Lifelong Intelligence Beyond the Edge using Hyperdimensional Computing

Xiaofan Yu
x1yu@ucsd.edu
University of California San Diego
La Jolla, California, USA

Anthony Thomas
ahthomas@ucsd.edu
University of California San Diego
La Jolla, California, USA

Ivannia Gomez Moreno
ivannia.gomez@cetys.edu.mx
CETYS University, Campus Tijuana
Tijuana, Mexico

Louis Gutierrez
l8gutierrez@ucsd.edu
University of California San Diego
La Jolla, California, USA

Tajana Šimunić Rosing
tajana@ucsd.edu
University of California San Diego
La Jolla, USA

ABSTRACT

On-device learning has emerged as a prevailing trend that avoids the slow response time and costly communication of cloud-based learning. The ability to learn continuously and indefinitely in a changing environment, and with resource constraints, is critical for real sensor deployments. However, existing designs are inadequate for practical scenarios with (i) streaming data input, (ii) lack of supervision and (iii) limited on-board resources. In this paper, we design and deploy the first on-device lifelong learning system called LifeHD for general IoT applications with limited supervision. LifeHD is designed based on a novel neurally-inspired and lightweight learning paradigm called Hyperdimensional Computing (HDC). We utilize a two-tier associative memory organization to intelligently store and manage high-dimensional, low-precision vectors, which represent the historical patterns as cluster centroids. We additionally propose two variants of LifeHD to cope with scarce labeled inputs and power constraints. We implement LifeHD on off-the-shelf edge platforms and perform extensive evaluations across three scenarios. Our measurements show that LifeHD improves the unsupervised clustering accuracy by up to 74.8% compared to the state-of-the-art NN-based unsupervised lifelong learning baselines with as much as 34.3x better energy efficiency. Our code is available at https://github.com/Orienfish/LifeHD.

KEYWORDS

Edge Computing, Lifelong Learning, Hyperdimensional Computing

1 INTRODUCTION

The fusion of artificial intelligence and Internet of Things (IoT) has become a prominent trend with numerous real-world applications, such as in smart cities [10], smart voice assistants [55], and smart activity recognition [62]. However, the predominant current approach is cloud-centric, where sensor devices send data to the cloud for offline training using extensive data sources. This approach faces challenges like slow updates and costly communication, involving the exchange of large sensor data and models between the edge and the cloud [53]. Instead, recent research has shifted towards edge learning, where machine learning is performed on resource-constrained edge devices right next to the sensors. While most studies focused on inference-only tasks [32, 33, 50], some recent work has investigated the optimization of computational and memory resources for on-device training [15, 34]. Nevertheless, these efforts often rely on static models for inference or lack the adaptability to accommodate new environments.

To fundamentally address these issues, sensor devices should be capable of "lifelong learning" [42]: to *learn and adapt with limited supervision* after deployment. On-device lifelong learning reduces the need for expensive data collection (including labels) and offline model training, operating in a deploy-and-run manner. This approach enables autonomous learning solely from the incoming samples with minimal supervision, and is thus able to provide real-time decision-making even without a network connection. The lifelong aspect is essential for handling dynamic real-world environments, representing the future of IoT.

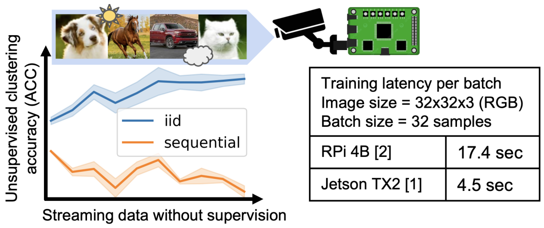
Although extensive research has investigated lifelong learning across various scenarios [42], existing techniques face challenges that render them unsuitable for real-world deployments. These challenges include:

(C1) **Streaming data input.** Edge devices collect streaming data from a dynamic environment. This *online* learning with *non-iid* data contrasts with the default *offline* and *iid* setting where multiple passes on the entire dataset are allowed [16].

(C2) **Lack of supervision.** Obtaining ground-truth labels and expert guidance is often challenging and expensive. Most lifelong learning methods rely on some form of supervision, such as class labels [28] or class shift boundaries [46], which are typically unavailable in real-world scenarios.

(C3) **Limited device resources.** Neural networks (NN) are known for their high resource demands [60]. Furthermore, the main techniques for lifelong learning based on NN, such as regularization [28] and memory replay [35], add extra computational and memory requirements beyond standard NNs, making them inadequate for edge devices.

Real-World Example. To illustrate the challenges faced, we present a real-world scenario in Fig.1. Consider a camera deployed in the wild continuously collecting data from surrounding environment. Our goal is to train an unsupervised object recognition algorithm on the edge device, purely from the data stream. We construct both iid and sequential (one class appears after the other) streams from CIFAR-100 [6], and adopt the smallest MobileNet V3 model [20] with the popular BYOL unsupervised learning pipeline [16]. As seen in Fig. 1, while the model shows improved accuracy with iid streams, it has a significant performance loss under sequentially



Training latency per batch Image size = 32x32x3 (RGB) Batch size = 32 samples	
RPi 4B [2]	17.4 sec
Jetson TX2 [1]	4.5 sec

Figure 1: Real-world example of on-device lifelong learning evaluated using the unsupervised clustering accuracy metric [63]. The training latency is measured on two typical edge platforms.

ordered data, highlighting the NN effect of "forgetting" in a streaming and unsupervised setting. In terms of efficiency, we measure the training latency of MobileNet V3 (small) [20] on two typical edge platforms, Raspberry Pi (RPi) 4B [2] and Jetson TX2 [1] by running 10 gradient descent steps on a single batch of 32 samples. Even on these very capable edge platforms, training takes up to 17.4 seconds, clearly unsuitable for real-time processing under 30 FPS. Therefore, a novel approach capable of handling non-iid data and offering more efficient updates is necessary to accommodate the continual changes in data.

To address challenges (C1)-(C3), we draw inspiration from biology, where even tiny insects display remarkable lifelong learning abilities, and do so using "hardware" that requires very little energy [4]. Hyperdimensional computing (HDC) is an emerging paradigm inspired by the information processing mechanisms found in biological brains [24]. In HDC, all data is represented using high-dimensional, low-precision (often binary) vectors known as "hypervectors," which can be manipulated through simple elementwise operations to perform tasks like memorization and learning. HDC is well-understood from a theoretical standpoint [56] and shares intriguing connections with biological lifelong learning [52]. Furthermore, its use of basic element-wise operators aligns with highly parallel and energy-efficient hardware, offering substantial energy savings in IoT applications [11, 23, 27, 65]. While HDC is reported as a promising avenue, the literature to date has not explored weakly-supervised lifelong learning using HDC.

In this work, we design and deploy LifeHD, the *first* system for on-device lightweight lifelong learning in an unsupervised and dynamic environment. LifeHD leverages HDC's efficient computation and advantages in lifelong learning, while effectively handling unlabeled streaming inputs. These capabilities extend beyond the scope of existing HDC designs, which have focused overwhelmingly on the supervised setting [23, 27]. Specifically, LifeHD represents the input as high-dimensional, low-precision vectors, and, drawing inspiration from work in cognitive science [5], organizes data into a two-tier memory hierarchy: a short-term "working memory" and a long-term memory. The working memory processes incoming data and summarizes it into a group of fine-grained clusters that are represented by hypervectors called *cluster HVs*. Long-term memory consolidates the frequently appeared cluster HVs in the working memory, and will be retrieved for merging and inference occasionally. We emphasize that LifeHD is designed to suit a variety of edge devices with diverse resource levels. More efficiency gains can be achieved by employing optimizations such as pruning and quantization [15, 61], but this is not the focus of our work.

Our basic approach in LifeHD is fully unsupervised. However, in reality, labels may be available (or could be acquired) for a small number of examples. We introduce LifeHD$_{\text{semi}}$ to exploit a limited number of labeled samples as an extension to the purely unsupervised LifeHD. Additionally, we propose LifeHD$_{\text{a}}$, which uses an adaptive scheme inspired by model pruning, to adjust the HD embedding dimension on-the-fly. LifeHD$_{\text{a}}$ allows us to further reduce resource usage (power in-particular), where necessary.

In summary, the contributions of this paper are:

(1) We design LifeHD, the first end-to-end system for on-device unsupervised lifelong intelligence using HDC. LifeHD builds upon HDC's lightweight single-pass training capability and incorporates our novel clustering-based memory design to address challenges (C1)-(C3).
(2) We further propose LifeHD$_{\text{semi}}$ as an extension to fully utilize the scarce labeled samples along with the stream. We devise LifeHD$_{\text{a}}$ that enables adaptive pruning in LifeHD to reduce real-time power consumption.
(3) We implement LifeHD on off-the-shelf edge devices and conduct extensive experiments across three typical IoT scenarios. LifeHD improves the unsupervised clustering accuracy up to 74.8% with 34.3x better energy efficiency compared to leading unsupervised NN lifelong learning methods [13, 14, 54].
(4) LifeHD$_{\text{semi}}$ improves the unsupervised clustering accuracy by up to 10.25% over the SemiHD [22] baseline under limited label availability. LifeHD$_{\text{a}}$ limits the accuracy loss within 0.71% using only 20% of LifeHD's full HD dimension.

The rest of the paper is organized as follows. We start by a comprehensive review of related works in Sec. 2. We then introduce salient background on HDC in Sec. 3 to help understanding. We formally define the unsupervised lifelong learning problem we target to solve in Sec 4. Afterwards, Sec. 5 describes the details of our major design LifeHD. Sec. 6 introduces LifeHD$_{\text{semi}}$ and LifeHD$_{\text{a}}$. Sec. 7 presents the implementation and results of LifeHD, while the evaluations of LifeHD$_{\text{semi}}$ and LifeHD$_{\text{a}}$ are reported in Sec 8. We add the discussions and future works in Sec. 9. The entire paper is concluded in Sec. 10.

2 RELATED WORK

Lifelong and On-Device Learning. Lifelong learning (or continual learning) is a large and active area of research in the broader machine learning community. Catastrophic forgetting is a major challenge in lifelong learning, and refers to a commonly observed empirical phenomenon in which updating certain machine learning models with new data severely degrades their ability to perform previously learned tasks [36]. Previous works proposed techniques such as dynamic architecture [31, 49], regularization by penalizing important weights [28, 66], knowledge distillation from past models [14] and experience replay using a memory buffer [35, 58]. The lifelong learning literature has examined a wide range of problem settings, ranging from the fully supervised case, in which tasks and class labels are provided, and the fully unsupervised case without any labels and prior knowledge [13, 57]. However, all of these works are based on deep NNs and require backpropagation, which is problematic for resource-constrained devices.

Neurally-inspired lightweight algorithms have recently been proposed for lifelong learning applications. FlyModel [52] and

SDMLP [8] use sparse coding and associative memory for lifelong learning. However, both approaches assume full supervision. STAM [54] is an expandable memory architecture for unsupervised lifelong learning, using layered receptive fields and a two-tier memory hierarchy. It learns via online centroid-based clustering pipeline, novelty detection and memory updates. Nevertheless, the memory in STAM is solely dedicated to image storage, while our LifeHD additionally emphasizes merging past patterns into coarse groups and shows more effective learning performance.

Recent works optimize the resource usage of on-device training via pruning and quantization [34, 45], tuning partial weights [9, 47], memory profiling and optimization [15, 61, 64], as well as growing the NN on the fly [67]. All these works optimize training given resource constraints and do not focus on lifelong learning. They are orthogonal to the contribution of LifeHD which focuses on adaptive and continual training. LifeHD can be further optimized by combining with such techniques.

Hyperdimensional Computing. HDC has garnered substantial interest from the computer hardware community as an energy-efficient and low-latency approach to learning, and has been successfully applied to problems such as human activity recognition [27], voice recognition [23], image recognition [11, 65], to name a few. The large majority of literature on HDC has focused on using the technique to perform supervised classification tasks. Among the limited literature for weakly-supervised learning with HDC, HDCluster [21] enabled unsupervised clustering in HDC with a new algorithm that is similar to K-Means. SemiHD [22] is a semi-supervised learning framework using HDC with iterative self-labeling. Hyperseed [41], C-FSCIL [18] and FSL-HD [65] adopted HDC or similar vector symbolic architectures (VSA) for unsupervised or few-shot learning. All above works did not consider the lifelong aspect and used offline training on a static dataset. To the best of the authors' knowledge, LifeHD is the first work that designs and deploys lifelong learning in edge IoT applications especially with zero or minimal amount of labels.

3 BACKGROUND ON HDC

Hyperdimensional Computing (HDC) is an emerging paradigm for information processing from the cognitive-neuroscience literature [24]. In HDC, all computation is performed on low-precision and distributed representations of data that accord naturally with highly parallel and low-energy hardware.

The first step in HDC is encoding, which maps an input $x \in \mathcal{X}$ to a distributed representation $\phi(x)$ living in some D-dimensional inner-product space $\mathcal{H}$, that we call the "HD-space." For instance, one might take $\mathcal{H} \subset \{\pm 1\}^D$, or $\mathcal{H} \subset \mathbb{R}^D$. We refer to points in the HD-space as *hypervectors*. Encodings of data can be manipulated so as to build more complex composite representations using a set of operators defined as follows:

(1) *Bind*: $\otimes : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$. Binding takes two hypervectors as inputs and returns a hypervector that is dissimilar to both inputs, and is intuitively used to represent tuples. For bipolar hypervectors (i.e., $\mathcal{H} \subset \{\pm 1\}^d$), the binding operator is typically element-wise multiplication.

(2) *Bundle*: $\oplus : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$. Bundling takes two hypervectors as input and returns a hypervector similar to both operands, and is intuitively used to build sets. The bundling operation is implemented through addition.

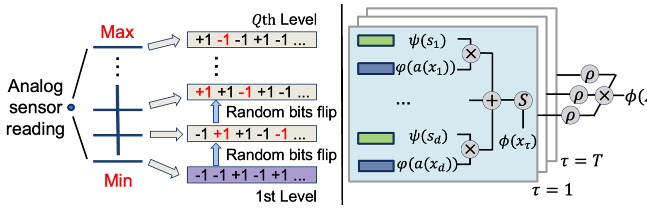


Figure 2: Spatiotemporal HDC encoding for time-series data. Left: random generation of level hypervectors. Right: the complete encoding process.

(3) *Permute*: $\rho : \mathcal{H} \to \mathcal{H}$. Permutation can be used to encode sequential information and is typically implemented using a cyclic shift.

The encoding function $\phi : \mathcal{X} \to \mathcal{H}$ embeds data from its ambient representation into HD-space. In general, encoding should preserve some meaningful notion of similarity between input points in the sense that $\phi(x) \cdot \phi(x') \approx k(x, x')$, where k is some similarity function of interest on $\mathcal{X}$. In this paper, we use spatiotemporal encoding for time series sensor data, and HDnn for more complex data, such as images, which we explain in the following.

Spatiotemporal Encoding. The spatiotemporal method [38] jointly encodes the analog information from each sensor (spatial) and at each time stamp (temporal) to a single hypervector. Suppose there are d-different sensors $s_1, ..., s_d$, each of which produce a real-valued reading $x_1, ..., x_d$, whereupon we may model the input at a particular moment in time by a set of tuples $\{(s_i, x_i)\}_{i=1}^d$. We pre-generate a set of base hypervectors to represent the values and sensors respectively. To represent a real valued feature $x \in \mathbb{R}$, we quantize the support of x into a set of bins with centroids $a_1, ..., a_Q$, and assign each bin an embedding $\varphi(a_i)$, which we call *level hypervectors*, such that $\varphi(a_i) \cdot \varphi(a_j)$ is monotonically decreasing in $|a_i - a_j|$. As shown in Fig. 2 (left), we initially generate a random hypervector for the first level. To maintain similarity between adjacent level hypervectors, for each subsequent level, we randomly flip a fraction of bits from the previous level as described in [56]. The fraction of flipping is denoted as P. This process is repeated until all Q level hypervectors are generated. To represent different sensor s, we assign each sensor a random embedding $\psi(s_i)$, which we call *ID hypervectors*, by sampling $\psi(s_i) \sim \text{Unif}(\{\pm 1\}^D)$.

The complete spatiotemporal encoding is visualized in Fig. 2 (right). We encode a pair (s, x) via $\psi(s) \otimes \varphi(a(x))$, where $a(x)$ is the centroid of the bin closest to x. This preserves both the level and sensor ID information. To encode the readings for all sensors we bundle together their individual embeddings and round to bipolar (e.g. $\{\pm 1\}$) precision: $\phi(x) = \text{Sign}\left(\bigoplus_{i=1}^d \psi(s_i) \otimes \varphi(a(x_i))\right)$. Finally, to represent a sequence of T readings: $X = \{x_1, ..., x_T\}$, we use permutation: $\phi(X) = \bigotimes_{\tau=1}^T \rho^\tau(\phi(x_\tau))$.

HDnn Encoding. In this work we use the recently proposed HDnn style encoding [11, 65] that combines a pretrained and frozen NN feature extractor with HDC's spatiotemporal encoding to obtain state of the art accuracy for sound and images. In HDnn the inputs to the spatio-temporal encoding, $s_1, ..., s_d$, are intermediate feature outputs of the pretrained and frozen NN (Fig. 3). For example,

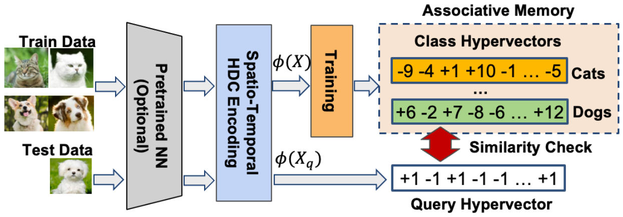


Figure 3: An overview of supervised HDC pipeline including encoding, training and inference (similarity check for classification). An additional pretrained NN is used as a feature extractor in the HDnn.

a section of MobileNet pretrained on ImageNet creates features which are then encoded into HD hypervectors for object recognition tasks. This only marginally increases the computational costs as no training is performed on NN, all the training happens in HD.

Supervised Training and Inference. A common use case of HDC, summarized in Fig. 3, is to fit classifiers. In particular, let us suppose that we see a set of N labeled samples $\{(X_i, y_i)\}_{i=1}^{N}$, where x_i is an input, and $y_i \in \{c_1, ..., c_J\}$ is a class label. In the traditional approach to classification, one simply represents each class via the bundle of its training data. That is: $\phi(c_j) = \bigoplus_{i:y_i=c_j} \phi(X_i)$. We store the trained class hypervectors in an associative memory. For example, in Fig. 3, we compute and store the class hypervectors of cats and dogs. During inference, we first encode the testing sample X_q into a query hypervector $\phi(X_q)$ using the same encoding procedure as for training. We then predict the label corresponding to the most similar class as measured by the cosine similarity, i.e., $\hat{y} = \text{argmax}_j \cos(\phi(X_q), \phi(c_j)) \propto \text{argmax}_j (\phi(X_q) \cdot \phi(c_j))/\|\phi(c_j)\|$.

4 PROBLEM DEFINITION

Before diving into our method, we first rigorously formulate the unsupervised lifelong learning problem using streaming sources, driven by real-world IoT applications.

Streaming Data. To represent continuously changing environment, we assume a well-known *class-incremental* model in lifelong learning, in which new classes emerge in a sequential manner [46]. We also allow data distribution shift within one class. This setting models a scenario in which a device is continuously sampling data while the surrounding environment may change implicitly over time, e.g., the self-driving vehicle as shown in Fig. 1. We require that all samples appear *only once* (i.e., single-pass streams).

Formally, we consider a scenario involving d sensors, each producing a real valued reading. We group readings into sliding windows of length T, and treat one such batch $X_i \in \mathbb{R}^{T \times d}$ as an input sample. Each input X_i is associated with an unknown label y_i. Importantly, the labels are not made available during training, nor the boundaries of class shift. Therefore the entire process is unsupervised. We represent the data stream associated with each class by $\mathcal{D}_j = \{X_1, X_2, ...\}$, and the set of streams for all classes by $\mathcal{D} = \{\mathcal{D}_1, ..., \mathcal{D}_J\}$. Note that the class-incremental streams can have imbalanced classes, i.e., $|D_i| \neq |D_j|, i \neq j$, and gradual distribution shift within each class.

Learning Protocol. Our goal is to build a classification algorithm that maps $\mathcal{X} \to \mathcal{Y}$. For evaluation, we use the common evaluation protocol in state-of-the-art lifelong learning works [13, 14, 54], in which we construct an iid dataset $\mathcal{E} = \{(X_k, y_k)\}$ for periodic testing, by sampling labeled examples from each class in a manner that preserves the overall (im)balance between the classes. Note, that even when one class has not appeared in the training data stream, it is always included in $\mathcal{E}$. Hence $\mathcal{E}$ is a global view of all classes that can potentially exist in the environment.

Unsupervised Clustering Accuracy. Since we do not give class labels or the total number of classes during training, the predicted label can be different from the ground-truth label. Therefore, for evaluation metric, we cannot adopt the simple prediction accuracy that requires exact label matching. Instead, we employ a widely used clustering metric known as unsupervised clustering accuracy (ACC) [63], which mirrors the conventional accuracy evaluation but within an unsupervised context.

Suppose ω_k is the predicted cluster of testing sample (X_k, y_k) in $\mathcal{E}$. ACC is computed as: $ACC = \max_m \frac{1}{|\mathcal{E}|} \sum_{k=1}^{|\mathcal{E}|} \mathbf{1}\{y_k = m(\omega_k)\}$, where m ranges over all possible one-to-one mappings between predicted clusters and ground-truth classes. Intuitively, this metric computes the accuracy under the "best" mapping between clusters and labels. The biggest advantage of ACC is that it does not require the number of clusters and classes to be equal. For instance, a cluster of pines and a cluster of redwood both belong to the ground truth label of trees. We treat such clustering result as a valid learning outcome, with a concrete visualization shown in Sec. 7.4.

5 LIFEHD

In this section, we present the design of LifeHD, the first unsupervised HDC framework for lifelong learning in general edge IoT applications. Compared to operating in the original data space, HDC improves pattern separability through sparsity and high dimensionality, making it more resilient against catastrophic forgetting [52]. LifeHD preserves the advantages of HDC in computational efficiency and lifelong learning, while handling the input of unlabeled streaming data, which has not been achieved in previous work [18, 21, 22, 41, 65].

5.1 LifeHD Overview

Fig. 4 gives an overview of how LifeHD works. The first step is HDC encoding of data into hypervectors as described in Sec. 3. Training samples X are organized into batches of size *bSize* and input into an optional fixed NN for feature extraction (e.g. for images and sound) and the encoding module. The encoded hypervectors $\phi(X)$ are input to LifeHD's two-tier memory design inspired by cognitive science studies [5], consisting of working memory and long-term memory. This memory system *intelligently* and *dynamically* manages historical patterns, stored as hypervectors and referred to as *cluster HVs*. As shown in Fig. 4, the working memory is designed with three components: novelty detection, cluster HV update and cluster HV merge. $\phi(X)$ is first input into novelty detection step (①). An insertion to the cluster HVs is made if a novelty flag is raised, otherwise $\phi(X)$ updates the existing cluster HVs (②). The third component, cluster HV merge (③), retrieves the cluster HVs from long-term memory, and merges similar cluster HVs into a supercluster via a novel spectral clustering-based merging algorithm [59]. The interaction between working and long-term memory happens as commonly encountered cluster HVs are copied to long-term memory, which we call *consolidation* (④). Finally, when the size

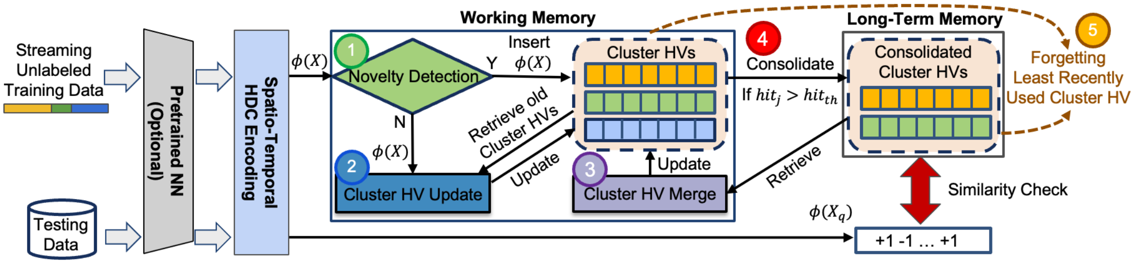


Figure 4: The end-to-end algorithm flow of LifeHD.

Table 1: List of important notations.

Symbol	Meaning
d	Number of sensor sources
T	Time window length of one input sample X
D	Dimension of the HD-space
Q	Number of quantized level for encoding
P	Fraction of random bit flip to generate level hypervector
ϕ	HDC encoding function
φ, ψ	Level and ID hypervector encoding function
$bSize$	Batch size of input samples
$\mathcal{M}, \mathcal{L}$	Set of cluster HVs stored in the working and long-term memory
M, L	Maximum number of cluster HVs in the working and long-term memory
$\mu, \hat{\sigma}$	Mean similarity and standard difference of between each cluster HV and its assigned inputs in the working memory
hit	The number of times that each cluster HV is hit in the working memory
hit_{th}	The hit frequency threshold to consolidate cluster HV from working to long-term memory
p, q	The most recent batch index when each cluster HV is accessed, for the working and long-term memory cluster HVs
γ	Hyperparameter for novelty detection sensitivity
α	Moving average update rate during cluster HV update
g_{ub}	Cluster HV merge sensitivity
f_{merge}	Cluster HV merge frequency
r	Average labeling ratio in LifeHD$_{\text{semi}}$
D_a	Dimension of the mask used in LifeHD$_{\text{a}}$

limit of either working or long-term memory is reached, the least recently used cluster HVs are forgotten (⑤).

All modules in LifeHD work collaboratively, making it adaptive and robust to continuously changing streams without relying on any form of prior knowledge. For example, in scenarios of distribution drift, LifeHD may generate new cluster HVs upon encountering drifted samples initially, which can later be merged into coarse clusters. This approach ensures that LifeHD can efficiently capture and retain historical patterns.

In the following, we discuss more details about the major components of LifeHD: novelty detection (Sec. 5.2), cluster HV update (Sec. 5.3), and cluster HV merging (Sec. 5.4). We summarize the important notations used in this paper in Table 1.

5.2 Novelty Detection

The initial novelty detection step (① in Fig. 4) is crucial for identifying emerging patterns in the environment. Suppose $\mathcal{M} = \{m_1, ..., m_M\}$ is the set of cluster HVs stored in the working memory. We gauge the "radius" of each cluster by tracking two scalars for each cluster HV i: μ_i and $\hat{\sigma}_i$, which represent the mean cosine difference and standard difference between the cluster HV and its assigned inputs. Given $\phi(X)$, we first identify the most similar cluster HV, denoted by j. LifeHD marks $\phi(X)$ as "novel" if it substantially differs from its nearest cluster HV. Specifically, this dissimilarity is measured by comparing $\cos(\phi(X), m_j)$ with a threshold based on the historical distance distribution of cluster HV j:

$$\text{If } \cos(\phi(X), m_j) < \mu_j - \gamma\hat{\sigma}_j, \text{ then flag novel.} \tag{1}$$

The hyperparameter γ fine-tunes the sensitivity to novelties.

LifeHD recognizes new $\phi(X)$ as prototypes and inserts them into the working memory. When reaching its size limit M, the working memory experiences forgetting (⑤ in Fig. 4). The least recently used (LRU) cluster HV, represented by $LRU = \text{argmin}_{i=1}^{M} p_i$, is replaced. Here p corresponds to the latest batch index where the cluster HV was accessed. A similar forgetting mechanism is configured for the long-term memory, where the last batch accessed is marked with q.

5.3 Cluster HV Update

If novelty is not detected, indicating that $\phi(X)$ closely matches cluster HV j, we proceed to update the cluster HV and its associated information (② in Fig. 4). This update process involves bundling $\phi(X)$ with cluster HV m_j, akin to how class hypervectors are updated as described in Sec. 3, and updaing μ_j and $\hat{\sigma}_j$ with their moving average:

$$m_j \leftarrow m_j \oplus \phi(X) \tag{2a}$$
$$\mu_j \leftarrow (1-\alpha)\mu_j + \alpha \cos(\phi(X), m_j) \tag{2b}$$
$$\hat{\sigma}_j \leftarrow (1-\alpha)\hat{\sigma}_j + \alpha|\cos(\phi(X), m_j) - \mu_j| \tag{2c}$$
$$hit_j \leftarrow hit_j + 1, p_j \leftarrow idx \tag{2d}$$

The hyperparameter α adjusts the balance between historical and recent inputs, where a higher α gives more weight to recent samples. Properly maintaining μ_j and $\hat{\sigma}_j$ is vital for tracking the "radius" of each cluster HV, affecting future novelty detection. We also increase the hit frequency hit_j and refresh p_j with current batch index idx. hit_j is further used to compared with a predetermined threshold hit_{th} to decide when a working memory cluster HV appears sufficiently frequently to be consolidated to long-term memory (④ in Fig. 4). p_j determines forgetting as described in the previous section. With this lightweight approach, LifeHD continually records temporal cluster HVs from the environment, while the most prominent cluster HVs are transferred to long-term memory.

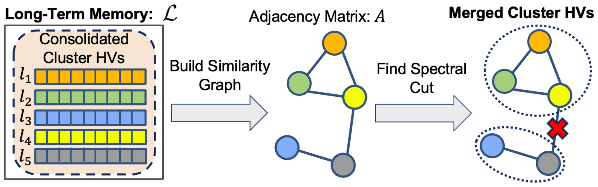


Figure 5: An intuitive visualization of cluster HV merging.

5.4 Cluster HV Merging

Cluster HV merge (③ in Fig. 4) has the dual benefit of reducing memory use and of elucidating underlying similarity structure in the data. Intuitively, a group of cluster HVs can be merged if they are similar to each other and dissimilar from other cluster HVs. For instance, one might merge the cluster HVs for Bulldog and Chihuahua into a single "Dog" cluster HV, that remains distinct from the cluster HV for "Tabby Cat".

To merge the cluster HVs, we first construct a similarity graph defined over the cluster HVs from the long-term memory. The cluster HVs correspond to nodes, and a pair of cluster HVs are connected by an edge if they are sufficiently similar. We then merge the cluster HVs by computing a particular type of cut in the graph in a manner similar to spectral clustering [40]. This graph based formalism for clustering is able to capture complex types of cluster geometry and often substantially outperforms simpler approaches like K-Means [59]. We detail the steps of cluster HV merging in LifeHD below, while Fig. 5 offers an illustrative overview.

Step 1: Preprocessing. Given the set of long-term memory cluster HVs $\mathcal{L} = \{l_1, ..., l_L\}$, we construct a graph $\mathcal{G}$ using the adjacency matrix $A \in \{0,1\}^{L\times L}$. Here, $A_{ij} = A_{ji} = \mathbb{1}[\cos(l_i, l_j) \geq \beta]$, with β as an adaptive threshold. In other words, an edge connects cluster HVs l_i and l_j if their similarity in HD-space surpasses β. A larger β implies that cluster HVs must be more similar to be considered for merging. In practice, we set $\beta = \frac{1}{M}\sum_{i=1}^{M} \mu_i$, representing the mean of the observed cluster HVs.

Step 2: Decomposition. We compute the Laplacian $W = D - A$, where D is the diagonal matrix in which $D_{ii} = \sum_j A_{ij}$. We then compute the eigenvalues $\lambda_1, .., \lambda_L$, sorted in increasing order, and eigenvectors $v_1, ..., v_L$ of W.

Step 3: Grouping. We infer $k = \max_{i\in[L]} \lambda_i \leq g_{ub}$, and merge the cluster HVs by running K-Means on $v_1, ..., v_k$. The upperbound g_{ub} is a hyperparameter that adjusts the granularity of merging, with a smaller g_{ub} leading to smaller k thus encouraging merging more aggressively.

Our merging approach is formally grounded, as discussed in [59]. It is a well-known fact that the eigenvectors of W encode information about the connected components of $\mathcal{G}$. When $\mathcal{G}$ has k connected components, the eigenvalues $\lambda_1 = \lambda_2 = ... = \lambda_k = 0$. To recover these components, K-Means clustering on $v_1, ..., v_k$ can be employed, as explained in [59]. However, practical scenarios may have a few inter-component edges that should ideally be distinct. For instance, when the similarity threshold is imprecisely set, erroneous edges may appear in the graph, causing $\lambda_1, ..., \lambda_k$ to be only approximately zero. Our merging approach is designed to handle this situation by introducing g_{ub}. The cluster HV merging is evaluated every f_{merge} batches, where f_{merge} is a hyperparameter that controls the trade-off between merging performance and computational latency. Both g_{ub} and f_{merge} are analyzed in Sec. 7.8, along with other key hyperparameters in LifeHD.

Time Complexity of Merging. A potentially limiting issue with spectral clustering is its time complexity, which is, in the worst case $O(L^3)$. However, this is not a concern in our setting. First, the number of cluster HVs in long-term memory (L) is typically small, around 50 in practice, resulting in modest worst-case complexity. Secondly, worst-case analysis is overly pessimistic, assuming a full eigendecomposition of the graph Laplacian (W). In practice, W is nearly always approximately low rank, meaning that only the first $k \ll L$ eigenvectors are needed. In such cases, fast randomized eigendecomposition algorithms can reduce the time complexity to linear in L [17]. Thus, while spectral clustering is sometimes colloquially thought of as an "expensive" procedure, this is true only in very unfavorable "worst-case" settings. In practice, its complexity is modest and acceptable for our situation, as shown in Sec. 7.5.

6 VARIANTS OF LIFEHD

While LifeHD is designed to cater to general IoT applications with streaming input and without supervision, real-world scenarios may vary. Some scenarios might have a few labeled samples in addition to the unlabeled stream, while others may require operation within strict power constraints. LifeHD offers extensibility to address these diverse needs. In this section, we introduce two software-based extensions: LifeHD$_{\text{semi}}$, which adds a separate processing path to manage labeled samples, and LifeHD$_{\text{a}}$, which adaptively prunes the HDC model using masking to handle low-power scenarios.

6.1 LifeHD$_{\text{semi}}$

While LifeHD excels in unsupervised scenarios, it does not harness labeled data when available. To address this limitation, we introduce LifeHD$_{\text{semi}}$ as an extension to enhance accuracy utilizing the limited labels. In Fig. 6, we provide an overview of LifeHD$_{\text{semi}}$. For each input batch idx, we consider two subsets: one labeled $(X_{l,idx}, y_{l,idx})$ and one unlabeled $X_{ul,idx}$. We denote the average labeling ratio throughout the data stream as $r = \frac{\sum_{idx} |X_{l,idx}|}{\sum_{idx} |X_{l,idx}| + |X_{ul,idx}|}$. Since obtaining external supervision is often challenging in dynamic environments, we focus on cases where $r \leq 0.01$.

LifeHD$_{\text{semi}}$ retains the two-tier memory structure of LifeHD but introduces modifications to the working memory components. In the LifeHD$_{\text{semi}}$ pipeline, the working memory undergoes three key steps. Firstly, labeled samples (X_l, y_l) update labeled class hypervectors following the conventional HDC methods outlined in Sec. 3. Next, we process unlabeled samples X_{ul} through novelty detection and HV update modules, mirroring LifeHD. Importantly, in LifeHD$_{\text{semi}}$, these operations are applied to both labeled HVs and

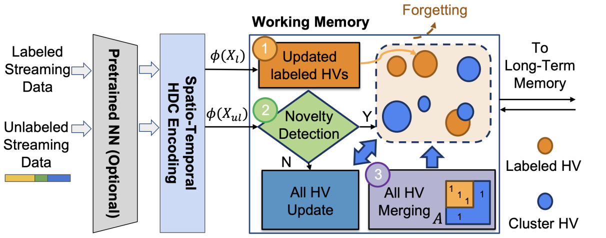


Figure 6: An overview of LifeHD$_{\text{semi}}$ which is designed to handle scarce labeled samples.

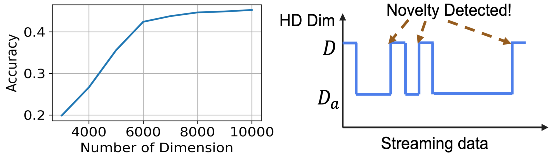


Figure 7: Left: An example of the impact of masking on HDC. Test on CIFAR-10 [29]. Right: The intuitions behind the design of LifeHD$_a$.

cluster HVs. Lastly, we introduce a merging step to group labeled HVs and cluster HVs that are closely related. To handle labeled HVs, we modify the adjacency matrix A by making it diagonal for labeled entries. For example, if the first J HVs correspond to labeled HVs, we ensure that $A_{1:J,1:J} = \text{diag}([1, ..., 1])$, while calculating the remaining values following LifeHD procedures. This strategy prevents the merging of labeled HVs with each other. With these adjustments, LifeHD$_\text{semi}$ offers a solution that retains the core elements of LifeHD while handling scarce labeled inputs.

6.2 LifeHD$_\text{a}$

While HDC computation is typically lightweight, there may be instances of energy scarcity (e.g., when powered by a solar panel) that call for a balance between accuracy and power efficiency. Similar to neural networks, one approach is to prune the HDC model using a mask, retaining the most crucial HDC dimensions post-encoding [25]. Dimension importance can be determined by aggregating all class hypervectors into one and sorting the values across all dimensions. Notably, direct reduction of the encoding dimension should be avoided, as it can degrade HDC's expressive capability and end up with corruption. Fig. 7 (left) visually demonstrates the impact of masking in supervised HDC tasks: retaining the top 6000 bits incurs only a 3% accuracy loss compared to using the full 10K-bit precision.

While the concept of masking has been employed in prior HDC studies [25], they are not directly applicable to LifeHD due to their offline training setting with iid data. With streaming non-iid data in LifeHD, the set of observed cluster HVs may only represent a subset of the potential classes, and the less significant bits could become crucial as new classes are introduced.

We introduce LifeHD$_\text{a}$, which enhances LifeHD through an adaptive masking approach applied to all cluster HVs in working and long-term memory. Let D_a represent the target dimension for reduction. The rationale behind LifeHD$_\text{a}$ is depicted in Fig. 7 (right). Whenever an original LifeHD detects novelty, we temporarily revert to the full dimension D for 2 batches, which is sufficient for LifeHD to consolidate new patterns in its memory. After these two batches, we assess the long-term memory cluster HVs by aggregating them and ranking the dimensions, and derive a mask retaining D_a dimensions with the largest absolute values. This mask is then applied to the following batches of $\phi(X)$ immediately after encoding, up until the next novelty is detected. Novelty detection is executed with the masked hypervectors. Importantly, LifeHD$_\text{a}$ can utilize the same novelty detection sensitivity as LifeHD since the most significant dimensions dominate the similarity check. In other words, the similarity results in LifeHD$_\text{a}$ using D_a dimension are similar as using the full dimension. LifeHD$_\text{a}$ offers an adaptive HDC model pruning interface with minimal accuracy loss and overhead.

7 EVALUATION OF LIFEHD

7.1 System Implementation

We implement LifeHD with Python and PyTorch [43] and deploy it on three standard edge platforms: Raspberry Pi (RPi) Zero 2 W [3], Raspberry Pi 4B [2], and Jetson TX2 module [1]. The selection of edge platforms represent three tiers with small, medium and abundant resources.

RPi Zero 2 W has a 1GHz quad-core Cortex-A53 CPU and 512MB SDRAM. RPi 4B enjoys a 1.8GHz quad-core Cortex-A72 CPU and 4GB SDRAM. The Jetson TX platform is equipped with a dual-core NVIDIA Denver 2 CPU, a quad-core ARM Cortex-A57 MPCore, an NVIDIA Pasca family GPU with 256 NVIDIA CUDA cores, and 8GB RAM. We measure the training latency per batch and the energy consumption using the Hioki 3334 powermeter [19].

We are aware that all NN-based feature extractors can be pruned and quantized to attain more efficient deployment on edge platforms [11], same for the NN-based baselines we compare to [13, 14]. However, NN model compression is not the primary focus of LifeHD. Existing compression techniques [26, 39] can be applied directly to the feature extractor in LifeHD. We leave LifeHD with acceleration design and emerging hardware deployment for future works.

7.2 Experimental Setup

We conduct comprehensive experiments to evaluate LifeHD on three typical edge scenarios. All three scenarios incorporate continuous data streams and expect lifelong learning over time. We summarize the experimental setup in Table 2.

Application #1: Personal Health Monitoring. Continuous health monitoring has emerged as a popular use case for IoT. We utilize the MHEALTH [6] dataset which includes measurements of acceleration, rate of turn, and magnetic field orientation on a smartwatch. MHEALTH differentiates 12 activities in daily lives and is collected from 10 subjects. Notably, MHEALTH employs raw time-series signals rather than processed frequency components as inputs. We use time windows of 2.56s ($T = 128$) with 75% overlap to generate the samples. In contrast to previous datasets, we strictly adhere to the temporal order during data collection.

Application #2: Sound Characterization. Continuous sound detection contributes to the characterization of urban environments. We choose the ESC-50 [44] dataset to emulate this scenario. This dataset comprises 5-second-long recordings categorized into 50 semantically diverse classes, including animals, human sounds, and urban noises. We construct the class-incremental streams by arranging the data in random order within each class.

Application #3: Object Recognition. Object recognition is a common use case for camera-mounted mobile systems, e.g., self-driving vehicles. We set up a class-incremental stream from CIFAR-100 [29], consisting of 32×32 RGB images of 20 coarse classes. We further evaluate the case of data distribution drift by examining gradual rotations occurring within each CIFAR-100 class.

On MHEALTH, LifeHD is fully dependent on the HDC spatiotemporal encoder to process the raw time-series signals. For ESC-50 and CIFAR-100, LifeHD utilizes the HDnn framework with a pretrained feature extractor before HDC encoding, same as in the state-of-the-art HDC works [18, 52]. Specifically, we adapt a pretrained ACDNet with quantified weights [37] for ESC-50. ACDNet

Table 2: Experimental setup of LifeHD across all datasets.

Dataset	Application Category	Classes (Balanced?)	Total Samples	Training Data Order	HDnn?	Pretrained Models	# of Params
MHEALTH [6]	Activity	12 (N)	9K	Temporal order during collection	N	-	-
ESC-50 [44]	Sound	50 (Y)	2K	Class-incremental, random within class	Y	ACDNet [37]	4.7M
CIFAR-100 [29]	Image	20 (Y)	60K	Class-incremental, random within class or gradual rotation within class	Y	MobileNet V2 [51] or MobileNet V3 small [20]	2.2M 927K

Table 3: Important hyperparameters configuration of LifeHD.

Dataset	HDC Encoding			LifeHD Design				
	D	Q	P	$bSize$	M	γ	g_{ub}	f_{merge}
MHEALTH	1000	5	0.01	32	50	3.0	0.2	25
ESC-50	10000	100	0.02	32	100	1.0	0.1	5
CIFAR-100	10000	100	0.01	32	100	1.0	0.1	150

is a compact convolutional neural network architecture designed for small embedded devices. For CIFAR-100, we use a MobileNet V2 [51] for accuracy evaluation and MobileNet V3 small [20] for efficiency evaluation, both pretrained on ImageNet [48]. For all pretrained NNs, we remove the last fully connected layer used for classification and keep the remaining weights frozen.

Table 3 summarizes the key hyperparameters in LifeHD, which are selected based on a separate validation set. We configure $\alpha = 0.1$ for moving-average update, $hit_{th} = 10$ for long-term memory consolidation. The long-term memory size L is set to 50 in all cases.

7.3 State-of-the-Art Baselines

We conduct a comprehensive comparison between LifeHD and state-of-the-art NN-based unsupervised lifelong learning baselines, which continuously train a NN for representation learning. The loss functions in these setups are defined in the feature space without relying on label supervision. During testing, we freeze the neural network and apply K-Means clustering on the testing feature embeddings to generate predicted labels. k is set to 50 which is the same number of cluster HVs as in LifeHD. Such a pipeline is widely used for lifelong learning evaluations [46, 54].

Fig. 8 presents a comparison of the pipeline setup using both the baselines and LifeHD on HDnn and non-HDnn frameworks respectively. To ensure fair comparisons, in HDnn framework on ESC-50 and CIFAR-100, we initialize the NN with the same pretrained weights for LifeHD and NN baselines. For the NN baselines on MHEALTH, we randomly initialize a one-layer LSTM of 64 units followed by a fully connected layer of 512 units. This architecture has achieved competitive accuracy as the Transformers-based designs on MHEALTH [12].

We compare LifeHD with the following baselines, which include all main lifelong learning techniques:

- **Finetune** is a naïve baseline that optimizes the NN model using the current batch of data without any lifelong learning techniques.
- **CaSSLe** [14] is a distillation-based framework that utilizes self-supervised losses. It leverages distillation between the representations of the current model and a past model. In the original paper, the past model is captured at the end of the previous task and prior to the introduction of a new task. However, since we do not assume awareness of task shifts, we simply freeze the model from the previous batch.
- **LUMP** [13] employs a memory buffer for replay and mitigates catastrophic forgetting by interpolating the current batch with previously stored samples in the memory.
- **STAM** [54] is brain-inspired expandable memory architecture using online clustering and novelty detection. We exclusively apply STAM to CIFAR-100 due to its demand for intricate dataset-specific tuning (e.g., number of receptive fields), and because the authors only released the implementation for the CIFAR datasets.
- **SupHDC** [18, 27] is the fully supervised HDC pipeline.

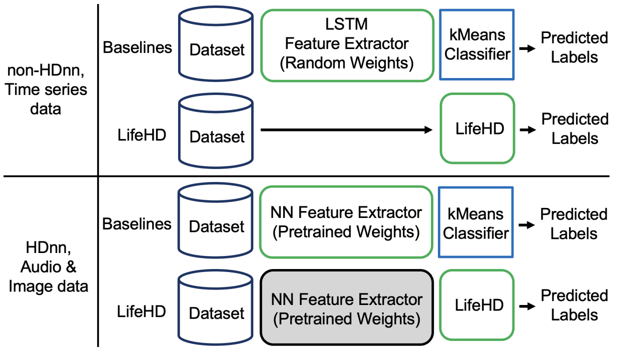
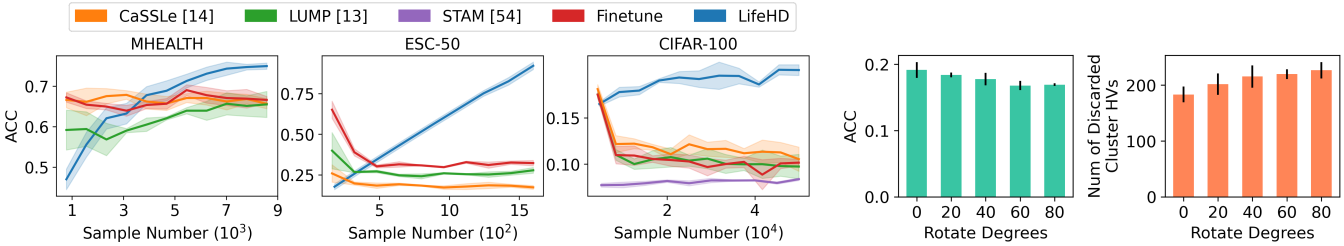


Figure 8: A graphical explanation of the pipeline setup. Green outlines denote the module trained with the streaming data. Blue outlines denote the unsupervised classifier trained during testing. Gray denotes when the module is frozen & not trained further.

All baselines are adapted from their original open-source code. For CaSSLe and LUMP, we employ BYOL [16] as the self-supervised loss function because it has showed superb empirical performance in lifelong learning tasks compared to other self-supervised learning backbones [14]. We use the memory buffer size of 256 for LUMP which is the same as in the original paper. We employ the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.03 across all methods, training each batch for 10 steps. All experiments are executed for 3 random trials.

7.4 LifeHD Accuracy

Results on Three Application Scenarios. Fig. 9 (a) details the ACC curve of all methods as streaming samples are received. All NN baselines start at higher accuracy, especially in ESC-50 (sounds) and CIFAR-100 (images), owing to the presence of a pretrained NN feature extractor within the HDnn framework. Meanwhile, LifeHD begins with lower accuracy as both the working and long-term memories are empty, needing to learn the cluster HVs and the optimal number of clusters. Notably, as streaming samples come in,



(a) The ACC curve of LifeHD vs. state-of-the-art NN baselines on three scenarios. (b) The final results of LifeHD in CIFAR-100 with rotation.

Figure 9: The unsupervised clustering accuracy (ACC) results of LifeHD on various input data streams.

Table 4: The gap of ACCs at the end of the stream between LifeHD and Supervised HDC [18, 27].

Method	MHEALTH	ESC-50	CIFAR-100
LifeHD	0.75	0.92	0.20
Supervised HDC [18, 27]	0.90	0.95	0.26
Gap	-0.15	-0.03	-0.06

all NN baselines experience a decline in ACC, underscoring the inherent challenges of unsupervised lifelong learning with streaming non-iid data and a lack of supervision. This is primarily due to the demand for extensive iid data and multi-epoch offline training for finetuning NNs, which is not feasible in our setting. CaSSLe [14] leads to forgetting due to its inability to identify suitable past models from which to distill knowledge. Similarly, LUMP [13] exhibits reduced ACC in ESC-50 and CIFAR-100, with only marginal ACC improvement in MHEALTH (time series), suggesting that its memory interpolation strategy may not be universally suitable for all applications. While the memory-based design of STAM [54] can mitigate forgetting, its efficacy in distinguishing patterns and acquiring new knowledge remains unsatisfactory. On the contrary, LifeHD demonstrates incremental accuracy across all three different scenarios, achieving **up to 9.4%, 74.8% and 11.8% accuracy increase** on MHEALTH (time series), ESC-50 (sound) and CIFAR-100 (images), compared with the NN-based unsupervised lifelong learning baselines at the end. Such outcome can be attributed to HDC's lightweight but meaningful encoding and the effective memorization design of LifeHD.

Results under Data Distribution Drift. We further evaluate LifeHD's performance under drifted data and present the final ACC along with the number of discarded cluster HVs in Fig. 9 (b). Specifically, we introduce gradual rotation to the CIFAR-100 samples within each class, ranging from no rotation to a substantial rotation angle of 80°. The other parameter settings remain the same as in Table 3. The number of discarded cluster HVs accounts for those that are either forgotten or merged. From Fig. 9 (b), we can observe the remarkable resilience of LifeHD to drifted data, with an ACC loss of less than 2.3% even under a severe rotation of 80°. This robustness stems from the general and uniform design of LifeHD to accommodate various types of continuously changing data streams. In cases of slight or minimal distribution drift, LifeHD updates existing cluster HVs; in instances of severe drift, new cluster HVs are created and subsequently merged if deemed appropriate. However, due to the finite memory capacities, more cluster HVs are subject to forgetting or merging under larger drifts, as shown in Fig. 9 (b) by the number of discarded cluster HVs, leading to ACC loss. In our experiments, LifeHD demonstrates minimal ACC loss even under substantial rotation shifts.

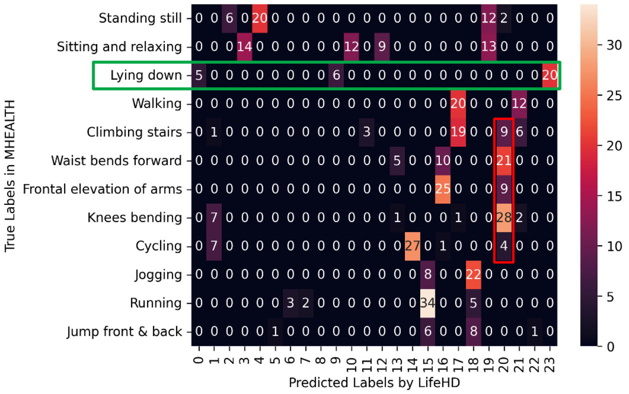
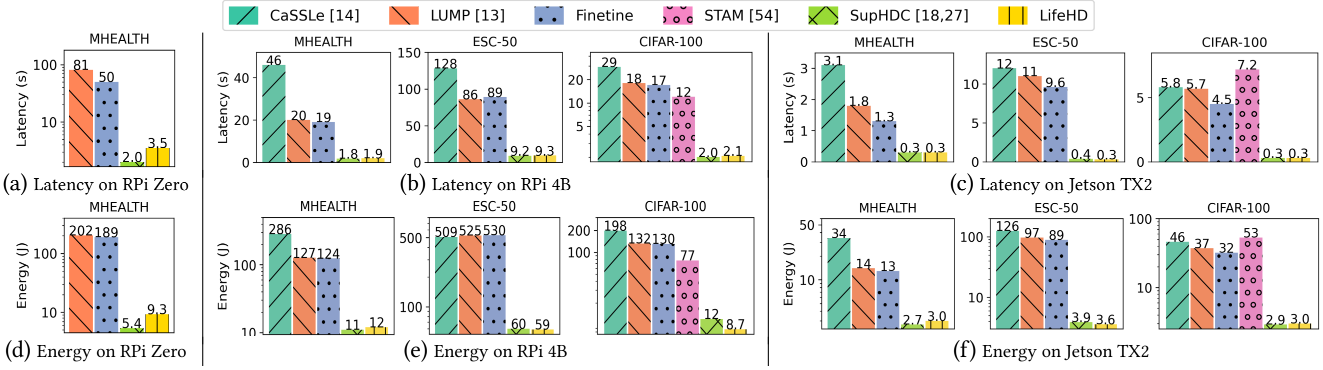


Figure 10: The confusion matrix of LifeHD on MHEALTH. The green box highlights smaller cluster HVs that form a single large ground truth class, which is a valid learning outcome. The red box highlights "boundary" cluster HVs that span multiple true labels, leading to lower ACCs.

Comparison with Fully Supervised HDC. Table 4 compares the average ACCs of supervised HDC method [18, 27] and LifeHD. Even without any supervision, LifeHD approaches the ACC of supervised HDC with **a gap of 15%, 3% and 6%** on MHEALTH, ESC-50 and CIFAR-100. A minimal ACC gap confirms the effectiveness of LifeHD in separating and memorizing key patterns. To help explain the small ACC loss even without supervision, we visualize the confusion matrix of LifeHD on MHEALTH in Fig. 10. MHEALTH has 12 true classes (y axis), whereas LifeHD maintains 23 cluster HVs in its long-term memory (x axis). ACC is evaluated by mapping the unsupervised cluster HVs to true labels. Although LifeHD cannot achieve precise label matching with true classes, it can preserve the essential patterns by using finer-grained clusters. For example, the green box in Fig. 10 highlights a valid learning outcome, where LifeHD uses predicted cluster HV No. 0, 9 and 23 to represent a bigger true class of "Lying down".

7.5 Training Latency and Energy

Fig. 11 provides comprehensive latency and energy consumption results to train one batch of samples on all three edge platforms. For CIFAR-100, we use the most lightweight MobileNet version, V3 small [20], as HDnn feature extractor and NN baseline, to assess LifeHD's efficiency gain over the most competitive mobile computing setup. On RPi Zero, we report results for the relatively lightweight NN-based baselines, Finetune and LUMP [13], using



(a) Latency on RPi Zero (b) Latency on RPi 4B (c) Latency on Jetson TX2

(d) Energy on RPi Zero (e) Energy on RPi 4B (f) Energy on Jetson TX2

Figure 11: Latency and energy consumption to train one batch of data using LifeHD and all baselines on off-the-shelf edge platforms.

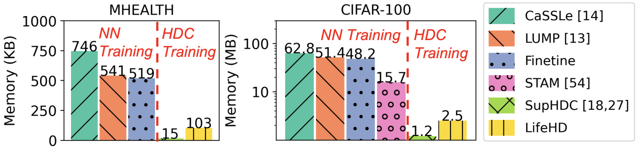


Figure 12: Peak memory footprint of all methods on MHEALTH (left) and CIFAR-100 (right) with batch size of 1. The results are representative for time series data and image data.

the smallest dataset, MHEALTH, while running CaSSLe [14] on MHEALTH would result in out-of-memory errors. As shown in Fig. 11, LifeHD is **up to 23.7x, 36.5x and 22.1x faster** to train on RPi Zero, RPi 4B and Jetson TX2, respectively, while being **up to 22.5x, 34.3x and 20.8x more energy efficient** on each, compared to the NN-based unsupervised lifelong learning baselines. In most settings, CaSSLe [14] is the most time-consuming because of the expensive distillation. LUMP [13] is slightly more expensive than Finetune due to its replay mechanism. STAM [54], implemented only on CPU, incurs the longest training latency on Jetson TX2, as it does not use GPU's acceleration capabilities. LifeHD is clearly faster and more efficient than all NN-based unsupervised lifelong learning baselines [13, 14, 54] due to LifeHD's lightweight nature. The overhead of LifeHD alongside fully supervised HDC, SupHDC [18, 27], is negligible on more powerful platforms like RPi 4B and Jetson TX2. Notably, in LifeHD, the cluster HV merging step for processing about 40 LTM elements takes 7.4, 0.86 and 0.66 seconds to run on RPi Zero, RPi 4B and Jetson TX2, respectively, which only executes once every f_{merge} batches. Further enhancements can be achieved using the acceleration techniques mentioned in Sec. 5.4.

Fig. 11 indicates LifeHD improves latency and energy efficiency the most on RPi 4B, as compared to RPi Zero and Jetson TX2 that represent more limited or powerful devices. This is because the high-dimensional nature of LifeHD requires a fair amount of memory, thus it cannot run efficiently on the highly restricted RPi Zero. The GPU resources on Jetson TX2 boost the NN-based baselines, narrowing the gap between them and LifeHD. We expect much larger efficiency improvements when LifeHD is accelerated using emerging in-memory computing hardware [11, 65].

7.6 Memory Usage

Fig. 12 provides a comprehensive summary of peak memory footprint for all methods on MHEALTH and CIFAR-100. We categorize the methods into NN training (Finetune, LUMP [13], CaSSLe [14], STAM [54]) and HDC training (Supervised HDC [18, 27] and our LifeHD). Following [30], we calculate the peak memory of NN training as the sum of model, optimizer and activation memories, plus additional memory consumption for lifelong learning. Specifically, CaSSLe [14] requires additional memory for training a predictor and inference from a frozen model, LUMP [13] needs extra memory for replay. For HDC-based methods, each dimension of the cluster HV is represented as a signed integer and stored in a byte. In addition to the working and long-term memories, we also consider the storage of bipolar level and ID hypervectors for encoding, and the frozen MobileNet for HDnn encoding in CIFAR-100. Notice that our focus here is on comparing full-precision memory usage, and optimization techniques like quantization can be applied to all methods in the future.

The results in Fig. 12 highlight LifeHD's memory efficiency. LifeHD conserves 80.1%-86.2% and 84.1%-96.0% of memory compared to NN training baselines on MHEALTH (non-HDnn) and CIFAR-100 (HDnn), respectively. This remarkable efficiency stems from LifeHD's HDC design, which dispenses with the memory-intensive gradient descents in NNs. STAM [54], with its hierarchical and expandable memory structure, consumes 6.3x the memory of LifeHD, as it stores raw image patches across all hierarchies. Compared to fully supervised HDC, SupHDC [18, 27], LifeHD introduces a modest memory increase to accomplish the challenging task of organizing label-free cluster HVs. LifeHD proves advantageous for edge applications with only 103 KB and 2.5 MB of peak memory required for MHEALTH and CIFAR-100.

7.7 Ablation Studies

The design of LifeHD consists of several key elements: the two-tier memory organization, novelty detection and online update, and cluster HV merging that manipulates past patterns. We conduct experiments to assess the contribution of each element. Using the configuration in Table 3, we evaluate the performance of (i) LifeHD without long-term memory, using only a single layer memory, (ii) LifeHD without merging, employing only novelty detection, online update and forgetting, and (iii) complete LifeHD. We present the ACC and the number of cluster HVs in LTM during MHEALTH training in Fig. 13, chosen as a representative scenario. LifeHD without LTM (green dashdot line) forces cluster HV merging to

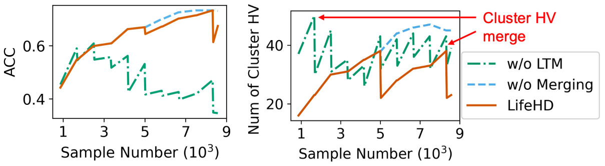


Figure 13: Ablation study of LifeHD on MHEALTH. The number of cluster HVs reported for LifeHD without long-term memory (LTM) is for the working memory, since no LTM is allowed.

take place in working memory, where the large number of temporary cluster HVs creates less important nodes in the graph and corrupts the graph-based merging process, as shown in Fig. 13 (left). This necessitates the design of the two-tier memory architecture and merging with LTM elements. LifeHD without merging (blue dashed line) consumes 1x more memory in the LTM, making it unsuitable for resource-constrained edge devices. Our design of LifeHD (red solid line) strategically combines similar cluster HVs with minor loss on the clustering quality, achieving ACC similar to those without merging while conserving memory storage.

7.8 Sensitivity Analysis

Fig. 14 summarizes the sensitivity results of key parameters in LifeHD, while the less sensitive ones such as α and hit_{th} are omitted due to space limitation. The default setting is the same as in Table 3.

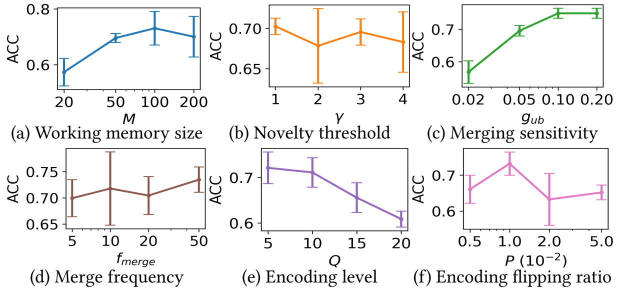
Working Memory Size. Fig. 14 (a) shows ACCs using working memory sizes of 20, 50, 100 and 200. In general, a larger working memory allows more temporary cluster HVs at the cost of higher memory consumption. $M = 100$ produces optimal results, while further increasing the memory size reduces clustering quality. This occurs because excessively large working memory retains outdated prototypes, degrading lifelong learning performance.

Novelty Threshold. In Fig. 14 (b), we present the final ACCs for different novelty detection thresholds (γ). A lower γ results in more frequent novelty detections and increased loads on the working memory, while a higher γ may lead to overlooking significant changes. Remarkably, LifeHD demonstrates resilience to variations in γ, a phenomenon that we attribute to the combined impact of novelty detection and merging processes.

Merging Sensitivity. Fig. 14 (c) shows ACC using various merging thresholds (g_{ub}). g_{ub} determines the number of clusters (k) to merge in the cluster HV merging step (Sec. 5.4). A low value for g_{ub} results in overly aggressive merging, leading to the fusion of dissimilar cluster HVs and a degraded ACC. A larger g_{ub} adopts a conservative merging strategy and encourages finer-grained clusters, albeit at the expense of increased resource demands.

Merging Frequency. Fig. 14 (d) shows the final ACCs for different merging frequencies (f_{batch}). LifeHD shows its robustness across various f_{batch} values, partly due to the presence of g_{ub} to prevent aggressive merging. Less frequent merging (larger f_{batch}) raises the risk of forgetting important patterns as of memory constraints. More frequent merging (smaller f_{batch}) increases the computational burden due to the spectral clustering-based algorithm.

Encoding Level and Flipping Ratio for Spatiotemporal Encoding. Fig. 14 (e) and (f) show the ACCs for various quantization encoding levels (Q) and flipping ratios (P) during the spatiotemporal encoding. Both parameters are important for preserving the similarity in HD-space after encoding. Optimal Q depends on the sensor sensitivity, with finer-grained sensors requiring more quantization levels. P determines the similarity between adjacent levels of hypervectors. For personal health monitoring, such as MHEALTH, $Q = 10, P = 0.01$ usually gives the best results.



(a) Working memory size (b) Novelty threshold (c) Merging sensitivity
(d) Merge frequency (e) Encoding level (f) Encoding flipping ratio

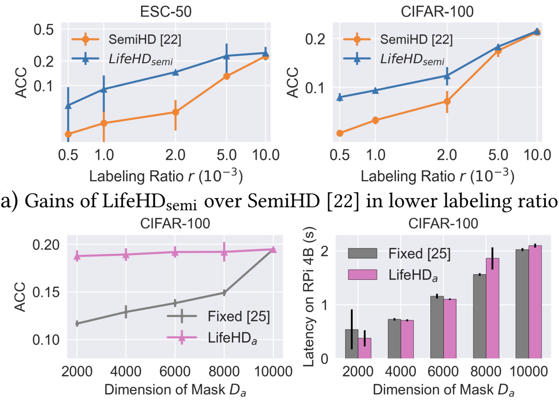
Figure 14: Sensitivity of various hyperparameters in LifeHD, using MHEALTH dataset as a representative.

8 EVALUATION OF LIFEHD$_{\mathbf{semi}}$ AND LIFEHD$_{\mathbf{a}}$

In this section, we compare LifeHD$_{\text{semi}}$ and LifeHD$_{\text{a}}$, our proposed extensions from LifeHD, with existing designs that are similar.

Performance of LifeHD$_{\mathbf{semi}}$. To evaluate LifeHD$_{\text{semi}}$ in a low-label scenario, we compare it with SemiHD [22], which is the state-of-the-art HDC method for semi-supervised learning. We adapt SemiHD [22] for single-pass settings, introducing a pseudolabel assignment threshold. When the cosine similarity of an unlabeled sample to the nearest class hypervector surpasses the threshold, we assign that class as its pseudolabel. The sample is then employed to update the class hypervector in SemiHD. We explore various threshold values and choose the optimal result for comparison. Fig. 15 (a) compares LifeHD$_{\text{semi}}$ and SemiHD [22] on ESC-50 and CIFAR-100 across various labeling ratios $r < 0.01$. The advantages of LifeHD$_{\text{semi}}$ are most prominent when labels are limited, the weakly supervised scenario is LifeHD$_{\text{semi}}$'s major focus. LifeHD$_{\text{semi}}$ improves ACC by up to 10.25% and 3.6% on ESC-50 and CIFAR-100 respectively. This outcome arises from the unsupervised nature of LifeHD, allowing it to autonomously organize prominent cluster HVs, especially when all samples from a class lack labels. As the labeling ratio increases, LifeHD$_{\text{semi}}$'s advantage over SemiHD diminishes, because more labels bolster SemiHD's performance.

Performance of LifeHD$_{\mathbf{a}}$. LifeHD$_{\text{a}}$ provides an interface to trade minimal performance loss for efficiency gains, by adaptively pruning out the insignificant dimensions. We compare LifeHD$_{\text{a}}$ with previous HDC works employing a fixed mask throughout training [25], and the results are presented in Fig. 15 (b) for CIFAR-100, including ACC and training latency per batch on RPi 4B. Fixed masks negatively impact HDC learning, especially with smaller dimensions. Such masks fail to adapt to new hypervectors in class-incremental streams, where less significant dimensions may become crucial later in training. LifeHD$_{\text{a}}$ addresses this issue by adjusting the mask upon novelty detection, leading to a degradation of only 0.71% in ACC and 4.5x efficiency gain compared to the complete LifeHD, using only 20% of the full HD dimension of LifeHD. The overhead of adaptively adjusting the mask is negligible when novelty detection occurs infrequently.



(a) Gains of LifeHD$_{\text{semi}}$ over SemiHD [22] in lower labeling ratios.

(b) LifeHD$_{\text{a}}$ vs. using fixed mask (Fixed) [25] under various D_a.

Figure 15: Results of LifeHD$_{\text{semi}}$ and LifeHD$_{\text{a}}$ compared to existing HDC techniques for similar goals.

9 DISCUSSIONS AND FUTURE WORKS

Problem Scale. One limitation of LifeHD is the relative small problem scale (e.g., the image size of CIFAR-100 is restricted to 32x32) due to the essential difficulty of unsupervised lifelong learning problem, including single-pass non-iid data and no supervision. For the same reason, there remains a disparity in accuracy between unsupervised lifelong learning and fully supervised NNs, as substantiated by prior research [13, 54]. In order to scale LifeHD to more challenging applications such as self-driving vehicles, one possible direction is to leverage the pretrained foundation model as a frozen feature extractor in the HDnn framework, which we leave for future investigation.

Hyperparameter Tuning. While we recognize that hyperparameters can influence the performance of LifeHD, such an issue is not exclusive to LifeHD, but has persistently been a challenge in machine learning research [7]. In LifeHD, the impact of hyperparameters can be mitigated through pre-deployment evaluation and component co-design. For example, encoding parameters such as Q, P can be tuned on similar health monitoring data sources prior to deployment. Meanwhile, the component of cluster HVs merging can increase LifeHD's resiliency to the novelty detection threshold γ, as a higher quantity of novel clusters can be merged in later stage of learning.

Limitations of HDC. HDC serves as the fundamental core of LifeHD. While HDC shows promise with its notable lightweight design, it is burdened by several limitations that remain active areas of research. First, for complex datasets like audio and images, HDC requires a pretrained feature extractor (the HDnn encoding) which may not exist for certain applications. Moreover, akin to any other architecture, HDC vectors face capacity limitations determined by the dimension of HD space, encoding method, and potential noise levels in the input data [56]. Due to these factors, careful evaluation and sometimes manual feature engineering are required to successfully deploy HDC for new applications.

Future Works. Although LifeHD focuses on single-device lifelong learning for classification tasks, the method can be extended for other types of tasks and learning settings, such as federated learning and reinforcement learning. We leave the investigation of these topics for future work.

10 CONCLUSION

The ability to learn continuously and indefinitely in the presence of change, and without access to supervision, on a resource-constrained device is a crucial trait for future sensor systems. In this work, we design and deploy the first end-to-end system named LifeHD to learn continuously from real-world data streams without labels. Our approach is based on Hyperdimensional Computing (HDC), an emerging neurally-inspired paradigm for lightweight edge computing. LifeHD is built on a two-tier memory hierarchy including a working and a long-term memory, with collaborative components of novelty detection, online cluster HV update and cluster HV merging for optimal lifelong learning performance. We further propose two extensions to LifeHD, LifeHD$_{\text{semi}}$ and LifeHD$_{\text{a}}$, to handle scarce labeled samples and power constraints. Practical deployments on typical edge platforms and three IoT scenarios demonstrate LifeHD's improvement of up to 74.8% on unsupervised clustering accuracy and up to 34.3x on energy efficiency compared to state-of-the-art NN-based unsupervised lifelong learning baselines [13, 14, 54].

ACKNOWLEDGMENTS

The authors would like to thank the anonymous shepherd, reviewers, and our colleague Xiyuan Zhang for their valuable feedback. This work was supported in part by National Science Foundation under Grants #2003279, #1826967, #2100237, #2112167, #1911095, #2112665, and in part by PRISM and CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA.

REFERENCES

[1] 2023. Jetson TX2 Module. https://developer.nvidia.com/embedded/jetson-tx2. [Online].

[2] 2023. Raspberry Pi 4B. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. [Online].

[3] 2023. Raspberry Pi Zero 2 W. https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/. [Online].

[4] Aurore Avarguès-Weber et al. 2012. Simultaneous mastering of two abstract concepts by the miniature brain of bees. *Proceedings of the National Academy of Sciences* 109, 19 (2012), 7481–7486.

[5] Alan Baddeley. 1992. Working memory. *Science* 255, 5044 (1992), 556–559.

[6] Garcia Rafael Banos, Oresti and Alejandro Saez. 2014. MHEALTH Dataset. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5TW22.

[7] Bernd Bischl et al. 2023. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13, 2 (2023), e1484.

[8] Trenton Bricken et al. 2023. Sparse Distributed Memory is a Continual Learner. In *International Conference on Learning Representations.*

[9] Han Cai et al. 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems* 33 (2020), 11285–11297.

[10] Ning Chen et al. 2016. Smart urban surveillance using fog computing. In *2016 IEEE/ACM Symposium on Edge Computing (SEC).* IEEE, 95–96.

[11] Arpan Dutta et al. 2022. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022.* 281–286.

[12] Ehab Essa and Islam R Abdelmaksoud. 2023. Temporal-channel convolution with self-attention network for human activity recognition using wearable sensors. *Knowledge-Based Systems* 278 (2023), 110867.

[13] Divyam Madaan et al. 2022. Representational Continuity for Unsupervised Continual Learning. In *International Conference on Learning Representations.*

[14] Enrico Fini et al. 2022. Self-supervised models are continual learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[15] In Gim and JeongGil Ko. 2022. Memory-efficient DNN training on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services.* 464–476.

[16] Jean-Bastien Grill et al. 2020. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems* 33

(2020), 21271–21284.

[17] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

[18] Michael Hersche et al. 2022. Constrained few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9057–9067.

[19] Hioki. 2023. Hioki3334 Powermeter. https://www.hioki.com/en/products/detail/?product_key=5812.

[20] Andrew Howard et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1314–1324.

[21] Mohsen Imani et al. 2019. Hdcluster: An accurate clustering using brain-inspired high-dimensional computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1591–1594.

[22] Mohsen Imani et al. 2019. Semihd: Semi-supervised learning using hyperdimensional computing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[23] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. 2017. Voicehd: Hyperdimensional computing for efficient speech recognition. In *IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 1–8.

[24] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation* 1 (2009), 139–159.

[25] Behnam Khaleghi, Mohsen Imani, and Tajana Rosing. 2020. Prive-hd: Privacy-preserved hyperdimensional computing. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[26] Hyeji Kim, Muhammad Umar Karim Khan, and Chong-Min Kyung. 2019. Efficient neural network compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 12569–12577.

[27] Yeseong Kim, Mohsen Imani, and Tajana S Rosing. 2018. Efficient human activity recognition using hyperdimensional computing. In *Proceedings of the 8th International Conference on the Internet of Things*. 1–6.

[28] James Kirkpatrick et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* (2017).

[29] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[30] Young D Kwon et al. 2023. LifeLearner: Hardware-Aware Meta Continual Learning System for Embedded Computing Platforms. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*.

[31] Soochan Lee et al. 2020. A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning. In *International Conference on Learning Representations*.

[32] Ji Lin et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.

[33] Ji Lin et al. 2021. Memory-efficient patch-based inference for tiny deep learning. *Advances in Neural Information Processing Systems* 34 (2021), 2346–2358.

[34] Ji Lin et al. 2022. On-device training under 256kb memory. *Advances in Neural Information Processing Systems* 35 (2022), 22941–22954.

[35] David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems* 30 (2017).

[36] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[37] Md Mohaimenuzzaman et al. 2023. Environmental Sound Classification on the Edge: A Pipeline for Deep Acoustic Networks on Extremely Resource-Constrained Devices. *Pattern Recognition* 133 (2023), 109025.

[38] Ali Moin et al. 2021. A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics* 4, 1 (2021), 54–63.

[39] James O' Neill. 2020. An overview of neural network compression. *arXiv preprint arXiv:2006.03669* (2020).

[40] Andrew Ng, Michael Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 14 (2001).

[41] Evgeny Osipov et al. 2022. Hyperseed: Unsupervised learning with vector symbolic architectures. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[42] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks* 113 (2019), 54–71.

[43] Adam Paszke et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[44] Karol J Piczak. 2015. ESC: Dataset for environmental sound classification. In *Proceedings of the 23rd ACM international conference on Multimedia*. 1015–1018.

[45] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. 2022. MiniLearn: On-Device Learning for Low-Power IoT Devices. In *International Conference on Embedded Wireless Systems and Networks*.

[46] Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. 2019. Continual unsupervised representation learning. *Advances in neural information processing systems* 32 (2019).

[47] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[48] Olga Russakovsky et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115 (2015), 211–252.

[49] Andrei A Rusu et al. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[50] Swapnil Sayan Saha et al. 2023. TinyNS: Platform-Aware Neurosymbolic Auto Tiny Machine Learning. *ACM Transactions on Embedded Computing Systems* (2023).

[51] Mark Sandler et al. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[52] Yang Shen, Sanjoy Dasgupta, and Saket Navlakha. 2021. Algorithmic insights on continual learning from fruit flies. *arXiv preprint arXiv:2107.07617* (2021).

[53] Shun Shunhou and Yang Peng. 2022. AIoT on Cloud. In *Digital Transformation in Cloud Computing*. CRC Press, 629–732.

[54] James Smith et al. 2021. Unsupervised Progressive Learning and the STAM Architecture. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. 2979–2987.

[55] Ke Sun, Chen Chen, and Xinyu Zhang. 2020. " Alexa, stop spying on me!" speech privacy protection against voice assistants. In *Proceedings of the 18th conference on Embedded Networked Sensor Systems*. 298–311.

[56] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. 2021. A theoretical perspective on hyperdimensional computing. *Journal of Artificial Intelligence Research* 72 (2021), 215–249.

[57] Matteo Tiezzi et al. 2022. Stochastic Coherence Over Attention Trajectory For Continuous Learning In Video Streams. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. 3480–3486.

[58] Rishabh Tiwari et al. 2022. Gcr: Gradient coreset based replay buffer selection for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 99–108.

[59] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17 (2007), 395–416.

[60] Erwei Wang et al. 2019. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 1–39.

[61] Qipeng Wang et al. 2022. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 450–463.

[62] Gary M Weiss et al. 2016. Smartwatch-based activity recognition: A machine learning approach. In *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*. IEEE, 426–429.

[63] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning*. PMLR, 478–487.

[64] Daliang Xu et al. 2022. Mandheling: Mixed-precision on-device dnn training with dsp offloading. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 214–227.

[65] Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2023. FSL-HD: Accelerating Few-Shot Learning on ReRAM using Hyperdimensional Computing. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[66] Junting Zhang et al. 2020. Class-incremental learning via deep model consolidation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1131–1140.

[67] Yu Zhang, Tao Gu, and Xi Zhang. 2020. MDLdroidLite: A release-and-inhibit control approach to resource-efficient deep neural networks on mobile devices. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 463–475.