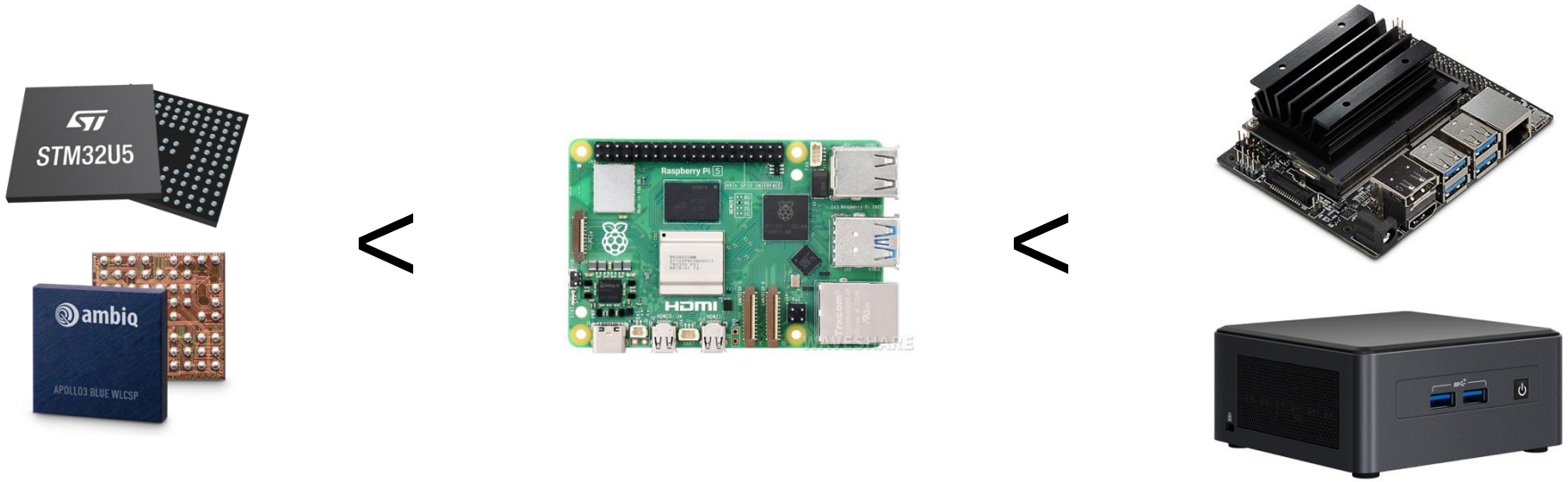
Ultra-Lightweight Edge Intelligence Using Vector Symbolic Architecture

Chae Young Lee
Department of Computer Science, Stanford University
chae@stanford.edu

AI on microcontrollers is extremely challenging



A typical MCU is equipped with:

- ARM Cortex M-series core
- Flat memory structure with flash <1MB and RAM <500KB
- Clock <200MHz
- Power consumption <5mW

“Ultra-lightweight” edge intelligence

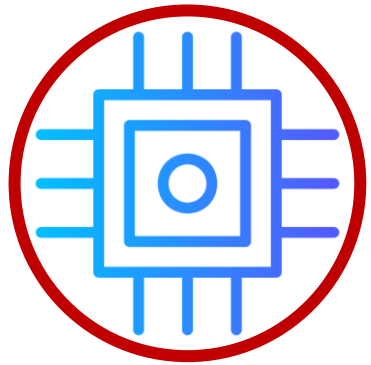
Providing intelligence with **ultra-light**

Power Consumption

<10mW

Memory Footprint

<100KB flash & RAM

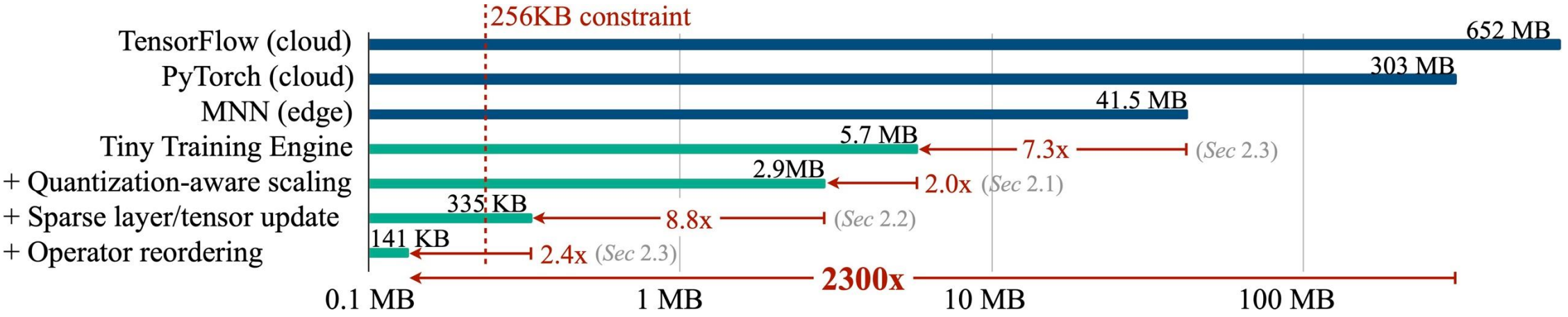


Compute Load

<100k CPU cycle

“Ultra-lightweight” edge intelligence

Deep neural networks (DNNs)



Lin et al. "On-device training under 256kb memory." NeuRIPS, 2022.

Vector Symbolic Architecture (VSA)

Information is represented as vectors (often, high-dimensional).

- 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.

Table of Content

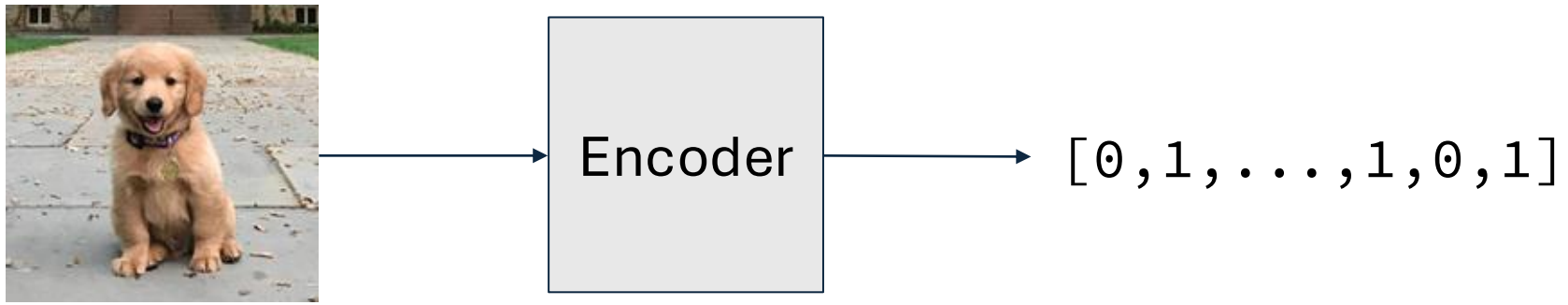
VSA Background

Vectors

In VSA, data is represented as fixed-length **vectors**.

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

Models: Binary Spatter Code (BSC), Multiply-Add-Permute (MAP)



Codebook

We store a codebook of vectors to represent atomic concepts.

These vectors are called **basis** vectors.

$$\mathbf{h}_a = [0,1,1,0,1,.....,1,0,1]$$

$$\mathbf{h}_b = [1,1,0,0,1,.....,1,1,1]$$

$$\mathbf{h}_c = [1,1,1,1,1,.....,0,0,1]$$

$$\vdots$$

$$\mathbf{h}_z = [0,0,1,1,0,.....,1,0,0]$$

VSA operators

Binding ($\mathbf{V} \otimes \mathbf{V}'$) and permutation ($p^k[\mathbf{V}]$) produce vectors that are **dissimilar** to the input.

Bundling ($\mathbf{V}_1 \oplus \ldots \oplus \mathbf{V}_m$) produces vectors that are **similar** to the input.

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 $V[i] \sim \text{Bern}(p=0.5)$	Hamming Distance $\text{HD}(V,V')$ $\text{HD}(V,V') = \frac{1}{N}\sum(V[i] \textbf{ xor } V'[i])$	XOR $V[i] \textbf{ xor } V'[i]$	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 $V[i] \sim \text{U}(-1,1)$	Cosine/Dot Product Similarity $\text{dot}(V,V') = \sum V[i]V'[i]$	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.

Yi et al., "Early Termination for Hyperdimensional Computing Using Inferential Statistics," ASPLOS, 2025.

Distance between vectors

Unrelated vectors 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?

Example: text encoding

Let's encode "catch" using three different encodings.

$$\mathbf{h}_{\text{catch}}^{\text{bind}} = \mathbf{h}_c \otimes p^1(\mathbf{h}_a) \otimes p^2(\mathbf{h}_t) \otimes p^3(\mathbf{h}_c) \otimes p^4(\mathbf{h}_h)$$

$$\mathbf{h}_{\text{catch}}^{\text{bundle}} = \mathbf{h}_c \oplus p^1(\mathbf{h}_a) \oplus p^2(\mathbf{h}_t) \oplus p^3(\mathbf{h}_c) \oplus p^4(\mathbf{h}_h)$$

$$\mathbf{h}_{\text{catch}}^{\text{trigram}} = \mathbf{h}_{\text{cat}} \oplus \mathbf{h}_{\text{atc}} \oplus \mathbf{h}_{\text{tch}}$$

Question: how similar is each encoding to words like "batch," "latch," and "pitch"?

Example: classification

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

For each class $c \in \{1,2,\dots,C\}$, a class vector $\mathbf{h}_c \in \mathbb{R}^n$ 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:

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

VSA in Python

```
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)
```

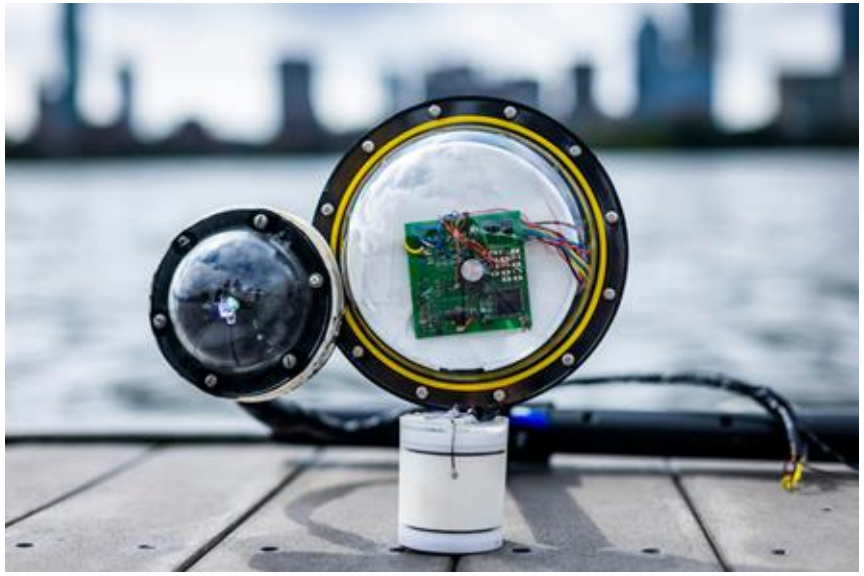
VSA for Image Classification

Background: low-power wireless cameras

BeetleCam[1]

WISPCam[2]

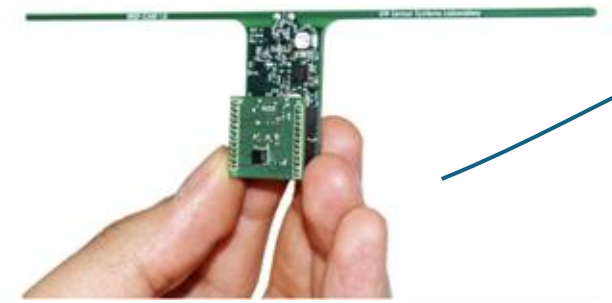
Underwater Wireless Camera[3]



[1] Iyer, Vikram, et al. "Wireless steerable vision for live insects and insect-scale robots." *Science robotics* 5.44 (2020): eabb0839.
[2] Naderiparizi, Saman, et al. "WISPCam: A battery-free RFID camera." *2015 IEEE International Conference on RFID (RFID)*. IEEE, 2015.
[3] Afzal, Sayed Saad, et al. "Battery-free wireless imaging of underwater environments." *Nature communications* 13.1 (2022): 5546.

Background: low-power wireless cameras

Battery-free
Wireless Camera



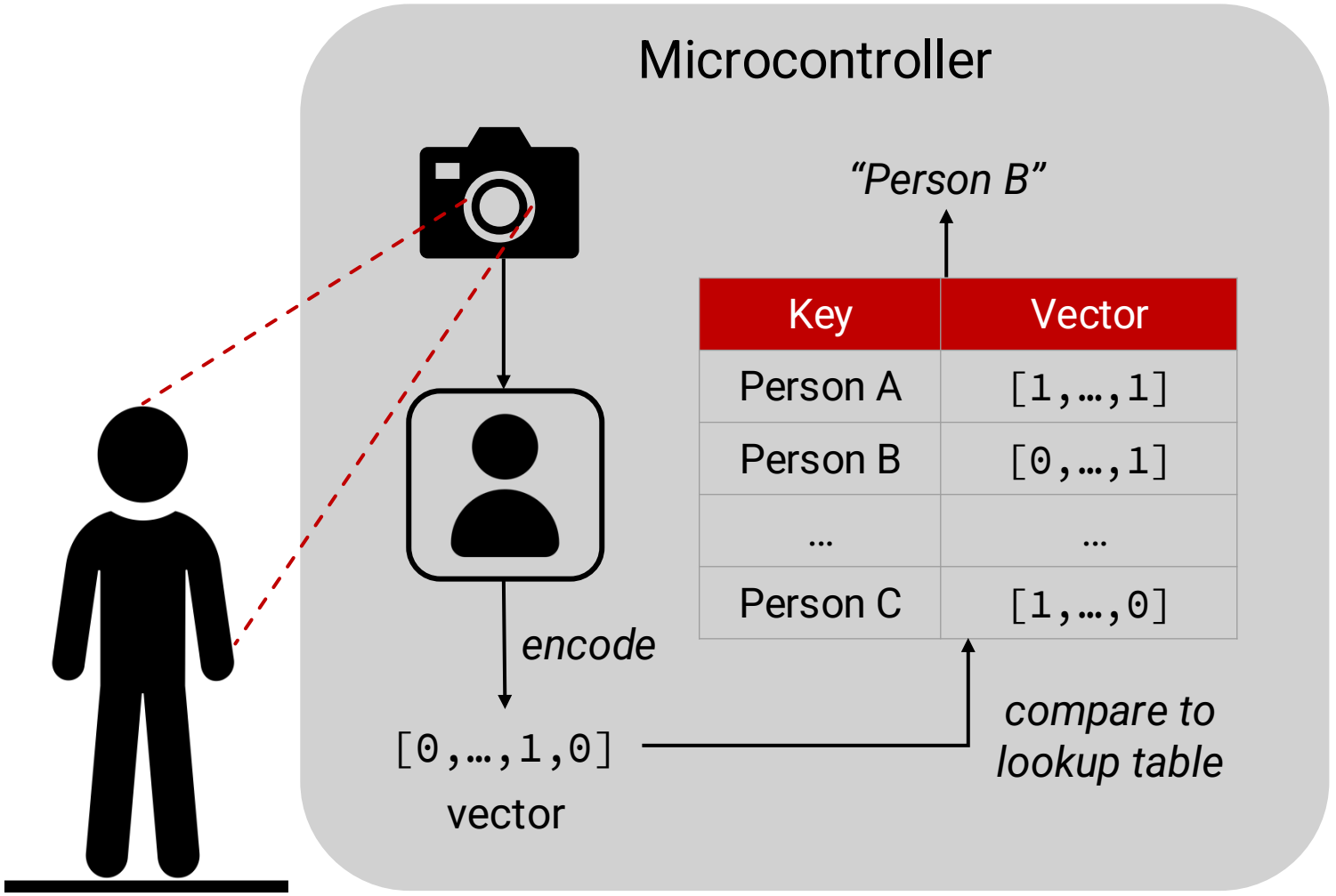
Low Bandwidth
Connection

- Very low power
- Can be battery-free

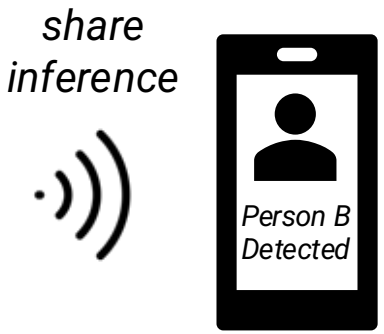
Image Transfer
Delays

Missing Pixels

Introducing HyperCam (MobiCom ‘25)



Compared to DNN, HyperCam achieves 55.8x latency and 27.7x memory usage improvements.



Introducing HyperCam

Key technical contribution:

	Codebook		Bind		Bundle		Sparse Bundle	
	Size	**Memory (KB)**	**# Ops**	**Time (ms)**	**# Ops**	**Time (ms)**	**# Ops**	**Time (ms)**
Naive	536	654	38400	2400	19200	2400	0	0
Rewrite 1	258	315	38400	2400	19200	2400	0	0
Rewrite 2	258	315	19200	1200	19200	2400	0	0
Rewrite 3	258	315	256	16	19456	2448	0	0
HyperCam	258	315	256	16	256	48	19200	2

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. HyperCam's codebook size is further reducing during implementation as described in Section 5.1.

Naïve VSA encoder is not resource-efficient for image classification.
HyperCam uses coding theory to optimize encoding operation.

VSA Image Encoder

At a glance: image → pixels → row, column, pixel intensity

Randomly initialize a codebook of basis vectors:

Row	$R: [1, \dots, h] \rightarrow \{0,1\}^n$
Column	$C: [1, \dots, w] \rightarrow \{0,1\}^n$
Pixel intensity	$V: [1, \dots, 256] \rightarrow \{0,1\}^n$

VSA Image Encoder

Given a grayscale image $I \in \{1, \ldots, 256\}^{h \times w}$ and codebooks R, C, V,

Per-pixel vector is:

$$\pi(i,j) = R(i) \odot C(j) \odot V\left(I_{i,j}\right)$$

Image encoder $\phi: \{1, \ldots, 256\}^{h \times w} \to \{0,1\}^n$ is:

$$\phi(I) = \sum_{i=1}^{h} \sum_{j=1}^{w} \pi(i,j)$$

VSA Image Encoder

That is, with $R: [h] \to \{0,1\}^n, C: [w] \to \{0,1\}^n, V: [256] \to \{0,1\}^n$,

$$\phi(I) = \sum_{i=1}^{h} \sum_{j=1}^{w} R(i) \odot C(j) \odot V\left(X_{i,j}\right)$$

Number of vectors in the codebook: $w + h + 256$

Number of encoding operations: $O(whn)$

	Naïve VSA Encoder	HyperCam Encoder
Stored Vectors	536	6
Binding Ops	38400	256
Bundling Ops	19200	256 + 38*

HyperCam's innovation is in optimizing the image encoding

HyperCam optimizes image encoding

Naïve image encoder

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

memory saving

Rewrite 1: permutation-based codebooks

$$\phi(I) = \sum_{i=1}^{h} \sum_{j=1}^{w} \rho^{i}[R(0)] \odot \rho^{j}[C(0)] \odot V(X_{i,j})$$

time, memory saving

Rewrite 2: row and column index coalescing

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

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

HyperCam optimizes image encoding

Observation: many pixels share the same intensity value. Can we group pixels by their intensity z and factor out $V(z)$? Recall Rewrite 2:

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

Rewrite 3: value hypervector factoring

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

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

HyperCam optimizes image encoding

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

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

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

HyperCam optimizes image encoding

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;
```

HyperCam optimizes image encoding

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]
```

HyperCam optimizes image encoding

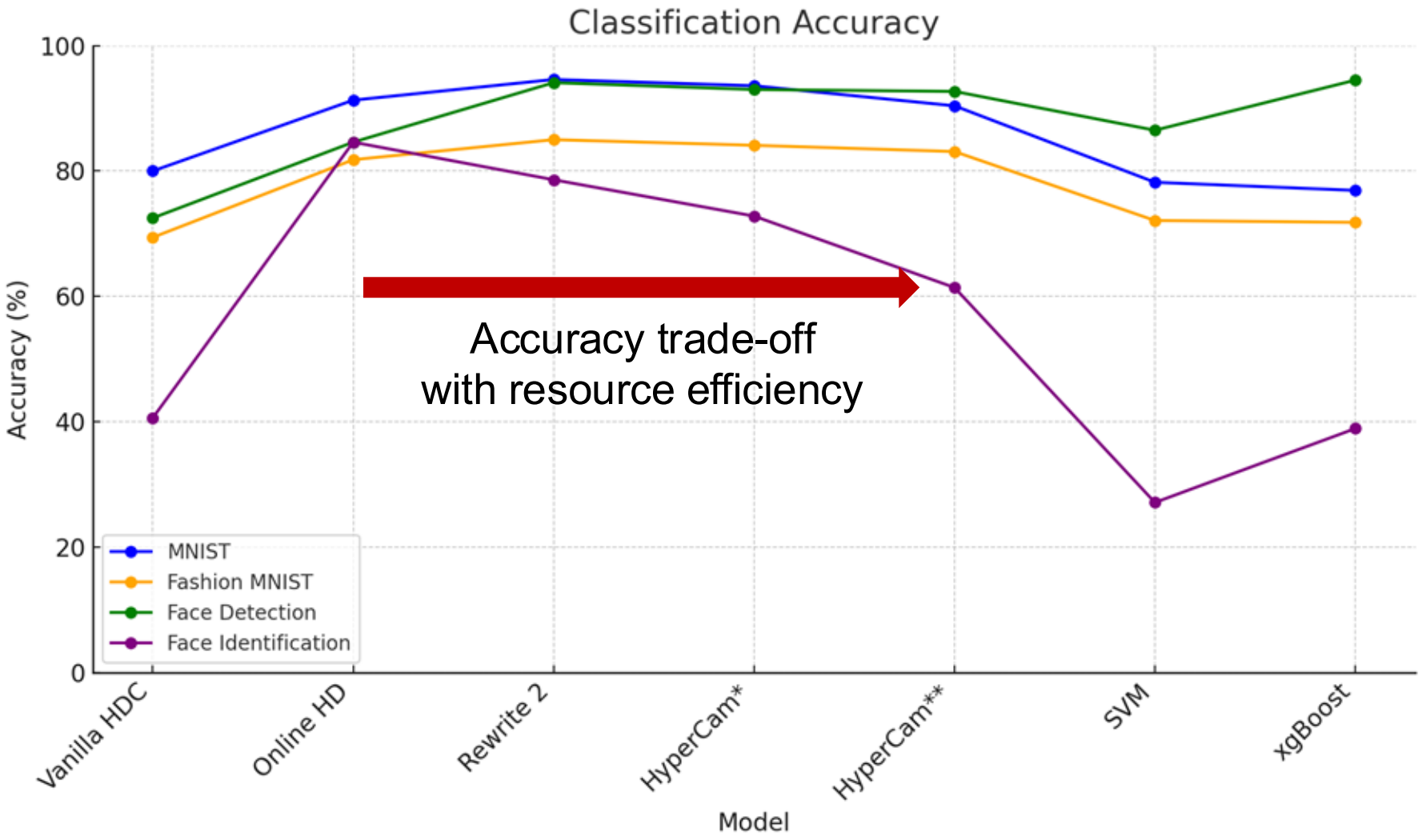
	Codebook		Bind		Bundle		Sparse Bundle	
	Size	Memory (KB)	# Ops	Time (ms)	# Ops	Time (ms)	# Ops	Time (ms)
Naive	536	654	38400	2400	19200	2400	0	0
Rewrite 1	258	315	38400	2400	19200	2400	0	0
Rewrite 2	258	315	19200	1200	19200	2400	0	0
Rewrite 3	258	315	256	16	19456	2448	0	0
HyperCam	258	315	256	16	256	48	19200	2

~38 normal bundle

Bundling: O(n)

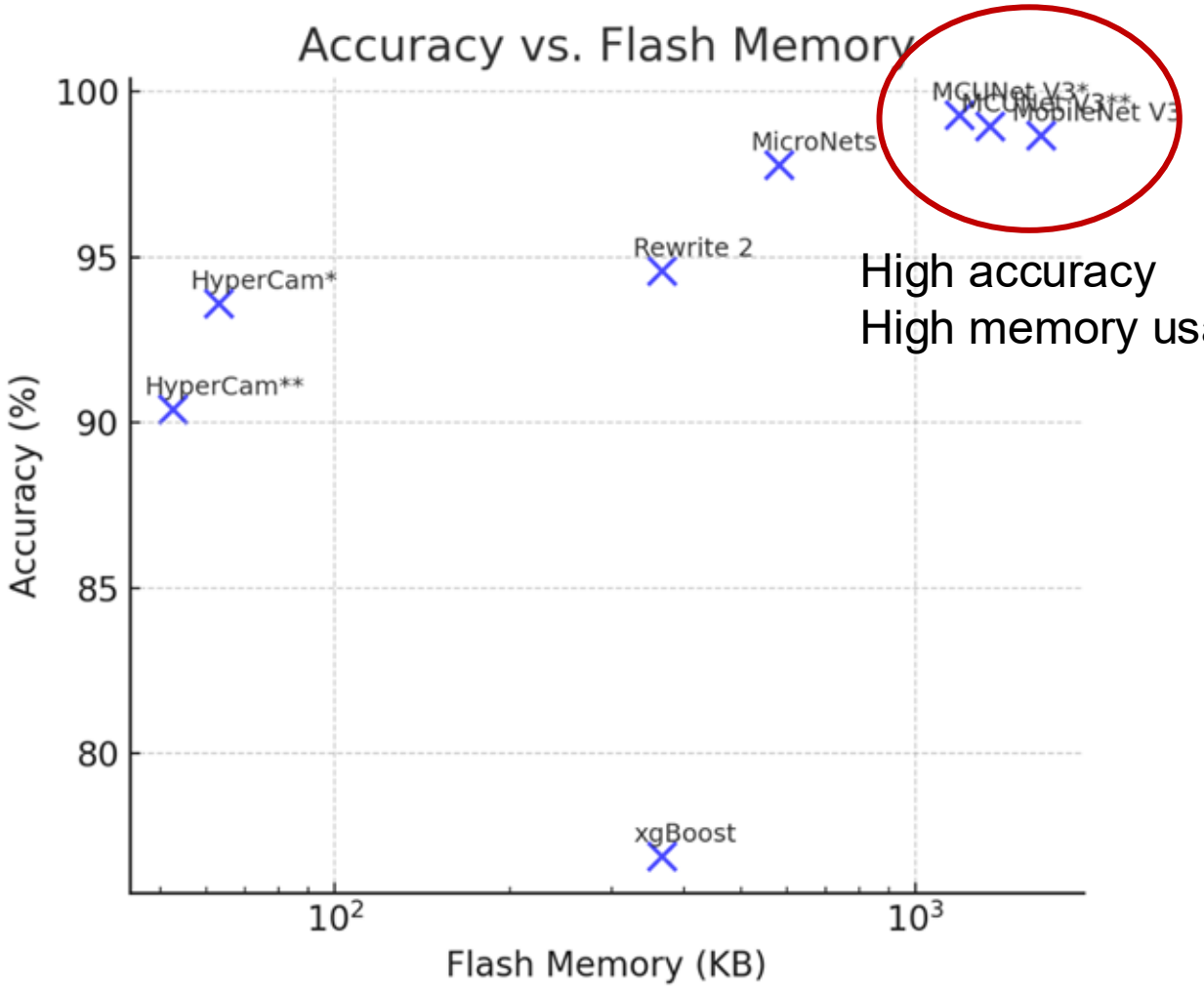
Sparse bundling: O(d)

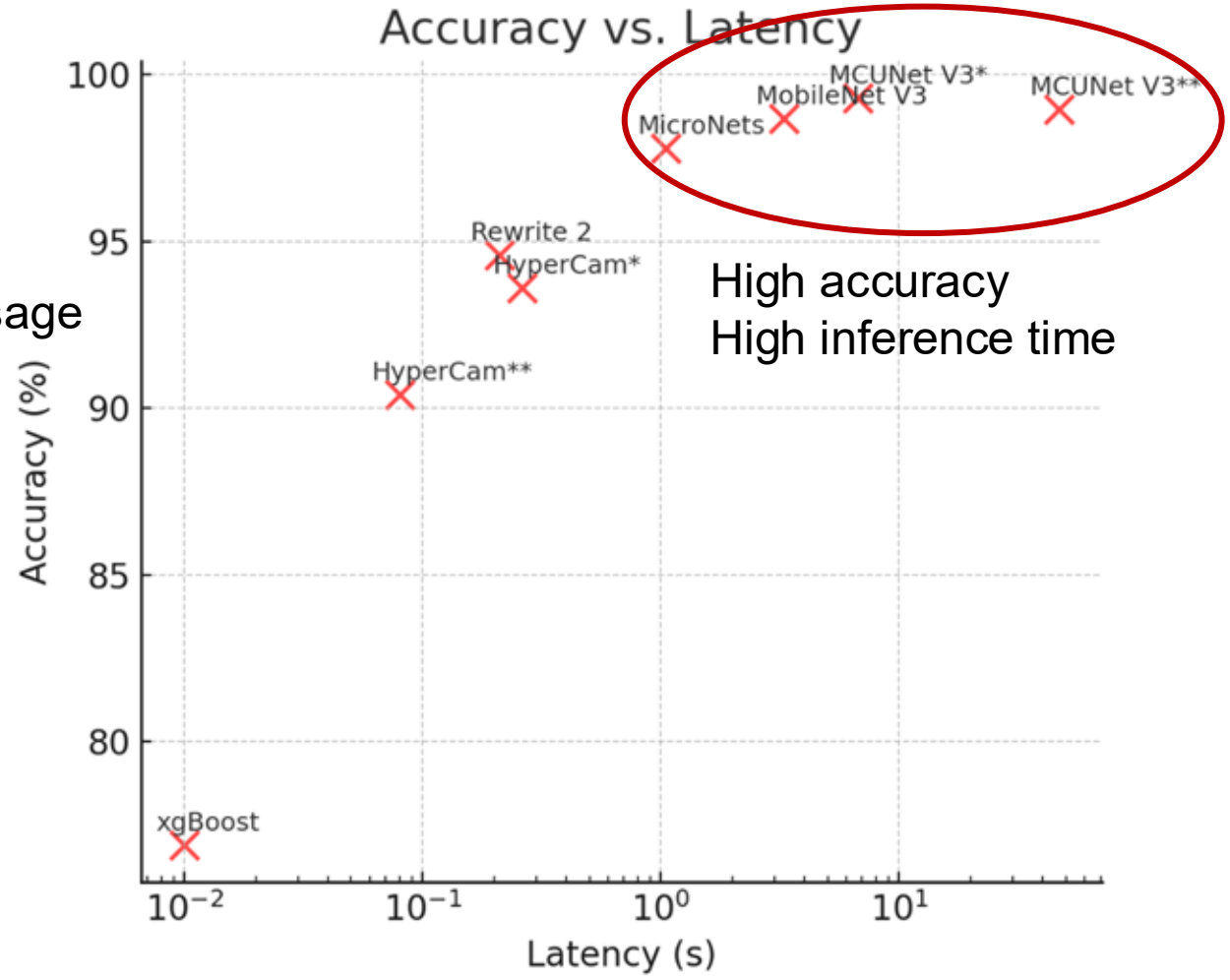
Evaluation against HDC and traditional ML models



Evaluation against neural networks

MNIST dataset





Evaluation against neural networks

MNIST dataset

	Accuracy (%)	Latency (s)	Flash (kB)
HyperCam (Rewrite 2)	94.60	0.21	365.02
HyperCam (count-sketch)	93.60	0.26	63.00
HyperCam (bloom filter)	90.34	**0.08**	**52.62**
MicroNets	97.82	1.05	582.16
MCUNet V3	**98.97**	6.70	1190.00

Conclusion

HyperCam's accuracy drop from Rewrite 2 is only **1-2%**, while reducing **6x** memory and **3-100x** processing time per frame.

Sparse bundling approximates 10,000 operations to 20.

DNNs have highest accuracy but ours is **55-398x faster** and **12-33x more lightweight** than DNNs.

VSA for Off-Policy Reinforcement Learning

Background: micro-robotics

RoBeetle[1]

KiloBot[2]

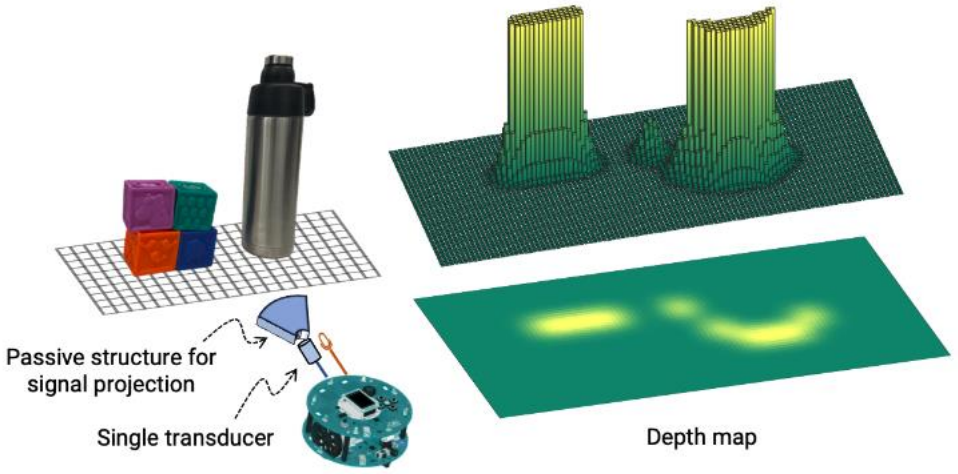
TerraSentia[3]

[1] Yang, Xiufeng, et al. "An 88-milligram insect-scale autonomous crawling robot driven by a catalytic artificial muscle." *Science Robotics, 2020*.
[2] Rubenstein, Michael, et al. "Kilobot: A low cost scalable robot system for collective behaviors." *2012 IEEE international conference on robotics and automation*. IEEE, 2012.
[3] Cuaran, Jose, et al. "Under-canopy dataset for advancing simultaneous localization and mapping in agricultural robotics." *The International Journal of Robotics Research, 2024.*

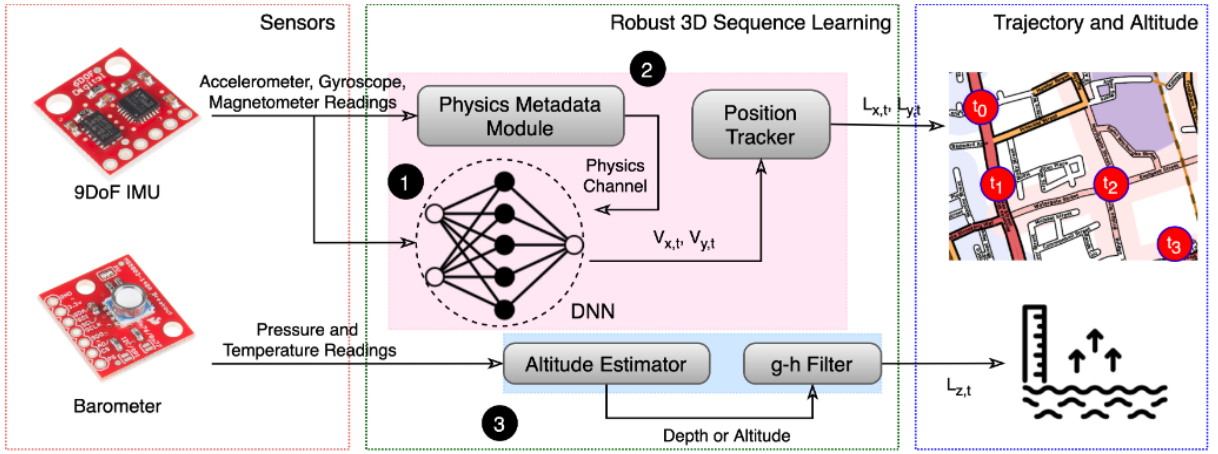
Background: micro-robotics

Existing navigation techniques

- Rely on cheap, low-power sensors
- Run lightweight processing pipeline onboard



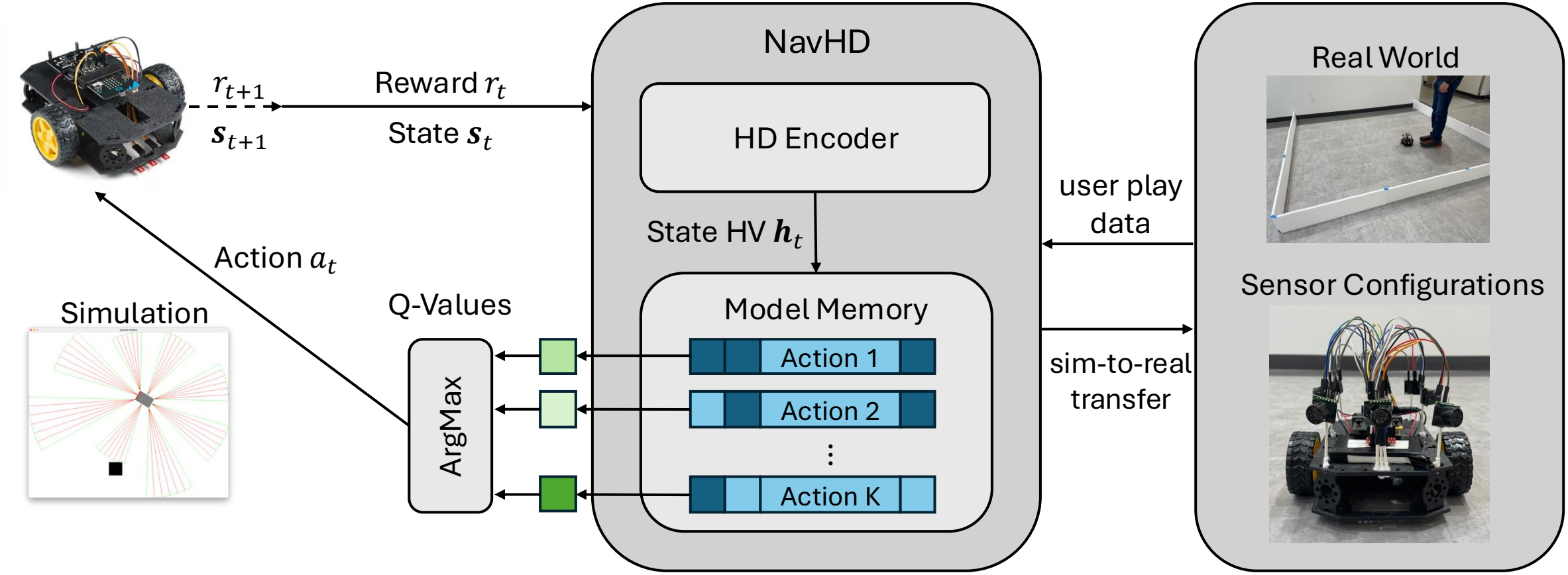
SpiDR (Bai et al., MobiSys ‘22)



TinyOdom (Saha et al., IMWUT ‘22)

Introducing NavHD

VSA-based Q-Learning framework

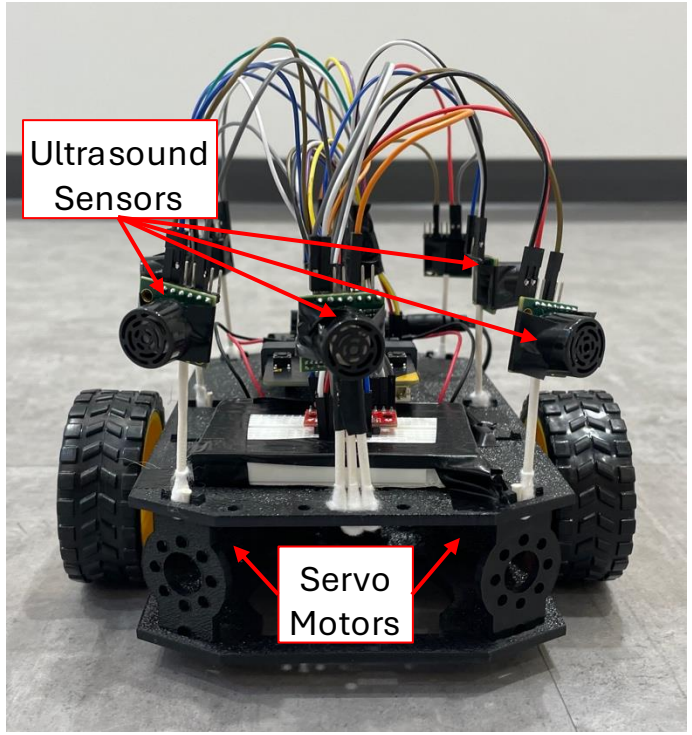


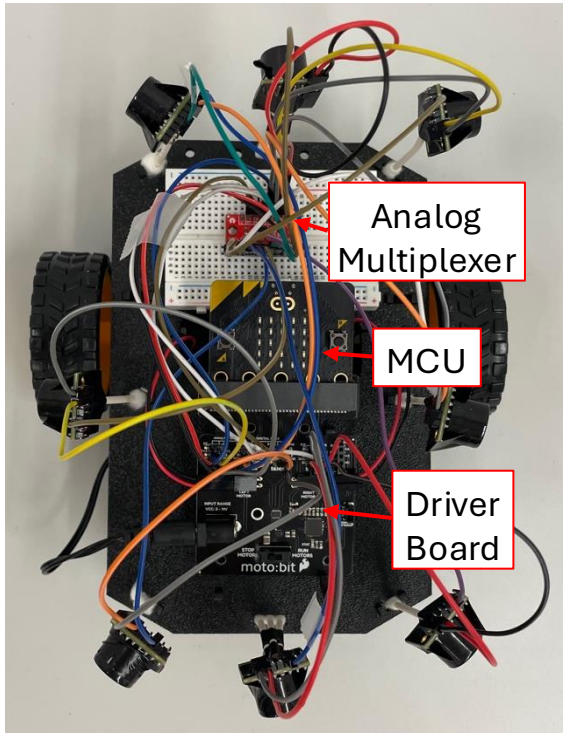
Zero-shot inference in the real world

Introducing NavHD

Core technical contribution:

The first fully-differentiable VSA encoder and Q-value estimator.





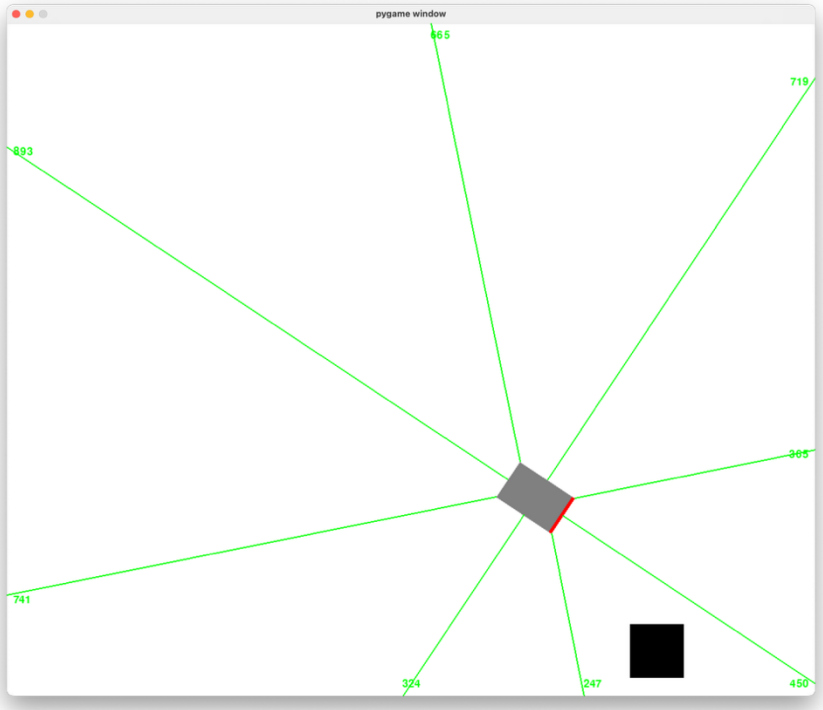
Component	Idle (mW)	Full Throttle (mW)
Micro:bit v2	39.0	39.0
Carrier Board	43.9	48.4
Motors	0.0	2307.0
Ultrasound Sensors	40.0	40.0
Multiplexer	0.2	0.2
Total	**123.1**	**2434.6**

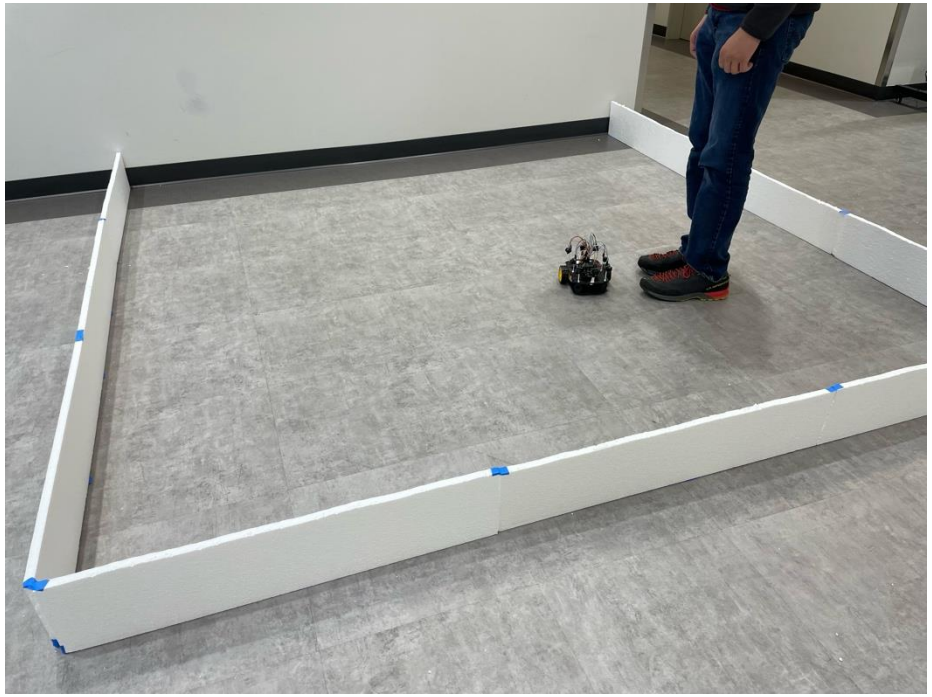
TABLE II: System power consumption.

Object Avoidance Task

Input: proximity sensor data

Output: action (4-class forward differential drive)





Encoder

Given encoder $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^n$,

input sensors $\mathbf{s}_t \in \mathbb{R}^d$ in d directions is encoded into $\mathbf{h}_t = \phi(\mathbf{s}_t)$.

Using a learned scaling matrix $\mathbf{B} \in \mathbb{R}^{n \times d}$ with $B_{ij} \sim N(0, \sigma^2)$ and a learned offset vector $\mathbf{b} \in \mathbb{R}^n$ with $b_i \sim \text{Uniform}(0, 2\pi)$, $\mathbf{h}_t$ is

$$\phi(\mathbf{s}_t) = \cos(\mathbf{B} \cdot \mathbf{s}_t + \mathbf{b}) \odot \sin(\mathbf{B} \cdot \mathbf{s}_t)$$

The resulting $\mathbf{h}_t$ is bounded within [-1, 1].

Action Predction

Given model memory $\mathbf{C} = \langle \mathbf{c}_1, \mathbf{c_2}, \ldots, \mathbf{c}_k \rangle$ consisting of k action vectors,

$$\text{sim}(\mathbf{h}_t, \mathbf{c}_k) = \mathbf{h}_t^T \cdot \mathbf{c}_k$$

Next action is determined by

$$\hat{y}_t = \text{argmax}_k \text{sim}(\mathbf{h}_t, \mathbf{c}_k)$$

NavHD Characteristics

- **Differentiable:** Including encoder and similarity function. Can be trained using backpropagation.
- **Exchangeable**: All parameters are i.i.d. random variables at initialization. Loss function is based on hypervector distances, which are symmetric. Gradient-based training preserves symmetry.

NavHD Q-Learning

Estimate Q-values using $Q(\mathbf{s}_t, a) = \mathrm{sim}(\phi(\mathbf{s}_t), \mathbf{c}_a)$. Using Bellman equation, the target Q-value is:

$$y_t = r_t + \gamma(1 - \mathrm{done}_t) \max_{a'} Q'(\mathbf{s}_{t+1}, a')$$

Then loss is computed as

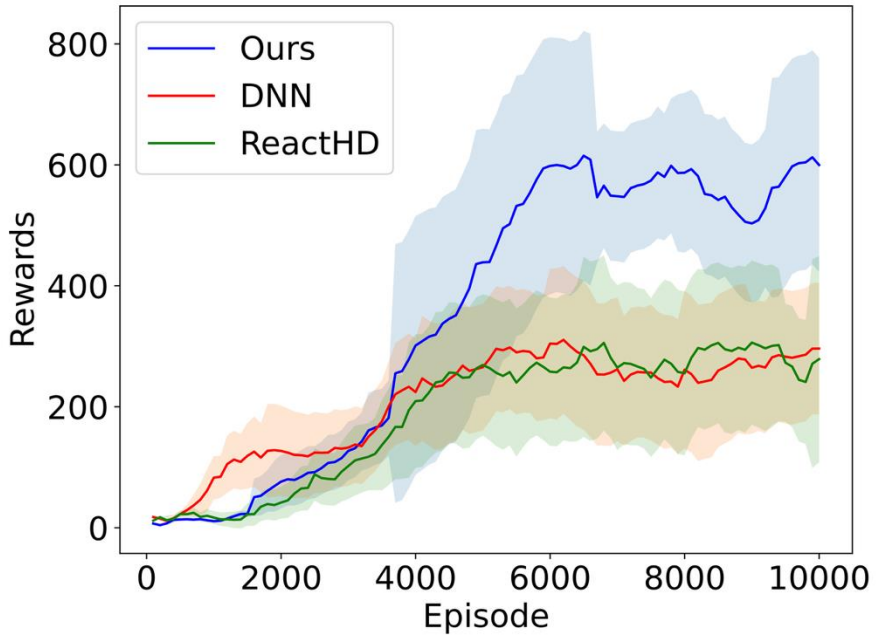
$$l_t = \mathrm{HuberLoss}(Q(\mathbf{s}_t, a_t; \mathbf{C}, \mathbf{B}, \mathbf{b}), y_t)$$

NavHD uses Double DQN and experience replay to enhance training stability.

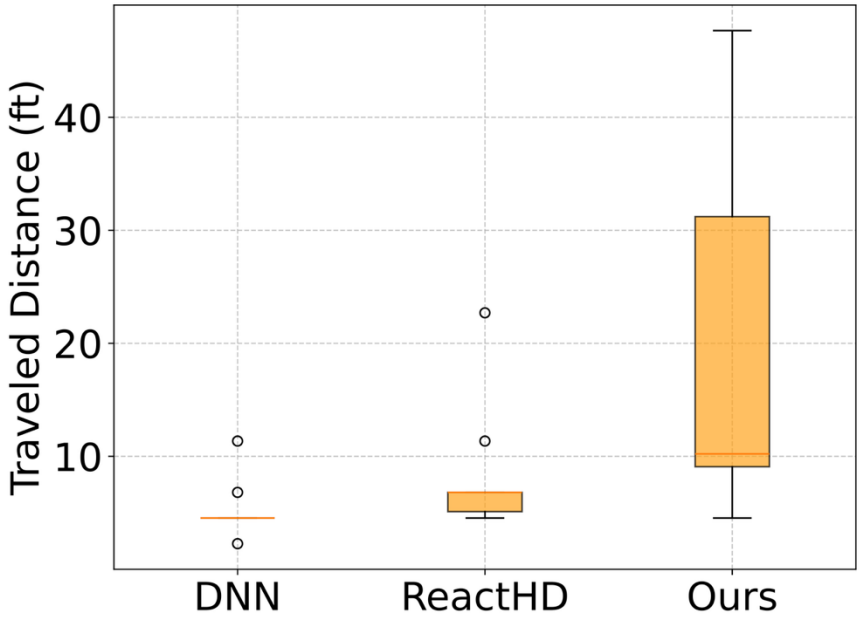
Learning Quality

DNN: four-layer with hidden size 256, 128, 64.

ReactHD: existing HD-based RL work. n=5,000.



(a) Hardware-optimized models in simulation



(b) Real world experiments

NavHD
Obstacle Avoidance

Resource Usage

TABLE IV: Resource usage on the target hardware.

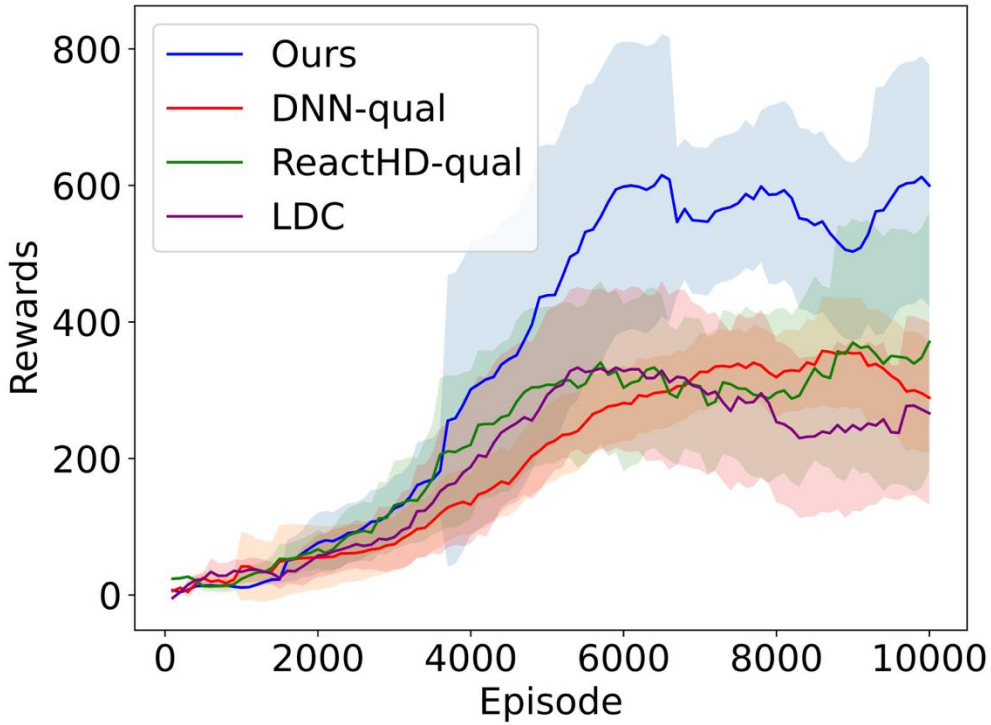
Model	Memory (kB)	Clock Cycle (k)	Energy (mJ)
DNN	263.8	2.9	3.4
ReactHD	19.5	8.4	9.9
NavHD	**10.2**	**0.9**	**1.1**

900 clock cyles in our MCU (64 MHz) takes ~14 us.

CPU-Only Experiments

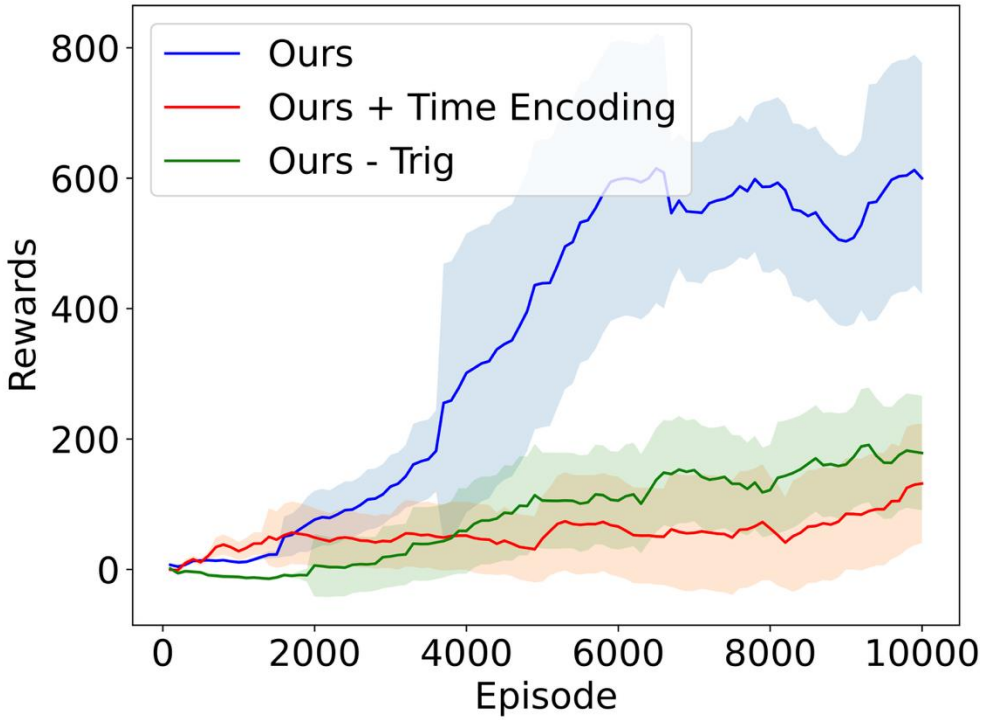
Models that don't fit in embedded hardware.

- **DNN-qual:** additional hidden layer of size 512.
- **ReactHD-qual:** uses hypervector length 10,000.
- **LDC:** state-of-the-art HD classifier + NavHD Q-learning



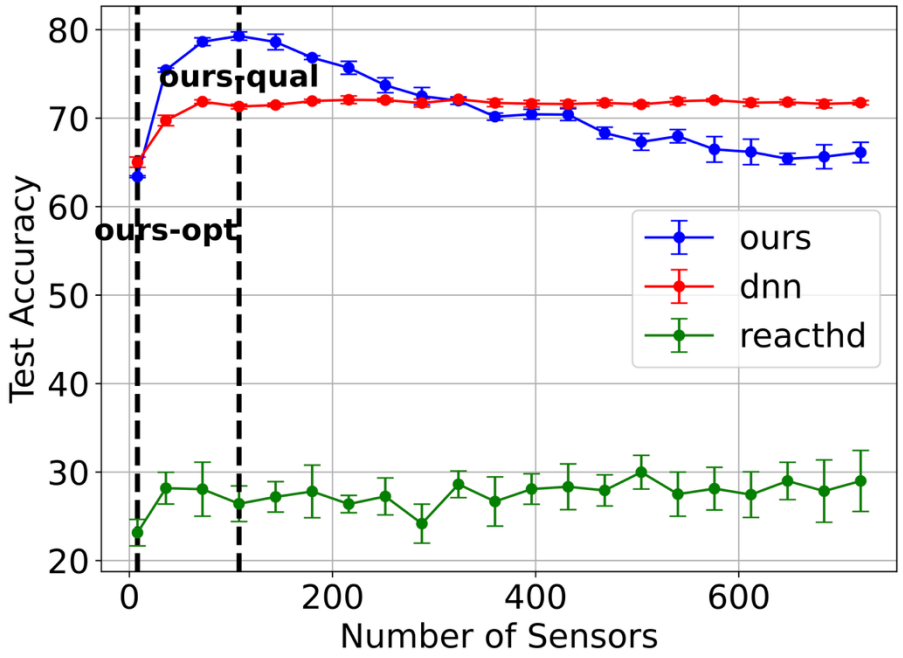
Encoder Ablation Study

- Explicit time encoding: $\phi(\mathbf{s_t}) = \mathbf{h}_t \oplus \rho^1(\mathbf{h}_{t-1}) \oplus \cdots \oplus \rho^L(\mathbf{h}_{t-L})$.
- Without activation function: $\phi(\mathbf{s_t}) = \mathbf{B} \cdot \mathbf{s}_t$

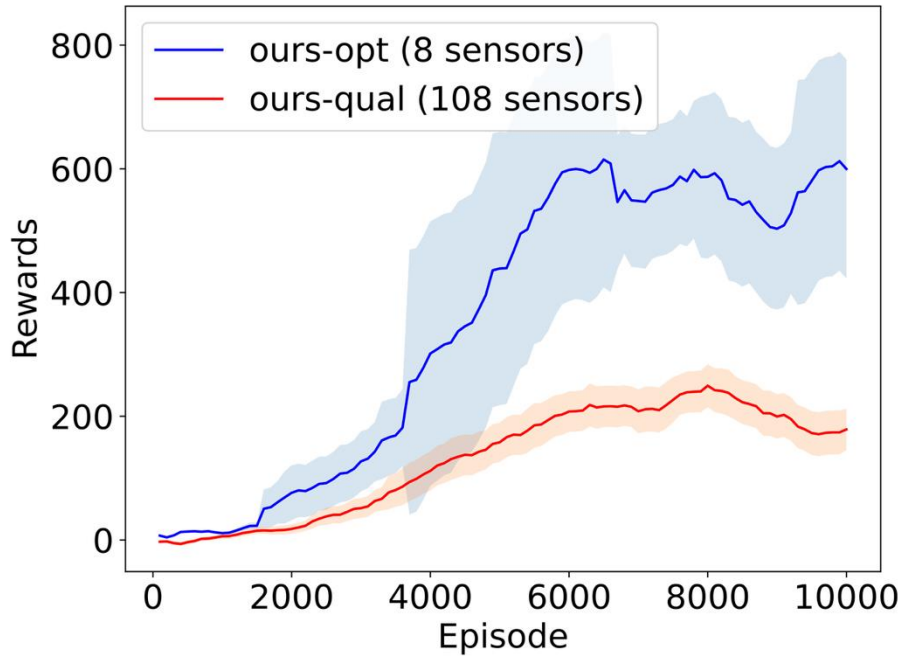


Sensor Count

Ours-qual (108 sensors), ours-opt (8 sensors).



(a) Imitation Learning



(b) Q-Learning

Conclusion & Takeaway

- NavHD outperforms both DNN and prior HD-based RL models in both accuracy and resource efficiency.
- NavHD uses ~10KB and can be deployed on extremely resource-limited hardware. If needed, model can be trained onboard.
- We found a sweet spot at 8 proximity sensors that gives us comparable accuracy to LiDAR while using ~100x less power.

Conclusion

Ongoing & future work

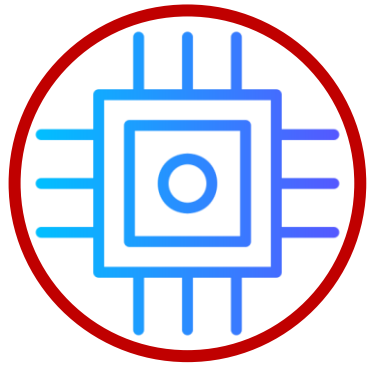
Countless possibilities for ML model that is

Low-power

<10mW

Small

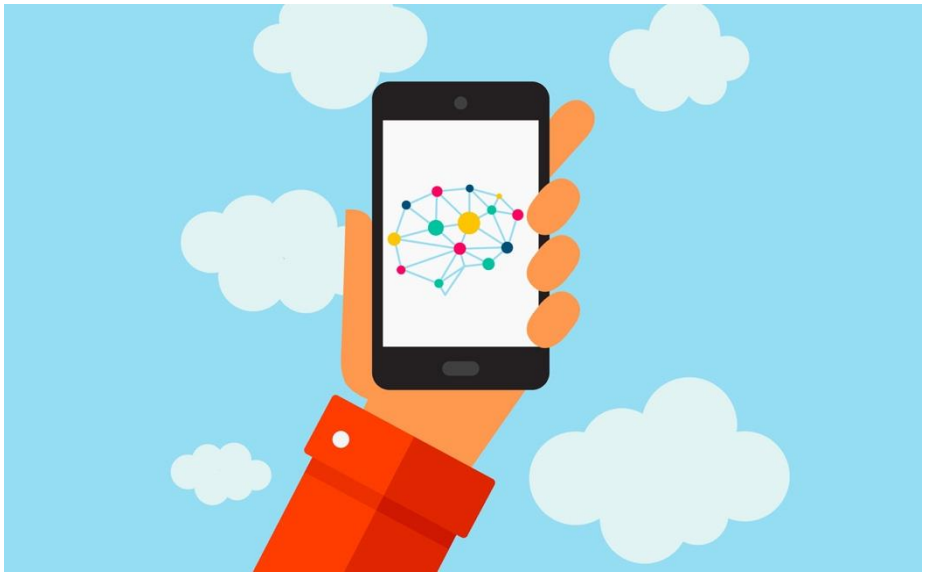
<100KB flash & RAM



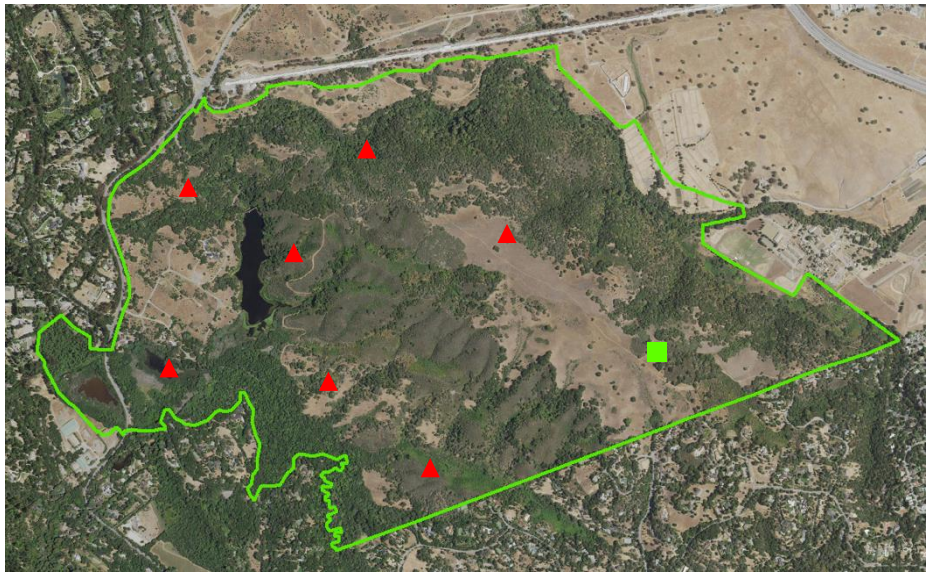
Fast

<100k CPU cycle

Ongoing & future work



Zero-Overhead
On-Device Training
Using <100KB



Large-Scale
Outdoor Sensor Network

Read my work her

Thank you for listening!

Stanford
SMART SENSING
SYSTEMS LAB

Chae Young Lee
Stanford University
chae@stanford.edu