

# HDDDB: Efficient In-Storage SQL Database Search Using Hyperdimensional Computing on Ferroelectric NAND Flash

Quanling Zhao<sup>\*1</sup>, Yanru Chen<sup>\*1</sup>, Runyang Tian<sup>1</sup>, Sumukh Pinge<sup>1</sup>, Weihong Xu<sup>2</sup>, Augusto Vega<sup>3</sup>, Steven Holmes<sup>3</sup>, Saransh Gupta<sup>3</sup>, Tajana Rosing<sup>1</sup>

<sup>1</sup> UCSD, <sup>2</sup> EPFL, <sup>3</sup> IBM

{quzhao,yac054,r3tian,spinge,tajana}@ucsd.edu

weihong.xu@epfl.ch

ajvega@us.ibm.com,{Steven.Holmes,saransh}@ibm.com

## Abstract

Hyperdimensional Computing (HDC) encodes information and data into high-dimensional distributed vectors that can be manipulated using simple bitwise operations and similarity searches, offering parallelism, low-precision hardware friendliness, and strong robustness to noise. These properties are a natural fit for SQL database workloads dominated by predicate evaluation and scans, which demand low energy and low latency over large fact tables. Notably, HDC’s noise-tolerance maps well onto emerging ferroelectric NAND (FeNAND) memories, which provide ultra-high density and in-storage compute capability but suffer from elevated raw bit-error rates. In this work, we propose HDDDB, a hardware–software co-design that combines HDC with FeNAND multi-level cells (MLC) to perform in-storage SQL predicate evaluation and analytics with massive parallelism and minimal data movement. Particularly, we introduce novel HDC encoding techniques for standard SQL data tables and formulate predicate-based filtering and aggregation as highly efficient HDC operations that can happen in-storage. By exploiting the intrinsic redundancy of HDC, HDDDB maintains correct predicate and decode outcomes under substantial device noise (up to 10% randomly corrupted TLC cells) without explicit error-correction overheads. Experiments on TPC-DS fact tables show that HDDDB achieves up to 80.6× lower latency and 12,636× lower energy consumption compared to conventional CPU/GPU SQL database engines, suggesting that HDDDB provides a practical substrate for noise-robust, memory-centric database processing.

## 1 Introduction

Predicate evaluation—evaluating boolean conditions over rows of a data table—is the front door of almost every SQL database query: it determines which data flow to downstream operators and thus often dominates end-to-end cost [50]. In workloads over large fact tables, these predicates are typically applied as scans; the cost is dominated by data movement rather than computation. All values must be read, compared, and filtered, saturating memory bandwidth and wasting energy shuttling bytes that will mostly be discarded. This has motivated a long line of near-data and in-storage processing

proposals that offload filtering into or near storage devices to reduce host–storage traffic for data-intensive queries [2, 16, 21, 23, 24, 27].

Hyperdimensional Computing (HDC) provides a particularly suitable algorithmic foundation for such workloads. HDC represents symbols and data as high-dimensional, often low-precision (e.g., binary) vectors—hypervectors (HVs)—and manipulates them using simple bitwise operations and similarity search [5, 11, 17, 44]. These operations map naturally to wide, bit-parallel datapaths and can be implemented close to where data resides, making HDC an attractive fit for in- or near-storage processing of scan-heavy SQL predicates. As a result, HDC can deliver low-latency, low-energy evaluation of simple decisions over large SQL fact tables, while its distributed representations also offer inherent robustness to device-level noise. Recent work has mapped HDC primitives onto emerging memory and logic devices, but these efforts have largely targeted AI or specialized database (e.g., Mass spectrometry) workloads rather than general SQL database workloads [8, 19, 20, 31, 43].

Emerging ferroelectric NAND (FeNAND) flash offers an appealing substrate for HDC-based SQL predicate processing. By leveraging ferroelectrics in CMOS-compatible processes, FeNAND promises ultra-high density, low power, and high-speed, three-dimensional integration—properties that are attractive for database storage and in-storage compute capabilities [22]. However, FeNAND’s narrow sense margins and retention-induced shifts introduce non-negligible readout noise and bit errors, which can severely strain conventional ECC-centric designs. In this context, HDC’s parallel, noise-robust computation is a natural match for noisy but highly efficient FeNAND arrays, and predicate scans in SQL databases are exactly the kind of simple, massively parallel decisions that HDC accelerates well.

In this work, we bring these threads together and propose HDDDB, a hardware–software co-design that combines HDC and FeNAND for fast and energy-efficient in-storage SQL predicate evaluation and analytics with massive parallelism and minimal data movement. Our contributions are as follows:

(1) HDDDB introduces, to the best of our knowledge, the first method for encoding conventional SQL tables into binary HDC representations while preserving decodability back to the original information to ensure that the data can be read from the hypervectors. On top of these encodings, HDDDB provides predicate evaluation algorithms for both string and numerical conditions using only standard HDC primitives—bitwise operations and Hamming-distance-based similarity search—making it the first system to apply

<sup>\*</sup>Both authors contributed equally to this research.

Author’s Contact Information: Quanling Zhao<sup>\*1</sup>, Yanru Chen<sup>\*1</sup>, Runyang Tian<sup>1</sup>, Sumukh Pinge<sup>1</sup>, Weihong Xu<sup>2</sup>, Augusto Vega<sup>3</sup>, Steven Holmes<sup>3</sup>, Saransh Gupta<sup>3</sup>, Tajana Rosing<sup>1</sup>

<sup>1</sup> UCSD, <sup>2</sup> EPFL, <sup>3</sup> IBM

{quzhao,yac054,r3tian,spinge,tajana}@ucsd.edu, weihong.xu@epfl.ch, ajvega@us.ibm.com,{Steven.Holmes,saransh}@ibm.com

HDC to SQL database workloads and providing a general framework that can be extended to broader SQL CRUD operations.

(2) On the hardware side of Hddb, we propose a mapping from binary HDC representations to FeNAND triple-level cell (TLC) that packs multiple bits into each cell. This enables exploiting FeNAND’s intrinsic advantages to execute HDC operations where the data resides. We further propose an In-Storage-Processing (ISP) accelerator. Integrating 3D FeNAND dies with near-storage processors (NSPs) to minimize data movement, our architecture is engineered to efficiently execute predicate-based filtering and additional analytics such as common aggregation functions. Together, the HDC encoding, predicate algorithms, and ISP design convert FeNAND MLC from a noisy storage medium into an efficient substrate for robust, energy-efficient processing platform for SQL databases.

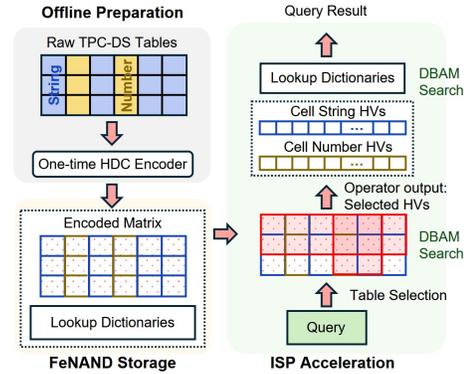
(3) We evaluate Hddb on TPC-DS [45] fact tables across multiple scaling factors, comparing against state-of-the-art (SOTA) SQL database engines and in-/near-storage baselines. Our results show that Hddb maintains correct predicate outcomes under substantial device noise (e.g., up to 10% randomly corrupted MLC cells) while running up to 80.6× faster and consuming 12,636× less energy.

## 2 Background and Related Works

**Hyperdimensional Computing (HDC):** HDC is an approach to representing symbols, sets, and relations with random, high-dimensional, and often low-precision codes (e.g.,  $10^3$ – $10^5$  binary bits), manipulated with a small set of algebraic operations [25, 38]. HDC has been applied to a wide range of applications, such as machine learning, associative memory, and many more [7, 15, 29, 49]. Previous works have shown HDC’s ability to encode different structures into vectors with information retrieval capability, such as graphs, ordered sets, or even functions [33, 36, 48]. Since information in HDC representation is distributed across many dimensions, small fractions of bit flips rarely affect the quality of results, yielding excellent performance in noisy conditions. This “redundancy by design” [44] contrasts with error-correcting codes that treat reliability separately from computation. In this work, we provide the first system that applies HDC to SQL-style database predicate evaluation and analytics.

**Ferroelectric NAND and In-Storage Processing:** As databases reach multi-terabyte and petabyte scales, the dominant cost in analytics is moving data from storage to compute [14]. ReRAM/PCM-based PIM approaches attack this bandwidth wall but lag 3D NAND in density and maturity [26, 28]. FeNAND instead offers ultra-high-density 3D NAND suitable for multi-terabyte fact tables [22], but its serial string structure, narrow sense margins, and retention drift raise raw bit-error rates [4]. These characteristics call for digital, page-parallel in-storage processing that can tolerate bit errors while exploiting FeNAND’s density—the regime that Hddb targets.

**Hardware Database Accelerators:** Beyond CPU and GPU database engines, prior work accelerates database queries using ASICs, FPGAs, and computational storage: Q100-style processors implement queries in custom datapaths [47], FPGA systems compile SQL to reconfigurable fabrics [10, 30], and SmartSSD-like devices attach FPGAs or embedded cores to SSDs [9]. These designs generally assume error-free memories and focus on mapping operators onto reliable DRAM, HBM, or flash. Hddb takes a different approach:



**Figure 1: End-to-end Hddb predicate evaluation pipeline and acceleration targets**

rather than adding an accelerator beside conventional NAND, it co-designs HDC encodings and FeNAND-resident in-storage processing so that the noisy flash array itself becomes the predicate engine. This cuts data movement and error-correction overhead, making predicate processing faster and more energy-efficient.

## 3 Hddb Algorithm Design

In SQL, predicate evaluation is invoked whenever the system evaluates Boolean conditions in clauses such as WHERE, JOIN ON, and HAVING, with WHERE-based filtering being the most common case. Because almost every analytic query includes one or more such predicates, and they are often applied as full-table scans, predicate evaluation frequently becomes the main bottleneck in both latency and energy [13, 14]. Conceptually, such queries reduce to evaluating a Boolean predicate on each row of a table to produce a selection mask, then projecting the required columns for the rows marked true. In practice, this is dominated by scanning: reading large volumes of data, comparing values against constants or ranges, forming a mask, and forwarding only selected rows. The overall dataflow of Hddb is shown in Figure 1. In Hddb, we represent data tables as long binary HVs and formulate common predicate evaluations as HDC primitives that can be executed directly in FeNAND storage.

**HDC Primitives:** We work in the standard binary HDC setting where an HV is a long bitstring. For a large dimension, independently sampled random HVs are nearly orthogonal: their Hamming distance concentrates, so flipping a modest fraction of bits barely changes relative similarity and yields noise-robustness and capacity. We use three classic HDC primitives. *Bind* combines two HVs with elementwise XOR ( $\otimes$ ), effectively “tagging” a value with a key and supporting approximate unbinding via  $(a \otimes b) \otimes a \approx b$ . *Bundle* ( $\oplus$ ) superposes multiple HVs by component-wise majority, forming a set-like representation that preserves membership in a distributed way. Finally, *associative recall* compares a (possibly noisy) query HV against a dictionary using Hamming similarity and returns the nearest match. Together, these operations let us encode, store, and access SQL tables in the HDC domain.

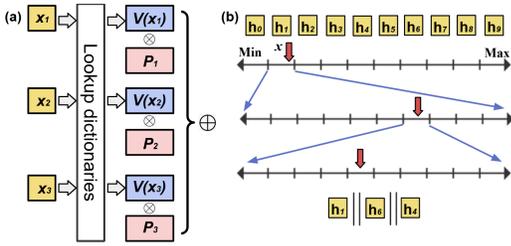


Figure 2: Mapping (a) String data (b) Numeric data into HV

### 3.1 Mapping SQL Table into Hypervectors

In SQL tables, every column behaves as either categorical or ordered data, which maps to “string-like” vs “numeric-like” for predicate semantics. For that reason, we propose different HDC encoding methods for string data and numerical data as shown in Fig 2. To encode a string  $x = x_1x_2 \cdots x_L$  over an alphabet  $\Sigma$  into a single binary HV, we first assign each symbol an i.i.d. random code  $V : \Sigma \rightarrow \{0, 1\}^D$  and each position a (near-orthogonal) positional HV  $P_i \in \{0, 1\}^D$ . Each character is bound to its position, and the string HV is the bundle:  $S(x) = \oplus (V(x_1) \otimes P_1, \dots, V(x_L) \otimes P_L)$ . This forms a decodable structure where the symbol at any position can be recovered with high probability via:

$$x_i = \arg \min_{s \in \Sigma} \text{hamming}(S(x) \otimes P_i, V(s)) \quad (1)$$

Intuitively, after bundling, the probe for position  $i$  ( $S(x) \otimes P_i$ ) is closer to the true symbol HV  $V(x_i)$  than any other symbols by a positive similarity margin. Since the string can vary in length, we use a special termination symbol to indicate the end of a string.

In SQL databases, numeric columns behave differently from strings because their domain is ordered and often wide, so we need encodings that preserve order and make range comparisons cheap. The usual HDC approach cuts the number line into bins and each bin gets a random HV. This works coarsely but breaks down at database scale: to avoid false merges across a large span, millions of bins are needed, which explodes dictionary size and makes numerical predicates expensive.

We encode a numeric value  $x \in [a, b]$  with a recursive multi-resolution scheme that reuses a small dictionary of  $m$  bin HVs across  $n$  levels. At level 1 we map  $x$  to its coarse bin index  $j_1(x) = \lfloor m \frac{x-a}{b-a} \rfloor$ ; at each deeper level  $\ell > 1$  we subdivide only the previously chosen bin and take  $j_\ell(x) = \lfloor m \frac{x - x^{(\ell-1)}}{(b-a)/m^{\ell-1}} \rfloor$  with  $x^{(\ell)} = x^{(\ell-1)} + \frac{b-a}{m^\ell} j_\ell(x)$ . Each digit  $j_\ell(x) \in \{0, \dots, m-1\}$  selects one HV from a shared dictionary  $\{\mathbf{h}_0, \dots, \mathbf{h}_{m-1}\}$ , and the final encoding is the concatenation of level HVs  $E_{\text{num}}(x) = [\mathbf{h}_{j_1(x)} \parallel \mathbf{h}_{j_2(x)} \parallel \dots \parallel \mathbf{h}_{j_n(x)}]$ . This realizes  $m^n$  distinct finest bins using only  $m$  HVs and  $n$  segments (memory  $O(mD)$  vs.  $O(m^n D)$  for naïve HDC method). This yields order-preserving numerical encoding that supports comparisons with simple HDC primitives, decodes cleanly, and is far more memory- and compute-efficient than a naïve HDC scheme that materializes one HV per fine bin.

### 3.2 Predicate Evaluation & Decode

For string predicates, the string encoder plays a hash-like role: it maps a string to an HV such that identical strings map to the same (or highly similar if under noise) HV, whereas different strings land near the expected hamming distance between two random HVs.

Effectively, for exact string matching predicates, the decision is whether a row’s HV is sufficiently similar to the query HV (with a predefined threshold), a check realized entirely with HDC primitives—so false matches are increasingly unlikely with larger dimension, and predicate evaluation results emerge without decoding. To evaluate a numerical predicate, we perform associative recall over the bin dictionary per level: for each segment of the stored numeric code, run Hamming nearest-neighbor against the  $m$  bin HVs to recover the level’s bin index. The recovered index sequence is then compared to the query’s index sequence. Both string and numeric predicates are inherently noise-robust in our HDC design. Bit flips on encoded tables only erode similarity search slightly, but not decisions, so predicate evaluation remains unaffected. This robustness is a direct consequence of distributed, high-dimensional HVs. After obtaining the predicate results, selected HVs can be decoded back to the original information via associative recall with the symbol dictionary.

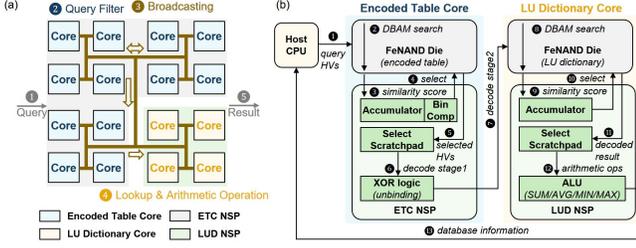
## 4 HDDB In-Storage Accelerator

In this section, we detail the hardware–software co-design of the HDDB in-storage accelerator. This architecture is built to efficiently execute the HDC predicate evaluation algorithms presented in Section 3 and supports additional analytics such as common aggregation functions  $\{\text{COUNT}, \text{SUM}, \text{AVG}, \text{MIN}, \text{MAX}\}$ . We first introduce the core organization and the overall system dataflow. We then describe the Dual Boundary Approximate Matching (DBAM) in-situ approximate searching technique, the design of specialized peripheral processors, and the column-wise data mapping strategy.

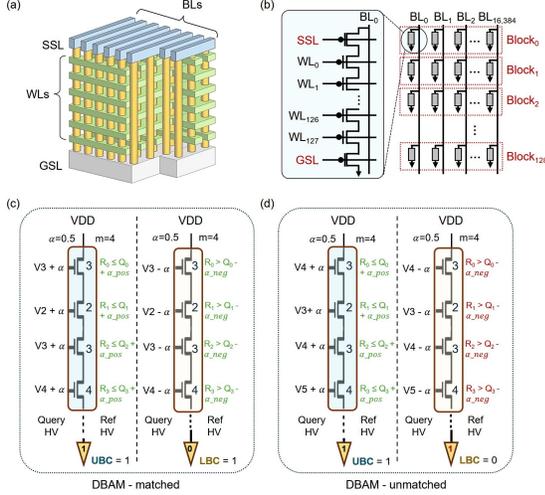
### 4.1 System Overview

The HDDB ISP acceleration system is a hardware–software co-design utilizing 3D heterogeneous integration to fuse 3D FeNAND storage tiles with specialized NSPs to minimize data movement. Driven by the high-throughput demands of hyperdimensional vector scans, this vertical stacking circumvents external I/O bottlenecks by exposing the massive internal bandwidth of the memory array directly to the compute logic. This interconnect links dedicated encoded table and lookup (LU) dictionary cores to their corresponding shared NSPs (ETC NSP and LUD NSP). The reconfigurable system-level architecture, illustrated in Figure 3(a), uses an H-tree network [39, 41], managing query broadcasting and data routing. The table cores store HDC-encoded database tables, while the dictionary cores hold LU dictionaries for decoding. The capacity ratio between two cores is configurable, as dictionary cores are often much smaller than table cores. Section 4.2 details the specific circuit-level implementation of heterogeneous peripherals.

The predicate evaluation dataflow, detailed in Figure 3(b), begins at the host CPU. The host sends the query HVs and encoding plan to the encoded table cores (Step ①). These cores execute a high-throughput in-situ DBAM search [31] against the stored encoded table HVs (Step ②): For string predicates, the query HV is directly compared against the target column. Numerical predicate search is more complex, leveraging the 4-level (100 bins/level) encoding. An NSP accumulator processes the resulting similarity scores (Step ③). For strings, this score directly identifies matching rows (Step ④). For numerical queries, the scores are used to find the bin index of



**Figure 3: Overview of HDDDB/ISP (a) Reconfigurable H-tree connected architecture (b) Predicate evaluation dataflow**



**Figure 4: FeNAND-based Dual Boundary Approximate Matching (DBAM) (a) 3D FeNAND array structure (b) 3D FeNAND planes (c) DBAM matched case (d) DBAM unmatched case**

each cell; these indices are then compared externally to the query’s target bin indices to select the correct rows (Step 4). The selected cell HVs are buffered in a select scratchpad (Step 5). An NSP parallel XOR logic unit then performs the decode stage 1, the HDC unbinding operation, on these selected HVs (Step 6). The unbound HVs are forwarded to the LU dictionary core for the decode stage 2 (Step 7). This core executes another DBAM search (Step 8), comparing the intermediate unbound HVs against the LU dictionary. A second NSP accumulator processes these similarity scores (Step 9) to select the final decoded results (Step 10), storing them in the LU core’s scratchpad (Step 11). If aggregation is required, an NSP ALU performs arithmetic operations on these results (Step 12). The final database information returns to the host CPU (Step 13).

## 4.2 In-situ Approximate Matching and Peripherals Design

**In-situ Approximate Matching:** The fundamental operation for predicate evaluation in HDDDB is the HDC similarity search (filtering). Unlike crossbar-based PIM architectures (e.g., ReRAM, PCM) that leverage parallel analog current summation, 3D NAND employs serial string connectivity that precludes standard matrix accumulation. To enable high-throughput search despite this structural constraint, we utilize the DBAM technique [31]. As shown in Figure 4(a), the FeNAND tile is organized into multiple blocks, and

bitlines (BLs) are shared across blocks. DBAM is engineered to exploit the serial connectivity of 3D NAND strings (Figure 4(b)) and maximize storage density by packing HVs into FeNAND Triple-Level Cells (TLC). Specifically, we map every three encoded bits to one cell (8 levels) using a fixed bijection Gray code. This encoding guarantees that threshold-voltage shifts between adjacent levels, a common issue in FeNAND, cause at most one bit to change. It thus preserves the similarity needed for reliable in-storage computation. The system reads these packed values by activating  $k$  wordlines (WLs) simultaneously, comparing  $k$  stored reference elements ( $r_i$ ) to  $k$  query elements ( $q_i$ ) in a single operation. A configurable tolerance margin ( $\alpha = 0.5$ ) provides robustness against  $V_{TH}$  shifts and other device non-idealities.

The search operation completes in exactly two sensing cycles. The Upper Bound Check (UBC) applies a WL voltage of  $q_i + \alpha_{pos}$ . Due to the serial string, current flows only if all  $k$  cells are below this bound, implementing a parallel AND operation.

$$UBC_j = \prod_{i=k}^{kj+k-1} [r_i \leq q_i + \alpha_{pos}] \quad (2)$$

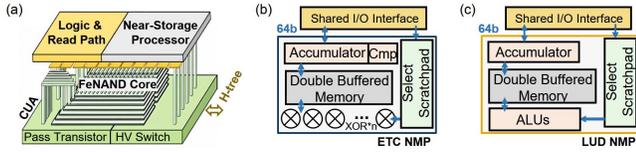
The Lower Bound Check (LBC) applies a WL voltage of  $q_i - \alpha_{neg}$  and blocks current if any cell exceeds the lower threshold.

$$LBC_j = 1 - \prod_{i=k}^{kj+k-1} [r_i < q_i - \alpha_{neg}] \quad (3)$$

As shown in Figure 4(c-d), these checks yield binary results per  $k$ -subset, which are aggregated by the NSP into a final similarity metric:  $Score = \sum_j (UBC_j + LBC_j)$ . This approach with  $k$ -way parallelism replaces conventional iterative MLC sensing [18, 32] with negligible hardware overhead. As demonstrated in FeNOMS [31] analysis, we select  $k = 8$  to achieve an optimal balance between parallel throughput and search accuracy. To support this array-level parallelism and process the resulting high-speed data streams, we implement a dedicated heterogeneous peripheral architecture.

**Peripheral Design:** The HDDDB/ISP peripheral design is a heterogeneous and reconfigurable architecture. The storage cores, shown in Figure 5(a), use 3D heterogeneous integration to stack the storage array atop a 65nm CMOS Under Array (CUA) for high-voltage management while placing performance-critical logic in a separate 7nm advanced logic layer. This upper layer integrates high-speed read path circuits including BL decoders and sense amplifiers alongside a dedicated NSP for each tile. By co-locating the NSP with the read circuitry, the design minimizes interconnect latency and maximizes internal bandwidth utilization. Each core is connected by the H-tree interconnect, providing low-latency, high-bandwidth data transfer between storage and processing.

While every core possesses an NSP, the specific logic configuration adapts to the core type to optimize area efficiency. The ETC NSP, detailed in Figure 5(b), executes the first stage of processing. It integrates a 7-bit bin index comparator to finalize numerical predicate selection by comparing the 4-level bin indices. It also contains a parallel bitwise XOR array for the HDC unbinding operation. The double-buffered memory structure concurrently supports intermediate result accumulation and output transfers. The LUD NSP, shown in Figure 5(c), contains a dedicated ALU that supports parallel SUM, AVG, MIN, and MAX operations for final aggregation.



**Figure 5: Design of (a) Heterogeneous integration (b) Encoded table near-storage processor (NSP) (c) LU dictionary NSP**

This decoupled architecture performs massive filtering in-situ. The filtered data is then processed in the NSPs for decoding and aggregation, enabling in-storage SQL analytics with massive parallelism and minimal data movement.

### 4.3 HDDB In-Storage Mapping and Scheduling

Figure 6(a) illustrates the multi-stage predicate evaluation algorithm. An HDDB subtable comprises  $M$  columns and  $N$  rows. The number of rows  $N$  scales with the Scale Factor (SF), while the column count  $M$  remains constant. The flow begins by selecting the target column based on a predetermined encoding plan. This plan serves as a schema map, specifying whether each column utilizes string or numeric encoding and delineating the dimension boundaries within the row vector. The in-storage DBAM engine then performs a parallel similarity search. This search adapts to the predicate type: string predicates involve direct HV comparison, whereas numerical predicates use the multi-level query HV to determine the correct bin index for each cell. Selected HVs are then unbound and used in a second DBAM search against the LU dictionary. This final lookup retrieves the decoded key values.

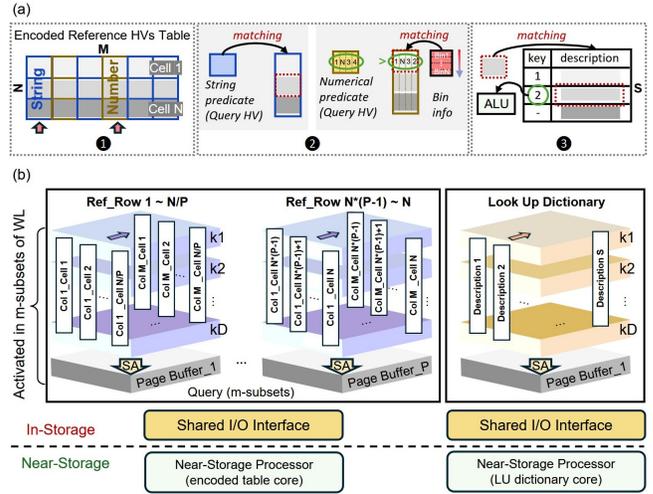
The HDDB system maps offline encoded tables onto the FeNAND array using a column-wise layout, as Figure 6(b) illustrates. Each cell of a given column is mapped onto a vertical FeNAND string. The data for a full column, containing up to 1.5 million rows (for an  $SF = 1$  table), is evenly distributed across multiple planes. This layout achieves high plane-level parallelism and scales easily with SF changes. LU dictionaries are stored separately in dedicated LU dictionary core using this same mapping strategy. This workflow utilizes two specialized NSPs. The ETC NSP accumulates the initial scores and performs the HDC XOR unbinding operation. The LUD NSP handles the final dictionary search and executes aggregation functions such as SUM, AVG, MIN, or MAX within.

## 5 Evaluation

Because large fact tables dominate memory footprint and scan cost in SQL workloads, we evaluate HDDB on TPC-DS fact table. In Section 5.1, we generate random string and numerical predicates to test that HDDB can perform predicate evaluation reliably and decode information accurately, even in the presence of substantial hardware noise. In Section 5.2, using these predicates, we construct two families of queries: (i) pure filter queries and (ii) filter+aggregation queries. For each family, we instantiate 1000 queries as common inputs, and use them to compare HDDB against SOTA database engines in terms of per query latency and energy consumption.

### 5.1 HDDB Algorithm Evaluation

We first evaluate the HDDB encoding and predicate algorithms in isolation. For encoding numerical data, we use an encoder with



**Figure 6: Mapping and scheduling (a) String and numerical cases (b) Data layout for encoded table and LU dictionary**

$m = 100$  bins per level and  $n = 4$  recursive levels (effectively  $100^4$  finest bins). To study how representation size affects both predicate correctness and decode accuracy, we sweep the per-row dimensionality between 70k and 110k bits. To simulate device noise, we map binary bits to TLC cells and, for a given noise level, randomly choose that fraction of cells and shift their programmed state by  $\pm 1$  level, reflecting the fact that most retention and sense-margin failures in MLC NAND appear as adjacent-level shifts. This model fits comfortably with prior work[31], which models cell-level non-idealities as Gaussian threshold-voltage noise,  $V_T \sim \mathcal{N}(0, 0.2^2)$  V within a 6.5 V window; the resulting  $V_T$  perturbations effectively induce these  $\pm 1$ -level errors near decision boundaries. For each row dimensionality, we encode the TPC-DS fact table into hyper-vectors, generate random string and numerical predicates, and run HDDB’s predicate evaluation algorithm to determine which rows satisfy each predicate. For every predicate, we measure quality as (i) whether the predicate outcome matches exactly the ground-truth SQL execution and (ii) whether we can correctly decode the original contents of the selected rows from their hyper-vectors. Figure 7(a) shows that under 100K to 110K dimensionality, HDDB’s predicate evaluation remains perfectly accurate up to a 0.15 noise level. In contrast, Figure 7(b) shows that decoding the original cell information is more sensitive to noise, but still able to achieve perfect decoding at a 0.1 noise level. This shows that increasing dimensionality improves HDDB’s ability to process predicates correctly and to reliably recover the original data for any downstream analytics.

### 5.2 HDDB In-Storage Accelerator Evaluation

**5.2.1 Experimental Setup. Baselines:** We compare our HDDB against widely used SQL engines: DuckDB [34], PostgreSQL [42], HeavyDB [1, 37], and a Dask-cuDF backed SQL engine [3, 35]. These represent SOTA or popular choices for SQL workloads on commodity platforms. Baselines were executed on platforms featuring the Intel Ultra7 155H/DDR5-5600 (16GB) and the NVIDIA RTX 4070/GDDR6 (8GB). To better contextualize HDDB relative to hardware database accelerators beyond CPU/GPU, we also report comparisons with recent accelerator-based systems, including

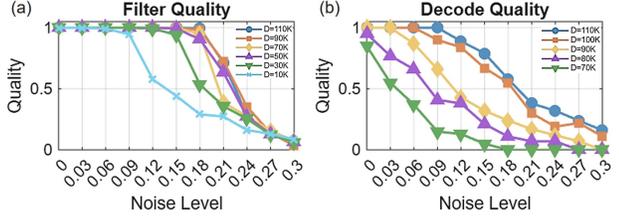


Figure 7: Hddb (a) Predicate evaluation (b) Decode quality

SQL2FPGA [27], Darwin [21], and pimDB [2]. For system evaluation in this section, we use a per-row dimension of 110K.

**Hardware Configurations:** We model the 3D FeNAND core based on the 3D NAND architecture [12, 40] with 128 WLS, 16,384 BLs, and 128 blocks, assuming a FeNAND z-scaling ratio of  $k = 4$ . This scaling ratio  $k = 4$ , derived from in-house modeling and inference from studies on ferroelectric material properties [6, 22, 46], is applied for the RC delay modeling and analysis to compare with conventional 3D NAND. System-level parameters are also detailed in Table 1.

Table 1: Hardware configurations of Hddb/ISP accelerator

3D FeNAND Core Parameters			
Parameter	WL = 128, BL = 16,384, #Blocks = 128, #Tiles = 1, 1 NSP/core		
SSL, WL, BL Pitch	220 nm, 500 nm, 100 nm	Capacity	3GB per TLC Core
GSL, SSL, Blocks	16, 16, 128	FeNAND z-scaling	$k = 4$
Plane area (mm <sup>2</sup> )	0.738198	WL, SSL, BL Read	1 V, 4.5 V, 0.2 V
Read Latency	50–100 $\mu$ s/page read	Read Energy	2.28 pJ/bit
Write Latency	0.2–0.8 ms/page program	Write Energy	34.2 pJ/bit
3D FeNAND System Parameters			
Organization	1 FeNAND flash controller/2TB	Interconnect	4 core per H-trees
NSP (Encoded table core)			
XOR logic	42 parallel bitwise XOR logic	Select Scratchpad	20 KB
Double buffered SRAM	2 KB (same as LUD NSP)	7-bit Bin Cmp.	5-parallel comparator
NSP (LU dictionary core)			
ALU	2 units (ADD/AVG/MIN/MAX)	Select Scratchpad	20 KB

**Evaluation Platform:** We develop an in-house simulator for the Hddb/ISP system. We model the 3D FeNAND arrays and logic & read path circuits in SystemVerilog to ensure device-level accuracy, and we synthesize the Register Transfer Level (RTL) design using Synopsys Design Compiler with a 64 nm CMOS PDK (1 V) and a 7nm CMOS advanced PDK (0.7V) at 1 GHz.

**5.2.2 Comparison with SOTA Database Engine.** We evaluate Hddb against two configurations (ISP-150GB and ISP-1TB) and four SOTA database engines. These include two GPU-accelerated systems (dask-cuDF-SQL [3, 35], HeavyDB [1, 37]) and two high-performance CPU-based systems (PostgreSQL [42], DuckDB [34]). Figure 8 shows the latency and energy consumption on a logarithmic scale across TPC-DS filter and aggregation workloads at SF=1, 5, and 10.

Figure 8(a) shows that both Hddb configurations consistently achieve the lowest latency across all workloads and scale factors. The Hddb (ISP-1TB) configuration achieves up to 78.7 $\times$  speedup over PostgreSQL and 19.8 $\times$  over HeavyDB for the SF=1 filter workload. The energy efficiency improvements shown in Figure 8(b) are more significant. Hddb consumes orders of magnitude less energy than any baseline. The Hddb (ISP-150GB) system is up to 12,636 $\times$  more energy-efficient than PostgreSQL. Meanwhile, the 1TB configuration saves 3,258 $\times$  the energy consumed by HeavyDB for SF=1 filter, demonstrating the benefits of eliminating data movement.

Table 2 further contextualizes Hddb against other hardware-accelerated database engines. While systems like SQL2FPGA, Darwin, and pimDB rely on FPGA co-processing or DRAM-based PIM,

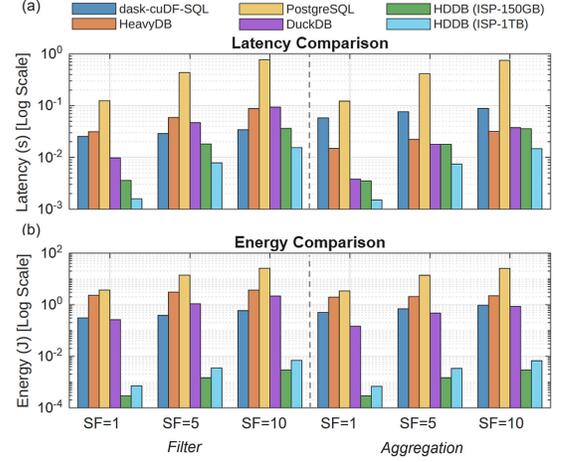


Figure 8: Performance comparison over SOTA database engine on average (a) Latency (b) Energy consumption

Hddb is a true ISP system on MLC FeNAND flash. This approach moves computation directly to the data, eliminating the I/O bottleneck rather than just reducing it. The resulting 80.6 $\times$  speedup of our system is highly competitive with the performance gains reported by these other specialized hardware solutions.

Table 2: Comparison of SOTA hardware-accelerated database engines (SF = 1)

Feature	Hddb (Ours)	SQL2FPGA [27]	Darwin [21]	pimDB [2]
Hardware	MLC Flash-based ISP	CPU-FPGA (Intel Xeon-Xilinx Alveo)	GDDR6 DRAM PIM	UPMEM DRAM PIM
Evaluation & Baselines				
Datasets	TPC-DS (Filter/Agg.)	TPC-H, TPC-DS (Q1-5)	TPC-H, Basic Query Operators	TPC-H (Scan/Select)
Baselines	SOTA: See Figure 8	Big Data Frmwk: Apache Spark	Trad. Impl: Optimized C++	Trad. Impl: Optimized C++
Performance & Efficiency				
Latency Speedup	Up to 80.6 $\times$	$\sim$ 10.1 $\times$	4 $\times$ –44 $\times$	Up to 68 $\times$
Energy Efficiency	88 $\times$ –12.6k $\times$	$\sim$ 9.2 $\times$	$\sim$ 7 $\times$	Not Reported

**5.2.3 Area and Power Analysis. Area and Power Breakdown:**

Table 3 presents the post-synthesis area and power breakdown for the ETC and LUD NSPs synthesized at 7nm, 1GHz, 0.7V. On-chip storage components, specifically the select scratchpad and double buffered SRAM, occupy  $\sim$ 80% of the total footprint in both processors. The specialized compute units remain compact: The 7-bit bin comparator and parallel XOR array in the ETC NSP together utilize less than 0.6% of the area while the ALUs in the LUD NSP require less than 1.9%. This demonstrates that offloading predicate logic to near-storage processors incurs minimal hardware overhead. **Overhead Analysis:** System-level analysis further validates that the logic fits easily within the system footprint: a single 3D FeNAND storage tile consumes 1.2964 mW and occupies 0.738 mm<sup>2</sup> [31], whereas the NSP occupies only  $\sim$ 0.013 mm<sup>2</sup>. Using the storage tile as a baseline, the NSP logic incurs a vertical area overhead of just 1.8% relative to the tile size. This indicates that the logic is densely integrated within the vertical shadow of the storage array without expanding the total chip footprint and ensures zero loss in FeNAND memory capacity. At the package level, the SM2508 controller adds a fixed power overhead of 3.5 W within a 225 mm<sup>2</sup> footprint; the scalable design results in a total system area of 75.1 mm<sup>2</sup> for a 150GB system and 512.6 mm<sup>2</sup> for a 1TB implementation.

**Table 3: Area and Power Breakdown of ETC NSP & LUD NSP**

Component	ETC NSP Logic		LUD NSP Logic	
	Area ( $\mu\text{m}^2$ )	Power (mW)	Area ( $\mu\text{m}^2$ )	Power (mW)
Accumulator	2520 (19.25%)	18.5 (46.49%)	2520 (18.99%)	18.5 (43.10%)
Select Scratchpad	8043 (61.45%)	15 (37.70%)	8043 (60.62%)	15 (34.95%)
Double buffered SRAM	2457 (18.77%)	5 (12.57%)	2457 (18.52%)	5 (11.65%)
7-bit Bin Comparator	20.37 (0.16%)	0.412 (1.04%)	–	–
Parallel XOR Array	49.31 (0.38%)	0.879 (2.21%)	–	–
ALUs	–	–	248.53 (1.87%)	4.424 (10.31%)
<b>Total</b>	<b>13089.68</b>	<b>39.791</b>	<b>13268.53</b>	<b>42.924</b>

## 6 Conclusion

In this paper, we presented HDDB, a hardware–software co-design that, to our knowledge, is the first system to apply Hyperdimensional Computing (HDC) to SQL databases and to execute large-scale predicate evaluation directly inside noisy FeNAND storage. HDDB merges HDC’s massively parallel, noise-tolerant computation model with FeNAND’s ultra-high density and in-storage compute capability. Our evaluation on TPC-DS fact tables shows that HDDB preserves correct predicate outcomes under substantial device noise while achieving up to  $80.6\times$  lower latency and  $12,636\times$  lower energy consumption than CPU/GPU SQL engines. These results indicate HDDB offers an efficient, noise-robust substrate for memory-centric SQL workloads. More broadly, HDDB points to a new class of database accelerators that co-design data representations, query algorithms, and emerging memories, and can be extended to richer SQL operators and queries.

## 7 ACKNOWLEDGMENT

This work was supported in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA (SRC grant number - 2023-JU-3135). This work was also supported by NSF grants #2003279, #1911095, #2112167, #2052809, #2112665, #2120019, #2211386.

## References

[1] [n. d.]. HeavyDB. <https://docs.heavy.ai/overview/overview>.  
 [2] Arthur Bernhardt, Andreas Koch, and Iliia Petrov. 2023. Pimdb: From main-memory dbms to processing-in-memory dbms-engines on intelligent memories. In *Proceedings of the 19th International Workshop on Data Management on New Hardware*. 44–52.  
 [3] Nils Braun and contributors. [n. d.]. dask-sql: A Distributed SQL Engine for Dask DataFrames. <https://dask-sql.readthedocs.io/>.  
 [4] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 438–449.  
 [5] William Youngwoo Chung, Hamza Errahmouni Barkam, Tamoghno Das, and Mohsen Imani. 2025. Robust Reasoning and Learning with Brain-Inspired Representations under Hardware-Induced Nonlinearities. In *GLSVLSI’25*. 968–975.  
 [6] Dipjyoti Das, Hyeonwoo Park, Zekai Wang, Chengyang Zhang, Prasanna Venkatesan Ravindran, Chinsung Park, Nashrah Afroze, et al. 2023. Experimental Demonstration and Modeling of a Ferroelectric Gate Stack with a Tunnel Dielectric Insert for NAND Applications. In *IEDM’23* (San Francisco, USA). IEEE. doi:10.1109/IEDM45741.2023.10413697  
 [7] Pieter Dewulf, Michiel Stock, and Bernard De Baets. 2024. The hyperdimensional transform: a holographic representation of functions. *IEEE Journal of Selected Topics in Signal Processing* 19, 1 (2024), 3–18.  
 [8] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. 2022. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *GLSVLSI’22*. 281–286.  
 [9] Dina Fakhry, Mohamed Abdelsalam, M Watheq El-Kharashi, and Mona Safar. 2023. A review on computational storage devices and near memory computing for high performance applications. *Memories-Materials, Devices, Circuits and Systems* 4 (2023), 100051.  
 [10] Jian Fang, Yvo TB Mulder, Jan Hidders, Jinho Lee, and H Peter Hofstee. 2020. In-memory database acceleration on FPGAs: a survey. *The VLDB Journal* 29, 1

(2020), 33–59.  
 [11] Lulu Ge and Keshab K Parhi. 2020. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.  
 [12] Po-Kai Hsu, Weihong Xu, Tajana Rosing, and Shimeng Yu. 2024. An In-Storage Processing Architecture with 3D NAND Heterogeneous Integration for Spectra Open Modification Search. In *MEMSYS* (Alexandria, VA, USA) (*MEMSYS ’23*). ACM, New York, USA, Article 14, 7 pages.  
 [13] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K Sjoerd Mullender, Martin L Kersten, et al. 2012. MonetDB: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.* 35, 1 (2012), 40–45.  
 [14] Mohsen Imani, Saransh Gupta, Sahil Sharma, and Tajana Simunic Rosing. 2018. NVQuery: Efficient query processing in nonvolatile memory. *TCAD* 38, 4 (2018), 628–639.  
 [15] Mohsen Imani, John Messerly, Fan Wu, Wang Pi, and Tajana Rosing. 2019. A binary learning framework for hyperdimensional computing. In *DATE’19*. IEEE, 126–131.  
 [16] Insoon Jo, Duck-Ho Bae, Andre S Yoon, Jeong-Uk Kang, Sangyeun Cho, Daniel DG Lee, and Jaeheon Jeong. 2016. YourSQL: a high-performance database system leveraging in-storage computing. *Proceedings of the VLDB Endowment* 9, 12 (2016), 924–935.  
 [17] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.  
 [18] Jaeyoung Kang et al. 2023. Massively Parallel Open Modification Spectral Library Searching with Hyperdimensional Computing. In *PACT ’22*. ACM.  
 [19] Jaeyoung Kang, Minxuan Zhou, Weihong Xu, and Tajana Rosing. 2025. RelHDX: Hyperdimensional Computing for Learning on Graphs With FeFET Acceleration. *IEEE Trans. Comput.* (2025).  
 [20] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. 2020. In-memory hyperdimensional computing. *Nature Electronics* 3, 6 (2020), 327–337.  
 [21] Donghyuk Kim, Jae-Young Kim, Wontak Han, Jongsoo Won, Haerang Choi, Yongkee Kwon, and Joo-Young Kim. 2024. Darwin: A DRAM-Based Multi-Level Processing-in-Memory Architecture for Column-Oriented Database. *IEEE Transactions on Emerging Topics in Computing* (2024).  
 [22] Min-Kyu Kim et al. 2021. CMOS-compatible ferroelectric NAND flash memory for high-density, low-power, and high-speed three-dimensional memory. *Science Advances* 7, 8 (2021), eabe1341. doi:10.1126/sciadv.abe1341  
 [23] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, and Sang-Won Lee. 2011. Fast, energy efficient scan inside flash memory SSDs. In *Proceedings of the International Workshop on Accelerating Data Management Systems (ADMS)*.  
 [24] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, Sang-Won Lee, and Bongki Moon. 2016. In-storage processing of database scans and joins. *Information Sciences* 327 (2016), 183–200.  
 [25] Denis Kleyko, Dmitri A Rachkovskij, Evgeny Osipov, and Abbas Rahimi. 2022. A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations. *Comput. Surveys* 55, 6 (2022), 1–40.  
 [26] Yun Long, Taesik Na, and Saibal Mukhopadhyay. 2018. ReRAM-based processing-in-memory architecture for recurrent neural network acceleration. *TVLSI* 26, 12 (2018), 2781–2794.  
 [27] Alec Lu, Jahanvi Narendra Agrawal, and Zhenman Fang. 2024. Sql2fpga: Automated acceleration of sql query processing on modern cpu-fpga platforms. *ACM Transactions on Reconfigurable Technology and Systems* 17, 3 (2024), 1–28.  
 [28] Sparsh Mittal. 2018. A survey of ReRAM-based architectures for processing-in-memory and neural networks. *Machine learning and knowledge extraction* 1, 1 (2018), 75–114.  
 [29] Peer Neubert, Stefan Schubert, and Peter Protzel. 2019. An introduction to hyperdimensional computing for robotics. *KI-Künstliche Intelligenz* 33, 4 (2019), 319–330.  
 [30] Philippos Papaphilippou and Wayne Luk. 2018. Accelerating database systems using FPGAs: A survey. In *FPL’18*. IEEE, 125–1255.  
 [31] Sumukh Pinge, Ashkan Moradifiroozabadi, Keming Fan, Prasanna Venkatesan Ravindran, Tanvir H Pantha, Po-Kai Hsu, Zheyu Li, Weihong Xu, Zihan Xia, Flavio Ponzina, et al. 2025. FeNOMS: Enhancing Open Modification Spectral Library Search with In-Storage Processing on Ferroelectric NAND (FeNAND) Flash. *arXiv preprint arXiv:2510.10872* (2025).  
 [32] Sumukh Pinge, Weihong Xu, Wout Bittremieux, Niema Moshiri, Sang-Woo Jun, and Tajana Rosing. 2024. RapidOMS: FPGA-based Open Modification Spectral Library Searching with HD Computing. In *BIOCAS’24*.  
 [33] Prathyush Poduval, Haleh Alimohamadi, Ali Zakeri, Farhad Imani, M Hassan Najafi, Tony Givargis, and Mohsen Imani. 2022. Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning. *Frontiers in Neuroscience* 16 (2022), 757125.  
 [34] Mark Raasveldt and Hannes Mühleisen. 2019. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 international conference on management of data*. 1981–1984.  
 [35] RAPIDS Development Team. [n. d.]. Dask-cuDF: GPU-accelerated Dask DataFrames. <https://docs.rapids.ai/api/dask-cudf/stable/>. Accessed: 2025-11-18.

- [36] Netanel Raviv. 2024. Linear Codes for Hyperdimensional Computing. *Neural Computation* 36, 6 (2024), 1084–1120.
- [37] Christopher Root and Todd Mostak. 2016. MapD: A GPU-powered big data analytics and visualization platform. In *ACM SIGGRAPH 2016 Talks*. 1–2.
- [38] Kenny Schlegel, Peer Neubert, and Peter Protzel. 2022. A comparison of vector symbolic architectures. *Artificial Intelligence Review* 55, 6 (2022), 4523–4555.
- [39] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars. In *ISCA'16*. 14–26. doi:10.1109/ISCA.2016.12
- [40] Wonbo Shim, Hongwu Jiang, Xiaochen Peng, and Shimeng Yu. 2021. Architectural Design of 3D NAND Flash based Compute-in-Memory for Inference Engine. In *MEMSYS (Washington, DC, USA) (MEMSYS '20)*. Association for Computing Machinery, New York, NY, USA, 77–85. doi:10.1145/3422575.3422779
- [41] Jiahao Song, Xiyuan Tang, Bocheng Xu, Haikang Diao, Haoyang Luo, Zihan Wu, Yuan Wang, Runsheng Wang, and Ru Huang. 2025. A 1131-kb/mm<sup>2</sup> 14.0-to-53.3-TOPS/W 8-bit Analog-Assisted Digital Compute-in-Memory With Hybrid Local-Refresh eDRAM for Attention Computing. *IEEE Journal of Solid-State Circuits* (2025).
- [42] Michael Stonebraker and Lawrence A Rowe. 1986. The design of Postgres. *ACM Sigmod Record* 15, 2 (1986), 340–355.
- [43] Simon Thomann, Hong LG Nguyen, Paul R Genssler, and Hussam Amrouch. 2022. All-in-memory brain-inspired computing using fefet synapses. *Frontiers in Electronics* 3 (2022), 833260.
- [44] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. 2021. A theoretical perspective on hyperdimensional computing. *JAIR* 72 (2021), 215–249.
- [45] Transaction Processing Performance Council (TPC). 2015. TPC Benchmark<sup>®</sup> DS Standard Specification Version 3.2.0. [https://www.tpc.org/tpc\\_documents\\_current\\_versions/pdf/tpc-ds\\_v3.2.0.pdf](https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf).
- [46] Prasanna Venkatesan, Chinsung Park, Taeyoung Song, Lance Fernandes, Dipjyoti Das, Nashrah Afroze, Priyanka Gundlapudi Ravikumar, Mengkun Tian, Hang Chen, Winston Chern, et al. 2024. Disturb and its mitigation in Ferroelectric Field-Effect Transistors with Large Memory Window for NAND Flash Applications. *IEEE Electron Device Letters* (2024).
- [47] Lisa Wu, Andrea Lottarini, Timothy K Paine, Martha A Kim, and Kenneth A Ross. 2014. Q100: The architecture and design of a database processing unit. *ACM SIGARCH Computer Architecture News* 42, 1 (2014), 255–268.
- [48] Dehao Yuan, Furong Huang, Cornelia Fermüller, and Yiannis Aloimonos. 2023. Decodable and sample invariant continuous object encoder. *arXiv preprint arXiv:2311.00187* (2023).
- [49] Quanling Zhao, Anthony Hitchcock Thomas, Ari Brin, Xiaofan Yu, and Tajana Rosing. 2025. Bridging the Gap Between Hyperdimensional Computing and Kernel Methods via the Nyström Method. In *AAAI*, Vol. 39. 22813–22821.
- [50] Chen Zou and Andrew A Chien. 2020. Empowering architects and designers: A classification of what functions to accelerate in storage. *UChicago CS TR-2020-02* (2020).