Early Termination for Hyperdimensional Computing Using Inferential Statistics

Pu (Luke) Yi
lukeyi@stanford.edu
Stanford University
Stanford, California, USA

Yifan Yang
yyang29@stanford.edu
Stanford University
Stanford, California, USA

Chae Young Lee
chae@stanford.edu
Stanford University
Stanford, California, USA

Sara Achour
sachour@stanford.edu
Stanford University
Stanford, California, USA

Abstract

Hyperdimensional Computing (HDC) is a brain-inspired, lightweight computing paradigm that has shown great potential for inference on the edge and on emerging hardware technologies, achieving state-of-the-art accuracy on certain classification tasks. HDC classifiers are inherently error resilient and support early termination of inference to approximate classification results. Practitioners have developed heuristic methods to terminate inference early for individual inputs, reducing the computation of inference at the cost of accuracy. These techniques lack statistical guarantees and may unacceptably degrade classification accuracy or terminate inference later than is needed to obtain an accuracy result.

We present Omen, the first dynamic HDC optimizer that uses inferential statistics to terminate inference early while providing accuracy guarantees. To realize Omen, we develop a statistical view of HDC that reframes HD computations as statistical sampling and testing tasks, enabling the use of statistical tests. We evaluate Omen on 19 benchmark instantiations of four classification tasks. Omen is computationally efficient, delivering up to 7.21–12.18× inference speed-ups over an unoptimized baseline while only incurring a 0.0–0.7% drop in accuracy. Omen outperforms heuristic methods, achieving an additional 0.04–5.85× inference speed-up over the unoptimized baseline compared to heuristic methods while maintaining higher or comparable accuracy.

CCS Concepts: • **Hardware** → **Emerging languages and compilers**; Memory and dense storage; • **Software and its engineering** → *Software notations and tools.*

Keywords: program optimization, unconventional computing, hyperdimensional computing, emerging hardware technologies, edge machine learning

ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0698-1/25/03
https://doi.org/10.1145/3669940.3707254

ACM Reference Format:
Pu (Luke) Yi, Yifan Yang, Chae Young Lee, and Sara Achour. 2025. Early Termination for Hyperdimensional Computing Using Inferential Statistics. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3669940.3707254

1 Introduction

Lightweight machine learning (ML) models have been effective in performing classification on the edge for biomedical, industrial, and environmental monitoring domains—examples include anomaly detection, wake word recognition, action prediction, and classification of signal patterns [22, 37]. In recent years, researchers and practitioners have developed ML classifiers that leverage hyperdimensional computing (HDC), an emerging computing paradigm, to perform inference on highly resource-constrained edge devices and error-prone emerging hardware platforms [6, 32, 39, 50, 67]. HDC classifiers are attractive targets for these platforms because their operators are resource efficient and amenable to optimization, the compute model is highly robust to errors, and inference results can be easily approximated.

1.1 HDC Classifiers and Early Termination

Hyperdimensional computing (HDC) is an emerging computing paradigm that uses high-dimensional vectors called hypervectors as the basic data type. In HDC, information is encoded in the distances between hypervectors. This representation is inherently error resilient and amenable to approximation. Data are encoded with hypervectors through the use of HD operators which compute hypervectors that are similar/dissimilar to their hypervector operands. Because all HD operators are element-wise operators and distances are approximable, HD computations are implementable as streaming computations that may be terminated early to obtain approximate results.

Early Termination. Due to the *fully distributed* property of HDC, which connects to the statistical property of exchangeability, the inference task is error-resilient and can be terminated at any time to obtain an approximation of the final classification result. HDC classifiers, therefore, support performance optimizations that terminate inference tasks early to reduce latency. In contrast, classical ML models typically deploy all-or-nothing inference algorithms in which the entire model must be applied to the inference input to produce a classification or offer coarse-grained early exit without guarantees on accuracy loss [23].

Prior work. Researchers have developed dynamic, heuristic approaches to terminate inference early in the runtime and methods to statically reduce the size of the hypervector [9, 30, 33, 54, 55, 63]. These methods lack statistical guarantees, and as a result produce unacceptable accuracy degradations or process more hypervector elements than necessary in certain scenarios (Section 7). Researchers have also developed methods that minimize hypervector size while providing theoretical guarantees; however, these methods make independence assumptions about encoded information that do not hold for classification tasks [74].

1.2 Early Termination with Omen

We present Omen, a dynamic HDC optimizer that uses inferential statistics to terminate classification early with accuracy guarantees. Omen deploys a termination algorithm inspired by Wald's sequential probability ratio test (SPRT) [70], where new data are processed sequentially until a statistically significant conclusion is reached. Omen offers the following benefits:

- **Fine-Grained Per-Input Early Termination.** Omen identifies effective termination points for individual inputs at the dimension granularity. Omen can therefore allocate compute resources based on the difficulties of inputs.
- **Accuracy Guarantees.** Omen works with a user-provided confidence level that serves as an upper bound on the accuracy drop resulting from early termination. These accuracy guarantees hold even with hardware-induced errors.
- **Practically Deployable.** Omen's termination algorithm is optimized to introduce minimal memory and compute overheads and can be readily deployed on embedded systems. Furthermore, because Omen requires little to no modification of the HDC-based ML pipeline, it can be readily integrated with various training and inference algorithms.

Results. Omen is computationally efficient, delivering up to 7.21–12.18× inference speed-ups over an unoptimized baseline while only incurring a 0.0–0.7% drop in accuracy. Moreover, compared to heuristic early termination methods, Omen achieves an additional 0.04–5.85× inference speed-up over the unoptimized baseline with higher or comparable accuracy in 19 benchmarks. We also demonstrate that Omen can be readily used without adaption to perform principled early termination in the presence of bit corruptions.

Key Insight. To realize Omen, we develop a new statistical view of HDC that reframes HD computations as statistical sampling and testing problems. We relate the statistical exchangeable property to the *fully-distributed* property of hypervectors, which enables such a view, and show that this property holds for hypervectors constructed using simple HD-operator-based or advanced optimization-based algorithms.

1.3 Contributions

- ***Statistical view of HDC*** (Section 4): We present a statistical view of HDC. To the best of our knowledge, this is the first work that frames HDC classification as a statistical sampling and testing problem, allowing the application of statistical methods to HDC classifiers.
- ***Omen optimizer*** (Section 5): We propose Omen, the first dynamic early termination algorithm for HDC classifiers that uses inferential statistics to improve inference performance while providing accuracy guarantees.
- ***Exchangeability*** (Section 6): We relate the *fully distributed* property of HDC to statistical exchangeability and prove that hypervectors constructed using advanced training algorithms are exchangeable.
- ***Evaluation*** (Section 7): We empirically demonstrate that Omen achieves significant performance improvements with minimal accuracy loss, and Omen's accuracy bounds hold robustly even in the presence of hardware errors.

2 Hyperdimensional Computing (HDC)

Hyperdimensional Computing (HDC), or Vector Symbolic Architectures (VSA), is a highly error-resilient, brain-inspired computing paradigm. The basic HDC datatype is a *hypervector*, V, a high-dimensional vector, and the value in each index $V[i]$ is referred to as a *hypervector element.* The datatype of the hypervector elements depends on the variant of HDC used; we will target two variants in this work, BSC (Binary Spatter Code) and MAP (Multiply-Add-Permute). BSC uses dense binary hypervectors with elements $V[i] \in \{0,1\}$, and MAP uses real-valued hypervectors with elements $V[i] \in [-1,1]$. Table 1 summarizes the HD operator implementations.

Distances. In HDC, information is represented as relative distances between hypervectors. BSC uses the Hamming distance $\mathrm{HD}(V,V')$ between hypervectors and MAP uses the cosine $\cos(V,V')$ or dot product $\mathrm{dot}(V,V')$ similarities between hypervectors. The size (N, number of elements) of the hypervector affects the resolution of encoded information and the granularity at which distances can be resolved.

HD Codebooks. To work with HDC, input datatypes, such as text, numbers, or images, must be encoded as a hypervector representation. The basic building block of an HD information encoding are basis or code hypervectors which represent atomic symbols in the problem domain. These hypervectors

Type	Hypervector	Random Generation	Distance Metric $[dis(V,V')]$	Binding $[V \otimes V']$	Bundling $[\oplus]$	Permutation $[p^k(V)]$
BSC [38]	Dense Binary $V[i] \in \{0,1\}$	Bernoulli $V[i] \sim \mathrm{Bern}(p=0.5)$	Hamming Distance $\mathrm{HD}(V,V')$ $\mathrm{HD}(V,V') = \frac{1}{N}\sum(V[i]\ \mathbf{xor}\ V'[i])$	XOR $V[i]\ \mathbf{xor}\ V'[i]$	Majority Vote $[(\sum_{j=1}^{m} V_j[i]) > \frac{m}{2}]$	Rotational Shift $V[(i+k) mod N]$
MAP [18]	Reals $V[i] \in [-1,1]$	Uniform $V[i] \sim \mathrm{U}(-1,1)$	Cosine/Dot Product Similarity $\mathrm{dot}(V,V') = \sum V[i]V'[i]$	Multiplication $V[i]V'[i]$	Normalized Addition $clamp(\sum V_j[i])$	Rotational Shift $V[(i+k) mod N]$

Table 1. BSC and MAP HDC Overview. Distance metrics compute a real value from two hypervectors. Other elementwise operators map hypervectors to hypervectors. *clamp* operation clamps the value to [−1,1] range.

are typically randomly generated, where each hypervector element $V[i]$ is independently and identically distributed (*iid*).

HD Operators. HDC's compute operators generate hypervectors with certain distance relationships to other hypervectors. The binding ($V \otimes V'$) and permutation ($p^k(V)$) operations produce hypervectors that are dissimilar (far) from the input hypervectors V and V'. The bundling operation ($V_1 \oplus ... V_m$) produces a hypervector that is similar (close) to each of the input hypervectors $V_1,...,V_m$. The implementation of HD operators depends on the HDC variant, as summarized in Table 1.

2.1 Example: Text Encoding

Consider the case where we want to encode the word **catch** as a hypervector. We first generate a random hypervector for each letter $\mathbf{a} \rightarrow V_{\mathbf{a}},...,\mathbf{z} \rightarrow V_{\mathbf{z}}$. These letter hypervectors are far apart from one another. Using HD operators, we can then build hypervector representations of strings from the letter hypervectors. We note that the datatype of the generated hypervector $V_{\mathbf{catch}}$ depends on the HDC variant used. The HD operators determine the distance relationships between words; all the following three commonly used encodings encode the word **catch** but have very different distance properties:

Binding-Based. We encode the position of each letter by permuting the letter hypervector and then bind them together. The resulting hypervector embedding for **catch** will be dissimilar to words that share letters in the same position, such as **patch** and **match**:

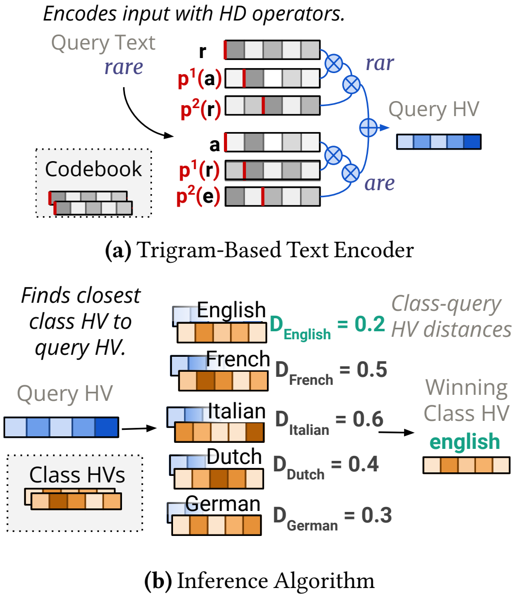
$$V_{\mathbf{catch}}^{\mathrm{bind}} = V_{\mathbf{c}} \otimes p^1(V_{\mathbf{a}}) \otimes p^2(V_{\mathbf{t}}) \otimes p^3(V_{\mathbf{c}}) \otimes p^4(V_{\mathbf{h}})$$

Bundling-Based. We encode the position of each letter by permuting the letter hypervector and then bundle them together. The resulting hypervector embedding for **catch** will be similar to words that share letters, such as **patch** and **match**:

$$V_{\mathbf{catch}}^{\mathrm{bund}} = V_{\mathbf{c}} \oplus p^1(V_{\mathbf{a}}) \oplus p^2(V_{\mathbf{t}}) \oplus p^3(V_{\mathbf{c}}) \oplus p^4(V_{\mathbf{h}})$$

Trigram-Based. We can build trigrams **cat**, **atc**, and **tch** with binding-based encoding and then bundle the trigrams together to build the word [36]. With this encoding, words that share trigrams will be similar to each other.

$$V_{\mathbf{catch}}^{\mathrm{trig}} = V_{\mathbf{cat}}^{\mathrm{bind}} \oplus V_{\mathbf{atc}}^{\mathrm{bind}} \oplus V_{\mathbf{tch}}^{\mathrm{bind}}$$



(a) Trigram-Based Text Encoder

(b) Inference Algorithm

Figure 1. HDC language classifier. HV stands for hypervector.

3 HDC Classification with Omen

HDC has been used extensively for performing lightweight and error-resilient classification tasks, and has been demonstrated to achieve higher classification accuracy than other lightweight ML approaches on a range of edge inference tasks [26, 56]. Omen enables early termination of inference tasks when a statistically significant result is reached, largely reducing the computational costs of inference. We next introduce Omen using a language classification problem as a running example. Section 3.1 overviews the anatomy of HDC classifiers using the HDC language classifier. Section 3.2 introduces the challenges associated with reducing hypervector size during inference, and Section 3.3 introduces Omen.

3.1 Anatomy of an HDC Classifier

Figure 1a and 1b presents the flow for an HDC language classifier. Given an input sentence, it encodes the sentence as a hypervector and classifies the sentence as one of the languages by computing the hypervector similarities. This classifier was previously applied to an in-memory computing substrate to enable language classification with high accuracy and very low energy consumption [39].

Architecture. The classifier is composed of an encoder (Figure 1a) that translates sentences to hypervectors. Hypervectors of training data are used to construct *class hypervectors* V_{English}, V_{French}, V_{Dutch}, V_{German}, and V_{Italian} that capture the salient features of each language. In the above language classifier, the encoder uses a pre-generated codebook to translate each symbol (letters and punctuations) into a hypervector, and the trigram encoding method described in Section 2.1 to encode sentences by regarding the sentence as a large "word".

Inference. During inference (Figure 1b), the classifier accepts a query sentence, translates it into a hypervector V_{query} using the encoder (Figure 1a), and then finds the class hypervector with the smallest distance to the query hypervector $\text{argmin}_{lang}(\text{dis}(V_{query}, V_{lang}))$.

Training. The HDC training algorithm computes class hypervectors from the training data, where each training input $t_{i,l}$ is the i-th input with label l. Given a training dataset, the simplest HDC training algorithm first encodes each training input $t_{i,l}$ as a hypervector $V_{i,l}$ using the classifier's encoder. The hypervectors are are then grouped by label and then bundled together $V_l = V_{1,l} \oplus V_{2,l} \ldots$ to produce a class hypervector V_l for each label. More advanced training algorithms achieve higher accuracy by iteratively re-bundling misclassified inputs (a kind of learning vector quantization (LVQ) algorithms [47]) or optimizing hypervector representations with gradient descent. These algorithms These algorithms are also supported by Omen and discussed in Section 6.2.4. In this example, we use iterative rebundle training.

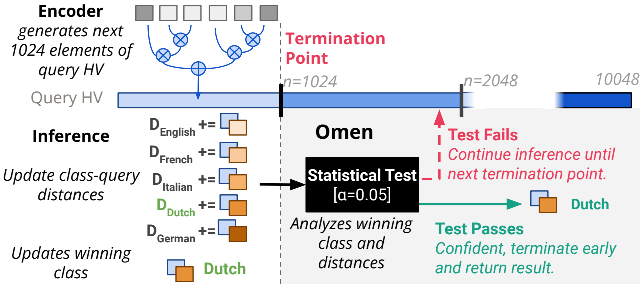
HDC Variant. This HDC classifier architecture is general and agnostic to the HDC variant or the hypervector size. In this example, we instantiate the HDC classifier to use the BSC variant and to work with N=10,048-element BSC hypervectors. Since each BSC element is a 0/1 value, each hypervector is 10,048 bits. We choose a hypervector size of 10,048 because it is a multiple of 64 and convenient for bit packing optimizations. The Hamming distance $\text{HD}(V, V')$ is used to compute distances between class and query hypervectors, and the XOR, majority vote, and rotational shift operations implement binding, bundling and permutation.

Dataset. In this example, we choose 10,000 sentences each from 5 languages, Dutch, English, French, German, and Italian, from the Leipzig Corpora Collection [21] language dataset. For each language, a random selection of 6/7 of all sentences is used as training samples, and the others as testing samples. We use only 5 languages to keep the example simple and tractable. The HDC language classifier achieves 98.5% test accuracy.

3.2 Challenges with Inference

As an example, we use the HDC classifier to classify two sentences Q1, Q2, where both sentences are Dutch:

- Q1. Door puur geluk leef je nog.
- Q2. Dringend burgemeesters uit hun ambt zetten als hoofd van de politie.



(a) Early termination with Omen.

	Size	V_{Dutch}	V_{English}	V_{French}	V_{German}	V_{Italian}
V_{Q1}	10048	**0.4790**	0.5001	0.4792	0.4916	0.4831
V_{Q2}	10048	**0.4033**	0.4534	0.4482	0.4388	0.4690
V_{Q1}	9048	0.4799	0.4997	0.4798	0.4911	0.4809
V_{Q2}	9048	**0.4048**	0.4546	0.4465	0.4418	0.4719

(b) Hamming Distances of Q1, Q2 queries. Minimum distances are marked as bold (correctly classified) or red (incorrectly classified).

p-values computed by the Wald's test		sorted by p-value (descending order) and compared to α/j (Holm-Bonferri method)			
name	p_j	j	name	$p_j < \alpha/j$	result
H_0^{English}	0.007560	1	H_0^{French}	$0.050722 \geq 0.0500$	FAIL
H_0^{French}	0.050722	2	H_0^{English}	$0.007560 < 0.0250$	PASS
H_0^{Italian}	0.000067	3	H_0^{German}	$0.004512 < 0.0166$	PASS
H_0^{German}	0.004512	4	H_0^{Italian}	$0.000067 < 0.0125$	PASS

(c) Omen's statistical test as applied to query Q2 for $n = 1024$. The unsatisfied test is marked as red.

Figure 2. Omen classifier with early termination. Tables report class-query hypervector distances and statistical tests for Q1, Q2 queries.

Figure 2b presents the hypervector distances for the Q1 and Q2 classification tasks. Using the entire 10,048-element hypervectors, the Q1 and Q2 queries are both correctly classified as Dutch sentences. V_{Q1} is almost as close to V_{French} as to V_{Dutch}, and V_{Q2} is much closer to V_{Dutch} than other V_{lang}'s.

We can reduce the memory and footprint of the model and the computation required for inference by reducing the size of the hypervectors N in the HDC classifier. This optimization sacrifices classification accuracy for resource usage. If we were to repeat the above inference with the BSC language classifier, but with slightly smaller 9,048-bit hypervectors, the Q1 query would be classified incorrectly, but the Q2 query would still be classified correctly. If we investigate the hypervector distances on this reduced-size model, we find that V_{Q1} becomes closest to V_{French}, although by a very small margin, resulting in a misclassification. In contrast, the V_{Q2} is robustly much closer to V_{Dutch} and is therefore still classified correctly.

The crux of the issue is that some queries are close to multiple classes and thus difficult to classify, requiring more bits to reach a conclusion, whereas other queries are much closer to one class and thus easy to classify. Therefore, it would be productive to dynamically adjust the number of processed bits depending on the difficulty of the individual inputs.

3.3 Early Termination with Omen

Omen enables dynamic early termination of HDC classifiers for individual inputs while providing an upper bound on the accuracy loss. Omen's workflow is shown in Figure 2a. Omen accepts a list of early termination points: for the current example, termination points are n=1024, 2048, 4096, and 9192 hypervector elements.

Omen keeps track of the running query-class distances and tests if the current classification result is statistically significant at each termination point. For Q2, at the termination point 1 where $n = 1024$, it is closest to class Dutch. To test the the statistical significance of the result, Omen applies a *statistical test* to the partially processed query and class hypervectors and the running query-class distances.

Rationale. Omen frames the classification task as a *statistical inference problem.* A statistical inference problem sets hypotheses about the underlying distributions, computes p-values that quantify probabilities of observing the current samples assuming the hypotheses are true, and then infers the truthfulness of the hypotheses based on p-values.

We develop a *statistical inference* view of HDC where distances between hypervector elements are treated as samples or observations of an unknown distance distribution. The running distances between class and query hypervectors can then be interpreted as approximations of the expected value of the underlying distance distribution. We can apply statistical tests to check whether the current observations support a hypothesis about the underlying distribution. Section 4 describes this view of HDC in more detail.

Assessing Statistical Significance. Suppose that, at a termination point, Omen finds that V_{Q2} is closest to V_{Dutch} (that is, a prefix of V_{Q2} is closest to the corresponding prefix of V_{Dutch}). Omen checks the statistical significance of the current observation. Omen forms a hypothesis H_1: V_{Q2} is closest to V_{Dutch} in expectation. If Omen accepts H_1, then Omen terminates at this point. Omen accepts a user-specified statistical significance parameter α that serves as an upper bound on accuracy drop due to early termination. Therefore, if H_1 is not true, the probability that Omen accepts H_1 and terminates with Dutch as the incorrect classification result should be at most α. In this example, we use $\alpha = 0.05$.

The Statistical Test. We next describe how the statistical test is performed. We first break H_1 down into 4 hypotheses H_1^{English}, H_1^{French}, H_1^{German}, and H_1^{Italian}, which are that V_{Q2} is closer *in expectation* to V_{Dutch} than to V_{English}, V_{French}, V_{German}, or V_{Italian}, respectively. We set 4 null hypotheses that we want to reject, H_0^{English}, H_0^{French}, H_0^{German}, and H_0^{Italian}, which are negations of the previous 4 hypotheses, respectively.

The way a statistical test rejects a null hypothesis H_0^{lang} is to compute a p-value p_{lang}, which is an upper bound probability that the current distances are observed if H_0^{lang} is true. Then, if the p-value is sufficiently small (e.g., $p_{lang} < \alpha$), we can reject H_0^{lang} confidently because even if H_0^{lang} is true, we make mistakes less than α of the time. Omen uses the Wald's test (Section 5.1) to derive the p-value for each null hypothesis. Because Omen tries to reject all 4 null hypotheses H_0^{English}, H_0^{French}, H_0^{German}, and H_0^{Italian}, the error probability for each null hypothesis accumulates. To upper bound the overall error probability, Omen uses the Holm-Bonferri method (Section 5.2) to derive a series of p-value thresholds for the null hypotheses from the statistical significance parameter α; briefly put, if the 4 p-values from the Wald's test are sorted into descending order, then the adjusted threshold for the j-th p-value is α/j. The p-values and thresholds for query Q2 at termination point $n = 1024$ are summarized in Figure 2c.

Continuing Execution. The above statistical test fails because the null hypothesis H_0^{French} has a p-value 0.050722 that exceeds the Holm-Bonferri adjusted threshold 0.0500. Therefore, Omen draws the conclusion that not enough hypervector elements have been processed to declare Dutch as the winning class and Omen will continue processing elements until the next termination point is reached. Omen repeats the statistical test at the second termination point after 2048 elements have been processed—at this point, the Dutch class is still the current winner. This time, the statistical test returns true, so Omen terminate inference early and returns Dutch as the classification result. For query Q2, inference is terminated after encoding and processing 2048 hypervector elements—just 20.4% of the whole hypervector.

Omen is Adaptive. Omen's early termination algorithm naturally adapts to differences in classification difficulty across inputs and to hardware error. For example, Omen does not terminate early for Q1, for it is significantly harder to classify Q1 query. Omen also naturally adjusts to any hardware errors and data corruptions. For example, if we repeat the same classification tasks but allow for bit corruptions during the distance computations, where the probability of corruption for each bit is 0.15, Omen correctly classifies Q2 but exits 1 termination point later, after processing 40.8% of the hypervector.

4 Statistical View of HDC

Omen performs early termination while also achieving a statistical upper bound α on the accuracy loss. To achieve this goal, we develop an alternative statistical interpretation of HDC classification that reframes HDC distance calculations as sampling from an underlying distribution. This view of HDC classification is critical to enabling early termination with guarantees, as it enables us to apply statistical tests to evaluate the significance of a particular distance calculation. In this section, we give an intuitive overview of the statistical view and focus on the simplest HD training algorithm

Notation	Meaning
α	significance level; upper bound of accuracy loss
V,v	V is a HV, and v is an element in V
N,n	number of total and processed HV elements $N \geq n$
D,d	vector of HV element distances, d is an element in D
P_V,P_D	underlying distribution of each v in V, or d in D
H_1,H_0,p	test and null hypothesis, p is p-value for H_0
$Z_n(D)$	average of first n samples from D, $Z(D) = \frac{1}{n}\sum_{i=1}^{n} D[i]$
$CD_n(D_1,D_2)$	average squared differences of n-prefixes of D_1 and D_2
π	a permutation $[n] \rightarrow [n]$ for some integer n
B,C	the number of basis and class HVs, respectively
W_i	a vector of the i-th dimensions of all B+C stored HVs

Table 2. Formalism symbols. HV is short for hypervectors.

that makes the hypervector elements independent and identically distributed (*iid*). The discussion and formalism of the advanced training algorithms are in Section 6. We summarize our use of notations in Table 2 for reference.

4.1 Fully Distributed Hypervectors

This statistical view of HDC applies if the hypervectors in the distance calculation are *fully distributed*, which intuitively means that the encoded information is fully distributed in all vector dimensions. This property of HDC was identified by prior work [38, 45]. We relate it to the statistical property *exchangeability* in Section 6. In this section, we consider a case of fully distributed hypervector, where each hypervector element $v = V[i]$ is *iid* from the same underlying distribution $v \sim P_V$. We call a hypervector *iid* if its elements are *iid*. If a hypervector is *iid*, then the following intuitions hold:

- The hypervector size, N, can be thought of as the maximum number of samples that may be taken from P_V. The number of hypervector elements n that have been processed corresponds to the number of samples that have been taken so far.
- Any element-wise operations ($\forall i, V[i] = op(V_1[i], V_2[i]..)$) over two or more independent *iid* hypervectors produces another *iid* hypervector.
- Permutation (such as rotational shifts) on *iid* hypervectors produces *iid* hypervectors.

4.1.1 HD Operators.

All randomly generated basis hypervectors are *iid* by construction. From this point, binding, bundling, and permutation operations all produce *iid* result hypervectors. Note that certain compositions of HD operators over random hypervectors produce hypervectors that are not *iid*, but the introduced element dependence does not affect the distance properties much, and these compositions can be replaced with alternatives that do not introduce dependence while achieving the same distance properties [74]. Therefore, we may assume compositions of the HD operators over random hypervectors produce *iid* hypervectors.

4.1.2 Element Distances/Similarities.

In HDC, hypervector distances/similarities are computed by aggregating over element-wise distances/similarities. Note that our statistical view applies to both distance and similarity measures, but we will present them as "distance" to be concise. For hypervectors V_1 and V_2, we can compute an element distance vector D where $D[i] = \text{dis}(V_1[i], V_2[i])$. Because the dis operator is element-wise, D is *iid* if V_1,V_2 are *iid*. We denote each element d in D is from distribution P_D ($d \sim P_D = \text{dis}(P_{V_1}, P_{V_2})$). For BSC, $\text{dis}(V_1[i], V_2[i]) = V_1[i]$ **xor** $V_2[i]$, the bit difference and for MAP, $\text{dis}(V_1[i], V_2[i]) = V_1[i]V_2[i]$, the product.

4.2 Distances as Expected Value Estimates

Given a *iid* distance vector D where $d \sim P_D$, the HDC distance metric $Z_n(D) = 1/n \cdot \sum_{i=1}^{n} D[i]$ estimates the expected element distance $E(P_D)$ by computing the average element distance over n processed samples. Note that the expected distance and the distance distribution are both *unknown* and has no closed-form solution in complex tasks such as classification and therefore can only be estimated by sampling the distribution. The number n of hypervector elements processed corresponds to the number of samples drawn from the underlying distributions and thus affects the quality of approximation. Conceptually, $Z_n(D) \rightarrow E(P_D)$ when n is sufficiently large.

4.2.1 Vanilla HD Classification.

In the simplest HD classification workflow, the encoder uses HD operators on randomly generated hypervectors, and the vanilla training algorithm bundles hypervector representations of training samples from a class as the class hypervector. Thus, the class hypervectors are *iid*. During inference, the classification of a test sample is based on the distances between the test sample hypervector and the class hypervectors. Since all hypervectors involved are *iid*, the associated distance vectors are also *iid*.

4.3 Distance Comparisons with Statistical Tests

We observe that with this alternate view of HDC, distance comparisons resemble sampling and testing problems, where inferential statistics, or statistical tests, can be used to assess the statistical significance of conclusions drawn from hypervector distance comparisons. Statistical tests offer guarantees that false conclusions are drawn with a probability of at most α, the user-defined significance level.

Distance Tests. We can interpret a given distance comparison as a statistical test. When comparing two hypervector distances computed with n elements, $Z_n(D_1) \leq Z_n(D_2)$, we essentially compare expected values of the underlying distance distributions $E(P_{D_1})$ and $E(P_{D_2})$, with $Z_n(D_1)$ and $Z_n(D_2)$ as their estimates. Therefore, we can formulate a *distance test* where we are testing whether the sampled hypervector distances D_1, D_2 support the test hypothesis $H_1 : E(P_{D_1}) \leq E(P_{D_2})$. We set the *null hypothesis* $H_0 : E(P_{D_1}) > E(P_{D_2})$, the negation of the hypothesis H_1 we want to obtain. When H_0 is true, we allow the incorrect conclusion H_1 to be drawn from the samples, with a probability of at most α. The statistical test computes a p-value p for H_0, where p is an upper bound on the probability that we observe D_1,D_2 when H_0 is true. If $p<\alpha$, we reject H_0 and conclude that H_1 holds. Otherwise, we cannot conclude H_1 although we observe $Z_n(D_1)\leq Z_n(D_2)$, because the observations D_1 and D_2 are not statistically significant.

5 Early Termination with Omen

In this section, we present the technical details of Omen that enables early termination of the inference for individual inputs while providing an upper bound on the accuracy loss. Omen adopts the statistical inference view of HDC presented in Section 4 and therefore works with classifier architectures that produce fully distributed vectors. Sections 5.1 and 5.2 present the statistical tests used by Omen to assess the statistical significance of the classification result and introduce the algorithmic optimizations made to improve computational efficiency. Section 5.3 presents the core inference algorithm and relates the statistical significance to accuracy loss.

5.1 Key Statistical Tests

Omen uses Wald's test [70] to perform distance tests and the Holm-Bonferri method to effectively draw conclusions from a set of distance tests. They are used together to assess the statistical significance of a classification result.

5.1.1 Distance Comparisons with the Wald's test. The Wald's test computes a p-value $W_n(D_1,D_2)$ for a hypothesis involving the expected value of two distributions $H_0: E(P_{D_1}) > E(P_{D_2})$, given n observations from D_1 and D_2. The p-value expression for applying the Wald's test to H_0 is

$$W_n(D_1,D_2)=1-\Phi\left(\frac{\sqrt{n}(Z_n(D_2)-Z_n(D_1))}{\sqrt{CD_n(D_1,D_2)-(Z_n(D_2)-Z_n(D_1))^2}}\right)$$

where Φ is the cumulative density function (CDF) of standard normal distribution, and n is the number of bits processed.

The CD_n function is cumulative squared differences between observations (distances) from two distributions, which is used to perform a *paired comparison*. CD_n is used to account for the fact that D_1 and D_2 may not be independent. For example, in classification, since the test input belongs to one class, closer to one class correlates to farther from other classes.

$$CD_n(D_1,D_2)=1/n\cdot\sum_{i=1}^{n}(D_1[i]-D_2[i])^2$$

5.1.2 Optimization: Precompute CD_n for Hamming Distance. The cumulative squared difference formula can be rewritten to eliminate a hypervector if the Hamming distance metric was used to construct D_1 and D_2 from hypervector elements and $D_1=V_1$ **xor** V and $D_2=V_2$ **xor** V. The cumulative squared difference can be rewritten to eliminate V, as $\forall i, V_1(i), V_2(i), V(i)$ are 0/1 bits:

$$\begin{aligned}
CD_n(D_1,D_2) &= 1/n\cdot\textstyle\sum_{i=1}^{n}(D_1[i]-D_2[i])^2\\
&= 1/n\cdot\textstyle\sum_{i=1}^{n}((V_1[i]\ \mathbf{xor}\ V[i])-(V_2[i]\ \mathbf{xor}\ V[i]))^2\\
&= 1/n\cdot\textstyle\sum_{i=1}^{n}(V_1[i]\ \mathbf{xor}\ V_2[i])^2\\
&= 1/n\cdot\textstyle\sum_{i=1}^{n}(V_1[i]\ \mathbf{xor}\ V_2[i])
\end{aligned}$$

Furthermore, if V_1 and V_2 are constant, such as two class hypervectors that are fixed after training, the cumulative square difference can be pre-computed for particular V_1 and V_2.

5.2 Holm-Bonferri Method

If only one distance comparison is performed, a statistical test, such as the Wald's test, may be used to determine the significance of the result. However, when multiple distance comparisons must be made, multiple tests must be performed to find out whether we can reject all null hypotheses, where the *multiple comparison problem* arises. It is the problem of accumulating error rate of rejecting null hypotheses when there is more than one statistical test.

The Holm-Bonferroni method [28] can be used to perform multiple tests that try to reject multiple ($m\geq 1$) null hypotheses $H_0^{(1)},H_0^{(2)},\ldots,H_0^{(m)}$. This algorithm guarantees that the probability of falsely rejecting *any* null hypothesis $H_0^{(i)}$ is within an upper bound of α. The algorithm achieves this by using successively stricter thresholds for the null hypotheses.

Algorithm. The algorithm first computes the p-value for each null hypothesis and then sorts all the null hypotheses by their p-values in descending order, thus assigning to each null hypothesis a distinct integer rank j, $1\leq j\leq m$, such that, for any $1\leq i_1,i_2\leq m$, if the rank of $H_0^{(i_1)}$ is less than the rank of $H_0^{(i_2)}$, then the p-value of $H_0^{(i_1)}$ is not less than the p-value of $H_0^{(i_2)}$. Then the significance test used for the null hypothesis $H_0^{(i)}$ is $p_i<\alpha_i=\frac{\alpha}{j}$, where j is the rank of $H_0^{(i)}$. If all p-value comparisons evaluate to true, we can reject all the null hypotheses. The probability that we draw this conclusion when any of the null hypothesis is true is upper bounded by α.

Optimization: rewriting p-value comparison. We can optimize the p-value comparisons for the Holm-Bonferroni method when the p-values was computed using the Wald's test. For the i-th hypothesis with a p-value $p_i=W_n(D_1,D_2)$ at rank j, the test can be rewritten as

$$\frac{n(nZ_n(D_2)-nZ_n(D_1))^2}{n(nCD_n(D_1,D_2))-(nZ_n(D_2)-nZ_n(D_1))^2}>\left(\Phi^{-1}\left(1-\frac{\alpha}{j}\right)\right)^2$$

where Φ^{-1} is the inverse of standard normal CDF. The rewrite enables several optimizations. It multiplies n to both the numerator and the denominator in the original expression in the fraction so that each normalized term $CD_n(D_1,D_2)$, $Z_n(D_2)$, and $Z_n(D_1)$ is multiplied by n, and no normalization is needed during computation. It also eliminates the square root operation and relocates Φ to the right-hand side. The $\left(\Phi^{-1}\left(1-\frac{\alpha}{j}\right)\right)^2$ term only involves the α parameter and the p-value rank j, and can be precomputed offline and stored.

5.3 Omen Inference Algorithm

The Omen inference algorithm uses the statistical view of HDC to perform early termination. The training algorithm works with full-length hypervectors of size N and produces

```
procedure INFERENCE(input,encoder,Vc,N,term_points,$\alpha$)
    k = len(Vc) // number of classes
    D = array(k,N) // element distances
    dist = zeros(k) // cumulative distances to classes
    for n in 1..N do
        vq = encoder(input, n) // encode the $n$th dimension
        for i in 1..k do
            D[i][n] = dis(vq,Vc[i][n])
            dist[i] += D[i][n]
        cand = argmin(dist) // argmax if dis is a similarity metric
        if n in term_points and confident(D,cand,n,$\alpha$) then
            return cand
    return cand
```

Figure 3. Omen's inference algorithm. N is hypervector size. Vc is class hypervectors. α is confidence level. cand is current inference result. `dis` returns distance between two hypervector elements. `confident` uses statistical tests.

a set of fully distributed class hypervectors. Omen deploys an HDC classifier that works with a fully distributed query hypervector V_q that is computed from the provided HD encoder.

5.3.1 Algorithm. Omen's algorithm is inspired from Wald's sequential probability ratio test (SPRT), which sequentially samples and terminates an experiment when the evidence is statistically significant in manufacturing quality control [70]. The SPRT is used under manufacturing conditions where sampling products is costly. We draw a connection between computational cost and product sampling cost.

Figure 3 shows the pseudo code for Omen's inference algorithm. Omen accepts as input a set of termination points, where each termination point j is reached after computing over n_j hypervector elements. We denote the number of termination points as m. Omen breaks up the inference task into $m+1$ units of work and checks if the inference task can be terminated early after each of the first m units of work. After a given work unit has been completed at the j-th termination point, n_j elements of V_q have been generated by the encoder. Omen computes the query-class element distances. After each unit of work is completed, Omen finds the current closest class **cand** to the query and assesses its statistical significance.

Next, Omen assesses the statistical significance of this classification result using the Holm-Bonferri method from Section 5.2 using the user-provided error threshold α. If the Holm-Bonferri method returns `true`, Omen terminates inference early and returns class `cand`. If the method returns `false`, Omen continues execution. If all statistical tests fail, Omen eventually returns the classification result after all N dimensions have been processed. Omen can optionally be configured to mark inputs that fail all statistical tests as difficult samples and pass them on to more reliable and sophisticated classifiers.

5.3.2 Guarantees on Classification Accuracy Loss. The statistical level α is an upper bound on the drop in classification accuracy resulting from processing fewer than N hypervector elements. Misclassifications that occur even with infinite hypervector size are attributed to *model error*. Specifically, *model error* occurs when $\exists i \neq t, E(P_{D_i}) < E(P_{D_t})$, where D_i is distance to the i-th class and t is the ground-truth label. Intuitively, this means the query is expected to be closer to other class than the ground-truth class. This form of error can only be mitigated with better encoder and classifier design.

We denote the model error rate mer as the percentage of test inputs in the test distribution that suffers from model error. Misclassifications that occur when the hypervector size is not large enough are attributed to *estimation error*. This happens when $\forall i \neq t, E(P_{D_i}) \geq E(P_{D_t})$ but the returned classification $cand \neq t$. Because $E(P_{D_{cand}}) \geq E(P_{D_t})$ but we get $Z_n(D_{cand}) < Z_n(D_t)$, this kind of error can be mitigated by computing more distance samples. The *estimation error rate* is therefore dependent on the number n of dimensions processed, which we denote as $\mathrm{eer}(n)$. The expected accuracy, when always processing all N hypervector dimensions without early termination, can therefore be defined as

$$\mathrm{acc}(N) = 1 - \mathrm{mer} - \mathrm{eer}(N)$$

Omen may terminate early, and may suffer from larger estimation error rate. Denoting $\mathrm{err}_{\mathrm{omen}}(N,\alpha)$ as the estimation error rate Omen has with confidence level α and hypervector size N, we can define Omen's accuracy as

$$\mathrm{acc}_{\mathrm{omen}}(N,\alpha) = 1 - \mathrm{mer} - \mathrm{err}_{\mathrm{omen}}(N,\alpha).$$

We also define $\mathrm{acc}_{\mathrm{opt}} = lim_{N\to\infty}\mathrm{acc} = 1 - \mathrm{mer}$, which is the best accuracy we can get out of the current model if we were to use infinitely large hypervectors and be free of estimation error. Because we have $\mathrm{err}_{\mathrm{omen}}(N,\alpha) > \mathrm{err}(N) > 0$, the accuracy relationship between the early-termination, normal inference, and optimal inference is $\mathrm{acc}_{\mathrm{omen}}(N,\alpha) < \mathrm{acc}(N) < \mathrm{acc}_{\mathrm{opt}}$. The Holm-Bonferri method guarantees that when $\forall i \neq t, E(P_{D_i}) \geq E(P_{D_t})$ (i.e., the input does not suffer from model error), we draw conclusion of $\forall i \neq cand, E(P_{D_i}) \geq E(P_{D_{cand}})$ for some $cand \neq t$ with a probability at most α. This translates to that α is an upper bound on Omen's estimation error rate, i.e., $\mathrm{err}_{\mathrm{omen}}(N,\alpha) \leq \alpha$. Therefore, we have:

$$\begin{aligned} \mathrm{acc}_{\mathrm{omen}}(N,\alpha) &= 1 - \mathrm{mer} - \mathrm{err}_{\mathrm{omen}}(N,\alpha) \\ &\geq 1 - \mathrm{mer} - \alpha \\ &\geq \mathrm{acc}_{\mathrm{opt}} - \alpha \\ &\geq \mathrm{acc}(N) - \alpha \end{aligned}$$

In summary, the statistical test employed by Omen ensures that the accuracy loss due to early termination over using the whole vector is at most α, or $\mathrm{acc}(N) - \mathrm{acc}_{\mathrm{omen}}(N,\alpha) \leq \alpha$.

6 Exchangeability

While Section 4 intuitively explains the *iid* property of HDC in the simple case, the state-of-the-art training algorithms often break the independence between the hypervector elements as they use aggregate information of all elements (e.g., hypervector distances) to update them. We find that although the learned hypervectors are not *iid*, they are *exchangeable*, which is a weaker statistical property than *iid* and relates nicely to the *fully distributed* property. In this section, we formalize exchangeability in Section 6.1, prove that the two representative training algorithms produce exchangeable hypervectors in Section 6.2 and discuss the relation to the *fully distributed* property and Wald's test in Section 6.3.

6.1 Definition of Exchangeablity

We formally define *exchangeability* in this section. We first define *permutation* and *symmetric functions* as follows.

Definition 1 (Permutation). *Denote $m \in \mathbb{Z}$ as an integer, and $[m]$ as the set of integers from 1 to m. A permutation π is a function $[m] \rightarrow [m]$, where $\{\pi(i) : i \in [m]\} = [m]$. Because π is a one-to-one mapping from $[m]$ to $[m]$, it is inversible, and we denote the inverse of π as π^{-1}. For a vector of n elements $W = (W_1, W_2, \ldots, W_n)$, we denote W shuffled by π as $\pi W = (W_{\pi(1)}, W_{\pi(2)}, \ldots, W_{\pi(n)})$.*

Definition 2 (Symmetric Function). *A function $f(X)$, where $X = (x_1, \ldots, x_n)$ is a n-element vector, is symmetric about X if $\forall \pi, f(\pi X) = f(X)$.*

The definition of exchangeability is taken from Kuchibhotla [48] (real-valued random variables, assuming the probability density function exists).

Definition 3 (Exchangeablility). *Suppose $W = (W_1, \ldots, W_n) \in \mathcal{R}^n$ is a vector of real-valued random variables, W_i's are exchangeable iff their joint probability density function $p(W)$ is symmetric about W.*

iid implies exchangeable because if W_i's are *iid*, we have $p(W) = \Pi_i p(W_i) = \Pi_i f(W_i)$, where f is the identical probability density function shared by all the W_i's. $p(W)$ function is symmetric about W because $p(\pi W) = \Pi_i f(W_{\pi(i)}) = \Pi_i f(W_i) = p(W)$ due to the commutativity of multiplication. However, the inverse is generally not true, because there can be dependence among W_i's.

6.2 Learned Hypervectors Are Exchangeable

In this section, we formalize some representative advanced HDC training algorithms and prove that they produce exchangeable hypervectors.

6.2.1 Primitives.

We first define some symmetry-related concepts, and prove some useful theorems.

Definition 4 (Element-wise Function). *A function $f(X) = Y$ where X, Y are n-element vectors is called element-wise if $f(\pi X) = \pi f(X)$.*

Lemma 1. *Composition of element-wise functions produces element-wise functions.*

Proof. If f and g are element-wise, for any n-element vector X, we have $f(g(\pi X)) = f(\pi g(X)) = \pi f(g(X))$, which implies that $f \circ g$ is element-wise. □

Lemma 2. *Composition of symmetric and element-wise functions are symmetric.*

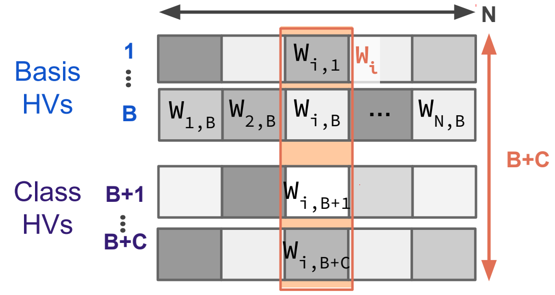


Figure 4. HD representation - W's definition.

Proof. If f is symmetric and g is element-wise, $f \circ g$ is symmetric because $\forall X, f(g(\pi X)) = f(\pi g(X)) = f(g(X))$, and $g \circ f$ is also symmetric because $\forall X, g(f(\pi X)) = g(f(X))$. □

Lemma 3. *For a function $f(X)$ symmetric about X, f's gradient $\nabla f(X) = (\frac{\partial f}{\partial x_1}(X), \ldots, \frac{\partial f}{\partial x_n}(X))$ satisfies $\nabla f(\pi X) = \pi \nabla f(X)$.*

Proof. $\nabla f(\pi X) = (\frac{\partial f}{\partial x_1}(\pi X), \ldots, \frac{\partial f}{\partial x_n}(\pi X))$. We have $\forall i \in [n]$,

$$\begin{aligned} \frac{\partial f}{\partial x_i}(\pi X) &= \lim_{\epsilon \rightarrow 0} 1/\epsilon \cdot (f(\pi X + \epsilon e_i) - f(\pi X)) \\ &= \lim_{\epsilon \rightarrow 0} 1/\epsilon \cdot \left(f(X + \epsilon e_{\pi(i)}) - f(X) \right) \\ &= \frac{\partial f}{\partial x_{\pi(i)}}(X) \end{aligned}$$

where e_i is a n-element vector with the i-th element being 1 and others being 0. The first and third equalities are the definition of the derivative, and the second equality exploits the symmetry of $f(x)$. Note that $\pi^{-1} e_i$ is $e_{\pi(i)}$ instead of $e_{\pi^{-1}(i)}$ (Definition 1). This is equivalent to $\nabla f(\pi X) = \pi \nabla f(X)$. □

Because the hypervectors are initialized to be exchangeable (e.g., zeros or random Gaussian), we only need to prove that each update of the training algorithms *preserves exchangeability*. A transformation from a vector of random variables to another vector of random variables is *exchangeability preserving* iff for all exchangeable input random variables, the transformed output random variables are also exchangeable. The following theorem states the condition for which a transformation on a vector of exchangeable random variables preserves exchangeability [11, 48].

Theorem 1 (Exchangeability Preservation). *Suppose $W = (W_1, \ldots, W_n) \in \mathcal{W}^n$ is a vector of exchangeable random variables. Fix a transformation $G : \mathcal{W}^n \rightarrow (\mathcal{W}')^m$. If for each permutation $\pi_1 : [m] \rightarrow [m]$ there exists a permutation $\pi_2 : [n] \rightarrow [n]$ such that*

$$\pi_1 G(W) = G(\pi_2 W), \quad \text{for all } W \in \mathcal{W}^n$$

then $G(\cdot)$ preserves exchangeability of W.

Corollary 1. *Element-wise functions are exchangeability preserving transformations.*

Proof. If $f : \mathcal{W}^n \rightarrow (\mathcal{W}')^n$ is element-wise, for each $\pi_1 : [n] \rightarrow [n]$, let $\pi_2 = \pi_1$, then we have $\pi_1 f(W) = f(\pi_2 W), \forall W \in \mathcal{W}^n$. By Theorem (1), f preserves exchangeability. □

6.2.2 HDC Representation W.

Our insight is that although the HDC training algorithms update hypervectors in a complex way, they do not break the symmetry among hypervector elements for all the hypervectors. To show that, we need to consider all hypervectors we have (basis hypervectors in codebooks and class hypervectors) as a whole. Denote N as our hypervector length, W_i as a vector of the i-th dimensions of all B basis hypervectors in codebooks and C class hypervectors. We illustrate W's definition in Figure 4. Using the same notation as theorem (1), denote $W=(W_1,...,W_N)$. W is initialized *iid*. We demonstrate that the training algorithms are exchangeability preserving transformations on W.

6.2.3 Implications of W's Exchangeability.

In this section, we prove that when the HDC representation W defined above is exchangeable, the encoding produces exchangeable hypervectors, and their distances to the class hypervectors are symmetric about W.

Theorem 2 (Exchangeable Encoding). *When W is exchangeable, HD encoding* $\mathrm{enc}(W,T)$ *produces a vector of exchangeable elements, where T is an input data to encode.*

Proof. HD encoding is a function $\mathrm{enc}(W,T)$ of W (specifically only the codebook part $W_{j,i}, i \leq B$) and an input T. enc is a composition of HD operators binding, bundling, and permuting. They are element-wise functions on the codebook.[1] Therefore, by Lemma (1), enc is an element-wise function. By Corollary (1), enc is exchangeability preserving, so when W is exchangeable, its output hypervector is also exchangeable. □

Corollary 2 (Exchangeable Element Distances). *When W is exchangeable, element distances* $\mathrm{ed}(W,T)$ *between an encoded hypervector* $\mathrm{enc}(W,T)$ *and class hypervectors are exchangeable.*

Proof. Denote $\mathrm{ed}(W,T)_{j,i} = \mathrm{dis}(\mathrm{enc}(W,T)_j, W_{j,B+i})$ as the j-th element distance from the encoded hypervector to the i-th class hypervector. Since $\mathrm{enc}(W,T)$ is element-wise, we have

$$\begin{aligned}\mathrm{ed}(\pi W,T)_{j,i} &= \mathrm{dis}\big(\mathrm{enc}(\pi W,T)_j,(\pi W)_{j,B+i}\big)\\ &= \mathrm{dis}\big((\pi\mathrm{enc}(W,T))_j,(\pi W)_{j,B+i}\big)\\ &= (\pi\mathrm{ed}(W,T))_{j,i}\end{aligned}$$

Therefore, $\mathrm{ed}(W,T)$ is by definition also an element-wise function (Lemma (1)) and preserves exchangeability (Corollary (1)), and the result length-N vector is exchangeable when W is exchangeable. □

Corollary 3 (Symmetric Distance). *When W is exchangeable, distances $d(W,T)$ between an encoded hypervector and the class hypervectors are symmetric about W.*

Proof. Denote $d(W,T)$, a vector of length C, as the HD distances from a encoded hypervector $\mathrm{enc}(W,T)$ to the C class hypervectors, that is, $d(W,T)_i = 1/N \cdot \sum_j \mathrm{ed}(W,T)_{j,i}$. $d(W,T)$ a composition of the element-wise function $\mathrm{ed}(W,T)$ and the symmetric function of averaging. Therefore, by Lemma (2), $d(W,T)$ is a symmetric function about W. □

6.2.4 Training Preserves Exchangeability.

We now formalize a distance-based iterative training algorithm and a gradient descent-based training algorithm, and prove that they preserve the exchangeablility of W.

Distance Based. In a distance-based iterative training algorithm, each transformation takes a training sample T with label t, encodes T into a hypervector $\mathrm{enc}(W,T)$ using the codebook, adds it to the class hypervectors with a weight vector $\mathrm{wt}(W,T,t)$ of length C. The weight applied to the hypervector of class t' is $\mathrm{wt}(W,T,t)_{t'}$, which is computed from the distance $d(W,T)_{t'}$ and whether $t'=t$. Usually, $\mathrm{wt}(W,T,t)_{t'} \geq 0$ when $t'=t$, and $\mathrm{wt}(W,T,t)_{t'} \leq 0$ when $t' \neq t$. OnlineHD training algorithm [26] is an example of this kind. It selects the update weights to prevent model saturation. The transformations can be formalized as follows. From the way the weights are computed, they can be rewritten as $\mathrm{wt}(W,T,t) = \sigma(d(W,T),t)$, where σ is a function that depends on the concrete update strategies and hyperparameters like learning rate. Because $d(W,T)$ is symmetric about W by Corollary (3), $\mathrm{wt}(\pi W,T,t) = \sigma(d(\pi W,T),t) = \sigma(d(W,T),t) = \mathrm{wt}(W,T,t)$. Therefore, $\mathrm{wt}(W,T,t)$ is also symmetric about W. The transformation G adds the encoded hypervector scaled by the weights to the class hypervectors

$$G(W)_{j,B+i} = W_{j,B+i} + \mathrm{wt}_i(W,T,t)\cdot\mathrm{enc}(W,T)_j.$$

Now we prove that G is element-wise. For each $\pi:[N]\longrightarrow[N]$,

$$\begin{aligned}(\pi G(W))_{j,B+i} &= G(W)_{\pi(j),B+i}\\ &= W_{\pi(j),B+i} + \mathrm{wt}_i(W,T,t)\cdot\mathrm{enc}(W,T)_{\pi(j)}\\ &= (\pi W)_{j,B+i} + \mathrm{wt}_i(\pi W,T,t)\cdot\mathrm{enc}(\pi W,T)_j\\ &= G(\pi W)_{j,B+i}.\end{aligned}$$

The second to last equation makes use of the fact that wt_i is a symmetric function about W and enc is an element-wise function. For $i \leq B$, since G does not change the $W_{j,i}, i \leq B$ (codebook), we also have $(\pi G(W))_{j,i} = (\pi W)_{j,i} = G(\pi W)_{j,i}$. By Corollary (1), G preserves exchangeability.

Gradient Descent. In a gradient descent-based training algorithm, each transformation uses the gradient of a loss function to update the hypervectors. The loss function is of the form $\mathrm{loss}(W,T,t) = \sigma'(d(W,T),t)$, where T is a training input with label t and σ' usually computes the loss function (e.g., cross-entropy) comparing the distance vector $d(W,T)$ applied by softmax to the ground-truth one-hot vector e_t. Due to the same reason as $\mathrm{wt}(W,T,t)$, $\mathrm{loss}(W,T,t)$ is also symmetric about W. There are different variants of gradient descent-based training algorithms. LeHDC [14] freezes the basis hypervectors and learns class hypervector representations. LDC [15] learns the basis hypervectors together with the class hypervectors. We prove a general theorem for them.

Theorem 3. *When the loss function given any input values is symmetric about W, gradient descent updates on W are exchangeability-preserving.*

[1] Strictly speaking, permuting is not an element-wise function by our definition. However, because it is a special kind of exchangeability preserving function, has similar properties as element-wise functions, and can be replaced by binding in the HD encoding [45], we omit this detail to keep the proof simple.

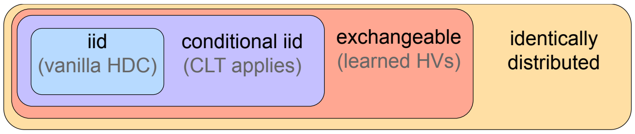


Figure 5. Relationship between related statistical concepts. Central limit theorem (CLT) applies to conditional *iid* variables, which can approximate exchangeable variables.

Proof. Because $\text{loss}(W,T,t)$ is also symmetric about W for all T and t, by Lemma (3), we can derive $\nabla_W \text{loss}(\pi W,T,t) = \pi \nabla_W \text{loss}(W,T,t)$ for any $\pi : [N] \to [N]$. Therefore, denoting $\text{grad}_{j,i}(W,T,t) = \frac{\partial \text{loss}(W,T,t)}{\partial W_{ji}}$ as the gradient for W_j at its i-th element, we have $\text{grad}_{\pi(j),i}(W,T,t) = \text{grad}_{j,i}(\pi W,T,t)$ for any π. Then, each update in the gradient descent-based training algorithm is a transformation G as follows

$$G(W)_{j,i} = W_{j,i} - \text{lr} \cdot \text{grad}_{j,i}(W,T,t)$$

where lr is a constant learning rate. We prove that G is element-wise. For each $\pi : [N] \to [N]$, we have

$$\begin{array}{rcl} (\pi G(W))_{j,i} & = & G(W)_{\pi(j),i} \\ & = & W_{\pi(j),i} - \text{lr} \cdot \text{grad}_{\pi(j),i}(W,T,t) \\ & = & (\pi W)_{j,i} - \text{lr} \cdot \text{grad}_{j,i}(\pi W,T,t) \\ & = & G(\pi W)_{j,i}. \end{array}$$

Therefore, by Corollary (1), G preserves exchangeability. □

The theorem naturally applies to LDC. For LeHDC, since it only updates the class hypervectors ($W_{j,B+i}, 1 \le i \le C$), we set $\text{grad}_{j,i}(W,T,t)$ to 0 for $1 \le i \le B$, which does not break Lemma (3). The proof works with this modification.

6.3 Discussion

In this section, we discuss the relation between exchangeability and the fully distributed property, and the applicability of Wald's test to exchangeable hypervectors. In Figure 5, we show a Venn diagram that illustrates the relationship between exchangeability and related concepts.

6.3.1 Relation to Fully Distributed Property. The *fully distributed* property of HDC intuitively means that the information encoded by the hypervector is equally distributed among the hypervector elements. Each element is not more or less important than or different from the other elements, while all elements as a whole represent the information. One may think that correlation-introducing training algorithms would break the fully distributed property and thus make the model less error-resilient. Counter-intuitively, prior work found that the training algorithms using gradient descent do not make the HDC classification model less error resilient [15]. This finding indicates that, while the hypervectors generated by these correlation-introducing training algorithms are not *iid*, the information is still fully distributed in them. Therefore, *iid* is possibly too strong and not a necessary condition for the fully distributed property. The question then arises, is there a less restrictive statistical property that satisfies the fully distributed property that makes the model error resilient?

We postulate that the *exchangeability* property offers the *fully distributed* property without the strict independence constraint. Exchangeable hypervectors have identically distributed elements. We believe the *identically distributed* property is necessary to ensure the hypervector is fully distributed because it intuitively specifies that each element is sampled from the same information distribution. Non-identically distributed hypervectors may have elements sampled from different distributions that contain disjoint information, and not all information is guaranteed to have the necessary redundancy that makes the representation error resilient.

However, we believe that the identically distributed property alone is not sufficient to ensure the vector is fully distributed, as it does not guarantee the information is *evenly* distributed throughout the vector. Consider an example where we have a hypervector V, and each element $v \sim P_V$ is identically distributed. If $V[1]$ and $V[2]$ are independently drawn from P_V, while other $V[i]$'s for $2 < i \le n$ are copies of $V[2]$, then $V[1]$ is more important and contains as much information about the underlying distribution as all the other elements combined. In fact, the information entropy of $V[1]$ is equal to that of all other $V[i]$'s, implying information is not evenly distributed through out the vector. The exchangeability property ensures the elements in the hypervector are symmetric, since elements may be permuted without changing the joint distribution function.

6.3.2 Assumptions Made by Wald's Test. We next provide a theoretical justification for applying Wald's test to exchangeable partial distances in HD inference. The Wald's test assumes that the samples follow the central limit theorem (CLT), such that the test statistic converges to a normal distribution. The hypervectors and distances described in Section 6.2 are exchangeable; however the CLT does not generally hold for exchangeable random variables.

We describe two methods to show CLT holds for exchangeable random variables. First, exchangeable random variables follow the CLT if the covariance of each pair of variables is zero, as this implies the variables are conditionally *iid* [7, 40]. Second, finite exchangeable random variables can be approximated with conditionally *iid* random variables [13], which follow CLT. We note the second method is always applicable, but the approximation bound is loose. We leave examining whether the first method applies to learned hypervectors to future work, this method is desirable as it does not introduce an approximation bound. We stress that, empirically, we find the Wald's test effectively and robustly bounds the accuracy loss by α (Section 7) for learned classifiers.

7 Evaluation

We evaluate Omen and compare it with other baselines in various benchmarks.

dataset	train/test	classes	description	NN acc
lang	42855/7145	5	language classification [21]	-
ucihar	7352/2947	6	activity recognition [3]	94.02 [12]
isolet	62371/560	26	voice recognition [4]	95.50 [69]
mnist	60000/10000	10	image classification [52]	95.83 [26]

Table 3. Dataset configurations.

Name	Algorithm	Encoder	Class	Size	Initial/Step
			HV		Terms (it/ct)
OHD	OnlineHD [26]	random	weighted ⊕	10048	512/64
LeH	LeHDC [14]	random	learned	5056	512/64
LDC	LDC [15]	learned	learned	256	16/4

Table 4. Hypervector (HV) Training algorithm descriptions.

7.1 Experimental Setup

Datasets. Following prior work on HDC [15, 26], we evaluate on four datasets fit for edge-inference scenarios. Table 3 summarizes train/test splits, the number of classes, and the state-of-the-art lightweight neural network accuracies for these datasets. To adapt these datasets for HDC, we use positional encoding or HD encodings from prior literature to embed information into hypervectors [26, 62]. The language dataset uses the trigram encoding-based architecture described in Section 3, which differs from its instantiation in prior work.

HDC Training Algorithms. We construct benchmark HD classifiers from the datasets using three training algorithms, where each algorithm is used with both the BSC and MAP-HDC variants. The algorithms are summarized in Table 4 and are described in Section 6.2.4. Table 4 summarizes each training method, reports the encoding/class hypervector dimension used for each training method (Size), the number of dimensions to the first termination point (it), and the number of dimensions between subsequent termination points (ct). Sizes are multiples of 64 to enable bit-packing, and it is generally $4x$ larger than ct to ensure enough information is processed before the first termination point is reached. For language dataset, only OnlineHD training algorithm and the HDC-BSC variant are used, as it is an illustrative example. This totals to $3\times2\times3+1=19$ dataset/training algorithm/HD combinations, which comprise the benchmarks of our evaluation.

Baselines. We evaluate Omen with confidence levels α = 1%,5% or 10%, capturing conservative, balanced, and aggressive optimization modes. We compare Omen to five baselines that capture different optimization and early termination strategies. The smaller vector (SV) baseline uses the first n dimensions of hypervectors for inference, where n is the average number of dimensions used by Omen α = 5% on the same benchmark. The remaining baselines implement heuristic early termination strategies. We perform hyperparameter tuning to find the optimal threshold for each dataset/training method combination that achieves a 1% accuracy loss on the training data. The Diff and Absolute strategies terminate early if the winner's marginal and absolute similarities exceed a class-agnostic threshold. Prior work [55] uses a table of class-specific termination thresholds derived from training data to enable specializing the termination point to each class. We adapt the threshold selection algorithm to consider in-class vs. not-in-class samples instead of correctly classified vs. misclassified samples, enabling threshold selection for classes which have no misclassified samples.

Execution Setup. We implement the baselines in C++ to measure the inference performance. For Omen, we precompute the squared distance differences CD for BSC at set termination points (described in Section 5.1.2), and lazily evaluate CD for MAP. We also apply bit-packing and precomputation optimizations to the heuristic baselines. Specifically, we precompute class statistics required for the normalized similarity metric used in the Diff, Absolute and Mean baselines. We conduct the evaluation on a Microcontroller (MCU) for the models that can fit in to mimic real edge inference scenarios, and on our local desktop otherwise. We use B-U585I-IOT02A, an evaluation board with a STM32U585AI MCU. This MCU has one Arm Cortex-M33 core, 2MB of flash memory, and 786KB of SRAM. In the MCU, the flash memory and SRAM sizes limit the hypervector's size and data types. We are able to fit all models in the MCU except the 10048-dimensional MAP OHD and the 5056-dimensional MAP LeH. The desktop machine has 64GB system memory and 24-core CPU (32 threads, Intel(R) Core(R) CPU i9-13900K with 8 performance cores @ 3.00 GHz and 16 efficiency cores @ 2.20 GHz). Our program was pinned on core 0 and is single-threaded.

Metrics. The absolute accuracy (acc) and average runtime is reported for the unoptimized HDC classifier. For each other baseline, the relative percent accuracy loss and the speedup relative to the unoptimized baseline are reported.

7.2 Evaluation Results

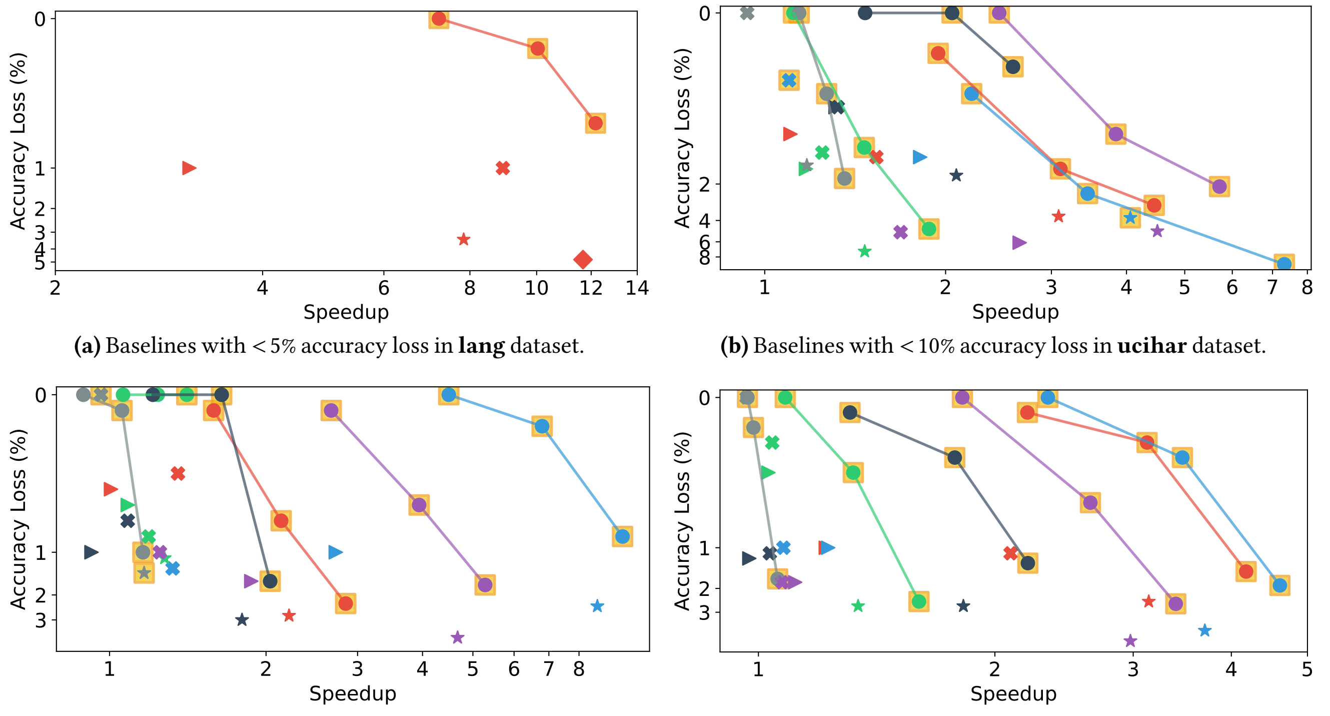
Figure 6 presents accuracy loss and speedups attained by Omen and the heuristic optimization methods, relative to an unoptimized baseline. The absolute accuracy and performance of the unoptimized baseline is summarized in Table 5.

Runtime. Omen offers 1.08–7.21× inference speed-up for $\alpha=1\%$, 1.24–10.04× inference speed-up for $\alpha=5\%$, and 1.41–12.18× inference speed-up for $\alpha=10\%$ on 16 out of 19 benchmarks, excluding 3 LDC-BSC benchmarks. Omen is able to deliver 1.36×, 1.16× and 1.06× speed-up with α = 10% on 3 LDC-BSC benchmarks, while the speedup when α = 1% or $\alpha=5\%$ is smaller, and we discuss the reasons in Section 7.2.1.

Other early termination baselines Diff, Absolute, and Mean offer less significant speedup comparing to Omen in iso-accuracy. In Figure 6, we find that almost all Omen baselines are on the (accuracy loss, speedup) Pareto Frontier in every benchmark. The only 5 exceptions are due to over-conservatism - they have 0 accuracy loss, and 4 of them are

dataset	lang	ucihar						isolet						mnist					
Training Algo	OHD	OHD		LeH		LDC		OHD		LeH		LDC		OHD		LeH		LDC	
HDC variant	BSC	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP	BSC	MAP
Accuracy (%)	98.5	83.5	91.9	88.1	95.3	93.5	94.6	85.2	94.4	88.5	96.3	91.3	96.2	85.5	96.9	93.0	97.7	96.2	97.5
Time (ms)	62.8	243.9	4.4†	123.1	1.6†	12.1	70.2	268.6	5.7†	135.4	1.9†	14.1	78.8	301.2	5.9†	151.1	2.3†	16.5	66.4

Table 5. Unoptimized Execution. Runtime measured on local laptop is marked as †. Other runtimes are collected on a MCU.



(a) Baselines with < 5% accuracy loss in **lang** dataset.

(b) Baselines with < 10% accuracy loss in **ucihar** dataset.

(c) Baselines with < 5% accuracy loss in **isolet** dataset.

(d) Baselines with < 5% accuracy loss in **mnist** dataset.

Figure 6. Accuracy Loss vs. Inference Speedup compared to the unoptimized baseline. Colors encode training algorithms. 🟥 is OHD-BSC, 🟦 is OHD-MAP, 🟩 is LeH-BSC, 🟪 is LeH-MAP, ■ is LDC-BSC, ■ is LDC-MAP. Marker shapes encode baselines. ● is Omen, ✖ is Diff, ► is Absolute, ◆ is Mean, ★ is SV. For each benchmark, we connect 3 Omen baselines in ascending α order with lines, and baselines on the Parato frontier are highlighted in 🟨.

dominated by other Omen baselines. In each benchmark, comparing to other baselines, there is at least one Omen baseline that delivers an additional 0.05–6.15× dimension reduction and 0.04–5.85× speedup over the Unoptimized baseline, while achieving higher or comparable accuracy.

Accuracy. Omen robustly achieves accuracies above the accuracy lower bound (Unoptimized accuracy - α). Omen's accuracy loss is 0.0–0.6%, with no accuracy loss on 13 benchmarks for $\alpha = 1\%$, and 0.0–2.4% for $\alpha = 5\%$. For $\alpha = 10\%$, Omen's accuracy loss is 0.0–4.7% on all benchmarks except OHD-MAP-ucihar. In OHD-MAP-ucihar the accuracy loss is 9.2%, close to the upper bound 10%. Omen α = 5% also achieves 0.8–7.2% less accuracy loss than SV, while processing the same average number of dimensions. We also note that HDC baselines achieve better accuracy than lightweight neural network-based models reported in Table 3.

Baseline	Encode	Distance Compute	Statistical Tests	**Total**
Unoptimized	243.40	0.10	-	243.50
Omen $\alpha = 5\%$	77.67	0.04	0.59	78.30

Table 6. Average latency (ms) breakdown for Unoptimized and Omen $\alpha = 5\%$ in OHD-BSC-ucihar benchmark.

We note that among 3 early termination heuristics, Diff is relatively stable in accuracy, with 0.0–5.0% accuracy loss, while Absolute is unstable and has > 30% accuracy loss in 3 benchmarks, and Mean has < 10% accuracy loss in only 1 benchmark. We conjecture that the poor accuracy of Mean is because this termination strategy must be paired with the specific training algorithm described in previous work [55].

7.2.1 Runtime vs. Dimensionality Reduction.

We show the relation between Omen's runtime speed-up ratio (SUR) and dimension reduction ratio (DRR) in Figure 7. We find

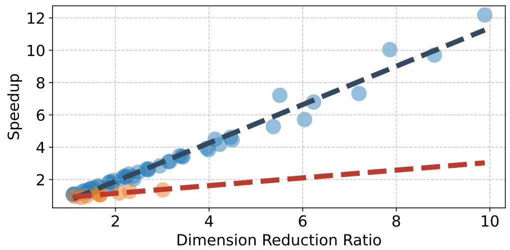


Figure 7. Speedup vs. Dimension Reduction for Omen with various α (1%, 5%, 10%). 🟥 is on LDC-BSC benchmarks and 🟦 is on other benchmarks. Dashed lines are fitted lines for the two sets of benchmarks, respectively.

Benchmark	ucihar		isolet		mnist	
	2BPC	3BPC	2BPC	3BPC	2BPC	3BPC
Accuracy	92.7	88.6	91.2	80.0	94.5	75.0

Table 7. Accuracy of Unoptimized Baseline in LDC-BSC benchmarks in presence of hardware errors.

that Omen's dimension reduction directly translates to inference speedup in most benchmarks. On 16 non-LDC-BSC benchmarks, SUR and DRR can be linearly fitted with $\text{SUR} = 1.187 \cdot \text{DRR} - 0.483$, with a small mean squared error of 0.144, indicating that Omen's runtime overhead is small. We show an example average latency breakdown of the Unoptimized and Omen $\alpha = 5\%$ baselines in the OHD-BSC-ucihar benchmark in Table 6. In this benchmark, encoding contributes the main latency, accounting for $> 99\%$ in both baselines. Omen's statistical tests account for only 0.76% of the total latency while significantly reducing the encoding overhead by 68.1% compared to the unoptimized baseline.

On 3 LDC-BSC benchmarks, the fitted line is $\text{SUR} = 0.239 \cdot \text{DRR} + 0.669$, with mean squared error 0.002. We think the slope is smaller for two reasons. First, the LDC-BSC benchmarks learn very compressed hypervector representations with $N = 256$, leaving less room for optimizations and making it more difficult to terminate early without losing accuracy. We note that other baselines also fail to optimize in these 3 benchmarks, either offering no dimension reduction and making inference slower, or suffering from a huge $> 30\%$ accuracy loss. Second, the LDC-BSC benchmarks use only binary/integer operators, while Omen's statistical tests use floating-point operations (FLOPs). We note that this issue can be potentially mitigated by approximating the floating-point value comparisons with integer operations. We leave more unsound optimizations of Omen to future work.

7.3 Robustness to Hardware Noise

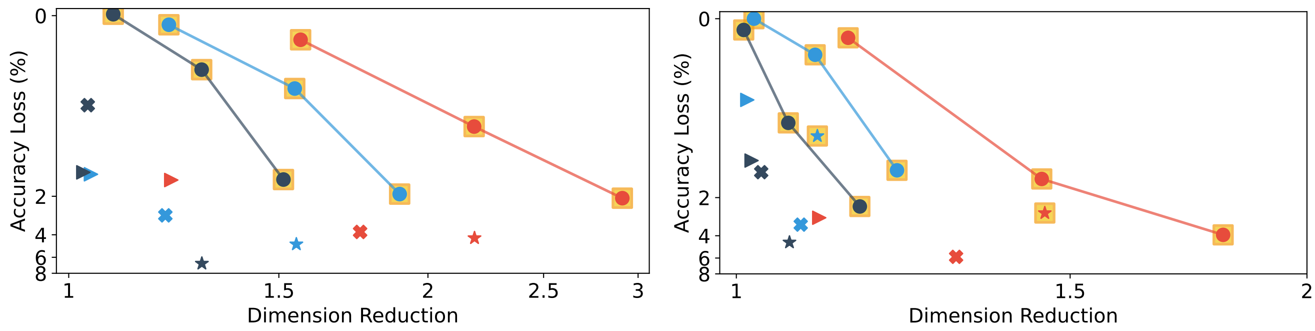
HDC is noise-resilient due to its *fully distributed* property and thus is well-suited for noisy emerging hardware technologies. Provided the noise distribution is identically distributed as hardware errors can be treated as samples from a single probability distribution, Omen's assumptions still hold and can be readily applied without adaption or tuning.

We evaluate Omen with a noisy hardware model in which bits are flipped with a fixed probability, referred to as the bit error rate (BER). The model approximates the error distribution on the resistive RAM (ReRAM) hardware [29, 51, 71]. ReRAM offers benefits such as larger memory density, non-volatility, and faster access, but is prone to bit corruption. We investigate BERs of 0.0215 and 0.1273, BERs collected from fabricated ReRAM hardware when setting the memory density to 2 bits per cell (2BPC), and 3BPC, respectively [71]. We evaluate the baselines on LDC-BSC benchmarks as they have the smallest model and are suitable for deployment in resource-constrained and potentially error-prone scenarios. We use the same parameters as in the error-free experiments.

Figure 8 shows baselines' accuracy loss and dimension reduction ratio relative to the unoptimized baseline (Table 7). Omen delivers a decent accuracy-performance trade-off with the accuracy loss robustly bounded by α. Across the 6 benchmarks, Omen $\alpha = 1\%$ delivers 1.01–1.56× dimension reduction with 0.0–0.2% accuracy loss, Omen $\alpha = 5\%$ delivers 1.07–2.19× dimension reduction with 0.3–1.4% accuracy drop, and Omen $\alpha = 10\%$ delivers 1.16–2.91× dimension reduction with 1.2–3.9% accuracy drop. All Omen baselines are on the Pareto Frontier. In contrast, Diff has an accuracy loss of 0.6–5.9%, and Mean has >20% accuracy loss on all benchmarks. Absolute becomes over conservative, as the overall similarities between hypervectors decrease under bit corruption, and thus it becomes harder to meet the absolute threshold. Absolute delivers only 1.02–1.22× dimension reduction, while still suffering a 0.6–2.9% accuracy drop. Omen with $\alpha = 5\%$ consistently outperforms all heuristics that have $< 10\%$ accuracy loss in both accuracy and dimension reduction. Although tuning the heuristics' thresholds with bit corruptions could improve accuracy, it is time-consuming, and the error characteristics in deployment may differ from the training environment.

8 Related Work

HDC/VSA. Hyperdimensional Computation (HDC) or Vector Symbolic Architectures (VSA) is a brain-inspired computational paradigm that has received increasing attention [5, 19, 24, 42, 44, 45]. HDC/VSA has shown great potential in various tasks, such as Genome Sequence Matching [8], Internet of Things systems [31], illness detection [20, 63], Out-of-Distribution Detection [72], image translation [66], neuro-symbolic AI [27], time-series classification [64], and robust hashing [25]. Various training frameworks for HDC/VSA based ML have been proposed, including iterative, online, optimization-based methods [6, 14, 15, 26, 56]. Due to its error resiliency, HDC/VSA is recognized as a suitable computing paradigm for emerging hardware technologies [32, 39, 50, 67]. Besides BSC and MAP that we target in this paper, other HDC/VSA systems can work with unit-length, real-valued, complex-valued, sparse binary, residue numbers and matrix-as-element hypervectors [1, 18, 49, 57–61, 65, 73]. Omen's



(a) Baselines with < 10% accuracy loss with 2BPC hardware error rate. **(b)** Baselines with < 10% accuracy loss with 3BPC hardware error rate.

Figure 8. Accuracy Loss vs. Dimension Reduction in LDC-BSC benchmarks compared to the UNOPTIMIZED baseline. Colors encode datasets. 🟥 is ucihar, 🟦 is isolet, ⬛ is mnist. Marker shapes encode baselines. ● is OMEN, ✖ is DIFF, ▶ is ABSOLUTE, ◆ is MEAN, ★ is SV. For each benchmark, we connect 3 OMEN baselines ($\alpha = 1\%$, 5% and 10% in order) with lines, and baselines on the Parato frontier are highlighted in 🟨. All MEAN baselines have too high accuracy losses and are all filtered out in the figures.

general idea can be applied to other HDC/VSA systems with slight modifications on the inferential statistics. We leave the application OMEN to other HDC/VSA systems, to other applications, and to combine with other techniques to future work.

Global HDC optimizations. Various theoretical results explored the relationship between the hypervector size and computation accuracy [10, 16, 17, 38, 41, 41, 43, 46, 57, 68, 75]. Exploiting these theories, Yi and Achour [74] proposed a static analysis framework that computes optimal hypervector sizes and parameters given the computation specifications. The theories are difficult to apply to practical machine learning, such as classification tasks, and researchers have developed various heuristics for global HDC optimizations in these tasks. Morris et al. traded off hypervector size and accuracy [54] by compressing the class hypervectors. Safa et al. and Imani et al. improved inference performance by sparsifying the class hypervectors [33, 63]. OMEN can be applied on top of most global HDC optimization techniques, as long as they do not break the fully distributedness of hypervectors.

Per-inference HDC optimizations. Chuang et al. [9] identified and exploited the difference in the difficulty of inference between test samples to improve HDC performance. They defined the confidence metric as the Hamming distance difference between the classes with the smallest distances and used the binary or integer HDC model depending on whether the confidence surpasses a threshold. Imani et al. [30] used a small 2000-bit model and a large 10000-bit model. They run inference first on the small model and checked whether the similarity value of any class surpasses a specified threshold. If not, the inference used the large model. Similar similarity threshold-based approaches have been used in genome sequence search and adaptive HD model quantization [8, 55, 76]. The confidence metrics in these techniques are heuristics and provide no guarantee on the accuracy loss. OMEN defines confidence as statistical significance and provides accuracy guarantees.

Applying statistical approaches to computing systems. Unlike heuristic approaches, statistical approaches are theoretically solid and offer guarantees. They are thus widely applied in various computer science problems, including machine learning [34], model checking [53], software testing [35], and data stream processing [2]. To our knowledge, no previous work has focused on edge setting and OMEN is the first statistical approach-based tool to optimize the performance of edge computing systems.

9 Conclusion

We presented OMEN, a dynamic HDC optimizer that improves inference performance by terminating the inference early on a per-input basis while providing guarantees on the accuracy loss. OMEN identified good termination points per input. We demonstrated that OMEN achieved a significant inference speedup with only a very small drop in accuracy, and OMEN's termination improved over heuristic approaches. OMEN could be used to perform an early termination even in the presence of hardware errors. OMEN required little-to-no modification of the HDC training algorithms and could be applied on top of other optimization methods. OMEN's new statistical perspective on HDC opens doors for applying statistical research to HDC classification problems. OMEN is highly configurable and extendable. Based on OMEN, future work could study optimal strategies for selecting termination points and more aggressive and perhaps unsound HDC performance optimizations.

Acknowledgements. We thank Arun Kumar Kuchibhotla for the helpful discussion on exchangeability, Christopher Matthew De Sa for serving as our shepherd and for helpful feedback to the paper, Denis Kleyko and Pentti Kanerva for helpful feedback to the paper, and the anonymous reviewer who took the time to write a review of ~5000 words with detailed comments and actionable advice. This research was supported by the Stanford SystemX Alliance.

A Artifact Appendix

A.1 Abstract

This artifact includes the implementation of the Omen HDC classification inference algorithm with early termination and various baseline algorithms, and scripts to compare them on 19 different benchmarks. The README file of the artifact lists how to reproduce the experiment results in the paper using a few commands. Our results shown in the paper contain runtime data collected in the Microcontroller (MCU), but we provide the option of running the same experiments on a local laptop, in which a similar trend shown in the paper can be observed. This helps more easily verify the claims in the paper. The code for runtime experiments is based on C++ and the other scripts are mostly based on Python. The datasets involved is either included in the repository or can be automatically downloaded by our pipeline scripts. We provide Docker configurations for the environment setup. CUDA is not required but can speed up the model training process. The estimated time to reproduce all the results is $\tilde{1}$ day using laptops without CUDA and a few hours with CUDA.

A.2 Artifact check-list (meta-information)

- **Algorithm:** Omen HDC inference algorithm.
- **Dataset:** Leipzig language dataset [21], UCIHAR [3], ISOLET [4], MNIST [52].
- **Hardware:** Microcontroller and Local Laptop.
- **Metrics:** Classification Accuracy, Average Number of Dimensions Used, Average Inference Runtime.
- **Output:** Figures, Latex/CSV tables.
- **Experiments:** Classification Inference.
- **How much disk space required (approximately)?:** 8GB.
- **How much time is needed to prepare workflow (approximately)?:** 10 minutes.
- **How much time is needed to complete experiments?:** 1 day without CUDA, a few hours with CUDA.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** BSD 2-Clause License.
- **Data licenses (if publicly available)?:** BSD 2-Clause License.
- **Workflow automation framework used?:** Yes.
- **Archived (provide DOI)?:** Yes. https://doi.org/10.5281/zenodo.14511963.

A.3 Description

A.3.1 How to access.

The artifact is available at https://github.com/y553546436/Omen-Artifact and archived at https://doi.org/10.5281/zenodo.14511963.

A.3.2 Hardware dependencies.

A common laptop. GPUs can speed up the experiments, but they are not necessary.

A.3.3 Software dependencies.

Python 3.10, C++ build environment, Makefile tools (all included in the provided Docker configuration).

A.3.4 Datasets.

Leipzig language dataset, UCIHAR, ISOLET, MNIST (included in the repository or auto-downloaded by our pipeline scripts).

A.4 Installation

Our pipeline scripts handle the build and compilations.

A.5 Experiment workflow

See `README.md` in the top directory of the git repository.

A.6 Evaluation and expected results

Following the workflow, the evaluation should exactly reproduce or show similar trends of all tables and figures in the paper's evaluation section (Section 7). Runtime data may have variations due to randomness and different platforms (Microcontrollers versus local), but similar trends should be observed, which supports the claims made in the paper.

References

[1] Diederik Aerts, Marek Czachor, and Bart De Moor. 2009. Geometric analogue of holographic reduced representation. *Journal of Mathematical Psychology* 53, 5 (2009), 389–398.

[2] Henrique CM Andrade, Buğra Gedik, and Deepak S Turaga. 2014. *Fundamentals of stream processing: application design, systems, and analytics*. Cambridge University Press.

[3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3.

[4] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.

[5] Sercan Aygun, Mehran Shoushtari Moghadam, M Hassan Najafi, and Mohsen Imani. 2023. Learning from hypervectors: A survey on hypervector encoding. *arXiv preprint arXiv:2308.00685* (2023).

[6] Toygun Basaklar, Yigit Tuncel, Shruti Yadav Narayana, Suat Gumussoy, and Umit Y Ogras. 2021. Hypervector design for efficient hyperdimensional computing on edge devices. *arXiv preprint arXiv:2103.06709* (2021). https://doi.org/10.48550/arXiv.2103.06709

[7] Julius R Blum, Herman Chernoff, Murray Rosenblatt, and Henry Teicher. 1958. Central limit theorems for interchangeable processes. *Canadian Journal of Mathematics* 10 (1958), 222–229.

[8] Hanning Chen, Yeseong Kim, Elaheh Sadredini, Saransh Gupta, Hugo Latapie, and Mohsen Imani. 2023. Sparsity Controllable Hyperdimensional Computing for Genome Sequence Matching Acceleration. In *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 1–6.

[9] Yu-Chuan Chuang, Cheng-Yang Chang, and An-Yeu Andy Wu. 2020. Dynamic hyperdimensional computing for improving accuracy-energy efficiency trade-offs. In *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 1–5.

[10] Kenneth L. Clarkson, Shashanka Ubaru, and Elizabeth Yang. 2023. Capacity Analysis of Vector Symbolic Architectures. arXiv:2301.10352 [cs.LG]

[11] Angela M Dean and Joseph S Verducci. 1990. Linear transformations that preserve majorization, Schur concavity, and exchangeability. *Linear algebra and its applications* 127 (1990), 121–138.

[12] Don Kurian Dennis, Durmus Alp Emre Acar, Vikram Mandikal, Vinu Sankar Sadasivan, Harsha Vardhan Simhadri, Venkatesh Saligrama, and Prateek Jain. 2019. Shallow RNNs: A method for accurate time-series classification on tiny devices. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 12916–12926.

[13] Persi Diaconis and David Freedman. 1980. Finite exchangeable sequences. *The Annals of Probability* (1980), 745–764.

[14] Shijin Duan, Yejia Liu, Shaolei Ren, and Xiaolin Xu. 2022. LeHDC: Learning-based hyperdimensional computing classifier. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 1111–1116.

[15] Shijin Duan, Xiaolin Xu, and Shaolei Ren. 2022. A brain-inspired low-dimensional computing classifier for inference on tiny devices. *arXiv preprint arXiv:2203.04894* (2022).

[16] Edward Paxon Frady, Denis Kleyko, and Friedrich T Sommer. 2018. A theory of sequence indexing and working memory in recurrent neural networks. *Neural Computation* 30, 6 (2018), 1449–1513. https://doi.org/10.1162/neco_a_01084

[17] Stephen I Gallant and T Wendy Okaywe. 2013. Representing objects, relations, and sequences. *Neural computation* 25, 8 (2013), 2038–2078. https://doi.org/10.1162/NECO_a_00467

[18] Ross W Gayler. 1998. Multiplicative binding, representation operators & analogy (workshop poster). (1998).

[19] Lulu Ge and Keshab K Parhi. 2020. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.

[20] Lulu Ge and Keshab K Parhi. 2022. Applicability of hyperdimensional computing to seizure detection. *IEEE Open Journal of Circuits and Systems* 3 (2022), 59–71.

[21] Dirk Goldhahn, Thomas Eckart, Uwe Quasthoff, et al. 2012. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages.. In *LREC*, Vol. 29. 31–43.

[22] Hui Han and Julien Siebert. 2022. TinyML: A systematic review and synthesis of existing research. In *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 269–274.

[23] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 11 (2021), 7436–7456.

[24] Eman Hassan, Yasmin Halawani, Baker Mohammad, and Hani Saleh. 2021. Hyper-dimensional computing challenges and opportunities for AI applications. *IEEE Access* 10 (2021), 97651–97664.

[25] Mike Heddes, Igor Nunes, Tony Givargis, Alexandru Nicolau, and Alex Veidenbaum. 2022. Hyperdimensional hashing: A robust and efficient dynamic hash table. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 907–912.

[26] Alejandro Hernández-Cano, Namiko Matsumoto, Eric Ping, and Mohsen Imani. 2021. OnlineHD: Robust, Efficient, and Single-Pass Online Learning Using Hyperdimensional System. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 56–61. https://doi.org/10.23919/DATE51398.2021.9474107

[27] Michael Hersche, Mustafa Zeqiri, Luca Benini, Abu Sebastian, and Abbas Rahimi. 2023. A neuro-vector-symbolic architecture for solving Raven's progressive matrices. *Nature Machine Intelligence* 5, 4 (2023), 363–375.

[28] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[29] E Ray Hsieh, Massimo Giordano, Bryce Hodson, Akash Levy, SK Osekowsky, Robert M Radway, Yu-Chuan Shih, Weier Wan, Tony F Wu, Xin Zheng, et al. 2019. High-density multiple bits-per-cell 1T4R RRAM array with gradual SET/RESET and its effectiveness for deep learning. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 35–6. https://doi.org/10.1109/IEDM19573.2019.8993514

[30] Mohsen Imani, Chenyu Huang, Deqian Kong, and Tajana Rosing. 2018. Hierarchical hyperdimensional computing for energy efficient classification. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6. https://doi.org/10.1145/3195970.3196060

[31] Mohsen Imani, Yeseong Kim, Behnam Khaleghi, Justin Morris, Haleh Alimohamadi, Farhad Imani, and Hugo Latapie. 2023. Hierarchical, Distributed and Brain-Inspired Learning for Internet of Things Systems. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. 511–522. https://doi.org/10.1109/ICDCS57875.2023.00083

[32] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan M Rabaey. 2017. Exploring hyperdimensional associative memory. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 445–456. https://doi.org/10.1109/HPCA.2017.28

[33] Mohsen Imani, Sahand Salamat, Behnam Khaleghi, Mohammad Samragh, Farinaz Koushanfar, and Tajana Rosing. 2019. SparseHD: Algorithm-hardware co-optimization for efficient high-dimensional computing. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 190–198.

[34] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[35] Ramesh Johari, Pete Koomen, Leonid Pekelis, and David Walsh. 2017. Peeking at a/b tests: Why it matters, and what to do about it. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1517–1525.

[36] Aditya Joshi, Johan T Halseth, and Pentti Kanerva. 2017. Language geometry using random indexing. In *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers 10*. Springer, 265–274.

[37] Rakhee Kallimani, Krishna Pai, Prasoon Raghuwanshi, Sridhar Iyer, and Onel LA López. 2024. TinyML: Tools, applications, challenges, and

future research directions. *Multimedia Tools and Applications* 83, 10 (2024), 29015–29045.

[38] Pentti Kanerva. 1997. Fully distributed representation. *PAT* 1, 5 (1997), 10000.

[39] 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. https://doi.org/10.1038/s41565-023-01357-8

[40] Michael Klass and Henry Teicher. 1987. The central limit theorem for exchangeable random variables without moments. *The Annals of Probability* (1987), 138–153.

[41] Denis Kleyko, Connor Bybee, Ping-Chen Huang, Christopher J Kymn, Bruno A Olshausen, Edward Paxon Frady, and Friedrich T Sommer. 2023. Efficient decoding of compositional structure in holistic representations. *Neural Computation* 35, 7 (2023), 1159–1186. https://doi.org/10.1162/neco_a_01590

[42] Denis Kleyko, Mike Davies, Edward Paxon Frady, Pentti Kanerva, Spencer J Kent, Bruno A Olshausen, Evgeny Osipov, Jan M Rabaey, Dmitri A Rachkovskij, Abbas Rahimi, et al. 2022. Vector Symbolic Architectures as a Computing Framework for Emerging Hardware. *Proc. IEEE* 110, 10 (2022), 1538–1571. https://doi.org/10.1109/JPROC.2022.3209104

[43] Denis Kleyko, Evgeny Osipov, Alexander Senior, Asad I Khan, and Yaşar Ahmet Şekercioğlu. 2016. Holographic graph neuron: A bioinspired architecture for pattern processing. *IEEE transactions on neural networks and learning systems* 28, 6 (2016), 1250–1262. https://doi.org/10.1109/TNNLS.2016.2535338

[44] Denis Kleyko, Dmitri Rachkovskij, Evgeny Osipov, and Abbas Rahimi. 2023. A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges. *Comput. Surveys* 55, 9 (2023), 1–52. https://doi.org/10.1145/3558000

[45] Denis Kleyko, Dmitri A Rachkovskij, Evgeny Osipov, and Abbas Rahimi. 2021. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations. *ACM Computing Surveys (CSUR)* (2021).

[46] Denis Kleyko, Antonello Rosato, Edward Paxon Frady, Massimo Panella, and Friedrich T. Sommer. 2023. Perceptron Theory Can Predict the Accuracy of Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* (2023), 1–15. https://doi.org/10.1109/TNNLS.2023.3237381

[47] Teuvo Kohonen and Teuvo Kohonen. 2001. Learning vector quantization. *Self-organizing maps* (2001), 245–261.

[48] Arun Kumar Kuchibhotla. 2020. Exchangeability, conformal prediction, and rank tests. *arXiv preprint arXiv:2005.06095* (2020).

[49] Christopher J Kymn, Denis Kleyko, E Paxon Frady, Connor Bybee, Pentti Kanerva, Friedrich T Sommer, and Bruno A Olshausen. 2023. Computing with Residue Numbers in High-Dimensional Representation. *ArXiv* (2023).

[50] Jovin Langenegger, Geethan Karunaratne, Michael Hersche, Luca Benini, Abu Sebastian, and Abbas Rahimi. 2023. In-memory factorization of holographic perceptual representations. *Nature Nanotechnology* 18, 5 (2023), 479–485.

[51] Binh Q Le, Akash Levy, Tony F Wu, Robert M Radway, E Ray Hsieh, Xin Zheng, Mark Nelson, Priyanka Raina, H-S Philip Wong, Simon Wong, et al. 2021. RADAR: A fast and energy-efficient programming technique for multiple bits-per-cell RRAM arrays. *IEEE Transactions on Electron Devices* 68, 9 (2021), 4397–4403. https://doi.org/10.1109/TED.2021.3097975

[52] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[53] Axel Legay, Benoît Delahaye, and Saddek Bensalem. 2010. Statistical model checking: An overview. In *International conference on runtime verification*. Springer, 122–135.

[54] Justin Morris, Mohsen Imani, Samuel Bosch, Anthony Thomas, Helen Shu, and Tajana Rosing. 2019. CompHD: Efficient hyperdimensional computing using model compression. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 1–6.

[55] Justin Morris, Si Thu Kaung Set, Gadi Rosen, Mohsen Imani, Baris Aksanli, and Tajana Rosing. 2021. AdaptBit-HD: Adaptive Model Bitwidth for Hyperdimensional Computing. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 93–100.

[56] Nuntipat Narkthong, Shijin Duan, Shaolei Ren, and Xiaolin Xu. 2024. MicroVSA: An Ultra-Lightweight Vector Symbolic Architecture-based Classifier Library for Always-On Inference on Tiny Microcontrollers. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 730–745.

[57] Tony A Plate. 1994. *Distributed representations and nested compositional structure*. Citeseer.

[58] Tony A Plate. 1995. Holographic reduced representations. *IEEE Transactions on Neural Networks* 6, 3 (1995), 623–641. https://doi.org/10.1109/72.377968

[59] Tony A Plate. 2000. Analogy retrieval and processing with distributed vector representations. *Expert systems* 17, 1 (2000), 29–40.

[60] Tony A Plate. 2003. Holographic Reduced Representation: Distributed representation for cognitive structures. (2003).

[61] Dmitri A Rachkovskij. 2001. Representation and processing of structures with binary sparse distributed codes. *IEEE Transactions on Knowledge and Data Engineering* 13, 2 (2001), 261–276. https://doi.org/10.1109/69.917565

[62] Dmitriy A Rachkovskiy, Sergey V Slipchenko, Ernst M Kussul, and Tatyana N Baidyk. 2005. Sparse binary distributed encoding of scalars. *Journal of Automation and Information Sciences* 37, 6 (2005).

[63] Ali Safa, Ilja Ocket, Francky Catthoor, and Georges Gielen. 2023. SupportHDC: Hyperdimensional Computing with Scalable Hypervector Sparsity. In *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference* (San Antonio, TX, USA) *(NICE '23)*. Association for Computing Machinery, New York, NY, USA, 20–25. https://doi.org/10.1145/3584954.3584961

[64] Kenny Schlegel, Peer Neubert, and Peter Protzel. 2022. HDC-MiniROCKET: Explicit time encoding in time series classification with hyperdimensional computing. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[65] Javier Snaider and Stan Franklin. 2014. Modular composite representation. *Cognitive Computation* 6 (2014), 510–527.

[66] Justin Theiss, Jay Leverett, Daeil Kim, and Aayush Prakash. 2022. Unpaired Image Translation via Vector Symbolic Architectures. In *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer Nature Switzerland, Cham, 17–32.

[67] Simon Thomann, Hong L. G. Nguyen, and Hussam Amrouch. 2022. HW/SW Codesign for Approximate In-Memory Computing. In *2022 23rd International Symposium on Quality Electronic Design (ISQED)*. 1–6. https://doi.org/10.1109/ISQED54688.2022.9806287

[68] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. 2021. Theoretical Foundations of Hyperdimensional Computing. *Journal of Artificial Intelligence Research* 72 (2021), 215–249. https://doi.org/10.48550/arXiv.2010.07426

[69] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*. 65–74.

[70] Abraham Wald. 1992. Sequential tests of statistical hypotheses. In *Breakthroughs in statistics: Foundations and basic theory*. Springer, 256–298.

[71] Anjiang Wei, Akash Levy, Pu (Luke) Yi, Robert Radway, Priyanka Raina, Subhasish Mitra, and Sara Achour. 2023. PBA: Percentile-Based Level Allocation for Multiple-Bits-Per-Cell RRAM. In *ICCAD*.

[72] Samuel Wilson, Tobias Fischer, Niko Sünderhauf, and Feras Dayoub. 2023. Hyperdimensional Feature Fusion for Out-of-Distribution

Detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2644–2654.

[73] Calvin Yeung, Zhuowen Zou, and Mohsen Imani. 2024. Generalized Holographic Reduced Representations. *arXiv preprint arXiv:2405.09689* (2024).

[74] Pu (Luke) Yi and Sara Achour. 2023. Hardware-Aware Static Optimization of Hyperdimensional Computations. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 222 (oct 2023), 30 pages. https://doi.org/10.1145/3622797

[75] Tao Yu, Yichi Zhang, Zhiru Zhang, and Christopher M De Sa. 2022. Understanding hyperdimensional computing for parallel single-pass learning. *Advances in Neural Information Processing Systems* 35 (2022), 1157–1169.

[76] Zhuowen Zou, Hanning Chen, Prathyush Poduval, Yeseong Kim, Mahdi Imani, Elaheh Sadredini, Rosario Cammarota, and Mohsen Imani. 2022. BioHD: an efficient genome sequence search platform using hyperdimensional memorization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 656–669.