to degrade with larger inputs due to its architectural complexity, making such architectures impractical for hardware implementation for larger input sizes. HLSCAM targets an input width of 150 bits, but with poor logic resource efficiency. These architectures may not scale well to accommodate very large input widths typical in IPv6 headers, 5-tuple flow records, or encoded biological sequences. A detailed discussion on these architectures with regard to resource efficiency is provided in Section 4.3.5.

2.2 Probabilistic Approaches

Probabilistic data structures, such as Bloom filters [5], Cuckoo filters [15], and hash tables [26, 36, 49], have been explored as potential CAM alternatives, particularly for handling larger inputs. Although these structures offer scalability and memory efficiency in terms of input bit widths and table size, they exhibit significant limitations as they trade off accuracy and determinism. While Bloom and Cuckoo filters offer compact storage and fast membership testing, they cannot retrieve the associated data or memory address, limiting their utility in key-value store or table lookup scenarios. Hash tables, including Cuckoo [36] and Peacock variants [26], function as key-value stores but demand higher memory footprints to achieve acceptable accuracy due to significant hash collisions [49]. Additionally, their non-deterministic memory access latency, especially under high load, makes them unsuitable for high-speed applications [51].

While probabilistic methods offer a path to scalability, it is crucial to distinguish between different architectural goals. One category, approximate associative memories, is tailored for similarity search. Approaches like Hamming-distance-based CAMs [17] and those using Locality-Sensitive Hashing (LSH) [33] are engineered to identify items that are similar but not necessarily identical to a query. By intentionally maximizing hash collisions for similar inputs, they excel in pattern recognition and approximate nearest-neighbor tasks. However, this inherent design for imprecision makes them unsuitable for mission-critical applications such as IP routing, which demand exact-match fidelity. These data structures neither replicate the essential CAM functionality nor achieve a high accuracy-memory trade-off. Consequently, there remains a critical need for innovative CAM architectures that reconcile scalability, low-latency operation, and hardware compatibility for contemporary applications.

There have been prior works on probabilistic data structures that have been used to augment CAMs. However, these works primarily focus on probabilistic membership check while the CAM architecture remains conventional. Ooka et al. [35] augment a conventional CAM-based name lookup engine with Bloom filters in a content-centric router. However, the probabilistic behavior is confined to the membership pre-check using Bloom filter, and it uses conventional CAM for the matching operation. Similarly, Pontarelli & Ottavi [39] use Bloom filters in conjunction with a conventional TCAM for error detection/correction, and the TCAM's normal lookup remains exact. pbCAM [52] uses multiple conventional CAM banks with hashed indexing and a Bloom filter to test for an entry's presence. While this probabilistically-banked design enables energy savings, the actual CAM lookup still uses exact matching once the correct bank is chosen. Byun et al. [6] go further by replacing CAMs entirely with a vectored Bloom filter for IP lookup. But it performs only membership query similar to Bloom filter, resembling a compressed lookup table, and does not provide the associative memory functionality that retrieves the address or value associated with a matched key. Memristor-based approximate CAM proposals [19, 45] focus on using analog approximation for BCAM or TCAM-like functionality, but the behavior remains fundamentally deterministic, with any observed inaccuracies arising from analog variability rather than deliberate probabilistic design based on hashing.

In summary, none of these works implements a content-addressable memory or associative memory array that is entirely probabilistic by architecture. They either incorporate probabilistic algorithms around an otherwise conventional CAM or treat probabilistic effects as errors to minimize, rather than enabling them as a feature. P-CAM, in contrast, uniquely proposes to make the CAM's match operation itself inherently probabilistic via its architecture and manages this uncertainty as part of normal operation to gain efficiency. This means P-CAM's behavior and contributions differ fundamentally from prior work as it includes deliberate probability in matching and storage, whereas the others either deliver strictly deterministic matches or introduce probability only in supporting mechanisms, not in the core CAM architecture.

2.3 The P-CAM Approach: A Probabilistic Alternative

In light of these limitations of CAMs, there is a growing need for scalable, low-power, and reconfigurable associative memory architectures. To overcome these scalability and efficiency limits of CAMs, this work explores a fundamental trade-off of sacrificing the guaranteed determinism of traditional CAMs for a probabilistic approach that enables massive scalability while maintaining constant-time lookups. We propose the Probabilistic Content Addressable Memory (P-CAM), based on the SPArch [50] architecture, that achieves significant memory efficiency and scalability, making it suitable for large-scale applications.

Unlike prior probabilistic structures that support only set membership checks, P-CAM introduces a distinct architectural approach that uses a fingerprint-based probabilistic data structure as its

primary storage and matching mechanism for associative lookup. To the best of our knowledge, this makes P-CAM the first CAM architecture to fully integrate such a design. P-CAM leverages hashing-based techniques to encode high-dimensional input data into compressed, fixed-size fingerprints. Unlike traditional CAMs that store the full-width key, P-CAM's memory footprint remains independent of the input key width, which significantly reduces power, chip area, and computational complexity. By tolerating a small, controlled rate of false positives and leveraging the inherent parallelism of hardware-friendly hashing, P-CAM achieves a balanced trade-off between speed, scalability, and flexibility, while preserving constant-time ($O(1)$) lookup performance. These properties make P-CAM especially suitable for applications with extremely large keys (hundreds of bits) in domains such as networking, bioinformatics, and machine learning, where conventional CAMs are limited by area and power constraints.

Because P-CAM uses a flexible hashing scheme to map keys to memory locations via a sketch array, rather than assigning a rigid, dedicated slot for each entry, it inherently supports multi-key-to-class associations and dynamic updates, making it ideal for modern, evolving workloads that require flexibility beyond simple exact-match searches. A sketch is a 2-dimensional array of memory that compactly store input data using multiple independent hash functions, enabling approximate query operations such as membership tests, frequency estimation, or similarity detection with bounded error [11]. Sketches trade exactness for space and time efficiency, and are particularly suited for high-speed data stream processing. As a sketch-based architecture, P-CAM is inherently suited for streaming data applications, where inputs must be processed and summarized in real time. Hence, the hardware implementation of P-CAM supports simultaneous update and query operations, enhancing its capability for real-time network flow monitoring [13] and similar data-intensive applications.

A high-level comparison of conventional BCAM, TCAM, and P-CAM is provided in Table 1. It is also important to note that a probabilistic CAM is distinct from an approximate-matching CAM. The latter are architectures designed to find matches within a specified distance metric, such as the Hamming distance. In contrast, probabilistic CAMs are algorithmic data structures that emulate CAM behavior with inherent and controllable error probability (false positives). The error probability of P-CAM and its implications for accuracy-critical applications are analyzed in Sections 3.3 and 3.4, including mechanisms to mitigate false positives and preserve correctness.

3 Architecture of P-CAM

The design of P-CAM adapts the core sketching mechanism from SPArch [50], an architecture originally designed for network flow measurement. By strategically omitting SPArch's counter array and modifying its sketching component, P-CAM is transformed into a novel, lightweight

Table 1. Comparison of BCAM, TCAM, and Probabilistic CAM

Feature	BCAM	TCAM	Probabilistic CAM
Match Type	Exact match	Exact + Wildcard	Exact match with error probability
Lookup Latency (cycles)	**Very Low (1 cycle)**	**Very Low (1 cycle)**	Low (constant, few cycles)
Lookup Latency (μs)	$\propto$ CAM size	$\propto$ CAM size	**Constant (independent of CAM size)**
Scalability	Low	Very Low	**High**
Deterministic Result	**Yes**	**Yes**	No (allows false positives)
Lookup Time Complexity	$O(1)$	$O(1)$	$O(1)$
Search Circuit Complexity	$O(n)$	$O(n)$	$O(1)$
Memory Efficiency	Moderate	Low	**High**
Hardware Complexity / Area	High	Very High	**Low**
Power Consumption	High	Very High	**Very Low**
Primary Use Case	Databases, caching	IP routing, ACL deep packet inspection firewall filtering	network traffic monitoring, Large-scale similarity search, machine learning, bioinformatics

n - total number of entries stored in the CAM

architecture that functions as a general-purpose associative memory, a distinct application from its conceptual predecessor.

Figure 2 shows the architecture of P-CAM. The sketch component is implemented as a two-dimensional memory array of dimensions $m \times d$, where d represents the number of rows and m the number of cells per row. Given an element x, it is mapped to a cell in each row i in the sketch using d independent hash functions, $h_i(x)$, for $i = 1, \ldots, d$. For each $x \in \mathcal{X}$, the value $h_i(x)$ is some integer j with $0 \leq j \leq m-1$, where $\mathcal{X}$ is the universe of all elements (or keys). A core component of P-CAM is the hash function. Since the memory requirement of P-CAM is fixed irrespective of the input size of entries, it is the hash function that takes care of the variability of input sizes. A hash function that can accommodate the largest possible input width is sufficient, as the smaller entries can be padded with zeroes to match the input block size of the hash function. The hash function used in P-CAM is detailed in Section 4.1.

Each cell in the sketch, termed as fingerprint-address cell (FAC), stores a tuple (f, A) consisting of a fingerprint (f) of the element and the corresponding address (A). In our design, the sketch refers to the arrays of FACs that collectively implement the probabilistic matching and compressed storage logic of P-CAM. The fingerprint is computed using a fingerprint function, $\mathtt{f}(x)$, which maps a key to a short fixed-size binary representation using a hash function. The address is generated using a simple counter module, *AdGen*, which increments with each new insertion and is computed as $A_{x_{new}} = A_{x_{prev}} + 1$, where $A_{x_{prev}}$ is the last assigned address. Since the address is incremented only on new insertions, it serves as a logical timestamp that tracks the relative age of each entry. To provide the functionality of a key-value store, a dedicated memory block, V, is allocated for storing values. The depth of the memory block is the number of elements n to be stored in the P-CAM, which is also the maximum value of the *AdGen*. Once the maximum value is reached, a *memory full* flag is raised, denying any further insertions. As the counter does not wrap around, it avoids the risk of misclassifying recent entries as older ones due to address reuse. The values are mapped to the memory using the addresses corresponding to the elements in the FACs: $V[A_x] \rightarrow$ value of element x. The total memory requirement of a single memory array would be $m \times (k + a)$ bits, where k and a are the sizes of f and A, respectively, in bits, and $a = \log_2(n)$. A comprehensive list of the notational conventions used in the P-CAM architecture is provided in Table 2.

The operations of P-CAM - update, query, and delete - are described in the following sections. The pseudocode for the update, query, and delete operations in P-CAM is provided in Algorithm 1.

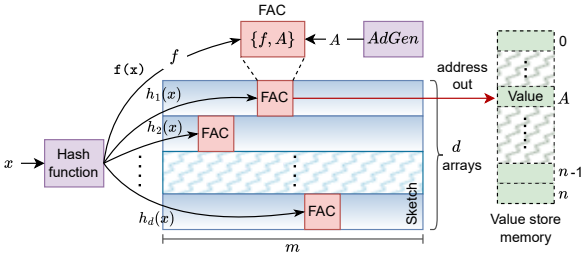

Fig. 2. Architecture of the proposed Probabilistic Content Addressable Memory (P-CAM), showing the sketch arrays composed of fingerprint-address cells (FACs), hash function, the address generator, and the value store memory.

Table 2. Notational conventions used in the p-CAM architecture and analysis.

x	Input key
b	Input bit width
d	Depth of the sketch/number of sketch memory arrays
m	Width of the sketch
n	Number of distinct entries in P-CAM
A	Address of the value store memory location
a	Address size in bits ($\log_2 N_c$)
FAC	Fingerprint-Address Cell
f	Fingerprint
k	Size of the fingerprint in bits
h_i	Hash function to index the i^{th} row of the sketch
$\mathsf{f}(x)$	fingerprint function
V	Value store memory
P_{fp}	Probability of false positives
α	Load factor ($\frac{n}{m}$)
c	Confidence value
T	Table size (= m)
w	Weighting factor for LUTRAMs
E	Normalized efficiency metric

3.1 Core P-CAM Operations

Update: To update the P-CAM with a new element x, we first compute its fingerprint ($f_x = \mathsf{f}(x)$) and identify the set of d hash-indexed FACs, $FAC[i, h_i(x)]$, for $i = 1, \ldots, d$. If one or more of the hash-indexed FACs are empty, they are directly updated with the new fingerprint and the address (A_x) from AdGen (Algorithm 1, lines 5-9). The content of the FAC becomes (f_x, A_x). If none of the indexed FACs are empty and no matching fingerprints are found, two scenarios are considered:
(i) Identical pairs: If multiple FACs contain identical fingerprint-address pairs, one of the identical FACs, either selected randomly or the first match in the array, is updated with the new fingerprint and address (Algorithm 1, lines 11-14).
(ii) Distinct pairs: If all fingerprint-address pairs are distinct, the FAC with the smallest address, which corresponds to the oldest entry (as the address acts as a logical timestamp tracking insertion order), is replaced with the new entry (Algorithm 1, lines 15-18). Although the probability of occurrence of this scenario is rare when the memory size is appropriately allocated, if overwriting the FAC is not desirable, insertion can be denied by raising a *memory full* flag to eliminate the possibilities of false negatives. The original SPArch algorithm permits insertion by replacing the oldest entry, as it is more beneficial for applications such as heavy-hitter detection, where older or outdated network flows are less relevant. Further discussion on the occurrence of false negatives and their elimination is presented in Section 3.5.

After every new update, the AdGen value is incremented. If at least one FAC contains a matching fingerprint, the item is considered already present, and no further action is taken (Algorithm 1, lines 20–21).
Query: Figure 3 shows the operational diagram of the P-CAM query operation. To query for an element x, its fingerprint ($f_x = \mathsf{f}(x)$) is computed, and the set of hash-indexed cells ($FAC[i, h_i(x)]$, for $i = 1, \ldots, d$) is retrieved. If at least one of the hash-indexed locations (FACs) is empty or if all locations are full but contain no matching fingerprints ($\forall (f_t, A_t) \in FAC[i, h_i(x)], f_t \neq f_x$), the element is considered absent (Algorithm 1, lines 24-25).

If all FACs are occupied and there is at least one matching fingerprint, the element is considered present if all corresponding fingerprint-address pairs are identical ($\{(f_t, A_t) \in FAC[i, h_i(x)] \mid f_t = f_x\}$); in this case, the associated address is returned (Algorithm 1, lines 26-28). In cases where

Algorithm 1: Update, query, and deletion in P-CAM

```
Function updateCAM(element, sketch, counters, AdGen):
    fingerprint ← element.fingerprint;
    address ← AdGen.address;
    for hash-indexed FACs in each row in sketch do
        if all or some FACs are empty then
            for empty FACs in rows do
                FAC.fingerprint ← fingerprint;
                FAC.address ← address;
            AdGen.address ← AdGen.address + 1;
        else if None of the FACs are empty & no matching fingerprint then
            if More than one FACs have identical FA pair then
                FAC_r ← random(identical FACs);
                FAC_r.fingerprint ← fingerprint;
                FAC_r.address ← address;
            else if None of the FACs are identical then
                FAC_m ← FAC(min(address));
                FAC_m.fingerprint ← fingerprint;
                FAC_m.address ← address;
            AdGen.address ← AdGen.address + 1;
        else if None of the FACs are empty & there is matching fingerprint then
            pass; // No action required
Function queryCAM(element, sketch, counters, AdGen):
    for hash-indexed FACs in each row in sketch do
        if One or more FACs are empty OR no matching fingerprints then
            match ← '0';
        else if one or more matching fingerprints are present then
            if all fingerprint-address pair is identical then
                address ← FAC.address[fingerprint match];
            if one or more matching fingerprint has different address pair then
                if there is a majority in fingerprint-address pairs then
                    address ← FAC.address(majority);
                else
                    address ← FAC.address(highest address);
            match ← '1'; output.address ← address;
Function removeCAM(element, sketch, counters, AdGen):
    Perform queryCAM to identify the matching FACs;
    for all matching FACs do
        FAC.fingerprint ← 'None';
        FAC.address ← 'None';
```

all FACs are full and multiple matching fingerprints are found, but at least one of the associated addresses differs (i.e., (f_x, A_t) with non-unique addresses A_t), two scenarios are considered:

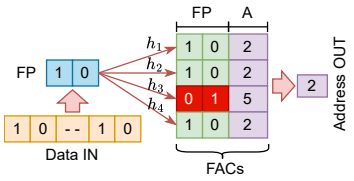

Fig. 3. Operational diagram illustrating the P-CAM query process. The fingerprint (FP) derived from the input data is compared with hash-indexed FACs, and the address (A) associated with the matching fingerprint is returned.

(i) Among all the non-identical fingerprint-address pairs, if an address A^* occurs as a majority, it is selected as the output (Algorithm 1, lines 30–31).
(ii) When no majority exists, the pair with the highest address ($\max\{A_t \mid (f_t, A_t) \in FAC[i, h_i(x)] \wedge f_t = f_x\}$) is chosen as the output (Algorithm 1, lines 32-33). This is because there is a higher probability that the older entries are replicated across multiple FACs than the newest entries. Since the address also serves as a logical timestamp, the highest address corresponds to the most recent entry.
Delete: To delete an item x, a query is first executed as outlined in the *query* operation to locate the matching FACs. The contents of these identified FACs are then cleared (Algorithm 1, lines 35–39).

3.2 Functionality as a Key-Value Store

To function as a key-value store, an additional memory block V to store the values is added, as shown in Figure 2. The addresses stored in the FACs serve as indices to the storage memory. While adding a new element, the *updateCAM* procedure from the Algorithm 1 is followed, and the corresponding memory location is updated using the generated address. If the element already exists, the matching address is retrieved and the associated value is updated at the memory location corresponding to that address. During a query, the value stored in V corresponding to the address of the element is retrieved.

Figure 4 further shows how P-CAM differs from conventional CAM in this context. Unlike traditional CAMs, which map each key to a fixed address location, P-CAM enables flexible key-to-value associations, allowing multiple distinct keys to be mapped to a specific address or class. This multi-key-to-class capability is particularly beneficial in classification tasks, where semantically equivalent keys can share a common output and memory location, thereby reducing storage overhead.

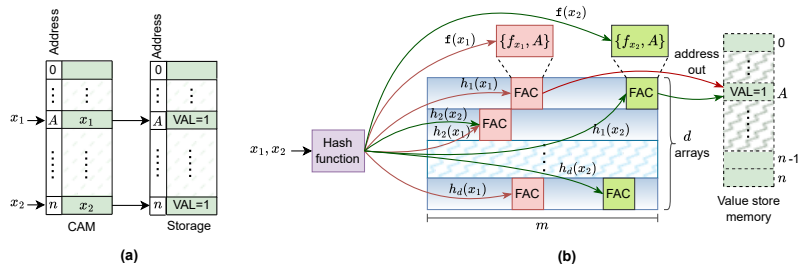

Fig. 4. Comparison of (a) conventional CAM and (b) P-CAM architecures, illustrating how P-CAM supports classification of multiple keys into the same value class. In this example, both keys x_1 and x_2 are associated with the same class value (VAL=1).

3.3 False Positive Probability Analysis

The Probability of false positives (P_{fp}) of P-CAM is determined by three factors: hash collisions (where two distinct elements must hash to the same location), fingerprint collisions (where two different elements produce the same fingerprint), and the address mismatches in the FAC. A hash collision occurs when two or more distinct elements are mapped to the same index within the same row by the hash function. If a fingerprint collision occurs, the associated addresses must also match for a false positive to occur. Address mismatches can only occur after a fingerprint collision during the update. If two elements share the same fingerprint but have different addresses (this situation arises during the first occurrence of an element when at least one of the hash-indexed FACs already contains the same fingerprint, but at least one of the FACs are empty, indicating that the incoming element is new), this difference helps distinguish them during both update and query operations. This differentiation adds an additional layer of verification, significantly lowering the likelihood of false positives, ensuring that fingerprint collisions alone are insufficient to cause a false positive.

Probability of hash collisions in the sketch: Let d represent the number of rows in the sketch, m the number of FACs in each row, and n the total number of entries in the sketch. Each of the n elements is hashed into all d rows using d independent hash functions. Applying the Birthday problem [57] and assuming uniform hashing, the probability that no hash collision occurs in a single row is:

$$P_{\text{no hash collision (row)}} = \prod_{i=0}^{n-1} \left(1 - \frac{i}{m}\right)$$

For large m and relative to n, this simplifies using an exponential approximation:

$$P_{\text{no hash collision (row)}} \approx e^{-\frac{n(n-1)}{2m}}, \text{for } n \ll m$$

This approximation is valid under the assumption that the number of inserted elements is much smaller than the total number of available rows, ensuring the expression remains within [0,1].

The probability of at least one hash collision in a row is:

$$P_{\text{hash collision (row)}} = 1 - e^{-\frac{n(n-1)}{2m}} \tag{1}$$

Probability of Fingerprint Collision: Let the fingerprint size be k bits. The probability of no fingerprint collision after inserting n elements is:

$$P_{\text{no fingerprint collision}} = \prod_{i=0}^{n-1} \left(1 - \frac{i}{2^k}\right)$$

Using the birthday problem approximation for large values of 2^k:

$$P_{\text{no fingerprint collision}} \approx e^{-\frac{n(n-1)}{2^{k+1}}}$$

The probability of at least one fingerprint collision is:

$$P_{\text{fingerprint collision}} = 1 - e^{-\frac{n(n-1)}{2^{k+1}}} \tag{2}$$

Combined Collision Probability: In a single row, the combined probability of hash and fingerprint collisions is:

$$P_{\text{combined collision (row)}} = P_{\text{hash collision (row)}} \cdot P_{\text{fingerprint collision}}$$

Address Mismatch Probability in the FACs: Let the size of addresses be a bits. The probability of an address mismatch is:

$$P_{\text{address mismatch (row)}} = \frac{1}{2^a} \tag{3}$$

Probability of false positives: Across all d rows, assuming independent hash functions, the P_{fp} considers the combined probabilities of hash collisions, fingerprint collisions, and address mismatches:

$$P_{\text{fp}} = \left(P_{\text{combined collision (row)}} \cdot P_{\text{address mismatch (row)}}\right)^d$$

Substituting $P_{\text{combined collision (row)}}$ and $P_{\text{address mismatch (row)}}$:

$$P_{\text{fp}} = \left(\left(1 - e^{-\frac{n(n-1)}{2m}}\right) \cdot \left(1 - e^{-\frac{n(n-1)}{2^{k+1}}}\right) \cdot \frac{1}{2^a}\right)^d \tag{4}$$

The same equation applies to the error probability of the key-value store architecture using P-CAM.

This analysis focuses on false positives because P-CAM can be configured to prevent false negatives entirely. Sketch-based structures are inherently immune to false negatives under normal operation, and when the P-CAM operates below full capacity with overwrites disabled (cf. Section 3.5), every inserted key's fingerprint is retained in all its hashed FACs. Thus, assuming no deletions or insert-time evictions, valid entries will always be detected upon query, making the error probability inherently one-sided. In scenarios where overwriting is enabled under high load, false negatives may occur due to evictions; however, this trade-off is optional and application-dependent. Mitigation of false negatives in P-CAM for deterministic scenarios is discussed in Section 3.5.

3.4 Managing Accuracy with Confidence Vector

The number of matches found in the FACs across the memory arrays can be encoded as a bit array of size d, referred to as the confidence vector. This vector indicates the likelihood of false positives: a higher number of set bits indicates greater confidence in the query result. A user-defined threshold can be applied to the confidence vector to decide whether to accept the result. For example, when $d = 4$, the result may be accepted only if at least two bits in the confidence vector are set. In hybrid systems where values are stored in external memory, low-confidence results can be verified through an additional lookup in the external memory. Since the number of matches is computed during the query process, reporting the confidence vector incurs no additional computational overhead. A more detailed evaluation of the query confidence is presented in Section 4.3.4.

In accuracy-critical applications, such as IP routing, relying solely on probabilistic matching may be insufficient due to false positives. In such cases, the confidence vector serves as a reliability filter: high-confidence results (e.g., matching all d arrays) are accepted immediately, while low-confidence matches can trigger a verification step against slower but exact backing memory. This selective confirmation mechanism effectively mitigates the impact of false positives, allowing the system to maintain 100% functional accuracy. From a design perspective, when the total number of entries (n) is known a priori, parameters such as the fingerprint size (k), number of FACs (m), and the number of memory arrays (d) can be configured to bound the false-positive probability, as indicated by Equation 4. This parameter tuning enables P-CAM to achieve a desired confidence level while optimizing the trade-off between accuracy and resource utilization.

3.5 Deterministic Operation and Error Mitigation

A critical aspect of P-CAM's behavior occurs when the memory approaches full capacity (the load factor, $\alpha = \frac{n}{m}$, approaches 1.0). As described in Algorithm 1, if a new entry hashes to d locations without finding an empty or matching FAC, the update policy replaces it with the oldest entry, identified by the smallest address. This eviction policy is inherited from the SPArch architecture, where it is beneficial for tracking dynamic data streams like network heavy-hitters, as older, less relevant entries are naturally discarded. However, for general-purpose CAM applications, this

replacement may introduce false negatives. This occurs only when all d candidate locations are occupied by distinct, non-matching fingerprints, forcing the eviction of the oldest among them. The probability of a specific entry being evicted is therefore a function of the load factor, the rate of new insertions, and the entry's relative age. Importantly, eviction does not always cause false negatives, as older entries are often stored in multiple locations due to fewer initial collisions.

To support applications where false negatives are unacceptable, P-CAM offers a configurable mechanism. As stated in the update algorithm in Section 3.1, insertions can be denied by raising a *memory full* flag instead of overwriting an existing entry. By monitoring this flag, the system can operate deterministically up to the capacity. For applications with static or quasi-static datasets, such as converged IP routing tables, the P-CAM can be pre-loaded, and subsequent updates can be disabled, thereby completely eliminating the possibility of false negatives. This allows P-CAM to serve deterministic use cases, provided the dataset fits within the configured memory capacity.

4 Hardware Implementation and Performance Evaluation

We implemented P-CAM on FPGA using Verilog HDL and assess the performance in terms of latency, resource utilization, and accuracy. The hardware architecture block diagram of P-CAM is shown in Figure 5. The implementation and evaluation are conducted for two different input sizes, 96 and 384 bits. The inputs are handled using hash functions with dedicated input sizes of 96 and 384 bits. A single hash function instance generates the required hash bits, and the hash output is split to produce the hash values that map the input x to the sketch memory. The address select logic outputs the presence, address of the queried input, and the confidence vector. An up-counter is used as the address generator, with the maximum value set to n, the number of elements to be stored.

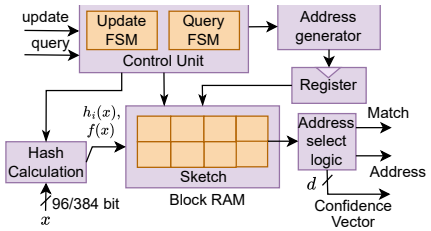

Fig. 5. Hardware architecture of P-CAM, with the sketch module implemented using dual-port Block RAMs and separate datapaths for update and query operations, enabling simultaneous lookups and insertions in real time.

4.1 Architectural Features

Simultaneous Update and Query: The hardware architecture of P-CAM supports simultaneous update and query operations, enhancing its applicability in high-throughput scenarios, particularly in applications such as real-time network flow monitoring. A generic probabilistic data structure typically uses a single datapath for update and query, which limits parallel operations and introduces data hazards during pipelining. In contrast, the P-CAM architecture consists of two separate hash function modules and two distinct datapaths for update and query, controlled by two finite state machines (FSM). This simultaneous operation is enabled by dual-port BRAMs used as the sketch memory, with dedicated ports for read and write operations. This design also improves the operating frequency compared to the generic single-datapath architecture by reducing the critical datapath.
Hash function: The reliance on hashing introduces potential security considerations. Specifically, adversarial inputs could be crafted to induce hash collisions, increasing the false positive rate or

corrupting stored entries. To mitigate such risks, P-CAM uses hash functions with strong avalanche properties to ensure uniform distribution of inputs. While we assume d independent hash functions, this is highly inefficient to implement in hardware. A common optimization is to use a more complex hashing scheme derived from two base hashes [23], but an even more hardware-efficient approach is to use a single, wide output hash function, and partitioning its large output to derive multiple, quasi-independent hashes. The effectiveness of this entire approach, however, hinges on the statistical quality of the chosen hash function.

The standard approach of using a hash function to compress a large key down to a small address (e.g., $\log_2(m)$ bits) is the fundamental source of hash collisions. P-CAM avoids this problem by using a hash function that does not compress the input and has strong avalanche properties. For this, we use Xoodoo-NC [48] as the hash function, which is a lightweight, reduced-round, non-cryptographic version of the Xoodoo permutation [12]. Xoodoo-NC with strong avalanche properties (entropy, weight, and dependence) and an output block size equal to or greater than the input block size acts as a collision-resistant mapping, not a collision-prone compression. This single hash function approach is not just a hardware optimization, but a more robust probabilistic design that replaces a lossy compression step with a high-fidelity permutation.

A 96-bit version of Xoodoo-NC is presented in [48], derived using only one *sheet* of the Xoodoo permutation state. A Xoodoo round consists only of shift, AND, and XOR operations, resulting in low logical depth and latency. The rounds are implemented using a fully unrolled architecture, completely implemented in combinational logic, allowing the entire hash to be computed in a single clock cycle. Xoodoo-NC can generate output widths of any multiple of 96 bits by increasing the number of rounds (beyond those required to satisfy avalanche properties) and concatenating the resulting outputs.

We extend the design to use the entire 384-bit Xoodoo permutation state for the 384-bit input version of the hash function. The 96-bit Xoodoo-NC hash function requires only 2.5 rounds of the permutation to satisfy the avalanche properties, whereas the 384-bit variant requires four rounds. In our implementation, we use 3 rounds for 96-bit Xoodoo-NC. The hash computation latency for the 384-bit input width is only 2.86 ns, and the hash function can process an input in every clock cycle. Due to its low logical depth and no memory usage, it introduces minimal resource and power overhead compared to commonly employed non-cryptographic hash functions such as FNV-1a and Murmur hash [14, 16]. The avalanche characteristics are represented using three metrics, namely avalanche dependence (D_{av}), avalanche weight ($\overline{w}_{av}$), and avalanche entropy (H_{av}), representing correlation, diffusion, and randomness, respectively. Table 3 shows the avalanche characteristics and hardware performance on a Kintex Ultrascale+ FPGA. For a detailed hardware analysis of Xoodoo-NC, description of avalanche metrics, and their computation, we refer to the original paper [48].

4.2 Evaluation Setup

The evaluation is conducted on two FPGA platforms: Kintex Ultrascale+ (xcku5p-ffvb676-2-e) and Virtex Ultrascale+ (xcvu9p-flga2104-2L-e). The Kintex UltraScale+ FPGA is used to evaluate smaller P-CAM configurations and allow a fair comparison with prior works. For larger table sizes, the

Table 3. Avlanche scores and hardware performance of 96-bit and 384-bit versions of Xoodoo-NC

Feature	Avalanche Values				Hardware Results (Kintex UltraScale+)			
	Rounds	D_{av}	$\overline{w}_{av}$	H_{av}	LUT	BRAM	Latency (ns)	f_{max}
Xoodoo-NC (384-bit)	4	384	191.83	383.99	1536	0	2.86	349 MHz
Xoodoo-NC (96-bit)	3	96	47.31	95.87	384	0	1.96	510 MhZ

Virtex UltraScale+ FPGA is employed due to the limited on-chip memory available on the Kintex device. All hardware results are obtained using Xilinx Vivado 2022.2.

Two datasets are used to evaluate the accuracy: a synthetic dataset and a real-world dataset. The synthetic dataset comprises unique key-value pairs, where keys are generated as 96-bit or 384-bit cryptographic values by truncating the 256-bit output of SHA-256 [32] or the 384-bit output of SHA-384 (to 24 or 96 hexadecimal characters) from 256-bit or 512-bit random inputs. The corresponding values are 32-bit integers arranged sequentially and then randomly shuffled to eliminate ordering bias. This dataset emulates real-world memory access patterns with uniformly distributed keys and non-contiguous values, offering high-entropy key-value associations.

The real-world dataset used is the RouteViews Prefix to AS mappings (pfx2as) [7], which maps IPv4/IPv6 prefixes to their originating Autonomous System (AS) numbers based on BGP routing data collected from globally distributed RouteViews [34] collectors. This mapping helps identify the network or AS responsible for routing traffic for specific IP ranges. Notably, AS numbers are not uniquely tied to individual prefixes, and a prefix may be associated with multiple ASes, making this dataset particularly suitable for evaluating system performance under challenging, multiclass real-world scenarios.

These datasets reflect common workloads in packet-processing applications such as IP routing, flow monitoring, and ACL lookups, where exact-match associative queries dominate.

4.3 Results

4.3.1 ***Accuracy Analysis.*** Accuracy is measured using the key-value storage architecture of P-CAM, since the presence of a matching key alone is insufficient to ensure correctness due to potential fingerprint collisions. In this setup, the stored value is retrieved and compared with the expected value to verify correctness. The accuracy is evaluated against the varying load factor (α) and fingerprint size. The load factor indicates how full the memory is, expressed as the ratio of the number of elements stored to the total memory size ($\frac{n}{m}$). Figure 6 plots the accuracy versus fingerprint size for various load factors using the synthetic dataset. As the load factor increases, the sketch becomes more saturated, leading to more hash collisions and reduced accuracy, as seen in Figure 6. When the load factor is below 0.5 and the fingerprint size is 8 bits, the accuracy remains $\approx$100% when $d > 2$. However, at higher load factors, lower d values experience a more pronounced decline in accuracy due to increased collisions in the FAC. Even so, the accuracy stays above 95% for d=2 at a load factor of 1.0. Furthermore, $d = 4$ maintains 100% accuracy across all load factors, while $d = 3$ achieves 99.6% at a load factor of 1.0.

While a 6-bit fingerprint suffices to reach peak accuracy at a load factor of 0.5, an 8-bit fingerprint is required at a load factor of 1.0. At lower load factors, the fingerprint size can be reduced to save memory without compromising accuracy. Figure 6 (a) shows that $d = 2$ has the lowest accuracy compared to $d = 3$ and $d = 4$. Regardless of the load factor, accuracy reaches $\approx$100% for $d = 4$ when the fingerprint size is eight bits or more. Similarly, $d = 3$ also reaches $\approx$100% accuracy at a load factor of 0.5 when the fingerprint size is eight bits or higher.

With the synthetic dataset, which contains only unique key-value pairs, the accuracy is slightly higher than with the real-world pfx2as dataset, as shown in Figure 6 (b). This difference arises because AS numbers in pfx2as are not uniquely tied to individual prefixes, meaning a key can map to multiple values and vice versa. Consequently, P-CAM may update the existing values of a specific key to its newest value based on the sequence of arrival. Nevertheless, the accuracy with the pfx2as dataset still approaches closer to the maximum observed value while using the synthetic dataset. We also evaluated the accuracy for the 384-bit-input implementation, which yields the same accuracy as the 96-bit version, as shown in Figure 6 (c).

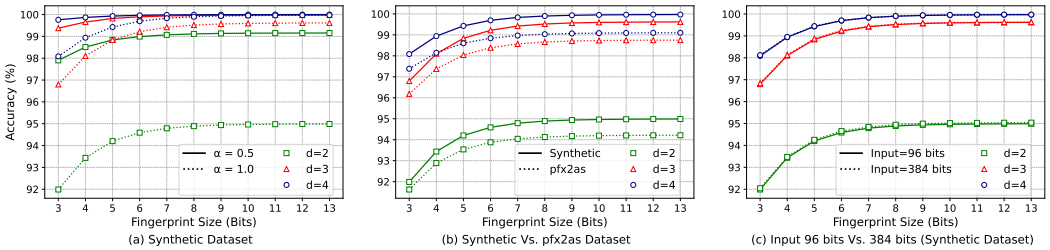

Fig. 6. P-CAM accuracy versus fingerprint size. (a) Accuracy on the synthetic dataset for load factors α=0.5 and α=1.0, with d=2,3,4. (b) Accuracy comparison between the synthetic and pfx2as datasets for d=2,3,4. (c) Accuracy comparison between input sizes 96 bit and 384 bits for d=2,3,4.

Accuracy-resource trade-off: In P-CAM, the accuracy is dependent on the parameters d, α, and f, while the hardware logic resource usage is independent of α. Under typical configurations (e.g., $d \geq 4$ and $f \geq 8$), accuracy remains stable across load factors (>99.9), as shown in Figure 6(a), and further increasing f yields negligible accuracy improvements while consuming more memory, as shown in Figure 7. As such, configuring the system with $d = 4$ and $f = 8$ offers a near-optimal trade-off between accuracy and resource usage. While smaller values of d and f exhibit modest accuracy degradation with increased load, reducing the load factor can restore high accuracy. For instance, with $d = 3$ and $f = 4$, P-CAM still achieves >99.9% accuracy for a load factor of 0.25. In contrast to deterministic CAMs, which guarantee perfect accuracy but incur substantial memory and logic overhead with increasing input width (cf. Section 4.3.3), P-CAM offers configurable accuracy with tunable memory use and width-independent storage. This tunability enables efficient deployment in high-throughput or resource-constrained environments without rigid accuracy–resource trade-offs.

4.3.2 ***Resource Utilization and Latency Analysis.*** The latency and resource usage on FPGA with regard to the increasing value of the memory array size m, which is also taken as the number of entries n or the table size, is shown in Figures 8, 9, and 10. The overall latency of the architecture is three clock cycles, corresponding to its three pipeline stages. As the table size grows, there is a gradual increase in the critical path delay. This increase is primarily attributed to routing delays, as shown in Figure 8. Notably, the logic delay component in the critical path remains nearly constant regardless of table size. The rise in critical path delay is therefore proportional to the increase in routing delay. Since critical path delay determines the maximum operating frequency and throughput, maintaining a constant logic delay enables the P-CAM to sustain high throughput even with larger input and table sizes. The timing analysis is based on automated placement and

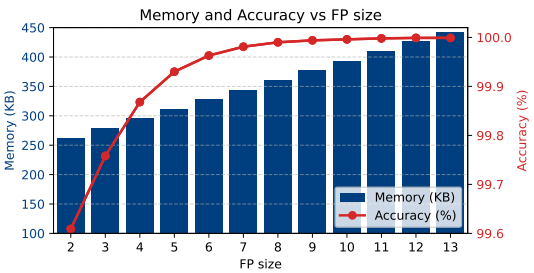

Fig. 7. Accuracy versus memory usage for P-CAM as a function of fingerprint size, evaluated at a constant load factor of 0.5.

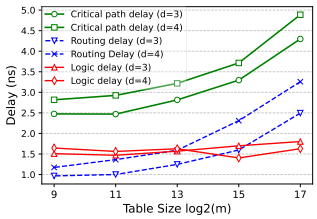

Fig. 8. Critical path delay of the FPGA implementation against increasing table size.

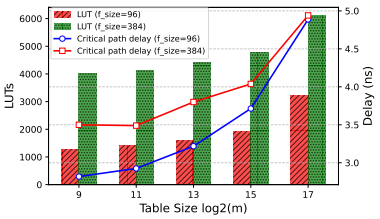

Fig. 9. Hardware resource usage and critical path delay of P-CAM for 96-bit and 384-bit inputs.

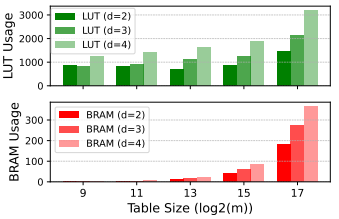

Fig. 10. LUT and BRAM usage against increasing table size.

routing, which accounts for minor variations in logic delay. Further optimization may be achieved through manual placement and routing.

The increase in LUT usage with increasing table sizes is minimal for P-CAM, as the computational overhead is primarily limited to hash computations. Although, as the table size becomes very large (e.g., when table size changes from 2^{15} to 2^{17}), there is a noticeable increase in LUT usage, as seen in Figures 9 and 10. This is mainly attributed to the increase in address width ($\log_2 m$) of the table, requiring more multiplexers, decode/control logic for data access, BRAM replication or cascading to meet higher memory demands, and parallelism or pipelining overheads. Nonetheless, such overheads are common to other memory architectures as well.

The usage of BRAM (36 Kb) also increases gradually with table size, as it stores both fingerprints and addresses. Yet, the memory requirement is significantly lower than that of conventional CAMs, especially for large key sizes, since CAMs must store full keys rather than compressed fingerprints. A more detailed discussion on the effect of input sizes and memory requirements is presented in the following section.

4.3.3 ***Scalability and Memory Efficiency.*** The memory requirement of P-CAM is fixed irrespective of the input size of entries. The memory footprint of P-CAM with a fixed fingerprint size of 8 bits and of traditional BCAM for varying input sizes are plotted in Figure 11. The larger the input size, the higher the memory efficiency of P-CAM, a significant contrast compared to CAM. In contrast, with smaller input sizes, the required bits per entry of each FAC would be substantial relative to the input size, particularly for larger tables. For instance, for an input size of 24 bits and a table size of 65,536, each FAC requires 24 bits, which is equal to the input size. With the increasing value of d, the overall memory requirement also grows.

For an input size of 96 bits and a fingerprint size of 8 bits, the size limit or the maximum table size of P-CAM is 65,536, as indicated by the red dot in Figure 11 that specifies the intersection point of CAM and P-CAM memory requirement. Beyond this point, the memory requirement of P-CAM exceeds that of CAM. The intersection point m_{max} defines the maximum table size at which P-CAM is more memory-efficient than CAM and is given by:

$$m_{\max} = 2^{\left(\frac{b}{d} - k\right)} \tag{5}$$

where b, d, and k represent the input bit-width, number of memory arrays, and fingerprint size, respectively.

Nevertheless, the memory usage of P-CAM can be reduced by either reducing the fingerprint size or limiting the table size. Importantly, even if P-CAM slightly exceeds CAM's memory usage beyond m_{max}, its computational complexity remains constant, and latency is unaffected, a key distinction from conventional CAM, which incurs increased delay and complexity with larger input or table

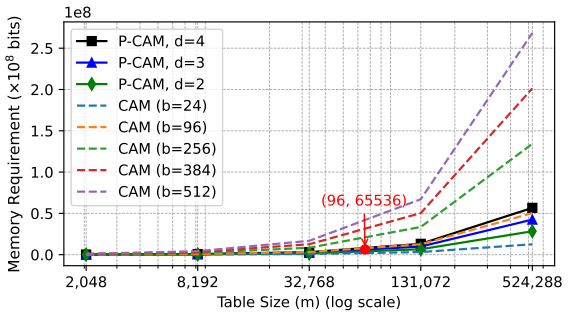

Fig. 11. Comparison of memory requirements w.r.t. input size for BCAM and P-CAM. For P-CAM, input size is not limited and the memory requirement is independent of input size. The red dot indicates the maximum table size for an input size of 96 bits and a fingerprint size of 8-bits.

sizes. As both input and table sizes grow, CAM becomes increasingly impractical due to its need to store full input keys. In contrast, P-CAM maintains a fixed memory requirement independent of input size. This fixed memory requirement is particularly advantageous for portability to other hardware platforms. By selecting a hash function capable of handling the largest expected input size and knowing the required table size, the P-CAM architecture remains unchanged, making it portable across various platforms, especially to ASIC. This advantage is highly relevant in applications such as networking, where CAMs are widely used in programmable switches.

4.3.4 ***Evaluation of Query Confidence.*** As described in Section 3.4, the d-bit confidence vector indicates the confidence of query operations, where the number of set bits indicates the number of matching FACs. A confidence value of $c = d$ corresponds to the highest confidence, while $c = 1$ indicates the lowest. The confidence of P-CAM queries is evaluated for load factors $\alpha = 0.5$ and 1.0, fingerprint sizes ranging from 3 to 10 bits, and the number of memory arrays $d = 4$. The resulting confidence percentages are plotted in Figure 12.

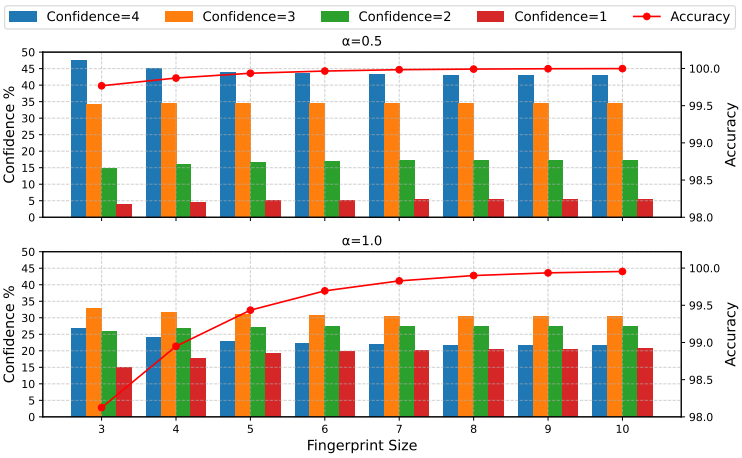

Fig. 12. P-CAM query confidence levels and overall accuracy versus fingerprint size. Both plots are shown for $d = 4$ at load factors $\alpha = 0.5$ and $\alpha = 1.0$.

Let conf(c) denote the confidence percentage of queries for which exactly c FACs match. At a lower load factor ($\alpha = 0.5$), query confidence is higher due to fewer FAC collisions. For example, with an 8-bit fingerprint, the combined percentage of conf(4) and conf(3) is approximately 78%, while conf(1) is only about 5.4%. As the load factor increases to $\alpha = 1.0$, FAC collisions become more frequent, reducing the overall query confidence, as reflected by the lower conf(4) and conf(3) values.

The impact of fingerprint size on confidence is also evident. Smaller fingerprints result in more collisions, thereby slightly lowering the confidence. However, as the fingerprint size increases, the confidence stabilizes. Specifically, the values of conf(4) and conf(3) decrease and saturate when the fingerprint size reaches 7 bits or more. Conversely, conf(2) and conf(1) exhibit the opposite trend as they are lower for smaller fingerprint sizes but increase and stabilize as the fingerprint size increases. The accuracy plot validates the confidence vector. It shows that accuracy is high and stable in low-collision scenarios (when $\alpha = 0.5$ or with large fingerprints), confirming that the confidence vector is a correct indicator of accuracy.

4.3.5 ***Comparison with State-of-the-Art Architectures.*** The comparison of P-CAM with state-of-the-art architectures is presented in Table 4. It is important to clarify the scope of this comparison. P-CAM, in its current form, performs probabilistic exact matching, whereas designs like DSCAM+ and Comp-TCAM provide deterministic exact and wildcard (ternary) matching. Therefore, this is not a direct feature-for-feature architectural benchmark. To our knowledge, no prior FPGA-based CAM-like architecture is built entirely on probabilistic data structures in the manner that P-CAM is, making a direct match-to-match comparison infeasible. Rather, we compare these architectures because they represent the state-of-the-art memory-optimized CAM designs implemented on FPGA, addressing the same problem domain, namely, memory efficiency, latency, and scalability for associative search, while deviating from conventional CAM architectures. While their functional capabilities differ, they serve as meaningful baselines to quantify the substantial resource and scalability benefits achievable in applications such as packet processing when a probabilistic engine like P-CAM is used in contexts where approximate exact-match functionality suffices. As we discuss in Section 4.4, extending P-CAM to support ternary operations is a direction for future work.

Solutions such as Comp-TCAM [20], SplitBucket [47], and DSCAM+ [55] primarily aim to reduce on-chip memory usage. These designs use LUTs or LUTRAMs as memory elements to reduce

Table 4. Comparison of P-CAM with State-of-the-Art FPGA-based CAM Architectures.

Architecture	FPGA	Table size	Input bitwidth	LUTs	BRAM (36Kb)	LUTRAM	Max. Freq. (MHz)	Latency (ns)	Searches/s (Million)	E_L	E_M	$E_{overall}$
Xilinx BCAM [1]	Virtex-US+	1024	150	2.9 K	12	0	333	–	–	52.07	0.36	0.46
HLSCAM BCAM [1]	Virtex-US+	1024	150	54.6 K	0	0	374	–	–	2.81	–	0.01
Comp-TCAM [20]	Kintex US	512	36	2.2 K	16	1536	585	–	–	7.83	0.03	0.05
SplitBucket [47]	Virtex-7	18,001	36	22 K	0	0	184	27.1	36.9	29.46	–	0.06
	Virtex-7	524,287	36	282.3 K	0	0	103	48.5	20.6	66.86	–	0.13
DSCAM+ [55]	Kintex US+	20,000	32	12.6 K	0.5	0	292	20.5	48.8	50.79	35.56	35.66
(BRAM cost=100)	Kintex US+	524,287	32	45.5 K	274	0	235	25.5	39.2	368.73	1.70	2.44
DSCAM+	Kintex US+	512	32	1.2 K	1	0	658	9.1	109.9	13.65	0.46	0.48
(BRAM cost=20)	Kintex US+	20,000	32	4.1 K	43	0	327	18.3	54.6	156.10	0.41	0.73
	Kintex US+	524,287	32	39.1 K	315.5	0	251	23.9	41.8	429.08	1.48	2.34
P-CAM (Our work)	Kintex US+	512	96	0.8 K	1.5	0	392	7.6	131.6	61.44	0.91	1.03
(d=3, α=1.0,	Kintex US+	20,000	96	1.2 K	40.5	0	303	9.9	101.0	1600.00	1.32	4.52
Accuracy=99.42)	Virtex US+	524,287	96	7.0 K	1152	0	100	30	33.3	7190.22	1.21	15.59
P-CAM (Our work)	Kintex US+	512	96	1.2 K	2	0	347	8.6	116.2	40.96	0.68	0.77
(d=4, α=1.0,	Kintex US+	20,000	96	1.9 K	54	0	270	11.1	90.1	1010.53	0.99	3.01
Accuracy=99.85)	Virtex US+	524,287	96	9.8 K	1536	0	101	29.6	33.8	5135.87	0.91	11.18
P-CAM (Our work)	Kintex US+	512	**384**	2.9 K	2	0	314	9.5	105.3	67.80	2.73	2.87
(d=4, α=1.0,	Kintex US+	20,000	**384**	3.8 K	54	0	256	11.7	85.5	2021.05	3.95	7.99
Accuracy=99.83)	Virtex US+	524,287	**384**	11.6 K	1536	0	100	29.9	33.4	17355.71	3.64	**38.35**

Max. Freq. refers to the maximum operating frequency of the synthesized hardware, which was used to compute the latency and throughput figures.

reliance on BRAMs on FPGAs. For instance, SplitBucket completely eliminates BRAM usage by relying solely on LUTs, while DSCAM+ adopts a hybrid strategy that balances the use of LUTs and BRAMs using a parameter called *BRAM cost*. Although these approaches reduce BRAM usage, they lead to a significant increase in computational complexity. It is also worth noting that the supported input widths in SplitBucket and DSCAM+ are relatively small, 36 bits and 32 bits, respectively. Scaling these architectures to larger input widths results in exponential growth in memory and logic complexity, making them impractical for high-capacity applications.

DSCAM+ is tailored for prefix-matching applications, but when it comes to generalized CAM applications, the Bitmap (BM) structure becomes a major bottleneck, as its memory requirements scale exponentially (2^s) with the subwordBM size (s). If the input size increases from 32 to 96 bits and s is scaled proportionally, the BM memory requirement may become infeasible for hardware implementation. Alternatively, reducing s may require increasing the subword sizes in later stages, leading to increased LUT usage and complexity in the MUX stage, ultimately causing higher latency and degraded clock frequency due to longer critical paths.

From Table 4, we observe that for larger table sizes, the BRAM requirement in P-CAM becomes higher than in DSCAM+. This outcome is to be expected for two main reasons. First, DSCAM+ support 32-bit entries while P-CAM supports 96 bits or more. Second, for an input size of 96 bits, P-CAM maintains a lower memory footprint than BCAM for table sizes up to 65,536 (see Section 4.3.3). Notably, P-CAM scales well with larger input sizes; for instance, at 384-bit input, the memory requirement of P-CAM will exceed that of BCAM only if the table size is larger than 2^{88}, demonstrating its excellent scalability.

Hardware Efficiency Metrics: To quantitatively compare P-CAM with the state-of-the-art architectures, we focus on normalized resource usage metrics that reflect architectural efficiency rather than absolute numbers. Specifically, we define two metrics: Memory Efficiency (E_M) and Logic Efficiency (E_L), which normalize the total information capacity (Table Size × Key Width) against consumed FPGA resources, where Table Size is the size of the memory array (m). For both metrics, higher values indicate more efficient use of hardware. These metrics also offer insights into scalability. Designs with higher (E_M) and (E_L) values can accommodate larger tables or wider keys with proportionally less resource growth. Moreover, reduced logic and memory footprint, reflected in these efficiency metrics, typically enables shallower pipelines and faster clocking, contributing to lower access delay in practice.

This normalization enables a fair comparison across designs with differing table sizes and bit-widths by distilling the performance down to a fundamental unit of stored bits per LUT or BRAM. While the comparison is not feature-to-feature, as P-CAM trades ternary capability for density, this normalization isolates the architectural overhead and quantifies the hardware cost of determinism. They highlight the resource penalty incurred by traditional architectures to support strict exact or ternary matching compared to P-CAM's more compact probabilistic approach.

Memory Efficiency: The memory efficiency measures how closely the actual BRAM usage matches the ideal storage requirement for the given table size (T) and bit-width (b):

$$E_{\mathrm{M}} = \frac{b \times T}{U_{\mathrm{BRAM}} \times S_{\mathrm{BRAM}}} \tag{6}$$

where U_{BRAM} is the actual number of BRAM blocks used, and S_{BRAM} is the storage capacity of one BRAM block (in bits).

Logic Efficiency: The logic resource efficiency quantifies the logic overhead expressed in terms of bits per LUT.

$$E_{\mathrm{L}} = \frac{b \times T}{U_{\mathrm{LUT}} + w \times U_{\mathrm{LUTRAM}}} \tag{7}$$

where U_{LUT} is the number of LUTs used, U_{LUTRAM} is the number of LUTRAM units used, and w ($0 < w < 1$) is the weighting factor for LUTRAMs, as LUTRAM is known to be more efficient than pure LUT-based storage logic. In Xilinx FPGAs, a reasonable assumption is that LUTRAM is ≈4× to ≈16× more efficient than LUTs. Thus, a good weighting factor is w=0.1.

Both metrics allow fair comparison of implementations across different parameter configurations by normalizing to the theoretical ideal requirements.

Overall Hardware Efficiency: To provide a single figure of merit for comparing different designs, we define an overall hardware efficiency metric that combines both logic and memory efficiency. This metric reflects the fact that FPGA performance depends on the balance between available logic resources (LUTs and LUTRAMs) and on-chip memory (BRAMs).

A straightforward approach is to compute a weighted sum:

$$E_{\mathrm{overall}} = w_L \times E_L + E_M \tag{8}$$

where w_L is a weighting factor that reflects the relative importance or scarcity of logic resources compared to BRAMs on the target device.

A single 6-input LUT can be configured as a 64-bit RAM [2], which means approximately 600–700 LUTs, including decoding logic, are needed to replicate the storage capacity of one BRAM. However, BRAMs are coarse-grained, i.e., if they are not fully utilized, some capacity is wasted. By contrast, LUTs are fine-grained and well-suited for implementing small buffers. In CAMs, caches, and other memory-intensive designs, it is preferable to map large storage blocks to BRAMs to maximize area efficiency. Taking these factors into account, we set the LUT weight factor w_L to $\frac{1}{500}$, which is a conservative and reasonable estimate, while also considering the FPGA resource availability. However, in designs where LUTs are critical resources, the fact that approximately 500 LUTs are needed to replicate the capacity of a single BRAM underscores the high cost of using LUTs for storage. Consequently, LUT usage should be weighted more heavily than BRAM usage when evaluating resource efficiency. Using the weighted sum provides greater interpretability and allows prioritization of resources according to the specific FPGA architecture or application requirements. A higher E_{overall} indicates that the design achieves greater useful capacity per unit of total hardware cost.

The quantitative results presented in Table 4 demonstrate the superior scalability and resource efficiency of P-CAM compared to existing solutions. This advantage is especially pronounced when processing wide inputs, a key limitation of prior architectures. For example, with 384-bit entries, P-CAM achieves an overall efficiency (E_{overall}) of ≈38, which is an order of magnitude greater than the scores reported for other state-of-the-art architectures, operating on much narrower input widths. This substantial gain is primarily due to P-CAM's exceptional logic efficiency (E_L), exceeding 17,000 bits per LUT-equivalent, underscoring its highly optimized architecture. In comparison, designs such as Comp-TCAM and DSCAM+ yield much lower logic efficiencies, reflecting their heavy dependence on logic resources and limited scalability. This becomes increasingly problematic when larger storage capacities are needed, where BRAMs offer far greater efficiency. Similarly, architectures like SplitBucket eliminate BRAM entirely, but at the expense of significantly increased logic consumption, resulting in poor overall efficiency. Collectively, these results show that P-CAM's balanced use of logic and memory resources enables a more practical and scalable solution for modern, high-capacity CAM applications.

4.4 Future Work

The proposed architecture currently supports (probabilistic) exact matching; however, many practical applications, such as prefix matching, require masked or ternary lookup capabilities. To address these needs, masking techniques can be integrated to emulate the functionality of TCAMs, enabling flexible prefix and exact pattern matching. Supporting ternary logic would likely require modifications to the FAC matching mechanism, such as introducing mask-aware fingerprint encoding or supporting multi-mode comparisons. The fundamental challenge is that the hash functions rely on the avalanche effect, changing a single input bit results in a completely different hash. This makes standard hashing incompatible with *don't care* bits. To overcome this, two potential approaches could be explored:

- Prefix-Partitioned P-CAM (for longest-prefix matching): Deploying multiple P-CAM instances in parallel, where each instance handles a fixed prefix length (e.g., /16, /24, /32), similar to Bloom filter chaining in longest-prefix matching designs.
- Key Decomposition (for masked matching in ACLs): Dividing the input key into smaller sub-fields (e.g., 16-bit chunks), performing exact probabilistic matches independently on each field, and aggregating results via logical operations to resolve wildcard semantics.

Future work should explore efficient hardware implementations of such masking schemes while minimizing area and power overhead.

For use cases involving noisy pattern matching or DNA sequence analysis, approximate lookup methods based on Hamming distance provide an effective solution to identify mutations. Conventional CAMs and TCAMs, with their strict exact or masked matching, are not well-suited for such tolerant comparisons. Furthermore, approximate lookups are also applicable to classification tasks, particularly when datasets are small and lack sufficient diversity to leverage machine learning models effectively. In this context, P-CAM with Hamming distance-based approximate lookup capability could replicate the behavior of classical machine learning models, such as k-nearest neighbor (kNN) classifiers, offering similar functionality while demanding significantly less memory.

Future work will explore the design of efficient hardware mechanisms to support approximate matching with configurable distance thresholds, enabling tunable tolerance for error-resilient applications. Moreover, future research could investigate hybrid architectures that combine exact, ternary, and approximate matching modes that may further enhance adaptability to diverse workloads. Additionally, integrating error-resilient encoding schemes could further enhance robustness in noisy environments. Benchmarking these capabilities against traditional classifiers in terms of accuracy, latency, and energy efficiency is another promising direction.

Additionally, extending the P-CAM framework to support multi-dimensional and hierarchical data structures may unlock new application domains, including real-time signal processing, bioinformatics, and security systems, where high-speed and flexible pattern matching is critical. Another exciting direction is to investigate how P-CAM could accelerate finite automata, such as deterministic (DFA) or non-deterministic finite automata (NFA), which are fundamental to deep packet inspection and regular expression matching. Traditional FPGA-based DFAs suffer from the state explosion problem, often requiring slow off-chip DRAM to store massive transition tables, which introduces latency bottlenecks. P-CAM, with its compact and parallel associative memory fabric, could serve as an efficient on-chip state transition memory. By mapping FA states and transitions onto P-CAM's BRAM-based structure, it may be possible to construct high-throughput, space-efficient automata processors tailored for streaming workloads. However, it also introduces the novel challenge of probabilistic transitions, where a false positive lookup could cause the automaton to enter an incorrect state. Future work will investigate the robustness of such systems and explore mitigation techniques to ensure correctness under probabilistic behavior.

5 Conclusion

This paper presented P-CAM, a novel associative memory architecture that addresses the scalability limitations of traditional CAMs by making a conscious trade-off of giving up deterministic, single-cycle lookups in favour of a highly scalable, memory-efficient probabilistic approach. Unlike conventional CAMs, which suffer from scalability limitations with increasing input width, P-CAM decouples memory requirements from key size through hashing-based fingerprinting. It supports dynamic updates, multi-key associations, and parallel insert/query operations, making it well-suited for high-throughput, low-latency workloads. Our hardware evaluation on Ultrascale+ FPGAs shows that P-CAM offers excellent resource efficiency and maintains low, constant latency even for large tables with wide input keys. For 384-bit input and a table size of 2^{19}, P-CAM achieves 15× improvement in overall hardware efficiency ($E_{overall}$) compared to the best-known deterministic design, which supports only 32-bit inputs. It further achieves 47× higher logic efficiency and 2× better memory efficiency. While the probabilistic nature introduces a small, controllable false-positive rate, we mitigate this through a confidence vector mechanism. Importantly, P-CAM sustains nanosecond-scale lookup latency even at high load factors, without increasing the LUT complexity. P-CAM thus establishes itself as a robust, scalable alternative for data-intensive domains like high-speed network monitoring and bioinformatics, where conventional exact-match memories are infeasible.

Acknowledgments

This work is supported by the ESCALATE project, funded by FWO under grant No. G0E0719N and SNSF under grant No. 200021L_182005, and by Cybersecurity Research Flanders under grant No. VR20192203.

References

[1] Mostafa Abbasmollaei, Tarek Ould-Bachir, and Yvon Savaria. 2025. HLSCAM: Fine-Tuned HLS-Based Content Addressable Memory Implementation for Packet Processing on FPGA. *Electronics* 14, 9 (2025), 1765. doi:10.3390/electronics14091765

[2] AMD. 2025. UltraScale Architecture Configurable Logic Block User Guide (UG574). https://docs.amd.com/r/en-US/ug574-ultrascale-clb?tocId=2DOfkt0abl3oJAYunYLu0A. Accessed: 2025.

[3] Nagender Bandi, Divyakant Agrawal, and Amr El Abbadi. 2007. Fast algorithms for heavy distinct hitters using associative memories. In *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS'07)*. 6. doi:10.1109/ICDCS.2007.110

[4] Masanori Bando and H Jonathan Chao. 2010. Flashtrie: Hash-based prefix-compressed trie for IP route lookup beyond 100Gbps. In *Proceedings of the 29th IEEE International Conference on Computer Communications (INFOCOM '10)*. 821–829. doi:10.1109/INFCOM.2010.5462142

[5] Burton H Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (1970), 422–426. doi:10.1145/362686.362692

[6] Hayoung Byun, Qingling Li, and Hyesook Lim. 2019. Vectored-Bloom filter for IP address lookup: Algorithm and hardware architectures. *Applied Sciences* 9, 21 (2019), 4621. doi:10.3390/app9214621

[7] CAIDA. 2024. RouteViews Prefix to AS mappings. https://catalog.caida.org/dataset/routeviews_prefix2as. Accessed: 2025.

[8] Dibei Chen, Zhaoshi Li, Tianzhu Xiong, Zhiwei Liu, Jun Yang, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2020. CATCAM: Constant-time alteration ternary CAM with scalable in-memory architecture. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. 342–355. doi:10.1109/MICRO50266.2020.00038

[9] Yu-Chieh Cheng and Pi-Chung Wang. 2015. Scalable multi-match packet classification using TCAM and SRAM. *IEEE Trans. Comput.* 65, 7 (2015), 2257–2269. doi:10.1109/TC.2015.2470242

[10] Avril Coghlan, Dónall A Mac Dónaill, and Nigel H Buttimore. 2001. Representation of amino acids as five-bit or three-bit patterns for filtering protein databases. *Bioinformatics* 17, 8 (2001), 676–685. doi:10.1093/bioinformatics/17.8.676

[11] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75. doi:10.1016/j.jalgor.2003.12.001

[12] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. 2018. The design of Xoodoo and Xoofff. *IACR Transactions on Symmetric Cryptology* 2018, 4 (2018), 1–38. doi:10.13154/tosc.v2018.i4.1-38

[13] Cristian Estan and George Varghese. 2002. New directions in traffic measurement and accounting. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '02)*. 323–336. doi:10.1145/633025.633056

[14] César Estébanez, Yago Saez, Gustavo Recio, and Pedro Isasi. 2014. Performance of the most common non-cryptographic hash functions. *Software: Practice and Experience* 44, 6 (2014), 681–698. doi:10.1002/spe.2179

[15] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*. 75–88. doi:10.1145/2674005.2674994

[16] Glenn Fowler, Landon Curt Noll, Kiem-Phong Vo, and Donald Eastlake. 2011. The FNV Non-Cryptographic Hash Algorithm. IETF Internet-Draft. https://datatracker.ietf.org/doc/html/draft-eastlake-fnv-03 Accessed: 2024.

[17] Esteban Garzón, Roman Golman, Zuher Jahshan, Robert Hanhan, Natan Vinshtok-Melnik, Marco Lanuzza, Adam Teman, and Leonid Yavits. 2022. Hamming distance tolerant content-addressable memory (HD-CAM) for DNA classification. *IEEE Access* 10 (2022), 28080–28093. doi:10.1109/ACCESS.2022.3158305

[18] Esteban Garzón, Robert Hanhan, Marco Lanuzza, Adam Teman, and Leonid Yavits. 2024. FASTA: Revisiting fully associative memories in computer microarchitecture. *IEEE Access* 12 (2024), 13923–13943. doi:10.1109/ACCESS.2024.3355961

[19] Catherine E Graves, Wen Ma, Xia Sheng, Brent Buchanan, Le Zheng, Si-Ty Lam, Xuema Li, Sai Rahul Chalamalasetti, Lennie Kiyama, Martin Foltin, et al. 2018. Regular expression matching with memristor TCAMs for network security. In *Proceedings of the 14th IEEE/ACM International Symposium on Nanoscale Architectures*. 65–71. doi:10.1145/3232195.3232201

[20] Muhammad Irfan, Hasan Erdem Yantır, Zahid Ullah, and Ray CC Cheung. 2021. Comp-TCAM: An adaptable composite Ternary content-addressable Memory on FPGAs. *IEEE Embedded Systems Letters* 14, 2 (2021), 63–66. doi:10.1109/LES.2021.3124747

[21] Weirong Jiang. 2013. Scalable ternary content addressable memory implementation using FPGAs. In *Proceedings of the IEEE/ACM International Symposium on Architectures for Networking and Communications Systems (ANCS '13)*. IEEE, 71–82. doi:10.1109/ANCS.2013.6665177

[22] Roman Kaplan, Leonid Yavits, Ran Ginosar, and Uri Weiser. 2017. A resistive CAM processing-in-storage architecture for DNA sequence alignment. *IEEE Micro* 37, 4 (2017), 20–28. doi:10.1109/MM.2017.3211121

[23] Adam Kirsch and Michael Mitzenmacher. 2006. Less hashing, same performance: Building a better bloom filter. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA '06)*. Springer, 456–467. doi:10.1007/11841036_42

[24] Dmitry Krotov and John Hopfield. 2020. Large associative memory problem in neurobiology and machine learning. *arXiv preprint arXiv:2008.06996* (2020). https://arxiv.org/abs/2008.06996

[25] Dmitry Krotov and John J Hopfield. 2016. Dense associative memory for pattern recognition. *Proceedings of the 29th International Conference on Neural Information Processing Systems (NIPS '16)* (2016). https://proceedings.neurips.cc/paper_files/paper/2016/hash/eaae339c4d89fc102edd9dbdb6a28915-Abstract.html

[26] Xiaozhou Li, David G Andersen, Michael Kaminsky, and Michael J Freedman. 2014. Algorithmic improvements for fast concurrent cuckoo hashing. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. 1–14. doi:10.1145/2592798.2592820

[27] Satendra Kumar Maurya and Lawrence T Clark. 2010. A dynamic longest prefix matching content addressable memory for IP routing. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 19, 6 (2010), 963–972. doi:10.1109/TVLSI.2010.2042826

[28] Chad R Meiners, Alex X Liu, and Eric Torng. 2010. *Hardware based packet classification for high speed internet routers*. Springer Science & Business Media.

[29] Chad R Meiners, Jignesh Patel, Eric Norige, Eric Torng, and Alex X Liu. 2010. Fast regular expression matching using small TCAMs for network intrusion detection and prevention systems. In *Proceedings of the 19th USENIX Security Symposium (USENIX Security '10)*. https://dl.acm.org/doi/10.5555/1929820.1929831

[30] Khader Mohammad, Aziz Qaroush, Mahdi Washha, and Baker Mohammad. 2017. Low-power content addressable memory (CAM) array for mobile devices. *Microelectron. J.* 67 (2017), 10–18. doi:10.1016/j.mejo.2017.07.001

[31] Ju Hyoung Mun and Hyesook Lim. 2015. New approach for efficient IP address lookup using a Bloom filter in trie-based algorithms. *IEEE Trans. Comput.* 65, 5 (2015), 1558–1565. doi:10.1109/TC.2015.2444850

[32] National Institute of Standards and Technology (NIST). 2015. *FIPS PUB 180-4: Secure Hash Standard (SHS)*. Technical Report FIPS PUB 180-4. National Institute of Standards and Technology, Gaithersburg, MD. doi:10.6028/NIST.FIPS.180-4

[33] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '12)*. IEEE, 3108–3115. doi:10.1109/CVPR.2012.6248043

[34] University of Oregon. [n. d.]. University of Oregon Route Views Project. http://www.routeviews.org/routeviews/. Accessed: 2025.
[35] Atsushi Ooka, Shingo Atat, Kazunari Inoue, and Masayuki Murata. 2014. Design of a high-speed content-centric-networking router using content addressable memory. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM Workshops '14)*. IEEE, 458–463. doi:10.1109/INFCOMW.2014.6849275
[36] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144. doi:10.1016/j.jalgor.2003.12.002
[37] Kostas Pagiamtzis and Ali Sheikholeslami. 2004. A low-power content-addressable memory (CAM) using pipelined hierarchical search scheme. *IEEE Journal of Solid-State Circuits* 39, 9 (2004), 1512–1519. doi:10.1109/JSSC.2004.831433
[38] Kostas Pagiamtzis and Ali Sheikholeslami. 2006. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits* 41, 3 (2006), 712–727. doi:10.1109/JSSC.2005.864128
[39] Salvatore Pontarelli and Marco Ottavi. 2012. Error detection and correction in content addressable memories by using bloom filters. *IEEE Trans. Comput.* 62, 6 (2012), 1111–1126. doi:10.1109/TC.2012.56
[40] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, et al. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 13–24. doi:10.1145/2678373.2665678
[41] VC Ravikumar, Rabi N Mahapatra, and Laxmi Narayan Bhuyan. 2005. EaseCAM: An energy and storage efficient TCAM-based router architecture for IP lookup. *IEEE Trans. Comput.* 54, 5 (2005), 521–533. doi:10.1109/TC.2005.78
[42] Renesas. [n. d.]. TCAM IP Solution. https://www.renesas.com/en/products/asic-ip/intellectual-property-ip/memory-ips/tcam-ip-solution. Accessed: July 2025.
[43] Victor Rios and George Varghese. 2022. MashUp: Scaling TCAM-Based IP Lookup to Larger Databases by Tiling Trees. *arXiv:2204.09813* (apr 2022). https://arxiv.org/abs/2204.09813
[44] Zareen Sadiq and Shehzad Hasan. 2021. MemCAM: A hybrid memristor-CMOS CAM cell for on-chip caches. *IEEE Access* 9 (2021), 21296–21305. doi:10.1109/ACCESS.2021.3055509
[45] Saad Saleh, Anouk S Goossens, Sunny Shu, Tamalika Banerjee, and Boris Koldehofe. 2024. Analog In-Network Computing through Memristor-based Match-Compute Processing. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '24)*. IEEE, 2518–2527. doi:10.1109/INFOCOM52122.2024.10621228
[46] Rama Sangireddy, Natsuhiko Futamura, Srinivas Aluru, and Arun K Somani. 2005. Scalable, memory efficient, high-speed IP lookup algorithms. *IEEE/ACM Trans. Netw.* 13, 4 (2005), 802–812. doi:10.1109/TNET.2005.852878
[47] Ideh Sarbishei, Shervin Vakili, JM Pierre Langlois, and Yvon Savaria. 2017. Scalable memory-less architecture for string matching with FPGAs. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '17)*. IEEE, 1–4. doi:10.1109/ISCAS.2017.8050818
[48] Arish Sateesan, Jo Vliegen, Joan Daemen, and Nele Mentens. 2022. Hardware-oriented optimization of Bloom filter algorithms and architectures for ultra-high-speed lookups in network applications. *Microprocessors and Microsystems* 93 (2022), 104619. doi:10.1016/j.micpro.2022.104619
[49] Arish Sateesan, Jo Vliegen, and Nele Mentens. 2022. An Analysis of the Hardware-Friendliness of AMQ Data Structures for Network Security. In *Proceedings of the 12th International Conference on Security, Privacy, and Applied Cryptography Engineering (SPACE '22) (Lecture Notes in Computer Science, Vol. 13785)*. Springer, Cham, Switzerland, 287–313. doi:10.1007/978-3-031-22829-2_16
[50] Arish Sateesan, Jo Vliegen, Simon Scherrer, Hsu-Chun Hsiao, Adrian Perrig, and Nele Mentens. 2024. SPArch: a hardware-oriented sketch-based architecture for high-speed network flow measurements. *ACM Trans. Priv. Secur.* 27, 4 (2024), 1–34. doi:10.1145/3687477
[51] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. 2005. Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*. ACM, New York, NY, USA, 181–192. doi:10.1145/1080091.1080114
[52] Tolga Soyata and John Liobe. 2012. pbCAM: probabilistically-banked content addressable memory. In *Proceedings of the IEEE International System-on-Chip Conference (SOCC '12)*. 27–32. doi:10.1109/SOCC.2012.6398372
[53] David E Taylor. 2005. Survey and taxonomy of packet classification techniques. *ACM Comput. Surv.* 37, 3 (2005), 238–275. doi:10.1145/1108956.1108958
[54] David E. Taylor and Edward W. Spitznagel. 2005. *On Using Content Addressable Memory for Packet Classification.* Technical Report WUCSE-2005-9. Washington University in St. Louis, Department of Computer Science and Engineering, St. Louis, MO. https://openscholarship.wustl.edu/cse_research/979
[55] Shervin Vakili. 2023. DSCAM+: Latency-Guaranteed FPGA-Based Content Addressable Memory for SDN-Enabled Forwarding Plane. In *Proceedings of the IEEE International Conference on High Performance Computing and Communications, Data Science and Systems, Smart City, and Dependability in Sensor, Cloud, and Big Data Systems and Applications (HPCC/DSS/SmartCity/DependSys '23)*. IEEE, 311–318. doi:10.1109/HPCC-DSS-SmartCity-DependSys60770.2023.00050

[56] Jagath Weerasinghe, Francois Abel, Christoph Hagleitner, and Andreas Herkersdorf. 2015. Enabling FPGAs in hyperscale data centers. In *Proceedings of the 2015 IEEE International Conference on Ubiquitous Intelligence and Computing, Autonomic and Trusted Computing, and Scalable Computing and Communications (UIC-ATC-ScalCom '15)*. IEEE, 1078–1086. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.199

[57] Eric W. Weisstein. 2002. Birthday Problem. https://mathworld.wolfram.com/BirthdayProblem.html. Accessed: 2024.

[58] Chengcheng Xu, Shuhui Chen, Jinshu Su, Siu-Ming Yiu, and Lucas CK Hui. 2016. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Commun. Surv. Tutor.* 18, 4 (2016), 2991–3029. doi:10.1109/COMST.2016.2566669