

# LOGHD: Robust Compression of Hyperdimensional Classifiers via Logarithmic Class-Axis Reduction

Sanggeon Yun

Dept. of Computer Science  
University of California, Irvine  
Irvine, CA, USA  
sanggeoy@uci.edu

Hyunwoo Oh

Dept. of Computer Science  
University of California, Irvine  
Irvine, CA, USA  
hyunwoo@uci.edu

Ryozo Masukawa

Dept. of Computer Science  
University of California, Irvine  
Irvine, CA, USA  
rmasukaw@uci.edu

Pietro Mercati

Intel Corporation  
Hillsboro, OR, USA  
pietromercati@gmail.com

Nathaniel D. Bastian

Dept. of Electrical Engineering and Computer Science  
United States Military Academy  
West Point, NY, USA  
nathaniel.bastian@westpoint.edu

Mohsen Imani

Dept. of Computer Science  
University of California, Irvine  
Irvine, CA, USA  
m.imani@uci.edu

**Abstract**—Hyperdimensional computing (HDC) suits memory, energy, and reliability-constrained systems, yet the standard “one prototype per class” design requires  $\mathcal{O}(CD)$  memory (with  $C$  classes and dimensionality  $D$ ). Prior compaction reduces  $D$  (feature axis), improving storage/compute but weakening robustness. We introduce LOGHD, a logarithmic class-axis reduction that replaces the  $C$  per-class prototypes with  $n \approx \lceil \log_k C \rceil$  bundle hypervectors (alphabet size  $k$ ) and decodes in an  $n$ -dimensional activation space, cutting memory to  $\mathcal{O}(D \log_k C)$  while preserving  $D$ . LOGHD uses a capacity-aware codebook and profile-based decoding, and composes with feature-axis sparsification. Across datasets and injected bit flips, LOGHD attains competitive accuracy with smaller models and higher resilience at matched memory. Under equal memory, it sustains target accuracy at roughly 2.5–3.0× higher bit-flip rates than feature-axis compression; an ASIC instantiation delivers 498× energy efficiency and 62.6× speedup over an AMD Ryzen 9 9950X and 24.3×/6.58× over an NVIDIA RTX 4090, and is 4.06× more energy-efficient and 2.19× faster than a feature-axis HDC ASIC baseline.

**Index Terms**—Hyperdimensional computing, LogHD, class-axis compression, bundle hypervectors, activation-profile decoding, bit-flip robustness

## I. INTRODUCTION

Machine-learning architectures operating under tight memory, energy, and reliability budgets increasingly depend on cross-layer co-design and in-/near-memory substrates. Recent surveys and system studies underline both the promise and the sensitivity of memory-centric ML to device non-idealities and limited precision, motivating methods that reduce footprint while preserving robustness [1]–[5]. Within this context, *hyperdimensional computing (HDC)*—a family of *vector-symbolic architectures (VSA)*—is a strong fit: its operations are inherently parallel and memory-centric, and recent cross-layer workflows and macro-level prototypes indicate practical routes to efficient HDC/VSA hardware in embedded and in-/near-memory settings [6], [7].

HDC encodes data as  $D$ -dimensional hypervectors and learns via light-weight algebraic operations that are parallel, memory-centric, and empirically tolerant to device noise [8]–[16]. A conventional HDC classifier stores one prototype per class and

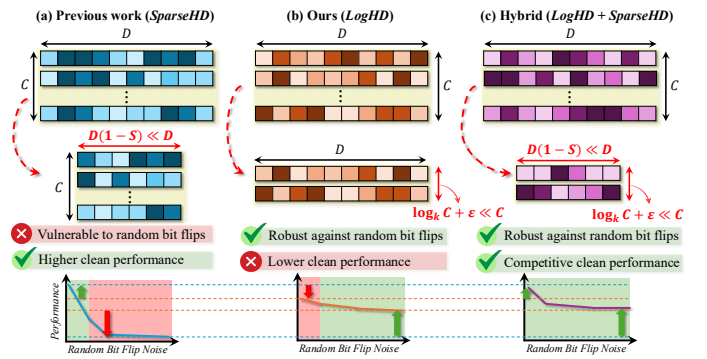


Fig. 1. **Comparison of approaches.** “Clean performance” denotes accuracy and throughput measured in fault-free conditions (no injected bit-flip noise). (a) Feature-axis compression (*SparseHD*): reduce  $D$  with sparsity  $S$ ; improves storage/compute, but robustness degrades as  $D$  shrinks. (b) LOGHD (class-axis): keep  $D$  and replace  $C$  prototypes with  $n \approx \lceil \log_k C \rceil + \epsilon$  bundles ( $\epsilon \geq 0$ ); memory scales logarithmically in  $C$  while maintaining high- $D$  robustness. (c) Hybrid (LOGHD + *SparseHD*): combine class- and feature-axis compression to tune memory, clean performance, and robustness.

predicts by comparing a query  $\phi(\mathbf{x})$  to all  $C$  prototypes [17]. The resulting memory cost  $\mathcal{O}(CD)$  dominates in multi-class, resource-constrained regimes, so shrinking this footprint without eroding accuracy or robustness is central.

Prior compression largely targets the *feature axis* by reducing  $D$  [18]–[21]. As a representative state-of-the-art example, *SparseHD* [18] sparsifies trained hypervectors to accelerate computation while keeping one prototype per class. However, operating at lower effective dimensionality weakens robustness to hardware noise—an effect consistent with observations from memory-centric and analog/near-memory accelerators [1], [2], [4], [5] as illustrated in Figure 1.(a).

We propose LOGHD, a *class-axis* compression scheme that replaces the  $C$  per-class prototypes with  $n$  bundle hypervectors  $\{\mathbf{M}_j\}_{j=1}^n$ , where  $n \geq \lceil \log_k C \rceil$  for an alphabet size  $k \geq 2$ . Each class receives a length- $n$   $k$ -ary code that specifies how its prototype contributes to each bundle. At inference, a query is compared to the  $n$  bundles to form an activation vector that is decoded via learned per-class profiles. This

reduces memory from  $\mathcal{O}(CD)$  to  $\mathcal{O}(nD) = \mathcal{O}(D \log_k C)$  without shrinking  $D$ . The approach is orthogonal to feature-axis methods (including SparseHD-style sparsification) and aligns with emerging HDC/VSA macros and cross-layer flows that benefit from storing fewer vectors rather than shorter ones [6], [7] as depicted in Figure 1.(b).

Preserving  $D$  has direct implications for robustness. In noisy memories and in-/near-memory fabrics, random bit flips or small analog perturbations distort stored hypervectors and degrade similarity computations [14], [22]. Concentration-of-measure effects stabilize similarities at large  $D$ , whereas reducing  $D$  amplifies score variance. By decreasing the *number* of stored hypervectors instead of their *length*, LOGHD achieves compactness while retaining high- $D$  tolerance to non-idealities, consistent with reliability analyses of analog and near-memory accelerators [3]–[5]. Orthogonal techniques—such as flexible numeric types and dynamic dimensional masking—can be layered with LOGHD when tighter budgets are required [23], [24]. Thus, we also study a *hybrid* design that combines LOGHD with SparseHD-style sparsification, offering additional memory savings with robustness between pure LOGHD and aggressive feature-axis compression as indicated in Figure 1.(c).

Our evaluation demonstrates that LOGHD delivers both compactness and resilience across platforms. Compared to a conventional, non-reduced HDC model, a LOGHD ASIC achieves  $498\times$  higher energy efficiency and  $62.6\times$  faster inference than an AMD Ryzen 9 9950X CPU, and  $24.3\times$  and  $6.58\times$  improvements over an NVIDIA RTX 4090 GPU. Against a strong feature-axis baseline, SparseHD, LOGHD on ASIC is  $4.06\times$  more energy-efficient and  $2.19\times$  faster under matched conditions. Robustness evaluations further show that, at equal memory budgets, LOGHD sustains target accuracy at roughly  $2.5$ – $3.0\times$  higher bit-flip probabilities than feature-axis compression. Finally, a hybrid configuration combining LogHD with SparseHD yields additional memory reduction while achieving intermediate robustness between pure class-axis and pure feature-axis approaches.

We summarize our contributions:

- 1) **Log-scale class-axis compression.** We introduce LOGHD, which replaces the  $C$  per-class prototypes with  $n \geq \lceil \log_k C \rceil$  bundle hypervectors. This reduces storage and per-query comparisons from  $\mathcal{O}(CD)$  to  $\mathcal{O}(nD)$  while preserving dimensionality  $D$ . For example, with  $k = 3$  and  $C = 26$ , only  $n = 3$  bundles are required, yielding  $8.7\times$  fewer stored prototypes and reducing comparisons per query from  $C$  to  $n$  without altering the encoder.
- 2) **Balanced decodability with light supervision.** A capacity-aware codebook and a profile-based decoder, complemented by light supervised refinement, balance bundle utilization and mitigate cross-class interference. This preserves reliable classification under superposition with inference cost scaling in  $n$  (much smaller than  $C$ ), and integrates naturally with hardware that benefits from storing fewer vectors.
- 3) **Efficiency and robustness.** Across platforms, LOGHD achieves up to  $498\times$  higher energy efficiency and  $62.6\times$  faster inference than conventional HDC, and is up to

$4.1\times$  more efficient and  $2.2\times$  faster than the state-of-the-art feature-axis compressor *SparseHD*. At equal memory budgets, it maintains accuracy under up to  $3\times$  higher bit-flip rates, with a hybrid LOGHD+SparseHD design offering further memory reduction and intermediate robustness.

## II. RELATED WORK

### A. Foundations of HDC

Hyperdimensional computing (also known as vector–symbolic architectures) represents symbols and structures as dense, high-dimensional hypervectors and manipulates them via binding, bundling, and permutation [8]. In classification, training samples for a class are superposed into a prototype hypervector; inference compares a query to all class prototypes using cosine or Hamming similarity [17]. The paradigm maps well to in-/near-memory accelerators because its operations are linear and massively parallel, and it exhibits empirical resilience to device non-idealities [14], [22], [25]. Comparative studies further catalog encoding/training choices and robustness trends, situating HDC among compact, hardware-friendly learners [11], [12], [21].

### B. Model-size reduction in HDC

Most prior work reduces parameters along the *feature axis* while retaining the one prototype per class layout. QuantHD focuses on precision reduction (binary/ternary encodings) so that Hamming distance replaces cosine similarity, improving memory footprint and compare cost without changing the number of stored prototypes [21], [26]. CompHD packs information within each prototype via structured splitting and binding with positional hypervectors followed by superposition; the query path mirrors these transforms at inference [19]. This yields a complex feature-axis reduction that preserves per-class prototypes and introduces additional binding/unbinding overhead and cross-talk considerations [21]. SparseHD sparsifies trained prototypes and co-optimizes the algorithm with reconfigurable hardware, achieving state-of-the-art efficiency–accuracy trade-offs among few-parameter compression methods while still storing one (sparsified) prototype per class [18], [21]. In our study, SparseHD is therefore used as the representative feature-axis compressor—both as a strong baseline for performance/robustness comparisons and as the feature-axis component in our hybrid LOGHD+SparseHD design—whereas CompHD is not used for hybridization due to its more complex binding/unbinding pipeline and sensitivity to cross-talk. Beyond these, accuracy-oriented updates such as OnlineHD and distributed/regenerative variants improve learning dynamics but leave the parameter layout unchanged, i.e., they do not reduce the number of class prototypes [11], [12], [17], [21]. In contrast, our approach compresses along the *class axis*: rather than shrinking each prototype, it replaces the  $C$  prototypes by  $n \approx \lceil \log_k C \rceil$  bundles and decodes in the induced activation space. To the best of our knowledge and as reflected in recent surveys [21], log-scale reduction in the *number* of stored class hypervectors has not been explored in HDC.

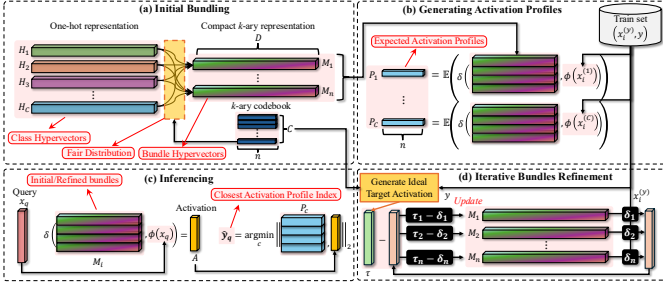


Fig. 2. **LOGHD overview.** (a) *Initial bundling*: assign each class a length- $n$   $k$ -ary code  $B_i$ ; map symbols via  $g$  and form bundles  $\{M_j\}$  by weighted superposition; a minimax-load selector balances per-bundle capacity. (b) *Activation profiling*: compute activation vectors  $A(\mathbf{x})$  against bundles and estimate per-class means  $\mathbf{P}_y$ . (c) *Inference*: classify by nearest profile in activation space. (d) *Iterative refinement*: nudge bundles toward code-implied targets with a perceptron-style update. Replacing  $C$  prototypes by  $n \approx \lceil \log_k C \rceil$  bundles reduces memory to  $\mathcal{O}(D \log_k C)$  while preserving dimensionality and robustness.

### C. Robustness to hardware noise

HDC’s robustness is rooted in concentration at high dimensionality: for a given bit-flip or analog error rate, larger  $D$  averages perturbations and stabilizes similarity scores, whereas reducing  $D$  increases score variance [14], [22]. Feature-axis compression (CompHD, SparseHD) therefore trades some of this averaging for footprint. By compressing along the class axis and preserving  $D$ , our method maintains the dimensionality-driven tolerance to non-idealities while still reducing overall model size, a behavior consistent with reliability observations reported for memory-centric ML on in-/near-memory substrates [21].

## III. METHODOLOGY

### A. Preliminaries

Let  $D$  denote the hypervector dimension and  $C$  the number of classes. Conventional HDC stores one prototype (class hypervector) per class,  $\{\mathbf{H}_i \in \mathbb{R}^D\}_{i=1}^C$ , obtained by superposing encoded training examples of that class. Given a query  $\phi(\mathbf{x}) = \mathbf{h} \in \mathbb{R}^D$ , the classifier computes scores  $s_i = \delta(\mathbf{h}, \mathbf{H}_i)$  and predicts  $\arg \max_i s_i$ . Throughout,  $\delta$  is cosine similarity,

$$\delta(\mathbf{u}, \mathbf{v}) = \left\langle \frac{\mathbf{u}}{\|\mathbf{u}\|_2}, \frac{\mathbf{v}}{\|\mathbf{v}\|_2} \right\rangle,$$

and superposition is the elementwise sum. The memory footprint of this baseline is  $\mathcal{O}(CD)$ .

LOGHD replaces the set of  $C$  prototypes with  $n$  *bundle hypervectors*  $\{M_j\}_{j=1}^n$ , where  $n \geq \lceil \log_k C \rceil$  for a user-chosen alphabet size  $k \geq 2$ . Each class is assigned a unique length- $n$   $k$ -ary code that prescribes how strongly the class prototype contributes to each bundle. A query is compared against the bundles to produce an  $n$ -dimensional activation vector, which is then decoded to a class label. This reduces model memory to  $\mathcal{O}(nD) = \mathcal{O}(D \log_k C)$  without altering the encoder  $\phi$ .

### B. Overall Pipeline

The complete pipeline (illustrated in Figure 2, panels  $a$ – $d$ ) proceeds as follows in concept. First, an initial bundling stage assigns each class a unique  $k$ -ary code and constructs  $n$  bundle hypervectors by weighted superposition of the class prototypes according to the code symbols; this step also balances

### Algorithm 1: LOGHD: Training and Inference

**Input** : Training set  $\mathcal{D} = \{(\mathbf{x}, y)\}$ , encoder  $\phi$ , alphabet size  $k$ , # bundles  $n \geq \lceil \log_k C \rceil$ , symbol weight  $g(s) = \frac{s}{k-1}$ , capacity surrogate  $U(w) = w^\alpha$ , epochs  $T$ , step size  $\eta$ .

**Output** : Bundles  $\{M_j\}_{j=1}^n$ , activation profiles  $\{\mathbf{P}_c\}_{c=1}^C$ , codebook  $B \in \{0, \dots, k-1\}^{C \times n}$ .

#### (1) Class prototypes

$\mathbf{H}_c \leftarrow \sum_{(\mathbf{x}, y) \in \mathcal{D}, y=c} \phi(\mathbf{x});$   
 $\mathbf{H}_c \leftarrow \mathbf{H}_c / \|\mathbf{H}_c\|_2$  for  $c = 1..C$ .

#### (2) Codebook (minimax-load greedy)

Initialize loads  $L_j \leftarrow 0$  for  $j = 1..n$ ; candidate pool  $\mathcal{Q} \subseteq \{0, \dots, k-1\}^n$ .

**for**  $c = 1$  **to**  $C$  **do**

$B_{c,:} \leftarrow \arg \min_{s \in \mathcal{Q}} [\max_j (L_j + U(g(s_j))) + \varepsilon \xi],$

$\xi \sim \text{Unif}[0, 1];$

$L_j \leftarrow L_j + U(g(B_{c,j}))$  for  $j = 1..n$ ;

$\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{B_{c,:}\}.$

#### (3) Initial Bundling

$M_j \leftarrow \sum_{c=1}^C g(B_{c,j}) \mathbf{H}_c;$   
 $M_j \leftarrow M_j / \|M_j\|_2$  for  $j = 1..n$ .

#### (4) Activation profiles

For each class  $c$ :

$\mathbf{P}_c \leftarrow \frac{1}{|\mathcal{D}_c|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_c} (\delta(M_1, \phi(\mathbf{x})), \dots, \delta(M_n, \phi(\mathbf{x}))).$

#### (5) Optional refinement ( $T$ epochs)

**for**  $t = 1$  **to**  $T$  **do**

**foreach**  $(\mathbf{x}, y) \in \mathcal{D}$  **do**

**for**  $j = 1$  **to**  $n$  **do**

$A_j \leftarrow \delta(M_j, \phi(\mathbf{x}));$

$\tau_j \leftarrow 2 \frac{B_{y,j}}{k-1} - 1;$

$M_j \leftarrow M_j + \eta (\tau_j - A_j) \phi(\mathbf{x});$

$M_j \leftarrow M_j / \|M_j\|_2.$

#### (6) Inference Predict( $\mathbf{x}_q$ )

$\mathbf{A} \leftarrow (\delta(M_1, \phi(\mathbf{x}_q)), \dots, \delta(M_n, \phi(\mathbf{x}_q)));$

$\hat{y} \leftarrow \arg \min_c \|\mathbf{A} - \mathbf{P}_c\|_2^2.$

the per-bundle load induced by the codebook to avoid over-capacity bundles (Figure 2.(a)). Second, using the training set, the method estimates a per-class expected activation profile—the mean vector of similarities between encoded examples of that class and the bundles (Figure 2.(b)). Third, inference compares a query’s activation vector to these profiles and returns the nearest profile in activation space (Figure 2.(c)). Finally, an optional refinement stage performs a small number of supervised updates to the bundles so that activations move toward code-implied targets, thereby mitigating cross-class interference introduced by superposition (Figure 2.(d)). Detailed procedures for each stage is described in algorithm 1.

### C. Initial Bundling

**Codebook and uniqueness.** Let  $B \in \{0, 1, \dots, k-1\}^{C \times n}$  be a *codebook* whose  $i$ -th row  $B_i = (B_{i,1}, \dots, B_{i,n})$  is the unique  $k$ -ary code assigned to class  $i$ . Uniqueness requires  $B_i \neq B_{i'}$  for all  $i \neq i'$ . When  $C = k^n$ , all length- $n$  codes are used and the codebook is effectively fixed. When  $C < k^n$ , there exist

$\binom{k^n}{C}$  valid choices; in this regime the structure of  $B$  strongly influences how many and how strong contributions land on each bundle.

**Load-aware fair code selection.** To guard against pathological codebooks that overburden a few bundles, LOGHD employs a capacity-aware selection heuristic that approximately balances the induced per-bundle load. Define a nonnegative symbol weight  $g : \{0, \dots, k-1\} \rightarrow \mathbb{R}_{\geq 0}$  and a nondecreasing capacity surrogate  $U : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ . We use  $g(s) = \frac{s}{k-1}$  to map symbols to relative contribution strengths and  $U(w) = w^\alpha$  with  $\alpha > 0$  to modulate how strongly heavy symbols are penalized in the load objective. For a candidate code  $s = (s_1, \dots, s_n)$ , its per-bundle capacity contributions are  $U(g(s_j))$ , and the cumulative load on bundle  $j$  induced by the current set of assigned codes  $\mathcal{S}$  is  $L_j = \sum_{c \in \mathcal{S}} U(g(B_{c,j}))$ .

The codebook is constructed greedily by repeatedly selecting the next code  $s^*$  that minimizes the worst-case updated load,

$$s^* = \arg \min_s \max_{1 \leq j \leq n} (L_j + U(g(s_j))) + \varepsilon \xi,$$

where  $\xi \sim \text{Unif}[0, 1]$  provides tie-breaking and diversity, and  $\varepsilon > 0$  is a tiny constant. After assigning  $s^*$  to the next class, we update  $L_j \leftarrow L_j + U(g(s_j^*))$ . When  $k^n$  is moderate we consider the full candidate set; when  $k^n$  is large we draw a sizable random candidate pool, which empirically suffices to flatten the loads while keeping selection time modest. This minimax-load criterion is a direct relaxation of the fair-distribution objective

$$B^* = \arg \min_B \max_{1 \leq j \leq n} \sum_{c=1}^C U(g(B_{c,j})),$$

and is the mechanism by which LOGHD avoids over-capacity bundles.

**Constructing the bundles.** Given the resulting codebook  $B$  and the class prototypes  $\{\mathbf{H}_i\}$ , the  $j$ -th bundle is formed by weighted superposition,

$$\mathbf{M}_j = \sum_{i=1}^C g(B_{i,j}) \mathbf{H}_i, \quad j = 1, \dots, n, \quad (1)$$

optionally followed by normalization  $\mathbf{M}_j \leftarrow \mathbf{M}_j / \|\mathbf{M}_j\|_2$  to stabilize cosine similarity. Intuitively, symbol 0 contributes nothing to a bundle, whereas larger symbols contribute proportionally more.

#### D. Generating Activation Profiles

Because each bundle aggregates multiple classes with heterogeneous weights, single-max decoding is no longer optimal. Instead, LOGHD estimates a per-class *expected activation profile*. For a sample  $\mathbf{x}$ , define its activation vector against the bundles as

$$A(\mathbf{x}) = (\delta(\mathbf{M}_1, \phi(\mathbf{x})), \dots, \delta(\mathbf{M}_n, \phi(\mathbf{x}))) \in \mathbb{R}^n.$$

For class  $y$ , the expected profile is the conditional mean

$$\mathbf{P}_y = \mathbb{E}_{\mathbf{x}|y} [A(\mathbf{x})] \approx \frac{1}{N_y} \sum_{i=1}^{N_y} A(\mathbf{x}_i^{(y)}),$$

where  $\{\mathbf{x}_i^{(y)}\}_{i=1}^{N_y}$  are the training examples of class  $y$ .

#### E. Inference

At test time, a query  $\mathbf{x}_q$  is encoded to  $\phi(\mathbf{x}_q)$ , compared with the bundles to obtain  $\mathbf{A} = A(\mathbf{x}_q)$ , and decoded by nearest-profile classification in activation space. We adopt the Euclidean metric,

$$\begin{aligned} \hat{y}_q &= \arg \min_{c \in \{1, \dots, C\}} \|\mathbf{A} - \mathbf{P}_c\|_2^2 \\ &= \arg \min_c \sum_{j=1}^n (\delta(\mathbf{M}_j, \phi(\mathbf{x}_q)) - \mathbf{P}_{c,j})^2, \end{aligned} \quad (2)$$

which we found to be robust across datasets. Alternatives such as cosine distance in the activation space perform similarly, and a Mahalanobis metric can further help when  $n$  is large relative to  $C$ ; these variations do not change the training procedure.

#### F. Iterative Bundle Refinement

Initial bundling by Equation 1 may introduce cross-class interference because multiple prototypes are superposed. LOGHD therefore supports a light-weight, supervised refinement that nudges each bundle so that the observed activations move toward class-dependent targets implied by the codebook. Let  $t : \{0, \dots, k-1\} \rightarrow [-1, 1]$  map symbols to target similarities via the linear scaling

$$t(s) = 2 \frac{s}{k-1} - 1,$$

so that  $t(0) = -1$  and  $t(k-1) = 1$ . For class  $y$ , define the target activation vector  $\boldsymbol{\tau}^{(y)} \in \mathbb{R}^n$  by  $\tau_j^{(y)} = t(B_{y,j})$ . For each training example  $\mathbf{x}^{(y)}$  we compute activations  $A_j = \delta(\mathbf{M}_j, \phi(\mathbf{x}^{(y)}))$  and update each bundle with a perceptron-style correction,

$$\mathbf{M}_j \leftarrow \mathbf{M}_j + \eta (\tau_j^{(y)} - A_j) \phi(\mathbf{x}^{(y)}), \quad j = 1, \dots, n, \quad (3)$$

optionally followed by normalization  $\mathbf{M}_j \leftarrow \mathbf{M}_j / \|\mathbf{M}_j\|_2$ . Here  $\eta > 0$  is a learning rate. The term  $(\tau_j^{(y)} - A_j)$  increases the projection of  $\mathbf{M}_j$  onto  $\phi(\mathbf{x}^{(y)})$  when the observed activation is below the target and decreases it otherwise. In practice a small number of passes over the training set suffices; excessive refinement may overfit the activation profiles.

#### G. Complexity and Memory

Conventional HDC requires  $\mathcal{O}(CD)$  memory for the prototypes and, per query,  $C$  similarities of  $D$ -dimensional vectors. LOGHD stores  $n$  bundles and therefore uses  $\mathcal{O}(nD)$  memory with  $n = \lceil \log_k C \rceil + \varepsilon$ , where a tiny redundancy  $\varepsilon \in \{0, 1, 2\}$  is sometimes added for robustness at negligible cost. Constructing the bundles by Equation 1 costs  $\mathcal{O}(nCD)$  arithmetic operations, dominated by superposition. The fair code selection considers  $n$  positions per candidate; with a candidate pool of size  $|\mathcal{Q}|$  the selection pass is  $\mathcal{O}(|\mathcal{Q}|n + Cn)$ , and  $|\mathcal{Q}| = k^n$  is feasible for moderate  $n$  while random subsampling is effective for larger  $k^n$ . Profile estimation requires encoding the training set and computing  $n$  similarities per example, which is  $\mathcal{O}(nND)$  where  $N$  is the number of training examples; accumulation of the class-wise means is negligible by comparison. Inference computes  $n$  similarities and then  $C$  distances in  $\mathbb{R}^n$ . Because  $n \ll C$  in the regimes of interest, LOGHD reduces both memory footprint and query-time computation, scaling logarithmically with  $C$  rather than linearly as in conventional HDC.

TABLE I  
DATASETS USED IN EVALUATIONS.  $C$  DENOTES THE NUMBER OF CLASSES.

Dataset	# Features	$C$	# Train	# Test	Description
ISOLET [27]	617	26	6,238	1,559	Voice recognition
UCIHAR [28]	261	12	6,213	1,554	Activity recognition (mobile)
PAMAP2 [29]	75	5	611,142	101,582	Activity recognition (IMU)
PAGE [30]	10	5	4,925	548	Page layout blocks classification

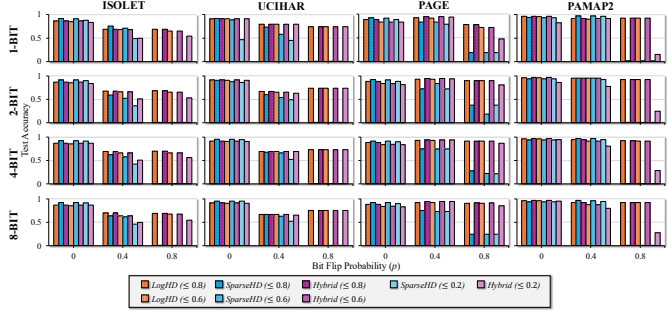


Fig. 3. Accuracy under random bit flips. Test accuracy versus bit-flip probability  $p$  at matched model-size budgets ( $\leq x$ ) across datasets, comparing SparseHD, LOGHD ( $k \in \{2, 3\}$ ), and Hybrid.

#### H. Practical Considerations

Normalization substantially improves stability when using cosine similarity; we normalize  $\phi(\mathbf{x})$ ,  $\mathbf{H}_i$ , and  $\mathbf{M}_j$  after construction and after each refinement update. The choices  $g(s) = \frac{s}{k-1}$  and  $U(w) = w^\alpha$  with  $\alpha \in [1, 2]$  work well empirically; larger  $\alpha$  penalizes heavy symbols more aggressively during code selection and promotes flatter bundle loads. Adding one or two redundant bundles beyond  $\lceil \log_k C \rceil$  often yields a small but reliable accuracy gain by improving separability in activation space without materially affecting the memory advantage.

## IV. EXPERIMENTS

### A. Experimental Setup

All models are implemented in NumPy. Evaluations use the datasets in Table I. Training is performed in 32-bit floating point; for each target precision (1, 2, 4, 8 bits) we apply post-training quantization to the learned model parameters and then evaluate on the test set. Conventional HDC, LOGHD, and SparseHD employ the same encoder and optimization hyperparameters to isolate the effect of the compaction mechanism. Iterative refinement runs for 100 passes over a randomly ordered training set with learning rate  $3 \times 10^{-4}$  and we set  $\alpha = 1$  in the capacity surrogate  $U$ . SparseHD uses dimension-wise sparsification only. Random bit flips are injected into the stored model state prior to each test evaluation: for SparseHD the flips are applied to non-pruned coordinates, and for LOGHD they are applied to

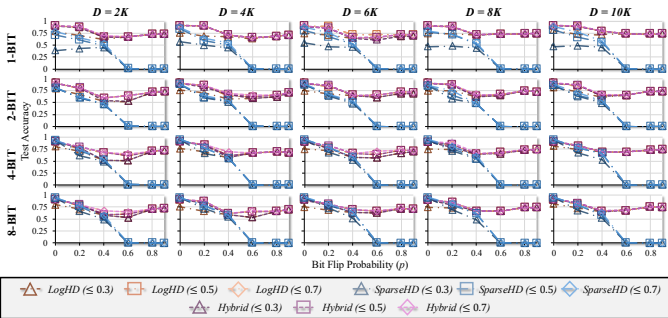


Fig. 4. Sensitivity to dimensionality and quantization. Test accuracy on UCIHAR versus bit-flip probability  $p$  for varying hypervector dimensionality  $D$  and numeric precision (1, 2, 4, 8 bits) at matched model-size budgets ( $\leq x$ ).

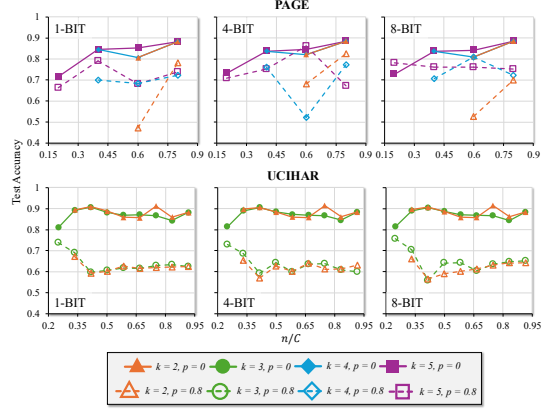


Fig. 5. Effect of alphabet size  $k$ . Test accuracy on PAGE and UCIHAR while varying  $n/C$  for different  $k$ , bit precisions, and flip probabilities  $p \in \{0, 0.8\}$ . For each  $k$ , the curve sweeps  $n$  starting at the feasibility limit  $n \geq \lceil \log_k C \rceil$ .

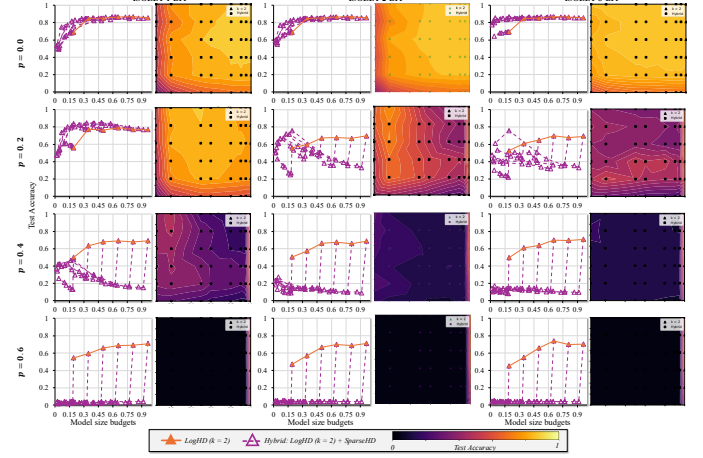


Fig. 6. Hybrid class- and feature-axis compression. Test accuracy on ISOLET for LOGHD with SparseHD-style sparsification, shown across bit precisions, flip probabilities, number of bundles  $n$ , and sparsity levels ( $1 - S$ ). The dotted traces showing accuracy as sparsity varies for a fixed base LOGHD model ( $n, k$ ). Heatmaps summarize accuracy as a function of the number of bundles  $n$  (rows) and retained feature fraction ( $1 - S$ ) (columns).

both the bundle hypervectors and the stored activation profiles. Test inputs are not corrupted. Unless noted, we fix  $D = 10,000$  and consider  $k \in \{2, 3\}$  for LOGHD.

### B. Performance under Bit-Flip Noise

Figure 3 reports test accuracy versus bit-flip probability  $p$  at matched model-size budgets across datasets, comparing SparseHD, LOGHD ( $k \in \{2, 3\}$ ), and a Hybrid composition. Budgets are reported in parentheses as ( $\leq x$ ), where  $x$  is the fraction of the conventional HDC footprint  $C \times D$ ; we fix the dimensionality to  $D = 10,000$ . For LOGHD—which replaces  $C$  prototypes with  $n \geq \lceil \log_k C \rceil$  bundles—the minimum feasible budget is  $\lceil \log_k C \rceil / C$ . With  $k \in \{2, 3\}$  and datasets having  $C=5$  (e.g., PAGE, PAMAP2), this lower bound is  $2/5 = 0.4$ , explaining the absence of a ( $\leq 0.2$ ) LOGHD point unless  $k$  is increased. Across all budgets, LOGHD sustains accuracy at higher fault rates by preserving  $D$  while reducing the number of stored vectors; SparseHD is competitive in the clean setting but degrades rapidly as  $p$  increases due to reduced effective dimensionality; the Hybrid lies between these two, extending compaction beyond LOGHD with smaller robustness penalties than pure feature-axis compression.

Figure 4 further examines sensitivity to dimensionality and quantization on UCIHAR under fixed memory budgets. We observe a clear trend: higher numeric precision and larger hypervector dimensionality  $D$  improve clean accuracy, while higher precision at small  $D$  leaves less redundancy per hypervector and thus amplifies the effect of bit flips, yielding larger accuracy drops under noise or dimensionality reduction. Although LOGHD can trail slightly in the fault-free setting, it exhibits markedly stronger robustness across all tested  $D$ , sustaining accuracy even at severe flip probabilities. The Hybrid composition mitigates LOGHD’s clean-accuracy gap, delivering competitive and often superior accuracy to SparseHD across dimensionalities and precisions while retaining much of LOGHD’s robustness. Taken together, these results indicate that combining class-axis compression—strong against bit-flip noise and quantization—with moderate feature-axis reduction—strong at preserving clean accuracy—yields compact models that are both high-performing and resilient at low memory budgets.

### C. Varying the Alphabet Size $k$

Figure 5 shows how alphabet size  $k$  influences accuracy as the budget scales with  $n$ . In the fault-free case ( $p=0$ ), performance is nearly identical across  $k$  once  $n \geq \lceil \log_k C \rceil$ , suggesting that profile-based decoding already ensures separability and that clean accuracy is largely task-dependent. Under high fault rates ( $p=0.8$ ), larger alphabets yield higher accuracy near feasibility. We hypothesize that this stems from two factors: (i) larger  $k$  reduces the required code length ( $n \approx \lceil \log_k C \rceil$ ), lowering activation dimensionality and thereby the number of independently corrupted coordinates; and (ii) multi-level coding provides finer weight granularity, enabling the capacity-aware selector to distribute bundle loads more evenly and mitigate cross-class interference. These advantages are most pronounced in noisy, memory-limited regimes (small  $n$ ) and diminish as  $n$  increases, where noise accumulation and narrower target steps reduce the margin benefits.

### D. Hybrid Class- and Feature-Axis Compression

Figure 6 evaluates combining LOGHD with SparseHD-style sparsification. At low fault rates ( $p \leq 0.2$ ), the hybrid can outperform pure LOGHD, as pruning weak coordinates regularizes cross-class leakage and sometimes improves clean accuracy. At higher fault rates ( $p \geq 0.4$ ), it degrades more quickly since reducing the effective dimensionality  $(1-S)D$  weakens the concentration-of-measure effects that stabilize cosine similarity, underscoring LOGHD’s advantage of preserving  $D$ . Sweeping sparsity at fixed  $n$  produces a U-shaped trend: light pruning removes signal and lowers accuracy, moderate pruning suppresses nuisance components and recovers accuracy up to an optimum, and aggressive pruning collapses class separation. The optimum shifts with precision, as lower bit-widths benefit more from sparsification (filtering quantization noise) while higher precisions tolerate denser representations before cross-talk dominates. Overall, reducing along the class axis (smaller  $n$ ) preserves robustness predictably across settings, whereas reducing along the feature axis (larger  $S$ ) introduces sharp uncertainty under faults; the hybrid offers a tunable middle

TABLE II  
HARDWARE EFFICIENCY RATIOS OF LOGHD (ASIC) RELATIVE TO BASELINES ON ISOLET ( $C=26, k=2$ ). RATIOS ARE LOGHD/Baseline.

Baseline	Platform	Energy eff. ( $\times$ ) $\uparrow$	Speedup ( $\times$ ) $\uparrow$
SparseHD	ASIC	4.06	2.19
Conventional HDC	CPU (Ryzen 9 9950X)	498.1	62.6
Conventional HDC	GPU (RTX 4090)	24.3	6.58

ground for tighter memory budgets, but its robustness ceiling remains bounded by the dimensionality reduction it imposes.

### E. Efficiency Gains Across Platforms

Table II summarizes hardware-level efficiency of LOGHD relative to strong baselines. Compared to SparseHD on the same ASIC class, LOGHD requires only about one quarter of the energy and half the latency, corresponding to  $4.06\times$  higher energy efficiency and  $2.19\times$  speedup. Relative to conventional HDC on general-purpose processors, the gains are substantially larger: up to  $498\times$  energy efficiency and  $63\times$  speedup versus CPU, and  $24\times$  and  $6.6\times$  versus GPU. These results confirm that compressing along the class axis while preserving dimensionality not only improves robustness under faults but also translates into favorable energy–latency trade-offs when mapped to dedicated hardware.

## V. CONCLUSIONS

We presented LOGHD, a class-axis compression scheme for hyperdimensional computing that reduces memory and compute from  $\mathcal{O}(CD)$  to  $\mathcal{O}(D \log_k C)$  while preserving dimensionality. By storing fewer hypervectors rather than shorter ones, LOGHD achieves robustness to device faults and quantization, and in hybrid form offers additional flexibility under tight budgets. Experiments across multiple datasets show that LOGHD sustains accuracy under up to  $3\times$  higher bit-flip rates than feature-axis compression, and ASIC implementations deliver up to  $498\times$  energy efficiency and  $63\times$  speedup over CPU baselines. Together, these results position class-axis compaction as a scalable, hardware-friendly approach for noise-robust HDC model compression, well suited for TinyML and edge deployments.

**Limitations and Future Work.** This work does not include a formal theoretical analysis of how class-axis reduction interacts with hypervector capacity and separability as dimensionality and class complexity grow. Exploring these relationships, along with richer fault models, is left to future work.

## ACKNOWLEDGEMENTS

This work was supported in part by the DARPA Young Faculty Award, the National Science Foundation (NSF) under Grants #2431561, #2127780, #2319198, #2321840, #2312517, and #2235472, the Semiconductor Research Corporation (SRC), the Office of Naval Research through the Young Investigator Program Award and Grants #N00014-21-1-2225 and #N00014-22-1-2067, Army Research Office Grant #W911NF2410360, and DARPA under Support Agreement No. USMA 23004. Additionally, support was provided by the Air Force Office of Scientific Research under Award #FA9550-22-1-0253, along with generous gifts from Xilinx and Cisco.

## REFERENCES

- [1] C. Lammie, H. Benmezziane, W. Simon, E. Ferro, A. Vasilopoulos, J. Büchel, M. Le Gallo, I. Boybat, and A. Sebastian, "Deep learning software stacks for analogue in-memory computing-based accelerators," *Nature Reviews Electrical Engineering*, pp. 1–13, 2025.
- [2] O. Krestinskaya, M. E. Fouda, H. Benmezziane, K. El Maghraoui, A. Sebastian, W. D. Lu, M. Lanza, H. Li, F. Kurdahi, S. A. Fahmy, et al., "Neural architecture search for in-memory computing-based deep learning accelerators," *Nature Reviews Electrical Engineering*, vol. 1, no. 6, pp. 374–390, 2024.
- [3] H. S. Tsai, "Architecture and programming of analog in-memory-computing accelerators for deep neural networks," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 666–666, IEEE, 2024.
- [4] G. Burr, H. Tsai, W. Simon, I. Boybat, S. Ambrogio, C.-E. Ho, Z.-W. Liou, M. Rasch, J. Büchel, P. Narayanan, et al., "Design of analog-ai hardware accelerators for transformer-based language models," in *2023 International Electron Devices Meeting (IEDM)*, pp. 1–4, IEEE, 2023.
- [5] M. M. Frank, N. Li, M. J. Rasch, S. Jain, C.-T. Chen, R. Muralidhar, J.-P. Han, V. Narayanan, T. M. Philip, K. Brew, et al., "Impact of phase-change memory drift on energy efficiency and accuracy of analog compute-in-memory deep learning inference," in *2023 IEEE International Reliability Physics Symposium (IRPS)*, pp. 1–10, IEEE, 2023.
- [6] M. M. R. Nayan, C.-K. Liu, Z. Wan, A. Raychowdhury, and A. J. Naeemi, "Hydra: Sot-cam based vector symbolic macro for hyperdimensional computing," *arXiv preprint arXiv:2504.14020*, 2025.
- [7] S. Du, M. Ibrahim, Z. Wan, L. Zheng, B. Zhao, Z. Fan, C.-K. Liu, T. Krishna, A. Raychowdhury, and H. Li, "Cross-layer design of vector-symbolic computing: Bridging cognition and brain-inspired hardware acceleration," *arXiv preprint arXiv:2508.14245*, 2025.
- [8] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional vectors," *Cognitive Computation*, 2009.
- [9] S. Yun, R. Masukawa, S. Jeong, and M. Imani, "Neurohash: A hyperdimensional neuro-symbolic framework for spatially-aware image hashing and retrieval," *arXiv preprint arXiv:2404.11025*, 2024.
- [10] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.
- [11] Z. Zou, Y. Kim, F. Imani, H. Alimohamadi, R. Cammarota, and M. Imani, "Scalable edge-based hyperdimensional learning system with brain-like neural adaptation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.
- [12] J. Wang, S. Huang, and M. Imani, "Disthd: A learner-aware dynamic encoding method for hyperdimensional classification," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.
- [13] S. Yun, R. Masukawa, H. Chen, S. Jeong, W. Huang, A. Rezvani, M. Na, Y. Yamaguchi, and M. Imani, "Hyperdimensional intelligent sensing for efficient real-time audio processing on extreme edge," *IEEE Access*, 2025.
- [14] S. Yun, H. E. Barkam, P. R. Genssler, H. Latapie, H. Amrouch, and M. Imani, "Hyperdimensional computing for robust and efficient unsupervised learning," in *2023 57th Asilomar Conference on Signals, Systems, and Computers*, pp. 281–288, IEEE, 2023.
- [15] S. Yun, H. Chen, R. Masukawa, H. Errahmouni Barkam, A. Ding, W. Huang, A. Rezvani, S. Angizi, and M. Imani, "Hypersense: Hyperdimensional intelligent sensing for energy-efficient sparse data processing," *Advanced Intelligent Systems*, vol. 6, no. 12, p. 2400228, 2024.
- [16] S. Yun, R. Hassan, R. Masukawa, and M. Imani, "Missionhd: Data-driven refinement of reasoning graph structure through hyperdimensional causal path encoding and decoding," *arXiv preprint arXiv:2508.14746*, 2025.
- [17] A. Hernández-Cano, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 56–61, IEEE, 2021.
- [18] M. Imani, S. Salamat, B. Khaleghi, M. Samragh, F. Koushanfar, and T. Rosing, "Sparsehd: Algorithm-hardware co-optimization for efficient high-dimensional computing," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 190–198, IEEE, 2019.
- [19] J. Morris, M. Imani, S. Bosch, A. Thomas, H. Shu, and T. Rosing, "Comphd: Efficient hyperdimensional computing using model compression," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2019.
- [20] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.
- [21] P. Vergés, M. Heddes, I. Nunes, D. Kleyko, T. Givargis, and A. Nicolau, "Classification using hyperdimensional computing: A review with comparative analysis," *Artificial Intelligence Review*, vol. 58, no. 6, p. 173, 2025.
- [22] H. E. Barkam, S. Yun, H. Chen, P. Gensler, A. Mema, A. Ding, G. Micheliogiannakis, H. Amrouch, and M. Imani, "Reliable hyperdimensional reasoning on unreliable emerging technologies," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2023.
- [23] H. Li, F. Liu, Y. Chen, Z. Wang, S. Huang, N. Yang, D. Lyu, and L. Jiang, "Fate: Boosting the performance of hyper-dimensional computing intelligence with flexible numerical data type," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 1269–1282, 2025.
- [24] F. Liu, H. Li, Z. Wang, D. Lyu, and L. Jiang, "Hyperdyn: Dynamic dimensional masking for efficient hyper-dimensional computing," in *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, IEEE, 2025.
- [25] H. E. Barkam, S. Yun, P. R. Genssler, Z. Zou, C.-K. Liu, H. Amrouch, and M. Imani, "Hdgin: Hyperdimensional genome sequence matching on unreliable highly scaled fefet," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2023.
- [26] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2268–2278, 2019.
- [27] R. Cole and M. Fandy, "ISOLET." UCI Machine Learning Repository, 1991. DOI: <https://doi.org/10.24432/C51G69>.
- [28] J. Reyes-Ortiz, D. Anguita, A. Ghio, L. Oneto, and X. Parra, "Human Activity Recognition Using Smartphones." UCI Machine Learning Repository, 2013. DOI: <https://doi.org/10.24432/C54S4K>.
- [29] A. Reiss, "PAMAP2 Physical Activity Monitoring." UCI Machine Learning Repository, 2012. DOI: <https://doi.org/10.24432/C5NW2H>.
- [30] D. Malerba, "Page Blocks Classification." UCI Machine Learning Repository, 1994. DOI: <https://doi.org/10.24432/C5J590>.