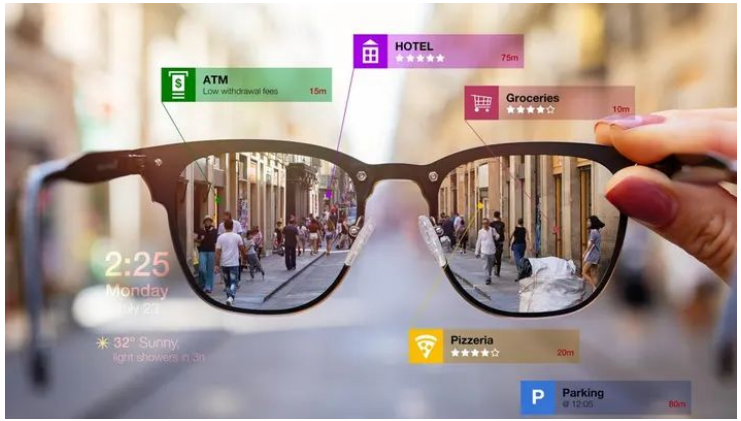
Energy-Efficient ML Using Hyperdimensional Computing

Chae Young Lee

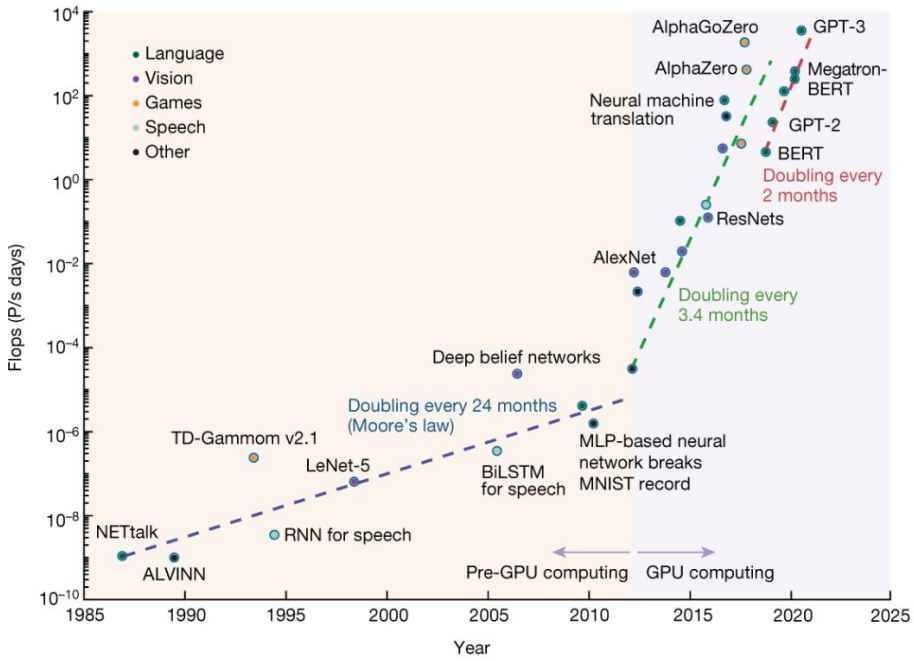
Advisor: Zerina Kapetanovic, Sara Achour

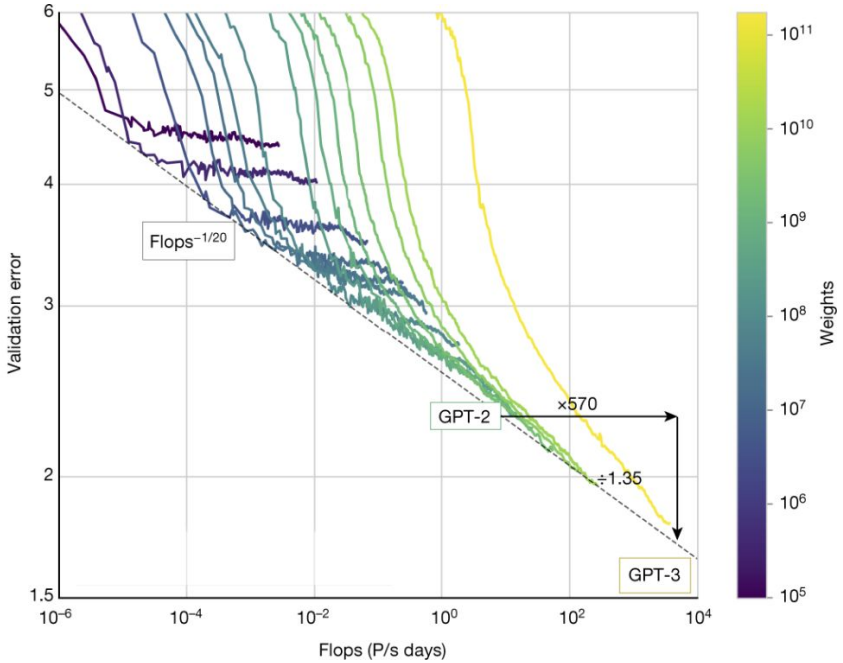
Connecting AI to the real world



(clockwise) Forbes, Anduril, Bay Alarm Medical, AIGen

Performance vs. Energy





Boahen, K. Dendrocentric learning for synthetic intelligence. *Nature* **612**, 43–50 (2022).

Inspiration from human brain

Slide from Prof. Yeseong Kim (DGIST)

Hyperdimensional Computing (HDC)

Motivated by the observation that human brain operates on **high-dimensional** representation of data.

A paradigm of computing in **hyperspace** using **hypervectors**.

Key characteristics:

- Naturally error-resilient.
- Use simple operators, thus hardware-friendly.

Pentti Kanerva

P. Kanerva, “Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors,” Cognitive Computation, vol. 1, no. 2, pp. 139-159, 2009.

Part 1: HDC Background

Part 2: HDC as an image classifier

Can HDC replace DNNs at low-energy setting?

Part 1: HDC Background

Part 2: HDC as an image classifier

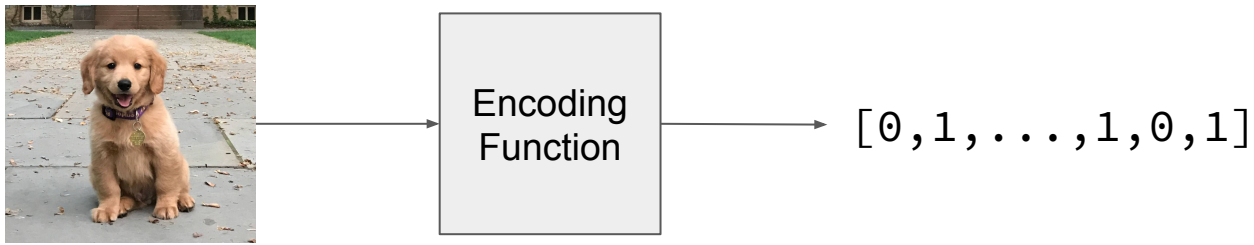
Can HDC replace DNNs at low-energy setting?

Hypervector

In HDC, data is represented as **hypervectors**.

A hypervector can represent a concept, a word in a language, a pixel in an image, or whatever you want to model!

Encoding Φ: $R^d \rightarrow R^n$ or $\{0, 1\}^n$



Codebook

We assign randomly generated vectors to atomic concepts.

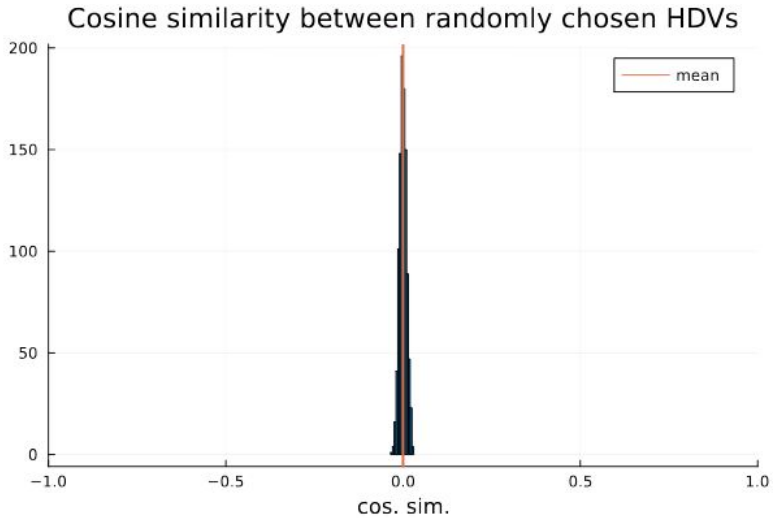
These vectors are called **basis hypervectors**.

```
hv(a) = [0,1,1,0,1,.........,1,0,1]

hv(b) = [1,1,0,0,1,.........,1,1,1]

hv(c) = [1,1,1,1,1,.........,0,0,1]

                  :

hv(z) = [0,0,1,1,0,.........,1,0,0]
```

Randomness

Observation: two randomly generated hypervectors differ in ~5000 bits. Take a third vector at random, it differs from first two in ~5000 bits.

Randomness leads to error-resiliency.



Michiel Stock's Blog Post "Tutorial on Hyperdimensional Computing"

Similarity between hypervectors

Unrelated hypervectors will be far apart. Related ones will be significantly closer.

Using distance/similarity metric, we detect **patterns**.

$$\langle dist(X,Y) \rangle = 1/N \underbrace{\sum XOR(X,Y)}$$

How many bits unmatch?

Operators (1) bundling

- Produces a HV that is **similar** to the input
- BSC: bitwise majority vote

$$hv(r) = hv(a) \oplus hv(b)$$

$$= \{ a, b \}$$

Operators (2) binding

- Produces a HV that is **dissimilar** to the input
- BSC: bitwise XOR

$$hv(r) = hv(a) \odot hv(b)$$

$$= \langle a, b \rangle$$

Operators (3) permutation

- Produces a HV that is **dissimilar** to the input, can encode **ordering**
- BSC: bit shifting

$$hv(r) = p_1(hv(a))$$

$$hv(a) = p_{-1}(hv(r))$$

Encoding

$$\text{hv}(\text{“cat”})$$

$$= \text{hv}(\text{“c”}) \odot p_1(\text{hv}(\text{“a”})) \odot p_2(\text{hv}(\text{“t”}))$$

$$\text{hv}(\text{“orange cat”})$$

$$= \text{hv}(\text{“orange”}) \oplus p_1(\text{hv}(\text{“cat”}))$$

Example: Classification

Let $x \in \mathbb{R}^d$ be an input data and $\phi : \mathbb{R}^d \to \mathbb{R}^n$ be an encoding function.

For each class c ∈ {1,2,...C}, a class hypervector $\mathbf{h}_c \in \mathbb{R}^n$ is computed by

$$\mathbf{h}_c = \bigoplus_{\mathbf{x} \in \text{class } c} \phi(\mathbf{x}).$$

Then, for a query data $\mathbf{q} \in \mathbb{R}^d$, **label prediction** is to find a class with the highest similarity:

$$\text{label} = \arg\max_c (\text{sim}\,(\phi(\mathbf{q}), \mathbf{h}_c))$$

HDC in Code

```
hv = np.random.randint(2, size=SIZE)

def bundle(x1, x2):
     return (np.add(x1, x2) >= 1) * 1

def bind(x1, x2):
     return np.bitwise_xor(x1, x2)

def permute(x, i):
     return np.roll(x, i)

def distance(x1, x2):
     return np.divide(np.sum(np.bitwise_xor(x1, x2)), SIZE)
```

More resources on HDC

A curated list of HDC papers:

https://github.com/chaeyoung-lee/hdvsa-papers

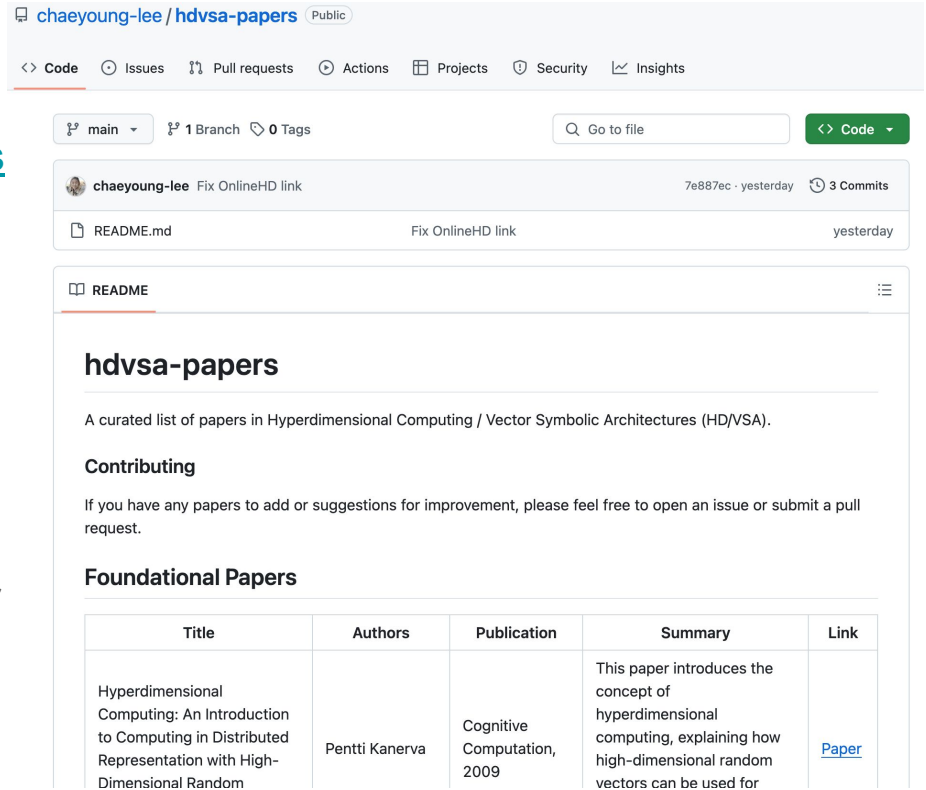
Community website:

https://www.hd-computing.com/

Webinar:

https://sites.google.com/view/hdvsaonline

Sara's class CS 349H: Software Techniques for Emerging Hardware Platforms



Part 1: HDC Background

Part 2: HDC as an image classifier

Can HDC replace DNNs at low-energy setting?

Image Encoding

Concepts: image → pixels → row, column, pixel intensity

Given a grayscale image I of size w x h,

We need basis hypervectors $R : [1, ..., h] \to \mathbb{R}^n, C : [1, ..., w] \to \mathbb{R}^n, V : [1, ..., 256] \to \mathbb{R}^n$

Image: $\phi(\mathbf{I}) = \oplus \mathbf{h}_{\mathbf{I}_{i,j}}$

Pixel: $\mathbf{h}_{\mathbf{I}_{i,j}} = R(i) \odot C(j) \odot V(I(i,j))$

Image Encoding

Given basis hypervectors $R : [h] \to \mathbb{R}^n, C : [w] \to \mathbb{R}^n, V : [256] \to \mathbb{R}^n$

$$\mathbf{h_I} = \sum_{i=1}^{h} \sum_{j=1}^{w} R(i) \odot C(j) \odot V(I(i,j))$$

basis hypervectors:

operations:

52% less memory
50% less binding
98% less bundling

Rewrite 1: Row and Column Index Coalescing

Consider a 2D image flattened into 1D array.

$$\mathbf{h_I} = \sum_{i=1}^{h} \sum_{j=1}^{w} R(i) \odot C(j) \odot V(I(i,j))$$
$$= X(i \cdot w + j)$$

Instead of *h*+*w*, we use *hw* basis hypervectors.

$$X : [1, ..., hw] \rightarrow \mathbb{R}^n$$

Rewrite 2: Permutation-based Codebooks

Instead of storing *hw* basis hypervectors, can we generate them on-the-fly?

$$dist(X(0), p^1[X(0)]) \approx 0.5$$

$$\begin{aligned}\mathbf{h_I} &= \sum_{i=1}^{h}\sum_{j=1}^{w} X(iw+j) \odot V(I(i,j)) \\ &= \sum_{i=1}^{h}\sum_{j=1}^{w} p^{iw+j}[X(0)] \odot V(I(i,j))\end{aligned}$$

Rewrite 3: Value Hypervector Factoring

$$\mathbf{h_I} = \sum_{i=1}^{h} \sum_{j=1}^{w} p^{iw+j}[X(0)] \odot V(I(i,j))$$

bundling operations is still *wh*, this needs to be significantly reduced.

Observation: many pixels share the same intensity value.

Can we group pixels by their intensity *z* and factor out *V(z)*?

$$\mathbf{h_I} = \sum_{z=1}^{256} V(z) \odot \left[\sum_{(i,j) \in Pix(z)} p^{iw+j}[X(0)] \right]$$

Pix(z) represents the set of all pixel positions where the image is z.

Rewrite 3: Value Hypervector Factoring

$$\mathbf{h_I} = \sum_{z=1}^{256} V(z) \odot \left[\sum_{(i,j) \in Pix(z)} p^{iw+j}[X(0)] \right]$$

Bundling operator is not associative, so some information on **the relative weight of different pixel values** are lost in quantization.

$$\mathbf{h_I} = \sum_{z=1}^{256} |Pix(z)| V(z) \odot \left[\sum_{(i,j) \in Pix(z)} p^{iw+j}[X(0)] \right]$$

Rewrite 4: Sparse Bundling

Instead of computing a full sum, can we **approximate it using a small subset**?

$$\mathbf{h_I} = \sum_{z=1}^{256} |Pix(z)| V(z) \odot \left[\sum_{(i,j) \in Pix(z)} p^{iw+j} [X(0)] \right]$$

$$SparseBundle(Pix(z)) \approx \sum_{(i,j) \in Pix(z)} p^{iw+j} [X(0)]$$

Algorithm 2 Image Encoding with Sparse Bundling

```
function ENCODEIMAGE(Img: image)
    int[256][n] hvs;
    uint[256] cnts;
    uint8[n] imgHV;
    for v in 0..256 do
        sumHVs[v] = NewSparseHV()
        cnts[v] = 0
    for k in 0..w · h do
        v = Img[k]
        SparseBundleElem(sumHVs[v], k)
        cnts[v] += 1
    for v in 0..255 do
        for i in 0 ··· n − 1 do
            imgHV[i] += cnts[v] · (sumHVs[v][i] xor get-
ValueCB(v,i))
    for i in 0 ··· n − 1 do
        imgHV[i] = 1 ? imgHV[i] > |wh|/2 : 0;
    return imgHV;
```

Rewrite 4: Sparse Bundling

Bloom Filter / **Count-Sketch** approximate superposition of sets.

$$SparseBundle(S) \approx CountSketch\left(\sum_{s \in S} p^s[\mathbf{h}]\right)$$

$$\approx BloomFilter\left(\bigcup_{s \in S} p^s[\mathbf{h}]\right)$$

Algorithm 1 Sparse Bundling Algorithm

```
1:  bool bloom = false; //use a bloom filter or count-sketch
2:  uint n = 10000; // hypervector size
3:  uint d = 20; // density - the number of hashes per bundle
4:  // random set of size d from values {0, 1, ··· , n − 1}
5:  uint8[d] indices ← rand(0,n,size=d,replace=False);
6:  // random vector with values {−1, 1}
7:  int8[d] CS ← rand([-1,1],size=d);
8:  function SparseBundleElem(hv,s)
9:      for j in 0...d − 1 do
10:         k = (indices[j]+s) % n
11:         if bloom then
12:             hv[k] = 1
13:         else
14:             hv[k] = hv[k] + CS[j]
```

Computational Savings

	Naive	Rewrite 1	Rewrite 2	Rewrite 3	HyperCam
Codebook	536	258	258	258	258
Bind	38400	38400	19200	19200	19200
Bundle	19200	19200	19200	19456	256
SparseBundle	0	0	0	0	19200

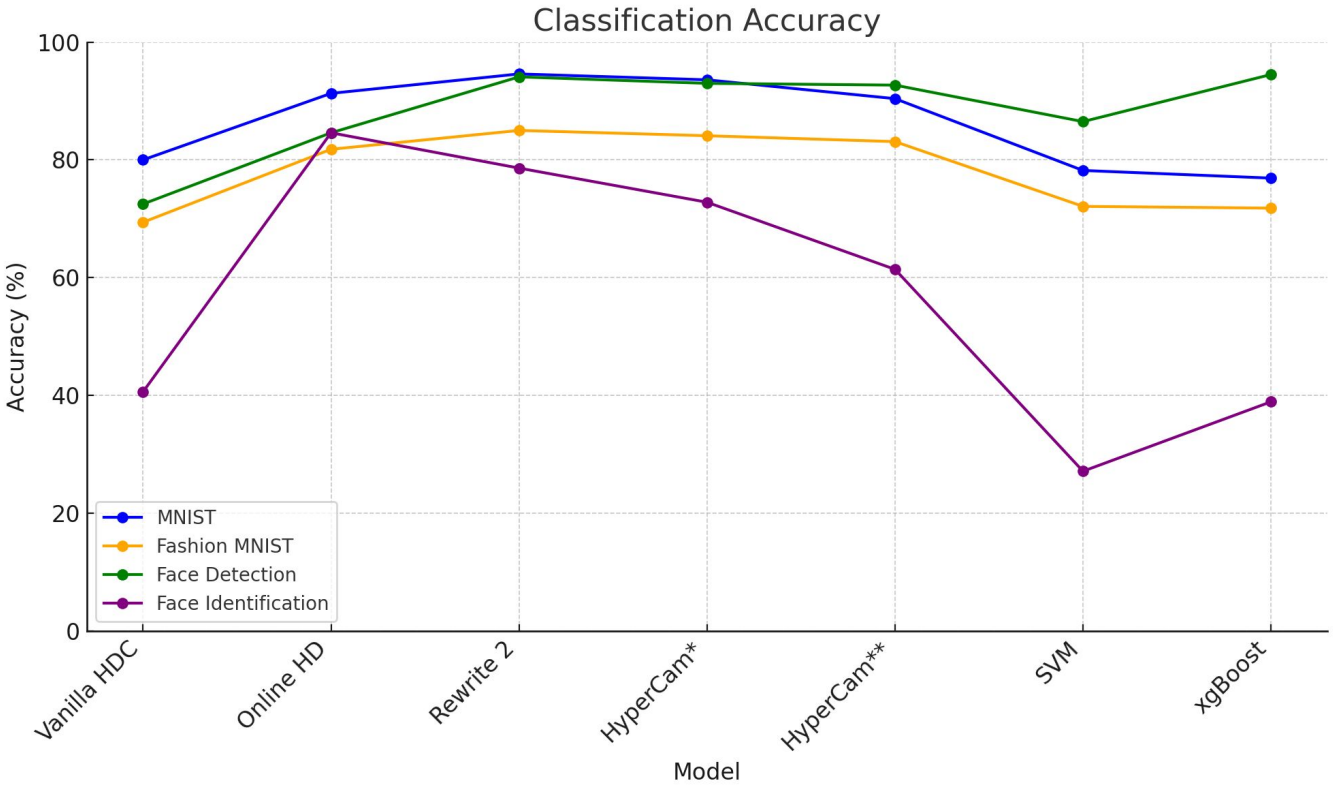
Table 1: Comparison of encoding methods based on the size of the codebook and the number of bind, bundle, and sparse bundle operations. Each bundling operation involves 10000 bit-wise addition, whereas each sparse bundling operation involves 20.

Bundling: O(n)
Sparse bundling: O(d)

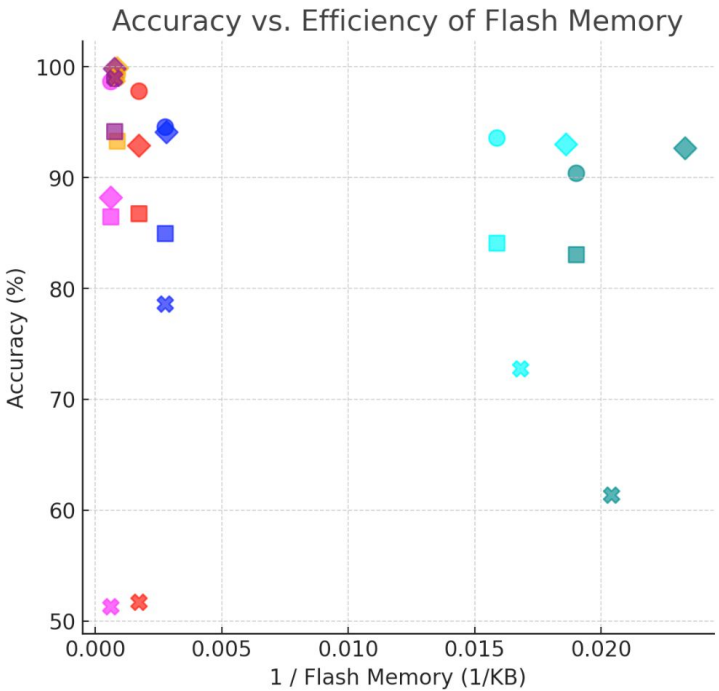
C. Lee et al., “HyperCam: Low-Power Onboard Computer Vision for IoT Cameras,” Will Appear on MobiCom, 2025

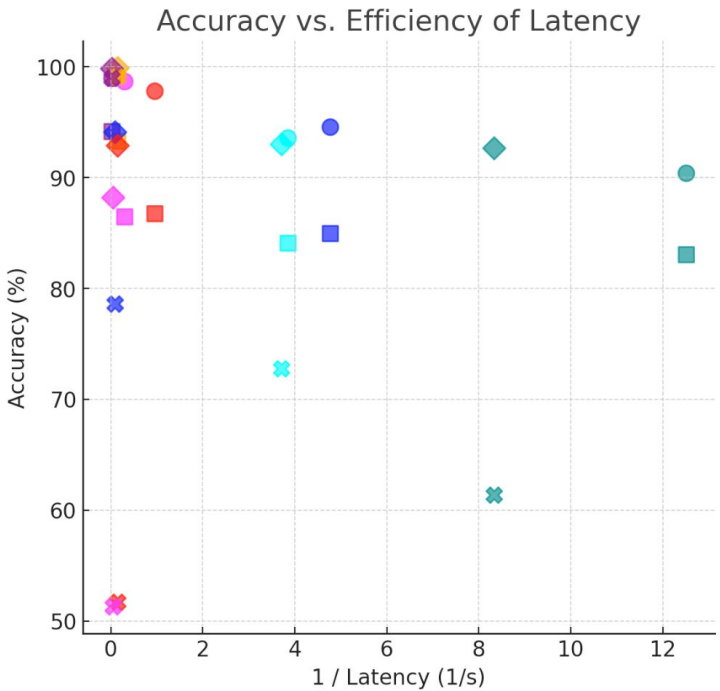
Training Algorithm

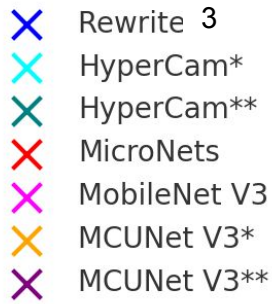
Evaluation against HDC and traditional ML models



Evaluation against tiny ML models







HDC that learns

HD classifiers are inefficient

Current HDC training algorithms heavily **rely on heuristics / trial-and-error** to search for class hypervectors.

Unlike in DNNs, **no principled approaches** to optimally train an HDC model and rigorously learn the class hypervectors that provide the best possible accuracy performance for HDC.

Model saturates easily and common pattern dominates each model's IM.

Label prediction is essentially a Binary Neural Network

Bipolar HDC

Encoding: $En(x)$: $R^N \rightarrow \{-1, 1\}^D$

Label prediction:

$$k^\star = \underset{k}{argmin}\ Hamm(En(x), c_k)$$
$$= \underset{k}{argmax}\ cosine(En(x), c_k)$$
$$= \underset{k}{argmax}\ En(x)^T c_k$$

Forward propagation in BNN?

What if basis vectors are learnable?

enc(x) = cos(x*b^T+c) * sin(x*b^T)

f(x) = softmax(w * enc(x))

Use CE loss and Adam optimizer to learn w, b, c

Binary model:

HD as a Q-value estimator

$Q(s\_t) = w * enc(x)$

$Q(s\_t,a\_t) \leftarrow Q(s\_t,a\_t) + \alpha(r\_t+\gamma a' maxQ(s\_t+1,a') - Q(s\_t,a\_t))$