NP-CAM: Efficient and Scalable DNA Classification using a NoC-Partitioned CAM Architecture

HPCA 2026 Submission **#NaN** – Confidential Draft – Do NOT Distribute!!

***Abstract*—The rapid advancement in genomic sequencing technologies has resulted in an explosion of data, creating substantial computational bottlenecks in DNA analysis workloads. Applications such as DNA classification are particularly impacted due to their reliance on intensive, large-scale pattern matching. Existing hardware accelerator and software solutions are increasingly unable to manage the scale and energy demands of these datasets, highlighting the need for architectures that can perform faster and more efficient pattern matching. To address these challenges, we propose NP-CAM: a data-optimized, CAM-based accelerator designed for parallel and energy-efficient DNA classification. NP-CAM harnesses a network-on-chip to implement a novel optimized indexing and CAM partitioning scheme that reduces the active search space, allowing significant scalability. We demonstrate results for NP-CAM on commodity 10T binary CAM cell designs. Our experimental evaluations show that NP-CAM achieves a simultaneous $65\times$ improvement in sequence throughput and an over $173\times$ improvement in energy efficiency over state-of-the-art hardware solutions on existing small viral workloads. We go on to demonstrate feasibility for larger bacterial and fungal workloads, enabling scalable DNA classification in the era of large-scale genomic data.**

I. Introduction

Explosive genomic data growth has outpaced our ability to process it. Next-generation sequencing (NGS) technologies, advancing faster than Moore's Law, have fueled this surge [1]. As of October 2024, GenBank contained over 34 trillion nucleotides [2], up from just 5 trillion in 2018—a 40% annual increase. From 2006 to 2023, the cost to sequence a human genome dropped from over $10 million to just $200 [3], [4]. This data boom motivates population-scale sequencing, but many end applications are now *computationally* bottlenecked: NGS devices generate 4–200 Gbp/hr, while software read mappers process only 0.2–1.7 Gbp/hr [5].

Bioinformatics workloads often involve large-scale pattern matching between reference and query sequences. Applications include alignment [6], [7], genome assembly [8], and DNA classification [9], [10]. Pathogen detection—a critical bioinformatic task—demands fast, energy-efficient DNA classification under tight resource constraints [11]. Most DNA classifiers count and compare matching substrings (k-mers) between query and reference sequences (Fig. 1). This reliance on k-mer matching makes classification an ideal candidate for hardware acceleration [12]–[17]. In particular, content addressable memories (CAMs) support constant-time parallel search across all entries, making them well-suited for high-throughput k-mer matching. However, existing CAM-based accelerators [12]–[14] still struggle with the power demands as the size and number of sequences increase (Fig. 4). For this reason, CAM-based

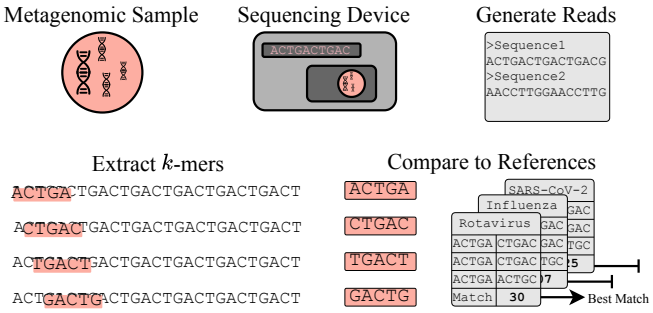


Fig. 1. Online DNA classification pipeline. A metagenomic sample is sequenced by an NGS device. From the reads, k-mers are extracted and compared against known reference genomes. Sequences are classified based on match counts.

solutions have been limited to pathogens with the smallest genomes: viruses [12]–[14]. For detection of pathogens with larger genomes, like bacteria or fungi, software solutions [9], [10] are relied upon.

We present NP-CAM, a scalable, parallel, and energy-efficient CAM-based accelerator for DNA classification. Unlike previous works [12]–[14], NP-CAM *partitions* the reference k-mer space into many independent on-chip search regions using an indexing step. Partitioning enables: **1) energy-efficient search** by activating only a single subspace per query, and **2) high throughput** via parallel searches across disjoint partitions. Our primary contributions are:

- We introduce *hierarchical adaptive partitioning (HAP)*, a data-driven scheme for splitting search space, and implement it using a butterfly network topology.
- We design NP-CAM as an end-to-end, commercially-plausible pathogen detection system, and evaluate trade-offs in energy, area, and throughput across partitioning granularity and workloads.
- We demonstrate up to $65\times$ higher throughput and $173\times$ better energy efficiency over best prior CAM accelerators on viral workloads, and extend classification to larger bacterial and fungal genomes previously infeasible for on-chip acceleration.

NP-CAM enables scalable, high-performance DNA classification across large reference datasets, making it well-suited for emerging genomics workloads.

II. Background

A. Metagenomics and DNA Classification

Metagenomics aims to characterize the biological makeup of an environment by analyzing DNA sequences. This has

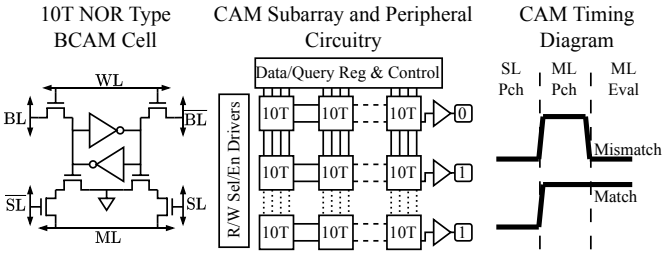


Fig. 2. Content addressable memory cell design for conventional 10T binary CAM with NOR-type matchline. Array architecture along with match and mismatch timing diagram.

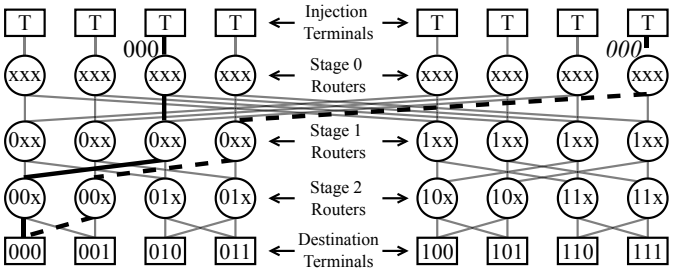


Fig. 3. A 2-ary 3-fly butterfly network. Example directs incoming packets from injection terminals to 000 destination terminal by fixing one digit per stage (MSB to LSB here).

broad applications in pathogen detection, disease diagnosis, and phylogenetic analysis. Detecting pathogens quickly and efficiently is especially important for tracking infectious disease outbreaks, particularly in resource-limited settings. While advances in next-generation sequencing (NGS) technologies (e.g., Illumina [18], PacBio [19], ONT [20]) have dramatically increased sequencing throughput, analyzing this data remains a major computational bottleneck—particularly for large-scale DNA classification. Early DNA classification relied on alignment-based methods [21], [22], which are accurate but computationally intensive. Probabilistic models offer high sensitivity but are often too slow for large-scale workloads [23], [24]. In contrast, k-mer-based methods achieve comparable accuracy with significantly lower compute cost [9], [10]. These algorithms compare query k-mers against reference k-mers, tallying match scores for each class and classifying accordingly.

Platforms like the handheld Oxford Nanopore's MinION [25] enable real-time sequencing in portable and resource-constrained contexts such as remote wastewater surveillance [26], pathogen tracking [27], [28], point-of-care diagnostics [29], [30], field epidemic surveillance [31], and embedded environmental metagenomic monitoring [32]. In these scenarios, the CPU-based Kraken2 is insufficient: it processes only 1.3 Gk-mer/hr while demanding 80 GB of RAM and server-level compute [10], [33]. Real-world applications necessitate high-throughput low-power classification pipelines that Kraken2 cannot support. Due to the need for an alternative to Kraken2, hardware accelerators for the memory-intensive k-mer-based classifiers have been developed [12]–[17], [34].

Algorithm 1: Software algorithm for DNA classification with k-mer matches (based on [9], [10]).

```
//Offline Preprocessing
foreach genome Ref in Database do
    foreach k-mer r in Ref do
        hash_table[Ref][r]++;
//Online Classification
foreach read Query in Sample do
    foreach k-mer q in Query do
        foreach genome Ref in Database do
            scores[Query][Ref]+= hash_table[Ref][q];
return C(scores);          //C is a classifier
```

B. Content Addressable Memory (CAM)

Content addressable memories (CAMs) store data in arrays of cells that perform bitwise comparisons in parallel. CAMs return the locations of matching entries in a single-cycle, $\mathcal{O}(1)$ operation by simultaneously activating all cells [35]–[38]. This in-memory search capability makes CAMs attractive for high-throughput matching tasks. Figure 2 illustrates the key components of a CAM array, including word lines, bitlines, cells, and peripheral circuitry. In the CAM cell of Fig. 2, if the search line (SL) input does not match the stored data bit, a pair of series-connected NMOS transistors is activated, creating a pull-down path that discharges the match line (ML). At the array level, a row is considered a match only if all CAM cells in that row maintain a high ML. Any single mismatch triggers a discharge, pulling the row ML low.

Various cell-level optimizations have reduced CAM energy consumption [12]–[14], [39], [40]. However, these approaches only improve constant factors and do not address fundamental scalability or support multi-query parallelism. CAMs are typically implemented as large monolithic arrays. Although physically tiled to mitigate line capacitance and delay, they still operate as a logical whole. Each search activates all cells, causing power consumption to scale with the total array size. This architecture becomes a bottleneck for large-scale workloads like DNA classification, where reference sets can span millions of k-mers. This has held back designs like DASH-CAM and EDAM, which applied CAMs to DNA classification due to their fast search capability, to small viral workloads [12], [14], [41]. Full array activation becomes increasingly energy-prohibitive for larger classification workloads. For example, classifying across 10 fungal genomes can require 64MB of CAM storage and over 200W of search power [12]. To overcome these limitations, we propose hierarchical adaptive partitioning (HAP), an algorithm-hardware co-design that partitions the CAM into smaller, independently searchable sections. By leveraging patterns in the reference data, HAP enables scalable CAM usage orthogonal to cell-level optimizations, allowing concurrent and energy-efficient matching across disjoint partitions.

C. Butterfly Networks

A m-ary n-fly butterfly network is a multistage interconnection topology comprising n stages of routers, each with a radix (degree) of m [42], [43]. This structure connects m^n input terminals to m^n output terminals through a series of n

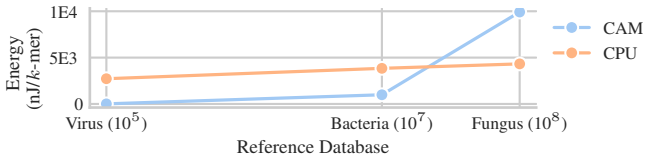


Fig. 4. Monolithic CAM vs CPU k-mer classification energy derived analytically. CPU outperforms prior CAM designs on fungal datasets due to efficient sublinear search algorithms.

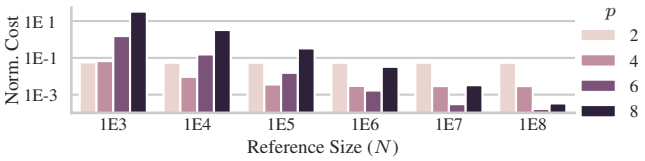


Fig. 5. Effective search cost (Eq. 1) for 16-mer datasets with varying sizes. Results are normalized to monolithic baseline and presented on a log-log axis.

switching stages. The topology maintains diameter logarithmic to the total number of nodes and enforces deterministic routing, making it a staple in parallel computing and network-on-chip (NoC) designs [44]–[48]. Each node in the network is assigned a unique address represented in radix-m with n digits. Routing a packet involves traversing the network stages, where at each stage i, the router examines the i-th digit of the destination address to determine the appropriate output port. This process effectively "fixes" one digit per stage, ensuring that after n stages, the packet reaches its correct destination, (see example in Fig. 3). From the packet's perspective, the traversal resembles navigating through a tree structure, where each decision point (router) directs the packet closer to its destination based on the corresponding digit in the address. This deterministic routing guarantees a unique path from any source to any destination, simplifying the routing logic and enabling predictable performance. In our work, we use the butterfly network topology to facilitate scalable and parallel indexing into independent search-space partitions.

III. Search Space Partitioning

This section presents the scalability limits of monolithic CAM search and motivates our partitioning-based solution. We introduce theoretical cost models for partitioned search and examine two approaches: a naive prefix method and a data-aware hierarchical adaptive partitioning (HAP) scheme. Using real genomic datasets, we compare their theoretical search costs across different partitioning granularities. Finally, we describe how the indexing step in both schemes maps naturally onto a butterfly NoC.

A. The Problem With Monolithic Searches

Though CAMs offer $\mathcal{O}(1)$ search time, there is no free lunch: their area and energy scale linearly with database size ($\mathcal{O}(N)$). Figure 4 illustrates this tradeoff between CAM and CPU systems across datasets of 10 species. For small N (10^4–10^5), CAMs outperform CPUs in performance, energy, and area, thanks to low constant overheads and in-memory parallelism. CPUs, constrained by memory bandwidth and the Von Neumann bottleneck, struggle in this regime. As N increases, however, CPUs can exploit sublinear search strategies (e.g., binary search), and their constant-factor overhead becomes less significant. For large databases, CPUs ultimately surpass monolithic CAMs in energy efficiency. This tradeoff has confined prior CAM-based works [12]–[14] to small viral genomes (typically 10^4 base pairs). Yet many pathogenic genomes are orders of magnitude larger: bacterial genomes span $\sim 10^6$ bp, while fungal genomes exceed 10^7 bp. We address this scalability gap with a partitioned CAM architecture tailored for large-scale DNA classification, enabling on-chip detection of complex pathogens.

B. Prefix Partitioning

To optimize large-scale CAM-based search tasks, such as DNA classification, we propose a search-space partitioning approach. By using selected positions in each query k-mer as an index, we divide the reference k-mers into disjoint groups, activating only the CAM subarray relevant to the query. This reduces energy consumption, as CAM search energy scales with the number of active entries.

From a reference genome of length N, we extract all k-mers using a sliding window of unit stride, yielding $N - k + 1$ substrings. These k-mers form a monolithic search space of size $N \times k$. We partition this space using p selected positions in each k-mer, producing $P = 4^p$ total partitions. Since these index positions are matched before dispatching a query to a partition, only the remaining $k - p$ characters need to be stored and compared within each subspace. The number of stored k-mers remains N, but each query only searches a fraction of the total space. Figure 8 illustrates this scheme.

To model the tradeoff between partitioning and search cost, we define the following objective function:

$$O_{\text{ideal}}(k, p, N) = \underbrace{p \times P}_{\text{indexing}} + \underbrace{(k-p) \times \frac{N}{P}}_{\text{regular partition search}}, \; 0 \leqslant p \leqslant k \quad (1)$$

The first term captures the cost of indexing logic shared across queries, while the second represents the average number of entries searched within a partition. Fig. 5 plots this cost, normalized to the monolithic baseline, across varying p and dataset sizes N. We observe that the optimal partitioning granularity increases with N, as more partitions reduce per-query cost. However, for very large p (e.g., $p = 8$), the indexing overhead dominates, making such granularities suboptimal even for massive datasets. In contrast, small p consistently offers modest but suboptimal improvements.

C. Hierarchical Adaptive Partitioning (HAP)

While simple, prefix partitioning has notable drawbacks. The index positions are fixed, and the resulting partitions are often highly imbalanced. In Fig. 6, we show how even small values of p can produce skewed partition sizes when using prefix-partitioning. This imbalance increases fragmentation and energy usage, as large partitions dominate the search workload. To address this, we introduce *hierarchical adaptive partitioning (HAP)*, a greedy algorithm that iteratively splits the dataset to balance partition sizes. Figure 8 illustrates the

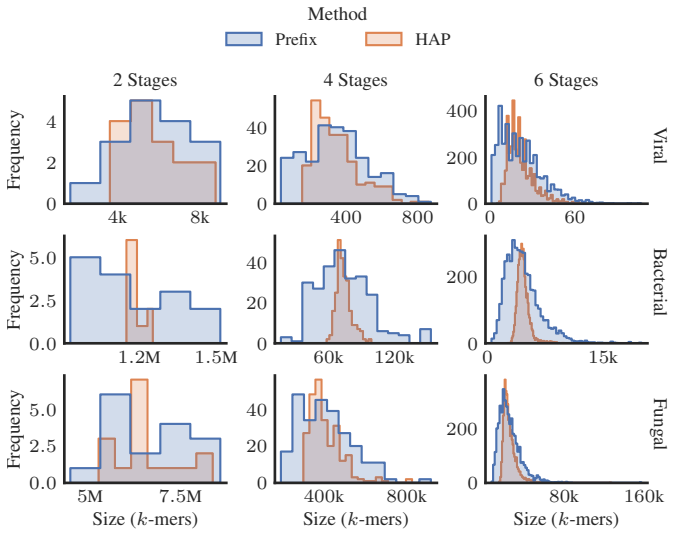


Fig. 6. DNA partition size distributions plotted for different partitioning schemes and /stages p. HAP generates more balanced partitions than prefix-based partitioning.

offline construction and online usage of both prefix-based and HAP trees. The figure compares how each method partitions a monolithic k-mer reference set (steps ①–④) and routes queries during classification (steps ❶–❹). Prefix-based partitioning splits on fixed positions, while HAP adaptively selects split points to balance partition sizes. During search, queries in the HAP tree land in smaller, more efficient partitions than in the prefix tree. A recursive formulation of HAP is given in Alg. 2. In Fig. 6, we illustrate the improvement in partition size balance from using HAP over prefix-partitioning. Because HAP adaptively splits partition nodes based on the current data distribution, the variation in partition sizes is reduced.

Our previous objective function (Eq. 1) assumed uniform

Algorithm 2: Hierarchical Adaptive Partitioning (HAP)

Input: Pattern length k, total stages t, current stage s, used positions U, current set of patterns C
Output: List of final partitions
Function: `HAP` (k, t, s, U, C)

```
if s > t then
    return C;
best_var ← ∞;
foreach position p ∈ {0, ..., k}\U do
    var ← evaluate_split (C, p);
    if var < best_var then
        best_var ← var;
        best_pos ← p;
G ← split_patterns(C, best_pos);
R ← ∅;
foreach group g in G do
    R ← R ∪ HAP(k, t, g, s+1, U ∪ {best_pos});
return R;
```

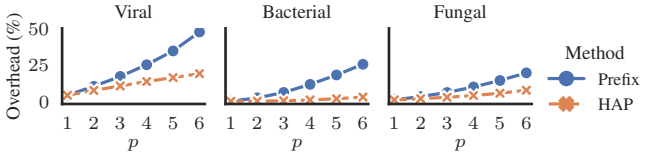


Fig. 7. Search overheads on real data due to imbalanced partitions. We calculate the weighted partition search cost (Eq. 2) for prefix-based, HAP, and perfectly-uniform partitioning.

query distribution, yielding an average partition cost of $(k-p) \times \frac{N}{P}$. However, in DNA classification, query distributions mirror reference distributions, since the source organism is expected to be present in the reference set. To better reflect true costs, we weight each partition by its share of the total reference set. Let S_i be the number of k-mers in partition i. The expected cost becomes:

$$\mathbb{E}[O_{\text{real}}(k,p,N)] = \underbrace{p \times P}_{\text{index}} + \underbrace{(k-p)\sum_{i=0}^{P-1} |S_i|\frac{|S_i|}{N}}_{\text{weighted partition search}},\ 0 \leqslant p \leqslant k \tag{2}$$

When partitions are uniform ($|S_i| = N/P$), Eq. 2 reduces to Eq. 1. We analyze the impact of the weighted partition search term on our proposed partitioning schemes. Both schemes incur overhead from partition imbalance, but prefix partitioning suffers more: in Fig. 7 prefix overheads exceed 40% above ideal, while HAP stays below 20%.

D. Butterfly Network Implementation

Hierarchical adaptive partitioning (HAP) constructs a decision tree that maps naturally onto a butterfly network. We implement HAP on a radix-4, p-stage network, where p is the partitioning granularity. Each stage performs a data-dependent split based on one k-mer index, routing queries along one of four paths corresponding to the nucleotide at that position. Fig. 9 shows a 3-stage (p=3) configuration. The first-stage routers all use the same split position, 3, to route queries. For instance, if the nucleotide at index 3 is 'A', the packet is routed to the rightmost region of the network. At the next stage, all four second-stage quadrants use independently chosen positions. Since both k-mers routed to the rightmost quadrant in the first stage, they again both split based on the same position—index 4. In the third and final stage, each k-mer is routed to a different subregion of the network: one splitting at index 0 and one splitting at index 2. From the final split directions, k-mers are sent to the CAM containing the corresponding partition. Our butterfly network implements parallel HAP trees with programmable routing logic at each node.

Prefix partitioning is a special case of HAP where the split positions are fixed ahead of time and shared across all routers in each stage. In this case, the k-mer is partitioned based on the first p characters, every router at a given stage uses the same fixed index (e.g., stage 0 uses position 0, stage 1 uses position 1, and so on). This results in unbalanced partitions when the underlying prefix distribution is skewed.

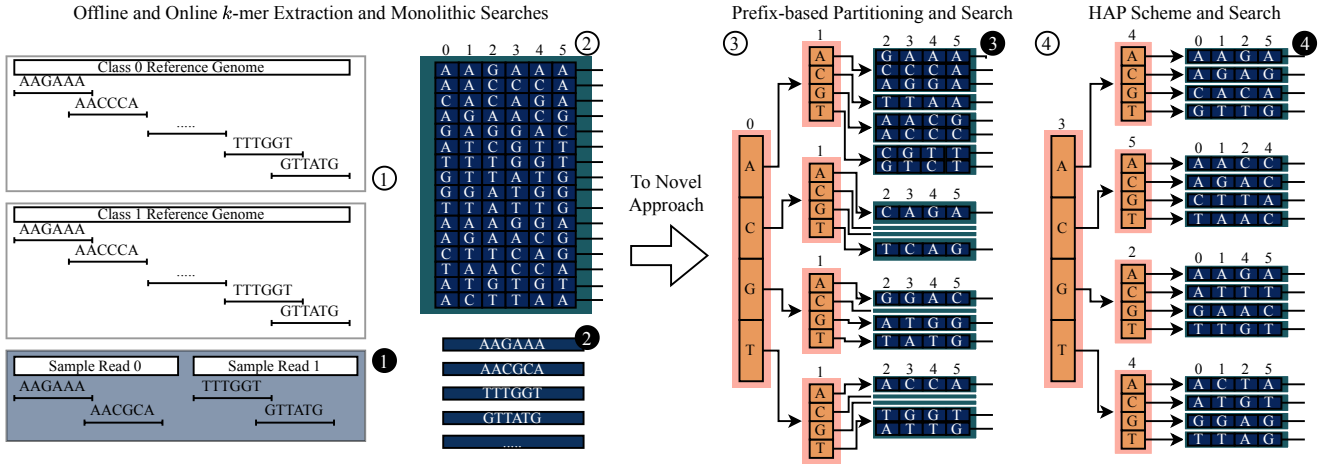


Fig. 8. Comparison of monolithic (prior work), prefix-based partitioning, and HAP schemes. Steps ①–④ depict offline dataset construction. Prior works extract k-mers (①) to construct monolithic reference set (②). Prefix-based partitioning (③) splits the reference set using fixed indices (e.g., first two characters), leading to imbalanced partition sizes due to nonuniform prefix distribution. In contrast, HAP (④) recursively selects the split index that minimizes partition size variance, producing more balanced trees. Steps ❶–❹ illustrate the online search process. In the monolithic configuration (❶–❷), the query k-mer must be searched across all entries. In the prefix tree (❸), the query "AAGAAA" lands in a partition of size 3 by matching fixed prefix positions. In the HAP tree (❹), adaptive splits direct the same query to a partition of size 1, reducing energy.

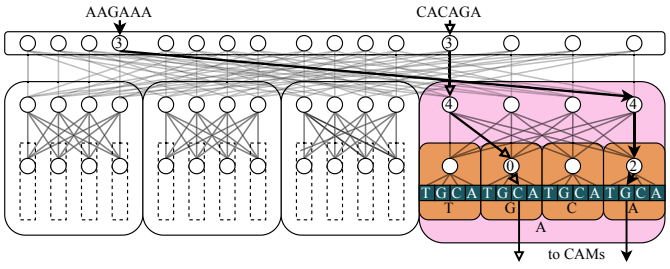


Fig. 9. A 3-stage ($p = 3$) configuration of HAP on a butterfly network topology. At top, 64 injection ports (not shown) feed into 16 radix-4 routers.

IV. NP-CAM

This section presents: an overview of the NP-CAM architecture, a detailed description of its two primary components—the HAPNet interconnect and SCore units, and the execution flow and system integration for end-to-end classification.

A. Architecture Overview

NP-CAM (Fig.10) consists of three main components: the frontend, which manages control and memory access; the in-memory search engine, which uses the HAPNet NoC to partition and search across CAM banks (SearchCores); and the backend, which performs result reduction on the same network.

1) Frontend: The frontend handles control, off-chip memory access, and preprocessing. A microcontroller configures the search engine by programming each HAPNet router's split position and assigning class IDs to CAM subarrays. This setup is performed offline, once per reference dataset. Incoming DNA reads are parsed using a bank of shift registers that extract k-mers via a sliding window. Input sequences are encoded in UCSC 2-bit format [49]. Each metagenomic sample may yield thousands of reads, with each read producing hundreds of k-mers. Shift registers operate on independent nucleotide streams from a slice of, one of, or multiple of these reads. Each register consumes a nucleotide, shifts left, discards the most significant position, and appends the new nucleotide. To maximize throughput, all P registers attempt to load P nucleotides and produce P k-mers per cycle. If contention arises in the search engine, backpressure may stall specific registers, preserving their contents. On-chip buffering serves to both support high-bandwidth streaming access to off-chip memory and dynamically adapt to network contention.

2) Search Engine: The search engine consists of the HAPNet and SCores (Search Cores); it is responsible for sorting, routing, and searching incoming k-mer queries, as well as accumulating class-specific match scores. Query k-mers enter the search engine through P injection ports of the HAPNet, connected to the frontend shift registers. Queries may arrive in any order; HAPNet deterministically routes each to its corresponding subspace partition while handling contention. Each routed k-mer exits HAPNet via one of P ejection channels, delivering it to the appropriate SCore. Each SCore contains a single CAM bank, hierarchically organized into mats, arrays, and subarrays (see Fig. 12), similar to prior CAM accelerators [50], [51]. Each SCore broadcasts the incoming k-mer to all CAM subarrays within the bank and accumulates matches using peripheral logic.

3) Backend: The backend comprises the local accumulator units within each SCore, the reversed HAPNet module, and the microcontroller running the `C()` classifier. The reversed HAPNet aggregates scores from all SCores and sends the result to the microcontroller. The butterfly network naturally supports low-latency reduction trees, and requires only a subset of the routers to support reduction. Our classifier `C()` function performs a scaled arg-max over class scores, normalizing each by the expected number of random matches based on genome size and k-mer length. Similar classifiers also apply lightweight logic on class scores and can be implemented on simple microcontrollers [12]–[14], [16].

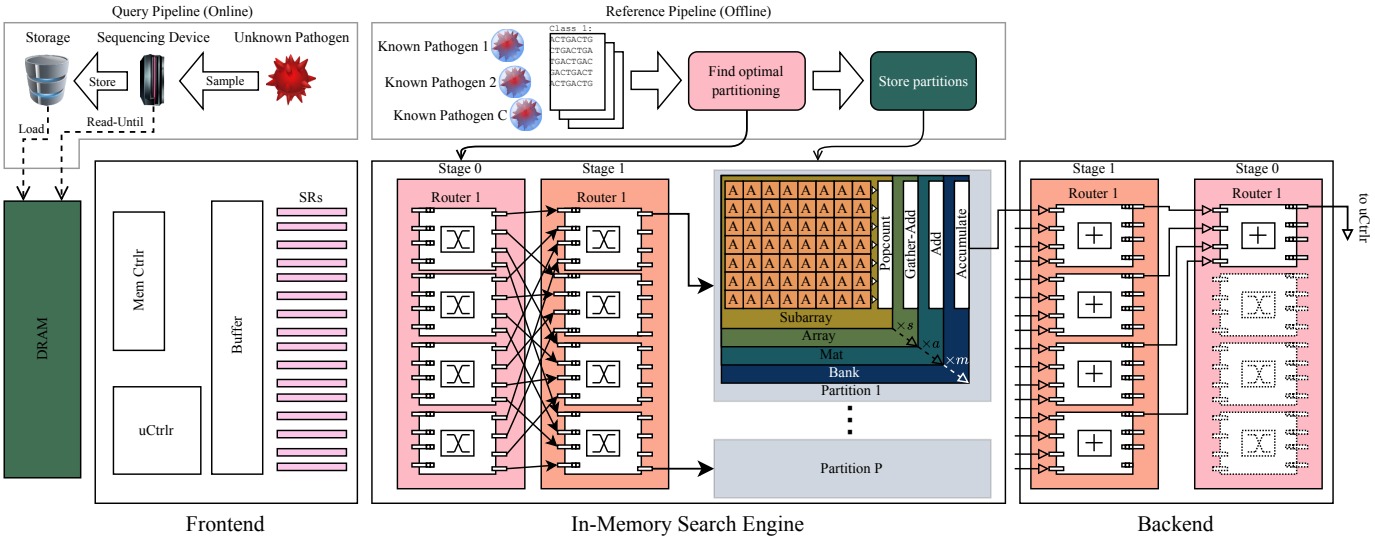


Fig. 10. NP-CAM architecture overview. The frontend preprocesses data input from DRAM through shift registers and controllers. The novel search engine contains HAPNet that scalably routes k-mers to corresponding SCores based on adaptive partitioning scheme. The SCores search the remainder of the query and accumulate matches for each class locally. The backend processes results from local accumulators via reduction on the same HAPNet.

B. HAPNet Network Topology & Router Microarchitecture

1) Forward Pass: Fig. 11 shows the router microarchitecture. Each router is radix 4, with four input and output channels corresponding to the four nucleotides. Internally, the router includes input buffers, a position register (identifying the current pivot index), a direction decoder, round-robin arbiter, credit-based flow control, and a 4-way integer adder for the backward reduction phase. The direction decoder inspects the nucleotide at the pivot position and selects the appropriate output channel. A round-robin arbiter resolves conflicts when multiple inputs request the same output, giving priority to each of the inputs in turn. Packets traverse the network in a pipelined manner: 1 cycle for router processing and 1 cycle for each hop. This lightweight design minimizes area and power overhead for the indexing process. HAPNet efficiently connects P shift register producers to P SCore consumers, managing contention and routing in parallel. The butterfly topology ensures scalable and deterministic routing for high-throughput DNA classification.

2) Backward Pass: As shown in Fig. 10, part of HAPNet supports a backward pass for class score reduction. In this phase, routers reverse their input and output directions: outputs become inputs and vice versa. Each reduction-stage router aggregates inputs from all four channels using an integer adder and forwards the result upstream. These modifications require minimal additional logic and apply only to the subset of routers involved in score reduction. The backward pass proceeds in a pipelined manner across all C classes and p network stages, completing in $C + p + 1$ cycles.

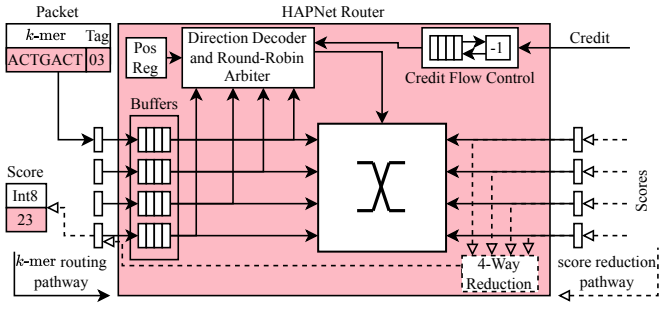


Fig. 11. Router microarchitecture for NP-CAM. The solid line shows k-mer routing in the forward pass; the dotted line shows score reduction in the backward pass.

C. SCore Memory Hierarchy & Microarchitecture

Each SCore (Search Core) consists of a single CAM bank and corresponds to one reference partition. We adopt a four-level hierarchical CAM architecture from prior accelerator designs [50], [51], consisting of bank, mat, array, and subarray levels (Fig. 12). In our notation, m is the number of mats per bank, a is the number of arrays per mat, s is the number of subarrays per array, and r is the number of rows per subarray. The number of banks is fixed at P, with each having a one-to-one correspondence with a reference partition from HAP (Sec. III-C). If a partition contains more k-mers than can fit in a single bank, a subsample is taken. In practice, around 20% coverage is necessary to maintain classification performance (Sec.V-C2); meaning the general size of the workload must be known to determine the appropriate CAM sizing.

Each SCore locally accumulates scores across C classes. We assign each subarray to a single class, enabling fine-grained representation without requiring per-row class tracking. When a k-mer is searched, matchlines in the subarray are activated, and a `POPCNT` operation counts matches. At the array level, scores

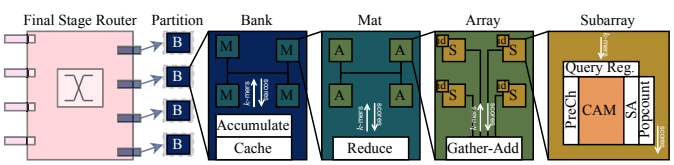


Fig. 12. The hierarchical CAM architecture used in SCores, with four levels: bank, mat, array, and subarray. Each SCore contains a single bank.

TABLE I
CAM CIRCUIT PARAMETERS FOR RELATED DESIGNS.

Design	This work [52]	DASH-CAM [12]	HD-CAM [13]	EDAM [14]
Technology (nm)	14	16	65	65
Cell type	10T	12T	30T	42T
Cell area (μm^2)	0.55	0.68	5.45	33
Cell search energy (fJ)	0.38	0.3	0.54	1.4
DNA encoding	binary	one-hot	one-hot	one-hot
Match type	exact	hamming	hamming	edit
Search time (ns)	1	1	1	0.75

from subarrays with the same class ID are aggregated using gather-add. Mats and banks sum partial scores into C-wide vectors, which are stored in a local SRAM-based cache.

We use a conventional 10T binary CAM (BCAM) design (Fig. 2, Table I). CAM cells store binary values $0, 1$, while DNA uses four bases A, C, G, T. To encode DNA in binary, we map each base to a unique 2-bit code stored across two adjacent CAM cells. For example, base 'A' is encoded as '00', and a row contains $2k$-p cells. This dual-cell scheme reduces memory overhead compared to the one-hot (4-cell) encodings used in other works.

D. DNA Classification in NP-CAM

1) Offline Processing: To perform DNA classification in NP-CAM, we generate a monolithic k-mer set, select a partitioning scheme, and program the search engine. The k-mer set is extracted from reference genomes of target pathogens, with each k-mer labeled by its class. A single NP-CAM configuration may support multiple partitioning schemes, provided they share the same granularity p. The HAPNet's position registers are programmed accordingly, and the reference dataset is partitioned. Each partition is assigned to a SCore, with subarrays allocated to classes in proportion to their relative size within the partition. Once the genome of a pathogen is known, the reference changing, and thus the offline partitioning and accelerator programming, occurs infrequently.

2) Online Classification: NP-CAM supports scalable, parallel, and energy-efficient online classification of metagenomic samples. To extract greater parallelism from the workload, NP-CAM supports *read-level parallelism*—up to 2^t reads per batch. Modern sequencing devices produce many reads from a sample in parallel; NP-CAM can consume many of these in real time. Each shift register will process a stream of nucleotides in a batch before the reduction step occurs. Depending on the batch size and available number of shift registers, reads may be concatenated with padding or split into overlapping

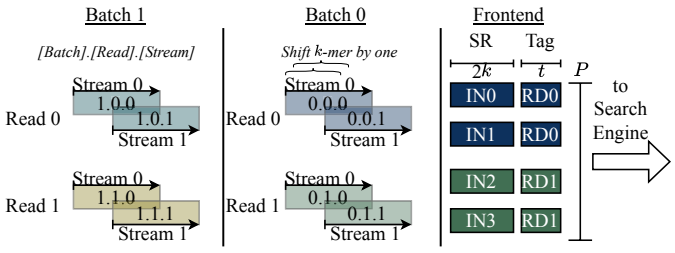


Fig. 13. Input format and execution flow. Supports batch size up to 2^t.

TABLE II
REFERENCE GENOME DATASETS OF VIRAL PATHOGENS, SOURCED FROM NCBI DATABASE [53].

Domain	Pathogen	Length	Notes
Viral	Lassa	10,700	Same viruses used in prior work [12]
	Influenza	13,600	
	Measles	15,900	
	Rotavirus	18,600	
	SARS-Cov-2	29,900	
Bacterial	E. Coli	4,640,000	Common human pathogenic bacteria
	Streptococcus	2,110,000	
	Salmonella	4,950,000	
	Staphylococcus	2,820,000	
	Tuberculosis	4,410,000	
Fungal	A. fumigatus	29,400,000	Critical or high priority according to WHO [54]
	Candida albicans	14,300,000	
	Candida auris	12,200,000	
	C. neoformans	18,900,000	
	Histoplasma	30,500,000	

TABLE III
SIMULATED READ DATASETS INFORMATION. REPORTED LENGTH AND ACCURACY VALUES ARE MEANS, READ COUNT IS PER CLASS.

Read Type	Simulator	Length	Accuracy	Reads	Total
Illumina	ART [55]	100	99.9%	10,000	1M
PacBio	PBSIM [56]	5,000	99.0%	200	1M
ONT	PBSIM [56]	10,000	90.0%	100	1M

subsequences to generate the streams. As each new k-mer is generated, a t-bit read-ID tag is appended. The tagged k-mer is then injected into one of P HAPNet ports. The k-mer and tag traverse p stages of the network and arrive at the corresponding SCore. The remaining $k-p$ nucleotides are compared against CAM entries, and class scores are accumulated locally. Scores for all active reads are cached in each SCore's accumulator unit. The read-ID tag indexes a direct-mapped score cache to read and write per-read scores efficiently. This approach improves utilization by allowing each forward pass to process many k-mers, and it reduces pipeline drain overhead. Total backward pass latency is reduced from $N_{\text{reads}}(c+p-1)$ cycles to $\frac{N_{\text{reads}}}{2^t}(2^t c + p - 1)$.

V. EXPERIMENTAL RESULTS

A. Experimental Setup and Methodology

1) Datasets and Software Baseline: The reference datasets, summarized in Table II, include viral, bacterial, and fungal genomes sourced from the NCBI database [53]. The query datasets, shown in Table III, consist of three simulated sequencing read types: 1) Illumina reads: short (100bp), high-accuracy (≈99.9%) [57]; 2) PacBio reads: longer (5,000bp), high-accuracy (≈99%) [19]; 3) ONT reads: ultra-long (10,000bp), lower-accuracy (≈90%) due to higher noise [20]. As a software baseline, we used Kraken2 with default settings and the standard RefSeq [58] `viral`, `bacteria`, and `fungi` databases [10]. Kraken2 was evaluated single-threaded on an Intel Xeon Gold 6254 CPU at 3.10 GHz.

2) Hardware Modeling and Cycle-Accurate Simulation: We model NP-CAM using CAM circuit parameters from [52]. The 10T-BCAM model was implemented and evaluated via SPICE

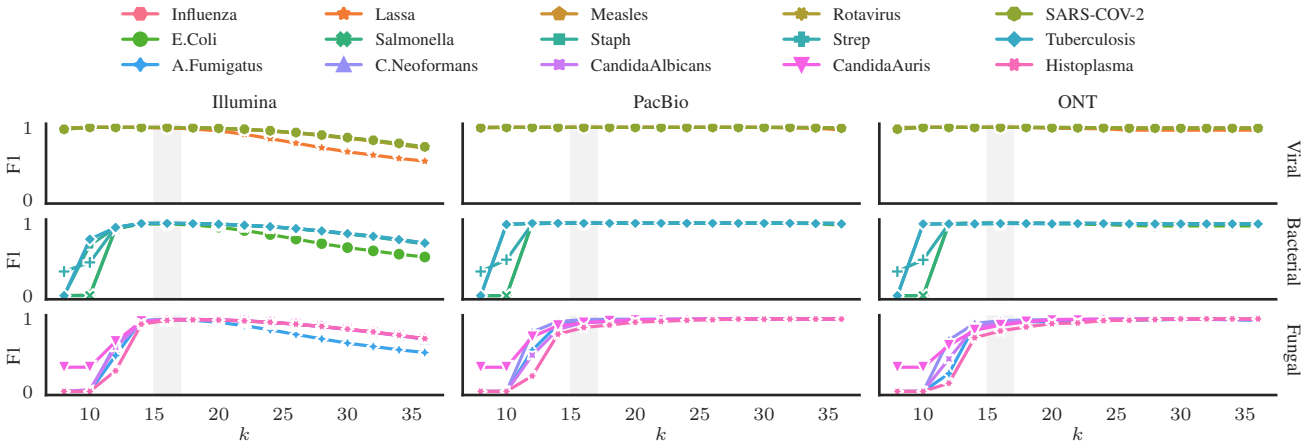


Fig. 14. Exact matching DNA classification performance for various k-mer lengths and pathogen/read type pairs. The k-mer length of 16 is chosen for its consistently high performance across all scenarios.

simulation. Peripheral circuitry and interconnects were modeled with a 20% cumulative energy and area overhead across the memory hierarchy. Digital components—including shift registers, buffers, HAPNet routers, and near-memory processing logic—were synthesized using Synopsys Design Compiler at a commercial 28nm node under SS corner conditions. Area and power estimates were scaled from 28nm to 14nm using the methodology of Stillmaker et al. [59]. Channel and wiring models were chosen conservatively to ensure area and power estimates remain realistic for large-scale HAPNet configurations. We estimate channel area as FbL, where F is feature width, b is channel bit width, and L is channel length, estimated as $L \approx \sqrt{AP/4}$ given router area A. Channel power is computed assuming 0.5pJ/bit/mm [60]. NP-CAM was evaluated using a custom, object-oriented, cycle-accurate simulator that faithfully models all components and tracks state across cycles for result validation.

B. NP-CAM Classification Accuracy

Figure 14 shows that classification accuracy is highly sensitive to the choice of k-mer length. Accuracy peaks in the k-mer length range of 14 to 20. Smaller k-mers, while more area- and energy-efficient for CAMs, produce overly frequent patterns. This reduces the ability to distinguish between classes—especially in the larger bacterial and fungal datasets. As k increases, Illumina reads—despite their high accuracy (99.9%)—show a sharper drop in F1 scores than PacBio and ONT. This is due to the short length of Illumina reads (100bp), which yield fewer k-mers and make each error disproportionately impactful. In contrast, PacBio and ONT reads produce many more k-mers, making them less sensitive to individual errors—even at higher k. *Based on these findings, we select k=16 as the optimal k-mer length for NP-CAM.* This value strikes a balance between error tolerance and discriminative power across all read types and reference datasets, matching the performance of approximate-matching accelerators [12]–[14] while minimizing CAM width.

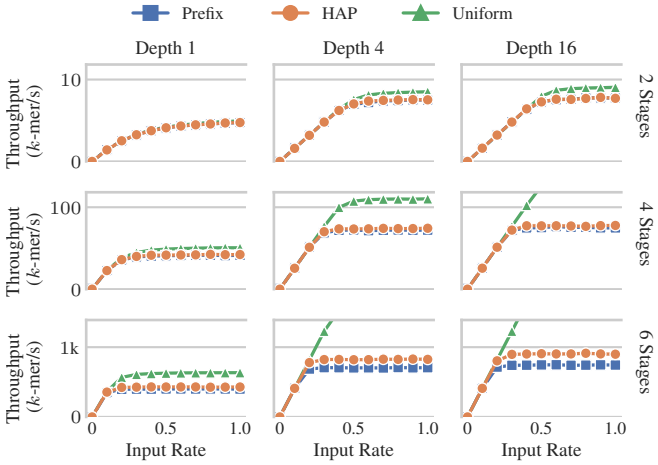


Fig. 15. Effect of partitioning scheme, granularity, and buffer depth on isolated network performance of HAPNet.

C. Design Space Exploration

1) HAPNet Architecture: We evaluate HAPNet performance across a range of design parameters. Throughput does not always match the injection rate due to contention: multiple inputs may compete for shared network resources, leading to increased latency and reduced throughput. We simulate 10,000-cycle trials to evaluate network throughput across a range of injection rates—defined as the probability that a k-mer is injected at each HAPNet input port per cycle. This methodology is standard in NoC research for assessing network performance. We sweep buffer depth and the number of HAPNet stages. Each configuration is evaluated using both prefix-based and HAP partitioning, classifying simulated reads (Table III). Uniform traffic is included as an idealized positive control. Results are shown in Figure 15. Saturation throughput improves with increasing buffer depth and more uniform traffic distributions. Larger HAPNet configurations benefit more from deeper buffers. *Overall, the optimal buffer depth is determined to be 4.* HAP consistently achieves higher throughput than naive prefix partitioning, due to its more balanced partition

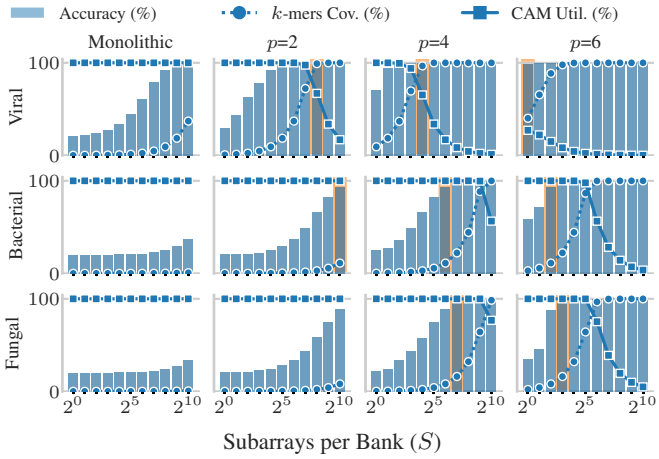


Fig. 16. Effect of CAM bank size (subarray count) on k-mer coverage, CAM utilization, and classification accuracy. Chosen configurations are highlighted, not all are shown.

sizes and less concentrated traffic.

2) SCore Architecture: We explore how SCore sizing affects fragmentation and storage overhead across partitioning granularities. The number of rows per subarray (r) is fixed to 32, 128, and 512 for viral, bacterial, and fungal datasets, respectively. We vary the number of subarrays (S) per bank from 1 to 1024. The memory hierarchy is sized to match the number of subarrays: S=mas, and to balance the interconnect requirements at each level: m=a=s. Subarrays are assigned to classes in proportion to each class's k-mer frequency within the partition (from HAP or prefix). For each class, k-mers are sampled without replacement. In Figure 16, we plot CAM utilization, k-mer coverage, and accuracy versus S for different datasets and values of p. Accuracy saturates when k-mer coverage reaches 20–40%; thus, *we set the storage requirements to 64K, 2M, and 16M k-mers for viral, bacterial, and fungal, respectively*—highlighted in Figure 16. When partitions are small and banks are large, many CAM entries go unused. This effect is especially pronounced for the viral dataset at p=6, where small partitions cause average utilization to fall below 50% even with a single subarray per bank. Figure 17 shows the effect of cache size on SCore performance. We sweep the tag bits from 0-7 and 0-8 for ONT and PacBio reads, allowing a single read to an entire dataset of 1Mbp to operate in parallel. For Illumina reads, we sweep the tag bits from p-1 to 2p+2, exploring the knee of the performance curve for each configuration of NP-CAM. Larger caches help maintain resource utilization at higher p, especially for Illumina reads, which produce fewer k-mers per read. *We identify 4 tag bits (16 cache entries per class) as optimal for p=1–3, and 6 tag bits (64 entries) as optimal for p=4–6 configurations.* This selection balances area and performance.

3) NP-CAM System-Level Evaluation: We illustrate the throughput of NP-CAM across configurations in Figure 18a. System throughput varies significantly with read type and partitioning granularity: higher partitioning provides increased hardware parallelism, and longer reads better exploit this parallelism. As p increases, throughput for short Illumina reads

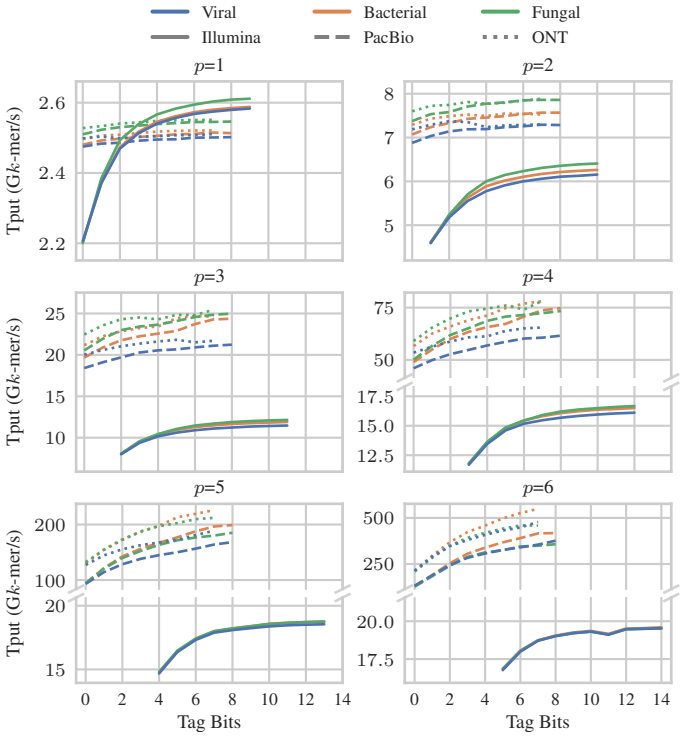
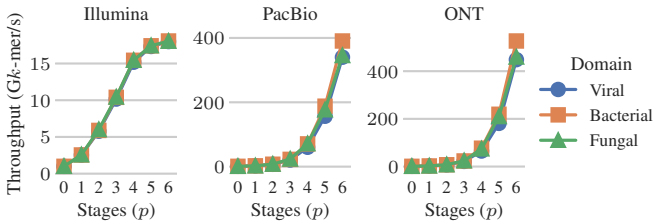


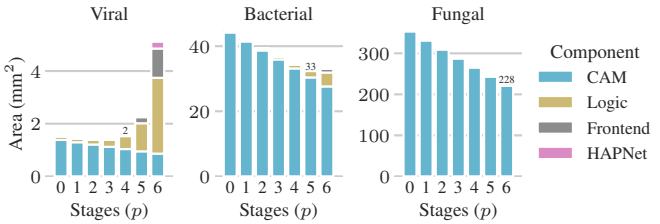
Fig. 17. Study on cache sizing effect on performance. Batch size doubles for each increment in Tag Bits parameter.

saturates rapidly due to bottlenecks in result aggregation, driven by the fixed latency per read. Longer PacBio and ONT reads benefit more significantly from increased p due to enhanced concurrency in k-mer searches. An observed secondary effect is the slightly higher throughput for bacterial pathogen detection compared to fungal pathogens in the long-read and high-p context. We attribute this to the fungal dataset containing multiple closely related species in the Candida genus, increasing k-mer overlap and network contention. For instance, at $p = 6$, ONT bacterial throughput reaches 527 Gk-mer/s compared to 460 Gk-mer/s for fungal datasets. Additionally, the small viral dataset when partitioned at $p = 6$ shows nonuniform partitions (Fig. 6), leading to contention in HAPNet, limiting maximum throughput (448 Gk-mer/s for ONT reads). Since each k-mer requires 2 bits, even NP-CAM's most bandwidth-intensive configuration ($p = 6$, ONT, bacterial) demands only 132 GB/s of DRAM bandwidth.

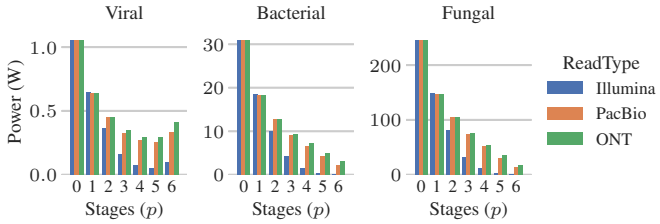
Figure 18b shows total area across configurations. Scales differ across domains due to increasing storage requirements for reference genomes. The area for viral classification remains below 2 mm^2 until $p = 5$, beyond which the increase in SCore logic dominates total area (area decreases monotonically to 1.4 mm^2 at $p = 3$ before increasing to 1.59 and 2.31 mm^2 at $p = 4$ and $p = 5$ respectively). For bacterial classification, total area initially decreases moderately before sharply increasing as p grows, reaching a minimal area of 32.7 mm^2 at $p = 5$. Fungal classification configurations exhibit the largest areas, driven by significantly larger reference genome sizes. At $p = 6$, the total fungal area is minimized at 228 mm^2, achieving a net area decrease of over 100 mm^2 compared to monolithic



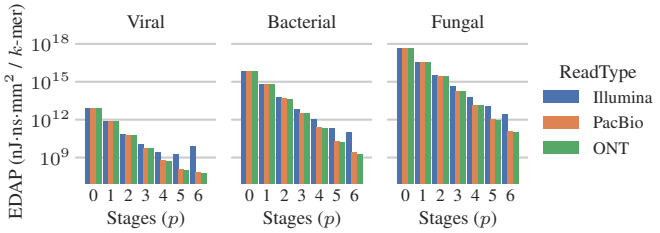
(a) Throughput primarily varies between read types. Longer ONT and PacBio reads can better exploit high p NP-CAM parallelism.



(b) Area varies significantly across domains due to greater memory requirements. Larger p first decrease total area before increasing.



(c) Power varies significantly across domains due to larger CAM footprints. Increasing p splits up larger CAMs into smaller partitions.



(d) Energy-Delay-Area product across different configurations. Multiple orders of reduction are achieved with NP-CAM partitioning.

Fig. 18. Design space exploration across partitioning granularity (p), reference domains (viral/bacterial/fungal), and read types (short Illumina/long PacBio and ONT). The p=0 configuration indicates the monolithic baseline. Other architectural parameters are fixed based on earlier design-space studies.

configurations due to reduced CAM widths.

Figure 18c shows power consumption across various NP-CAM configurations. Power consumption exhibits substantial variation across datasets due to differences in CAM footprint size. For viral datasets, increasing partition granularity initially reduces power from 1.05 W at $p = 0$ to less than 0.30 W at $p = 4$, as large CAM arrays are partitioned into smaller segments. However, overhead from additional logic at higher granularity limits further reductions. Bacterial and fungal datasets require more power (17.8 W at $p = 6$ for ONT reads on fungal) due to larger memory footprints. Deeper partitioning strategies significantly mitigate the total power.

Figure 18d shows the Energy-Delay-Area Product (EDAP) across different configurations. EDAP significantly decreases as partition granularity (p) increases, highlighting NP-CAM's efficient balance between performance, energy, and area. Monolithic fungal datasets have an EDAP of 4.3×10^{17} nJ·ns·mm^2/k-mer, reducing dramatically to 9.5×10^{10} nJ·ns·mm^2/k-mer at $p = 6$. The EDAP metric indicates several orders-of-magnitude improvement over baseline configurations, especially pronounced for large fungal datasets, demonstrating scalability.

D. Comparison with State-of-the-Art

To our knowledge, NP-CAM is the first CAM-based accelerator to scale to DNA classification tasks involving pathogen genomes exceeding 1 million base pairs. Table IV summarizes NP-CAM's performance alongside representative state-of-the-art hardware and software solutions across viral, bacterial, and fungal classification workloads. NP-CAM throughput and power results are reported as ranges corresponding to short and long reads, respectively. Kraken2 results are reported for a single core and ONT reads, which we empirically found to run 5–20× faster than PacBio and Illumina, respectively.

TABLE IV
STATE-OF-THE-ART DNA CLASSIFICATION SOLUTIONS.

Task	Solution	Tput (Gk-mer/s)	Power (W)	Area (mm^2)
Viral	NP-CAM (p=4)	15.2–65.1	0.07–0.29	1.59
	DASH-CAM [12]	1.00	0.88	1.57
	HD-CAM [13]	0.50	4.53	10.51
	EDAM [14]	0.67	3.93	16.00
	ClaPIM [16]	0.00015	0.37	0.03
	ClaPIM Filt. [16]	0.00440	*0.0015	*0.03
	KRAKEN2 [10]	0.00036	11.1	-
Bacterial	NP-CAM (p=5)	17.4–219.3	0.41–4.90	32.7
	DASH-CAM [12]	1.00	28.3	50.3
	HD-CAM [13]	0.50	145.0	336.3
	EDAM [14]	0.67	125.8	512.2
	ClaPIM [16]	0.00015	12.0	1.0
	ClaPIM Filt. [16]	0.00440	*0.048	*1.0
	KRAKEN2 [10]	0.00036	11.1	-
Fungal	NP-CAM (p=6)	18.0–459.2	0.78–17.8	228.1
	DASH-CAM [12]	1.00	226.5	403
	HD-CAM [13]	0.50	1159.6	2692
	EDAM [14]	0.67	1006.6	4098
	ClaPIM [16]	0.00015	95.63	8
	ClaPIM Filt. [16]	0.00440	*0.39	*8
	KRAKEN2 [10]	0.00036	11.1	-

*: plus CPU power/area (not reported)

MetaCache-GPU [61] is omitted due to its inferior performance to CPU-based Kraken2 [12]. All hardware solutions are normalized by scaling to a fixed reference dataset size: 64K, 2M, and 16M k-mers for viral, bacterial, and fungal tasks, respectively. For HD-CAM and EDAM, which target older technology nodes, we scale normalized area to 14nm [59].

Compared to prior CAM-based accelerators [12]–[14], NP-CAM offers significantly higher scalability in throughput, power efficiency, and area. Against the PIM-based ClaPIM [16], NP-CAM delivers orders-of-magnitude higher throughput and greater energy and area efficiencies. ClaPIM Filt. uses software-

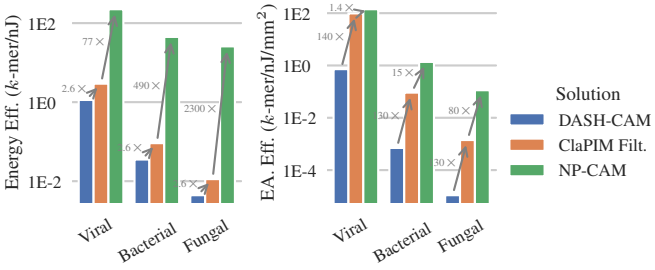


Fig. 19. Energy and energy-area efficiency of selected hardware solutions.

aided filtering to reduce search operations—similar in spirit to our HAP-based indexing. However, it lacks a hardware implementation of parallel filtering and search. In contrast, NP-CAM implements indexing and parallel search directly in hardware via HAPNet, enabling scalable, systematic partitioning. Furthermore, the memristive hardware of ClaPIM Filt. suffers from low endurance with a projected operational lifespan of just 9 hours to 1 year. We plot the energy efficiency and energy-area-product efficiency of the top hardware solutions in Figure 19. For ClaPIM Filt., we consider only the energy and area contributions of the hardware components. The results demonstrate that the efficiency improvements of NP-CAM increase with the dataset size, proving its more scalable nature.

VI. Discussion

Even under conservative modeling assumptions, NP-CAM significantly outperforms prior CAM-based accelerators for realistic pathogen detection workloads. Specifically, viral and bacterial evaluations at moderate granularities (p=1,2,3) already yield substantial improvements in throughput, energy efficiency, and scalability. These results position NP-CAM as a significant advancement in hardware-accelerated DNA classification. Nevertheless, we discuss future directions here.

Given that large fungal datasets occupy 200mm^2, NP-CAM's suitability for DNA classification beyond the context of pathogen detection is limited. However, it could be significantly enhanced by employing emerging memory technologies such as memristors, which offer higher density and lower power compared to traditional SRAM-based CAMs [39], [40], [62]. NP-CAM, unlike ClaPIM [16], does not require repeated writes, making it well-suited to memristive device characteristics. Alternatively, we suggest a multi-pass reference loading strategy that sequentially loads portions of the reference k-mer set into the CAM and accumulates query scores across these passes. For example, in a two-pass strategy, CAM area halves and throughput is reduced by roughly 50%, a linear trade-off. Accordingly, NP-CAM can scale effectively to extremely large genomic databases containing plant or even mammalian genomes ($\sim 10^9$ bp) while maintaining performance and energy efficiency advantages. Exploring scalability beyond pathogen detection is a potential line of work.

Butterfly networks are commonly used in off-chip networks but are less prevalent on-chip due to their long wire lengths at scale. For our largest configuration (p=6), channel routing accounts for over 50% of each router's area. Prior work [47] proposed the flattened butterfly, which collapses each row of routers into a single high-radix router to improve VLSI compatibility. Their implementation of a 3-stage radix-4 butterfly in 65nm validates our HAPNet design up to p=3. While they discuss scaling via increased concentration, dimensionality, or hybrid topologies, we conservatively model energy and area using a full butterfly topology. Future work may explore other network topologies such as flattened butterflies.

Our system uses *exact matching* and achieves high accuracy (Sec.V-B). Prior works have used custom CAM [12]–[14] or PIM [16] designs to implement *approximate matching*, which can improve performance on noisy datasets when using large k-mers [12]–[14], [16]. Our choice of a smaller k-mer simultaneously allows for high sensitivity on noisy ONT reads and minimizes hardware area. Still, we outline how our architecture can be extended to support arbitrary Hamming distance matching [12], [13]. Supporting Hamming matching within SCore units is straightforward: we can directly adopt CAM designs from prior work [12], [13]. However, this only applies to the $k-p$ length subsequence of the k-mer that arrives at the SCore. To support mismatches across the full k-mer, we can attach a mismatch counter to the packet header and enable multicast routing in the HAPNet. As a k-mer moves through the network, it is multicast to all directions that do not exceed the mismatch threshold. For each divergence from the matched index, the mismatch counter is incremented. Once it arrives at the SCore unit, the allowed Hamming distance is adjusted to account for any mismatches garnered through the indexing step. Exploring the tradeoff between classification sensitivity—across varying k-mer ranges and approximate matching techniques—and classifier performance is proposed as future work.

VII. Conclusion

We introduce NP-CAM, a scalable and energy-efficient accelerator tailored specifically for DNA classification workloads. To the best of our knowledge, NP-CAM is the first work to exploit the *parallelism* enabled by CAM partitioning in this manner, transforming the approach to CAM-based search operations. Our architecture leverages a novel indexing and partitioning scheme, HAP, to accelerate the entire DNA classification pipeline, directly addressing critical bioinformatics applications such as pathogen detection. Through comprehensive design space exploration, we investigate the impact of major architectural parameters on key application metrics. NP-CAM demonstrates superior scalability, achieving throughput improvements ($10^7\times$/$370\times$/$84000\times$) and energy-efficiency improvements ($620000\times$/$4600\times$/$1800\times$) over state-of-the-art software/CAM/PIM solutions on large fungal datasets.

These results position NP-CAM as an advancement, positioned to meet the computational demands of rapidly expanding genomic datasets, enabling faster, more efficient DNA classification in a wide range of real-world settings. Beyond DNA classification, NP-CAM's efficient and scalable matching infrastructure has broad applicability in domains such as database acceleration, network acceleration, and pattern detection.

REFERENCES

[1] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: astronomical or genomical?" *PLoS biology*, vol. 13, no. 7, p. e1002195, 2015.

[2] E. W. Sayers, M. Cavanaugh, L. Frisse, K. D. Pruitt, V. A. Schneider, B. Underwood, L. Yankie, and I. Karsch-Mizrachi, "Genbank 2025 update," *Nucleic Acids Research*, vol. 53, no. D1, pp. D56–D61, 11 2024. [Online]. Available: https://doi.org/10.1093/nar/gkae1114

[3] K. A. Wetterstrand, "Dna sequencing costs: Data from the nhgri genome sequencing program (gsp)," 2023, accessed: 2025-07-29. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data

[4] I. Illumina, "Illumina's revolutionary novaseqx exceeds 200th order milestone in first quarter 2023," Apr. 2023, pR Newswire press release. [Online]. Available: https://www.prnewswire.com/news-releases/illuminas-revolutionary-novaseq-x-exceeds-200th-order-milestone-in-first-quarter-2023-301789677.html

[5] M. Alser, J. Lindegger, C. Firtina, N. Almadhoun, H. Mao, G. Singh, J. Gomez-Luna, and O. Mutlu, "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 4579–4599, 2022.

[6] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 05 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty191

[7] G. M. Boratyn, C. Camacho, P. S. Cooper, G. Coulouris, A. Fong, N. Ma, T. L. Madden, W. T. Matten, S. D. McGinnis, Y. Merezhuk, Y. Raytselis, E. W. Sayers, T. Tao, J. Ye, and I. Zaretskaya, "Blast: a more efficient report with usability improvements," *Nucleic Acids Research*, vol. 41, no. W1, pp. W29–W33, 04 2013. [Online]. Available: https://doi.org/10.1093/nar/gkt282

[8] A. V. Zimin, G. Marçais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke, "The masurca genome assembler," *Bioinformatics*, vol. 29, no. 21, pp. 2669–2677, 08 2013. [Online]. Available: https://doi.org/10.1093/bioinformatics/btt476

[9] D. E. Wood and S. L. Salzberg, "Kraken: ultrafast metagenomic sequence classification using exact alignments," *Genome Biology*, vol. 15, no. 3, p. R46, March 2014. [Online]. Available: https://doi.org/10.1186/gb-2014-15-3-r46

[10] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with kraken 2," *Genome biology*, vol. 20, no. 1, p. 257, 2019.

[11] K. Bibby and J. Peccia, "Identification of viral pathogen diversity in sewage sludge by metagenome analysis," *Environmental Science & Technology*, vol. 47, no. 4, pp. 1945–1951, 2013, pMID: 23346855. [Online]. Available: https://doi.org/10.1021/es305181x

[12] Z. Jahshan, I. Merlin, E. Garzon, and L. Yavits, "Dash-cam: Dynamic approximate search content addressable memory for genome classification," *bioRxiv*, 2023. [Online]. Available: https://www.biorxiv.org/content/early/2023/10/02/2023.09.29.560142

[13] E. Garzón, R. Golman, Z. Jahshan, R. Hanhan, N. Vinshtok-Melnik, M. Lanuzza, A. Teman, and L. Yavits, "Hamming distance tolerant content-addressable memory (hd-cam) for dna classification," *IEEE Access*, vol. 10, pp. 28 080–28 093, 2022.

[14] R. Hanhan, E. Garzón, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits, "Edam: edit distance tolerant approximate matching content addressable memory," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 495–507. [Online]. Available: https://doi.org/10.1145/3470496.3527424

[15] I. Merlin, E. Garzón, A. Fish, and L. Yavits, "Diper: Detection and identification of pathogens using edit distance-tolerant resistive cam," *IEEE Transactions on Computers*, vol. 73, no. 10, pp. 2463–2473, 2024.

[16] M. Khalifa, B. Hoffer, O. Leitersdorf, R. Hanhan, B. Perach, L. Yavits, and S. Kvatinsky, "Clapim: Scalable sequence classification using processing-in-memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 9, pp. 1347–1357, 2023.

[17] T. Shahroodi, M. Zahedi, A. Singh, S. Wong, and S. Hamdioui, "Krakenonmem: a memristor-augmented hw/sw framework for taxonomic profiling," in *Proceedings of the 36th ACM International Conference on Supercomputing*, 2022, pp. 1–14.

[18] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-throughput sequencing technologies," *Molecular Cell*, vol. 58, no. 4, pp. 586–597, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1097276515003408

[19] A. Rhoads and K. F. Au, "Pacbio sequencing and its applications," *Genomics, Proteomics & Bioinformatics*, vol. 13, no. 5, pp. 278–289, 11 2015. [Online]. Available: https://doi.org/10.1016/j.gpb.2015.08.002

[20] M. Jain, H. E. Olsen, B. Paten, and M. Akeson, "The oxford nanopore minion: delivery of nanopore sequencing to the genomics community," *Genome Biology*, vol. 17, no. 1, p. 239, November 2016. [Online]. Available: https://doi.org/10.1186/s13059-016-1103-0

[21] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022283605803602

[22] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster, "Megan analysis of metagenomic data," *Genome research*, vol. 17, no. 3, pp. 377–386, 2007.

[23] A. Brady and S. L. Salzberg, "Phymm and phymmbl: metagenomic phylogenetic classification with interpolated markov models," *Nature methods*, vol. 6, no. 9, pp. 673–676, 2009.

[24] G. L. Rosen, E. R. Reichenberger, and A. M. Rosenfeld, "Nbc: the naive bayes classification tool webserver for taxonomic classification of metagenomic reads," *Bioinformatics*, vol. 27, no. 1, pp. 127–129, 2011.

[25] H. Lu, F. Giordano, and Z. Ning, "Oxford nanopore minion sequencing and genome assembly," *Genomics, proteomics & bioinformatics*, vol. 14, no. 5, pp. 265–279, 2016.

[26] X. Xu, Y. Deng, J. Ding, Q. Tang, Y. Lin, X. Zheng, and T. Zhang, "High-resolution and real-time wastewater viral surveillance by nanopore sequencing," *Water Research*, vol. 256, p. 121623, 2024.

[27] R. M. Leggett, C. Alcon-Giner, D. Heavens, S. Caim, T. C. Brook, M. Kujawska, S. Martin, N. Peel, H. Acford-Palmer, L. Hoyles *et al.*, "Rapid minion profiling of preterm microbiota and antimicrobial-resistant pathogens," *Nature microbiology*, vol. 5, no. 3, pp. 430–442, 2020.

[28] K. Loit, K. Adamson, M. Bahram, R. Puusepp, S. Anslan, R. Kiiker, R. Drenkhan, and L. Tedersoo, "Relative performance of minion (oxford nanopore technologies) versus sequel (pacific biosciences) third-generation sequencing instruments in identification of agricultural and forest fungal pathogens," *Applied and Environmental Microbiology*, vol. 85, no. 21, pp. e01 368–19, 2019.

[29] G. V. Radhakrishnan, N. M. Cook, V. Bueno-Sancho, C. M. Lewis, A. Persoons, A. D. Mitiku, M. Heaton, P. E. Davey, B. Abeyo, Y. Alemayehu *et al.*, "Marple, a point-of-care, strain-level disease diagnostics and surveillance tool for complex fungal pathogens," *BMC biology*, vol. 17, no. 1, p. 65, 2019.

[30] D. Sheka, N. Alabi, and P. M. Gordon, "Oxford nanopore sequencing in clinical microbiology and infection diagnostics," *Briefings in bioinformatics*, vol. 22, no. 5, p. bbaa403, 2021.

[31] E. M. de Vries, N. O. I. Cogan, A. J. Gubala, P. T. Mee, K. J. O'Riley, B. C. Rodoni, and S. E. Lynch, "Rapid, in-field deployable, avian influenza virus haemagglutinin characterisation tool using minion technology," *Scientific reports*, vol. 12, no. 1, p. 11886, 2022.

[32] B. L. Brown, M. Watson, S. S. Minot, M. C. Rivera, and R. B. Franklin, "Minion™ nanopore sequencing of environmental metagenomes: a synthetic approach," *Gigascience*, vol. 6, no. 3, p. gix007, 2017.

[33] J. Lu, N. Rincon, D. E. Wood, F. P. Breitwieser, C. Pockrandt, B. Langmead, S. L. Salzberg, and M. Steinegger, "Metagenome analysis using the kraken software suite," *Nature protocols*, vol. 17, no. 12, pp. 2815–2839, 2022.

[34] Y. Harary, P. Snapir, S. S. Tov, C. Kruphman, E. Rechef, Z. Jahshan, E. Garzón, and L. Yavits, "Gcoc: A genome classifier-on-chip based on similarity search content addressable memory," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 19, no. 3, pp. 484–495, 2025.

[35] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[36] K. J. Schultz, "Content-addressable memory core cells a survey," *Integration*, vol. 23, no. 2, pp. 171–188, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926097000217

[37] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1311–1330, 2015.

[38] M. Irfan, Z. Ullah, and R. C. C. Cheung, "Zi-cam: A power and resource efficient binary content-addressable memory on fpgas," *Electronics*,

vol. 8, no. 5, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/5/584

[39] X. Wang, Y. Yang, and M. Shang, "A novel content addressable memory based on hybrid memristor-cmos architecture," in *2018 37th Chinese Control Conference (CCC)*, 2018, pp. 8502–8506.

[40] C. Li, C. Graves, X. Sheng, D. Miller, M. Foltin, G. Pedretti, and J. P. Strachan, "Analog content-addressable memories with memristors," *Nature Communications*, vol. 11, 04 2020.

[41] A. F. Laguna, H. Gamaarachchi, X. Yin, M. Niemier, S. Parameswaran, and X. S. Hu, "Seed-and-vote based in-memory accelerator for dna read mapping," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[42] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[43] N. E. Jerger, T. Krishna, and L.-S. Peh, *On-chip networks*. Springer Nature, 2022.

[44] T. J. LeBlanc, M. L. Scott, and C. M. Brown, "Large-scale parallel programming: experience with bbn butterfly parallel processor," *SIGPLAN Not.*, vol. 23, no. 9, p. 161–172, Jan. 1988. [Online]. Available: https://doi.org/10.1145/62116.62131

[45] P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a switch for network on chip applications," in *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, 2003, pp. V–V.

[46] H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel, "Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6.

[47] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 172–182.

[48] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, "Bts: an accelerator for bootstrappable fully homomorphic encryption," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 711–725. [Online]. Available: https://doi.org/10.1145/3470496.3527415

[49] R. M. Kuhn, D. Haussler, and W. J. Kent, "The ucsc genome browser and associated tools," *Briefings in bioinformatics*, vol. 14, no. 2, pp. 144–161, 2013.

[50] H. Farzaneh, J. P. C. De Lima, M. Li, A. A. Khan, X. S. Hu, and J. Castrillon, "C4cam: A compiler for cam-based in-memory accelerators," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 164–177.

[51] M. Li, S. Liu, M. M. Sharifi, and X. S. Hu, "Camasim: A comprehensive simulation framework for content-addressable memory based accelerators," *arXiv preprint arXiv:2403.03442*, 2024.

[52] W. Choi, J. Park, H. Kim, C. Park, and T. Song, "Half-and-half compare content addressable memory with charge-sharing based selective match-line precharge scheme," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 17–18.

[53] E. W. Sayers, E. E. Bolton, J. R. Brister, K. Canese, J. Chan, D. C. Comeau, R. Connor, K. Funk, C. Kelly, S. Kim, T. Madej, A. Marchler-Bauer, C. Lanczycki, S. Lathrop, Z. Lu, F. Thibaud-Nissen, T. Murphy, L. Phan, Y. Skripchenko, T. Tse, J. Wang, R. Williams, B. W. Trawick, K. D. Pruitt, and S. T. Sherry, "Database resources of the national center for biotechnology information," *Nucleic Acids Res*, vol. 50, no. D1, pp. D20–D26, Jan. 2022.

[54] W. H. Organization, *WHO fungal priority pathogens list to guide research, development and public health action*. World Health Organization, 2022.

[55] W. Huang, L. Li, J. R. Myers, and G. T. Marth, "Art: a next-generation sequencing read simulator," *Bioinformatics*, vol. 28, no. 4, pp. 593–594, 2012.

[56] Y. Ono, M. Hamada, and K. Asai, "Pbsim3: a simulator for all types of pacbio and ont long reads," *NAR Genomics and Bioinformatics*, vol. 4, no. 4, p. lqac092, 2022.

[57] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-throughput sequencing technologies," *Molecular cell*, vol. 58, no. 4, pp. 586–597, 2015.

[58] K. D. Pruitt, T. Tatusova, and D. R. Maglott, "Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins," *Nucleic acids research*, vol. 35, no. suppl_1, pp. D61–D65, 2007.

[59] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017300755

[60] A. B. Kahng, B. Lin, and S. Nath, "Orion3. 0: A comprehensive noc router estimation tool," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 41–45, 2015.

[61] R. Kobus, A. Müller, D. Jünger, C. Hundt, and B. Schmidt, "Metacache-gpu: ultra-fast metagenomic classification," in *Proceedings of the 50th International Conference on Parallel Processing*, 2021, pp. 1–11.

[62] C. E. Graves, C. Li, X. Sheng, D. Miller, J. Ignowski, L. Kiyama, and J. P. Strachan, "In-memory computing with memristor content addressable memories for pattern matching," *Advanced Materials*, vol. 32, no. 37, p. 2003437, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.202003437