HDTorch: Accelerating Hyperdimensional Computing with GP-GPUs for Design Space Exploration

William Andrew Simon[*], Una Pale[*], Tomas Teijeiro[*†], David Atienza[*]
[*]*Embedded Systems Laboratory (ESL), Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland*
[†]*Department of Mathematics, University of the Basque Country (UPV/EHU), Spain*
{*william.simon, una.pale, david.atienza*}*@epfl.ch, tomas.teijeiro@ehu.eus*

Abstract—**The HyperDimensional Computing (HDC) Machine Learning (ML) paradigm is highly interesting for applications involving continuous, semi-supervised learning for long-term monitoring. However, its accuracy is not yet on par with other ML approaches, necessitating frameworks enabling fast HDC algorithm design space exploration. To this end, we introduce HDTorch, an open-source, PyTorch-based HDC library with CUDA extensions for hypervector operations. We demonstrate HDTorch's utility by analyzing four HDC benchmark datasets in terms of accuracy, runtime, and memory consumption, utilizing both classical and online HD training methodologies. We demonstrate average (training)/inference speedups of (111x/68x)/87x for classical/online HD, respectively. We also demonstrate how HDTorch enables exploration of HDC strategies applied to large, real-world datasets. We perform the first-ever HD training and inference analysis of the entirety of the CHB-MIT EEG epilepsy database. Results show that the typical approach of training on a subset of the data may not generalize to the entire dataset, an important factor when developing future HD models for medical wearable devices.**

Index Terms—**Hyper-Dimensional Computing, Machine Learning, PyTorch, GPUs, CUDA**

1. Introduction

Recently, HyperDimensional Computing (HDC) has emerged as an alternative Machine Learning (ML) framework to more traditional models such as random forests or neural networks, where its novel data representation strategy enables various advantages from both hardware and software perspectives. HD computing has been used in a broad spectrum of applications, such as robotics [1], recommendation systems [2], language recognition [3] and more. Due to the current trends of using Artificial Intelligence (AI) and ML for personalized medicine [4] and wearable devices for health monitoring [5], many HDC biomedical applications have been proposed, varying from emotion recognition [6] and electromyogram gesture recognition [7] to epileptic seizure detection via EEG signals [8], [9].

The highly parallel nature of HDC algorithms lends motivation to the development of specific HDC hardware accelerators. While such accelerators will certainly be implemented in future products that rely on HD computing, they are expensive and limited in algorithmic flexibility, a necessity for research into the HDC design space. Therefore, open-source, flexible GPU-accelerated HDC frameworks are necessary to enable efficient HDC research.

In this context, we propose HDTorch, the first open-source, PyTorch-based library built for exploring the HDC paradigm. HDTorch unlocks the full potential of PyTorch applied to HDC algorithms, and further extends PyTorch with custom, CUDA-backed hypervector operations. HDTorch is highly customizable, enabling modification to hyperparameters and encoding/similarity strategies. We validate HDTorch's accuracy and runtime performance on four reference HDC benchmarks, demonstrating accuracy comparable to state-of-the-art works while greatly accelerating training and inference for classical and online HD strategies.

We further motivate HDTorch's utility for exploring the HDC design space by applying it to the CHB-MIT epilepsy database. HDTorch enables us to perform the first ever HDC training on the entire dataset by reducing training and inference time by over 70x. We draw several conclusions from the analysis that will be useful for future works applying HD computing to large, unbalanced datasets.

This work's contributions are summarized as follows:

- We introduce HDTorch, an open-source, PyTorch-based HDC framework with CUDA extensions for hypervector operations, namely, bit-(un)packing and bit-array summation in the horizontal/vertical dimensions.
- We benchmark classical/online HD computing with HDTorch, showing accelerations of (111x/68x)/87x for (classical/online) training and inference, respectively.
- We explore the accuracy/runtime impact of varying hyperparameters, namely, hypervector width and online HD training batch size.
- We motivate HDTorch's design space exploration utility by performing, to the best of our knowledge, the first HDC training on the entire 980-hour, 7 million datapoint CHB-MIT epilepsy detection dataset. We present novel observations on the utility of HD computing on large, unbalanced datasets.

The remainder of the paper is organized as follows. Section 2 details related work on HD computing. Section 3 provides a typical HDC workflow as motivation for HDTorch. Section 4 presents the HDTorch framework. Section 5 details the benchmarks, metrics, and testing environment we use to validate HDTorch. Section 6 presents our experimental results, while Section 7 concludes this work.

This work has been partially supported by the ML-Edge Swiss National Science Foundation (NSF) Research project (GA No. 200020182009/1), and the PEDESITE Swiss NSF Sinergia project (GA No. SCRSII5 193813/1). T. Teijeiro is supported by a Maria Zambrano fellowship (MAZAM21/29) from the University of Basque Country and the Spanish Ministry of Universities, funded by the European Union-Next-GenerationEU.

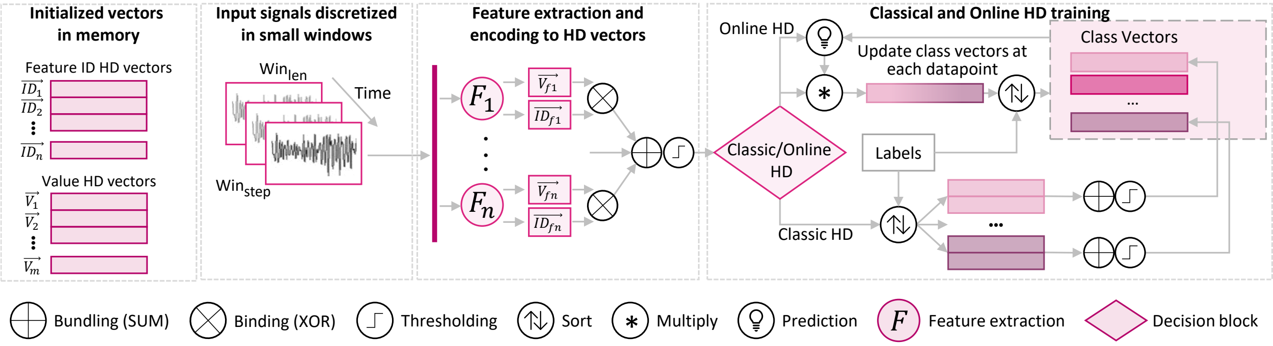


Figure 1. HD workflow for training classical and online HD models. Online training differs in that the class vectors are updated after every datapoint by multiplying its similarity to the target class by the vector before accumulating it into the class.

2. Related Work

2.1. Hyperdimensional Computing

HD computing is a machine learning strategy whose defining feature is its representation of datapoints as long ('hyper') vectors, which enables learning by 'accumulation' of said vectors belonging to the same class. HD computing relies on two conditions; first, any two randomly generated HD vectors are with high probability orthogonal, and second, a vector generated by vector accumulation will be more similar to its components than a vector not of its class [10]. HD computing has proven to be very appropriate for various forms of learning such as online [11], [12] on-device, semi-supervised [9], [13], and distributed learning [14]. Storing models in the form of hypervectors exhibits strong noise and data corruption resistance [15]. It also enables exploration of feature importance and selection [16].

From an algorithmic perspective, many HDC variations have been explored in literature, touching on almost every aspect of the HD training or inference flow. These include methods for initializing hypervectors [8], [17], [18], accumulating datapoints into class vectors [9], [13], [19]–[21], and calculating similarity between data and class vectors [10]. Exploring this design space necessitates analysis of the effects of various hyperparameter values such as hypervector lengths or online batch sizes, testing different encoding and learning strategies, and applying pre- or post-processing filtering to data or results. Unfortunately, due to the lack of publicly available libraries for fast processing and parallelization of HD computing on CPU or GPU, such analysis has thus far been performed on unoptimized HDC frameworks, greatly reducing research efficiency. For example, to the best of our knowledge, HD computing has not been tested on large datasets such as those for epilepsy detection. While previous works applied HD computing to epilepsy datasets, data subsets were always utilized [8], [18], [22]–[24]. Unfortunately, it has also been demonstrated that training on subsets of data may result in significant alterations to the final predictions in comparison to utilizing the entire dataset [9]. Thus, to develop suitable HDC algorithms, accelerating the exploration of the HDC design space is necessary.

2.2. Accelerating HD Computing

Several works have begun to explore strategies of accelerating HD computing via various software and hardware frameworks. Works mainly focus on 1) ASIC implementations [25]–[28] or 2) in-memory computing accelerators [29]–[31]. Other works explore GPU acceleration of HD computing. In [19], the authors implement an HDC architecture in PyTorch to compare against other ML algorithms on an embedded GPU, analyzing runtime, memory usage, and energy consumption. Similarly, the authors in [20] design a TensorFlow framework with HDC-specific extensions to accelerate the encoding and training process. Our proposal differs from the previous two in several ways. First, we utilize PyTorch as a base framework for HDTorch, as explained in Section 4. Second, as we want to encourage fast design space exploration in future works, we rely on native PyTorch operations where possible, and develop a set of custom CUDA functions where we identify HDC-specific bottlenecks that PyTorch cannot handle effectively. These functions are different from those accelerated in previous papers. Finally, we provide an open-source PyTorch library that encompasses these contributions.

3. Motivation: A Typical HD Workflow

To motivate the need for HDTorch, we present a typical HD workflow that may be applied to a dataset by a researcher. The design choices presented here are commonly used in previous literature, and are accelerated by HDTorch, as described in Section 4.2.

The first step when performing training or inference via HD computing is to encode raw input data into the HD space as hypervectors. One popular encoding method is that of ID-Level encoding [20], where each feature has an ID vector representing it, while all datapoints are discretized into a fixed number of bins, with each bin having its own representative HD Value vector. ID vectors ($\vec{ID}$) are randomly generated, while Value vectors ($\vec{V}$) may be randomly generated or, as in [17], generated using a linear scaling method so that vectors representing similar values are also similar. Using the predefined $\vec{ID}$ and $\vec{V}$ vectors, each data point is encoded to a hypervector $\vec{H}_i$ by binding each feature vector $I\vec{D}_{fi}$ with the bin value vector $\vec{V}_{fi}$ corresponding to the value of the feature. Encoded features are then accumulated as formulated in Equation 1. The final summed values are normalized using majority voting to regenerate the binary vector.

$$\vec{H}_i = \left\lfloor \sum_{f_i} I\vec{D}_{fi} \oplus \vec{V}_{fi} \right\rceil \tag{1}$$

Once encoded, the data can be processed in different ways. The most basic approach is so-called classical HD training, which utilizes single-pass accumulation of the encoded vectors belonging to the same class into a class vector. On the other hand, recent literature has proposed various improvements to classical HD training, such as iterative learning [13], 'multi-centroid learning' which utilizes more than one vector per class [9], and progressive, online HD learning [19]. In [24] the authors compared all these approaches on the use case of epileptic seizure detection and demonstrated that online training is potentially the most interesting, given its high performance, the fact that it utilizes each data point only once as opposed to iterative training, and its lower memory requirements than, for example, a multi-centroid approach. Online training improves accuracy by, instead of treating all data points as equally important, multiplying new data points by their similarity to current class vectors before being accumulated, as illustrated in Equations 2 and 3:

$$\vec{M}'_C \leftarrow \vec{M}_C + (\delta_C)\vec{H} \tag{2}$$

$$\vec{M}'_W \leftarrow \vec{M}_W - \gamma(1-\delta_W)\vec{H} \tag{3}$$

where $\vec{M}_X$ is either the correctly or incorrectly classified class vector, δ_X is the distance from the class vector (lower distance means more similarity), and γ is the learning rate. In this way, highly common class patterns are not allowed to saturate the class vectors, thus improving sensitivity to less common patterns. Online HD training is also applicable in continuously learning wearable devices, as it is capable of integrating new data in real time. The workflow is described visually in Figure 1.

Once class vectors have been learned from either the entirety of the training set in a single pass or over time via online training, test data can be compared to the class vectors to find the most similar vector, thus returning a predicted class. A variety of similarity metrics have been proposed, with the two most popular being Hamming distance and cosine similarity.

Then, let us consider a typical HD workflow that integrates the aforementioned steps. We will utilize as an example, subject 1 of the CHB-MIT epilepsy database consisting of 40 hours of data, split in 1 hour segments as in the original database, further described in Section 5.1.2. We use the HDC framework from [23], with ID-Level Encoding and Hamming distance similarity measurement. We perform both classical and online training using the common leave-one-out cross-validation strategy.

The results of CPU training and inference are as follows. Classical training takes 46 minutes, 22 of which are consumed by the encoding step. Online training takes 134 minutes, 28 of which are consumed by encoding. If extrapolated to the full 980 hours of data, with a time resolution of 0.5 seconds, resulting in more than 7 million samples included in the CHB-MIT database, classical/online training would take 19/54 hours, respectively.

It is no wonder, then, that previous works have analyzed only subsets of the dataset, as performing any sort of design space exploration is infeasible with such high runtimes. Thus, a GPU-accelerated, flexible HD computing framework is necessary to enable future fast, iterative research into the HDC design space.

TABLE 1. HDTorch Feature Overview

Features	Values
Customizable Hyperparameters	Hypervector Dimension Batch Size
Hypervector Flavors	Binary (0,1) Bipolar (-1,1)
Hypervector Generation Strategies	Random Scale Random [17], [18] Sandwich [23]
Available Binding Strategies	ID-Level Encoding [7] Feature Permutation [21] Feature Appending [16]
Available Similarity Metrics	Hamming [10] Cosine [10]
HD Computing CUDA Extensions	Binary (Un)Packing Horizontal/Vertical Summation

4. HDTorch

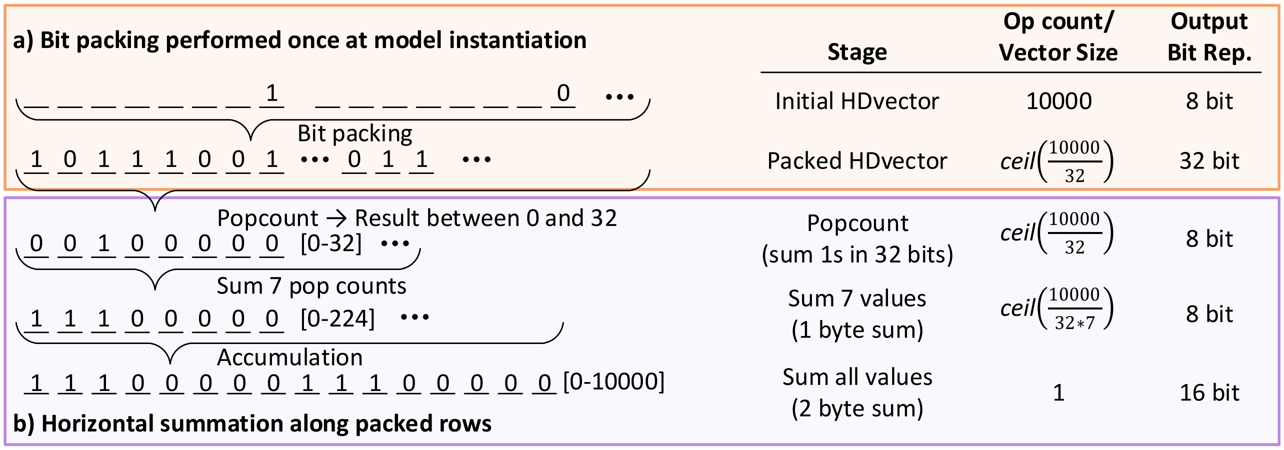
With this motivation in mind, we describe how HDTorch provides a flexible framework for performing HDC research on GPU-equipped platforms.

4.1. Implementing HD Computing in PyTorch

PyTorch is one of the two most popular Deep Learning (DL) frameworks utilized in research today, along with TensorFlow. While the debate on framework superiority is contentious, there can be no doubt that PyTorch is currently the most popular DL framework in research, being utilized in over 75% of research papers using either of the two frameworks [32]. As such, providing HDC support in PyTorch will enable the majority of the research community to explore HDC solutions on a wider range of topics more easily.

HDTorch is realized as a python library installable from PyPi[1]. Table 1 lists the features provided by HDTorch's base HD model class. Namely, it supports variable size binary or bipolar hyperdimensional vectors for any number

[1] https://pypi.org/project/hdtorch/



Stage	Op count/ Vector Size	Output Bit Rep.
Initial HDvector	10000	8 bit
Packed HDvector	$ceil\left(\frac{10000}{32}\right)$	32 bit
Popcount (sum 1s in 32 bits)	$ceil\left(\frac{10000}{32}\right)$	8 bit
Sum 7 values (1 byte sum)	$ceil\left(\frac{10000}{32*7}\right)$	8 bit
Sum all values (2 byte sum)	1	16 bit

Figure 2. Illustration of HDTorch's bit packing and horizontal summation operations. a) Bit packing reduces hypervector memory footprint and improves bitwise operation efficiency, and b) summation enables packed hypervectors to be accumulated efficiently.

of classes. Value and feature hypervectors may be generated randomly, or by a number of pseudo-random functionalities proposed previously in the literature [8], [17], [18]. A variety of feature/class binding methods are supported, including ID-Level encoding [20], vector permutation [21], and feature appending [16]. Pairwise similarity calculation via either hamming distance or cosine similarity [10] is supported.

4.2. HDTorch: HyperDimensional Extensions for PyTorch

While PyTorch's GPU optimization provides already excellent acceleration, its functions are not implemented with HD computing in mind and thus fail to fully optimize the HDC kernel. Namely, HD computing relies on operations between hypervectors containing 1000's of binary values. By default, PyTorch stores binary values as bytes, resulting in up to an 8x memory overhead per bit. Besides memory access, inefficiencies are also found in three key hypervector computations: namely, 1) the performance of binary `xor` operations between hypervectors during both training and inference, 2) the summation of ID-Level encoded vectors for all features, and 3) the summation along the class hypervector during hamming distance calculation. Encoding is usually the main bottleneck of HDC applications, taking up to 70% of training time [20], and is highly dependent on these inefficient operations. With this in mind, HDTorch provides four new functions with backing CUDA C++ code to address Python's shortcomings in regards to HD computing. These extensions are only applicable to binary hypervectors; bipolar hypervectors may still benefit from HDTorch's non-specific accelerations, as described in Section 6.

4.2.1. Bit (Un)Packing to Improve Memory and Bitwise Operation Efficiency. The first optimization HDTorch supports is the packing of a hypervector's bits into byte blocks. This reduces hypervector memory footprint by 8x and enables bitwise operations, specifically the bitwise `xor` necessary for ID-Value encoding and Hamming distance calculation. Individual bits are packed into 32-bit integers, enabling HDTorch to take advantage of CUDA's bit counting intrinsics as described below. A complementary unpacking operation is also supported to return packed hypervectors to their original state if necessary. Both operations are backed by CUDA code for GPU acceleration.

4.2.2. Horizontal/Vertical Summation for Highly SIMD Bitwise Operations. While PyTorch natively supports bitwise operations on packed bits, summation operations that can take advantage of the packed and binary nature of hypervectors are absent. Therefore, operations for performing horizontal and vertical summations are introduced to enable fast accumulations of binary vectors.

In the case of horizontal summation, CUDA's `popcount` intrinsic is utilized to count the number of bits set to 1 in a 32-bit integer. As it is known beforehand that only values of 0 or 1 are being accumulated, 8-bit summations can be used during accumulation, with intermediate accumulation of the 8-bit accumulators into the final output summation vector every seven additions to avoid overflow.

Figure 2 illustrates the bit-packing and horizontal summation HDTorch operations. Vertical summation is accomplished by transposing the input bit-array before performing horizontal summation on the intermediate array. The input array is tiled into subblocks of 128x128 bits, with each subarray assigned to a CUDA warp, which is in charge of transposing the tile. Once all warps have completed their transpositions, the tiles are transposed as they are written back to main memory.

5. Experimental Setup

To evaluate HDTorch's utility in terms of exploring the HDC design space, we perform a wide range of evaluations in terms of HDC training and inference strategies, datasets, and hyperparameter variations.

TABLE 2. HDTorch benchmark Datasets (FC: feature count, CC: class count, DS: dataset size in number of samples)

Dataset	Description	FC	CC	DS $[10^3]$
PAMAP	Activity recognition (IMU + HR)	31	5	∼80
UCIHAR	Activity recognition (Smartphone)	561	12	∼10
ISOLET	Voice recognition	617	26	∼8
MNIST	Handwritten digit recognition	784	10	∼70
CHB-MIT	Epilepsy detection	342	2	∼7056

5.1. Datasets

We draw results from experiments performed on five datasets covering a range of sizes, complexities, and use cases. Four datasets are standard HDC benchmarks, while the 5th is a large, highly unbalanced medical dataset, providing a more demanding scenario than the first four datasets.

5.1.1. Reference HD Computing Benchmarks. To compare HDTorch performance with HDC implementations available in the literature, we use four benchmark datasets from the online-available UCI repository [33]: 1) ISOLET is an audio dataset containing spoken letters of the English alphabet, 2) MNIST is an image dataset consisting of written digits, 3) UCIHAR is a dataset for classifying human activity from smartphone inertial sensors and, 4) PAMAP is a physical activity dataset containing both inertial sensors and a heart rate monitor. We chose these datasets as they are utilized in previous works demonstrating HD computing on GPUs [19], [20], with a wide range of Feature Counts (FC), Class Counts (CC) and Dataset Sizes (DS), as listed in Table 2.

5.1.2. Epilepsy Benchmark Use-Case. Beyond demonstrating HDTorch's utility on standard HD benchmarks, we wish to demonstrate its ability to enable analysis on large, computationally challenging datasets typical of real-world scenarios such as the ones for continuous monitoring of biomedical data. These datasets contain hundreds of hours of data and are usually highly imbalanced. For this reason, typically only subsets of the entire dataset are utilized for analysis, and it is possible that the results of such studies do not represent the actual performance of the algorithms in the final application (e.g. on a wearable device for long-term detection and monitoring). Thus, we test the ability of HDTorch to explore HD algorithms on the CHB-MIT epilepsy dataset, a widely used open source dataset for epilepsy detection [9], [34]–[36].

CHB-MIT is an EEG database collected by the Children's Hospital of Boston and MIT. It contains 980 hours of data recorded at 256Hz, consisting of 183 seizures from 24 subjects with medically-resistant seizures ranging in age from 1.5 to 22 years [34], [37]. On average, it has 7.6 ± 5.8 seizures per subject, and between 23 and 26 channels, of which the 18 channels that are common to all patients are utilized in this work.

We extract 19 features from each of the 18 channels, similar to [16], calculating them on 4 second windows with a moving step of 0.5 seconds. We organize the dataset in two manners before analysis: 1) Subsets of data for each patient that contain all seizure data and 10x more randomly selected non-seizure data (*Fact10*), and 2) the entirety of the dataset divided into approximately one-hour-long segments (*1HSeg*). Previous literature has demonstrated that utilizing different seizure to non-seizure ratios in dataset sub-selections can lead to highly overestimated performance [9]. Thus, in this work, we perform the first (to the best of our knowledge) assessment of HDC performance on the entire database, comparing it to the *Fact10* subset. Evaluation is performed in a time-series split cross-validation (TSCV) approach [38], where only previously acquired data can be used for training, as opposed to, for example, typical leave-one-out cross-validation. More specifically, we perform a cross-validation for each hour-long segment, with segment n as the test set and previous segments 0 to $n-1$ as the training set. This approach also reduces the runtime by approximately half, as it trains on less data in all cross-validations but the last one.

5.2. Evaluation Metrics

To confirm framework correctness, we evaluate performance in terms of accuracy, memory consumption, and training and inference speedup on the four datasets described in Section 5.1.1. We perform these analyses while varying model hyperparameters, specifically, the hypervector dimension D and the online training batch size, or the frequency with which class vectors are updated as a function of the arrival of new training data.

Concerning the epilepsy dataset use-case, we analyze acceleration due to the utilization of HDTorch for encoding, training, and inference for classical and online HD strategies. With this realized speedup, we are able to evaluate performance of classical and online HDC on the entirety of the CHB-MIT database. Thus, we compare episode detection accuracy for the two previously described dataset selections, *Fact10* and *1HSeg*. Performance is evaluated by concatenating predictions of all cross-validations. We perform moving average smoothing of the predicted labels with a window size of 5s, measure sensitivity (TPR), precision (PPV) and F1 score for both selections, and discuss the differences in performance.

5.3. Benchmarking Environment

Benchmarking and analysis are performed on a server system equipped with a 2-socket, 40-core Intel Xeon Gold 6242R processor capable of frequencies up to 4.1GHz and an NVIDIA Tesla V100 GPU. The software environment consists of Python v3.9.10, PyTorch v1.10.2, and the CUDA driver v11.6. Profiling is accomplished via PyTorch's native profiler, capable of profiling CPU and GPU runtime/memory consumption by function.

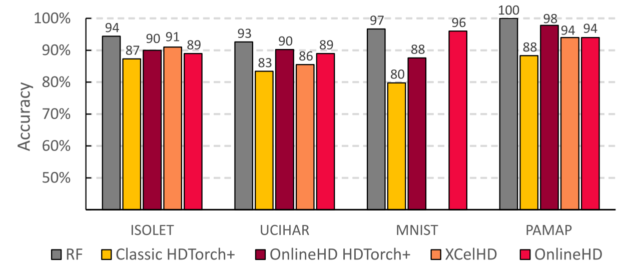


Figure 3. Performances of different HDC implementations, with Random Forest (RF) for reference, on 4 benchmark datasets.

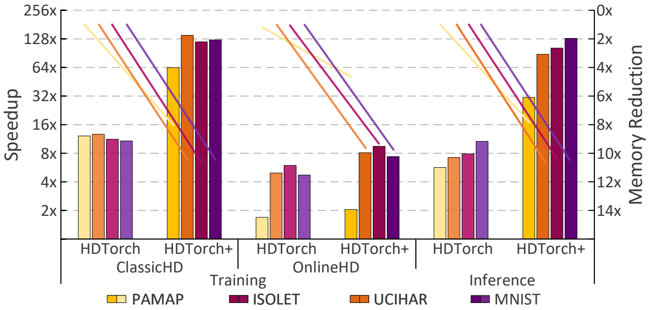


Figure 4. Speedup comparison between different HD learning implementations. 4 different benchmark datasets are used.

6. Results

The following sections detail the results of our experiments. For clarity, results utilizing hypervector CUDA extensions are marked as HDTorch+ in the following figures.

6.1. Model Accuracy Analysis

Figure 3 shows the accuracy of online and classical HD model implementations with respect to random forest performance. We also compare the online HD approach with two implementations found in the literature. *XCelHD* [20] uses a modified TensorFlow implementation of the online HD workflow described in Section 3, and *OnlineHD* [19] uses an original HD floating point model with a non-standard encoding approach implemented in PyTorch.

Online training improves performance in comparison to classical HD for all datasets. Furthermore, our online HD implementation using HDTorch is similar in accuracy to the implementations found the literature. It should be noted that random forest outperforms all HD computing implementations, indicating the necessity for further optimization of HD computing to reach state-of-the-art accuracy results. This situation further motivates the need for efficient design space exploration of HDC algorithms.

6.2. Time Performance Analysis

Figure 4 illustrates runtime accelerations achieved by HDTorch/HDTorch+ in comparison to HDTorch run on the CPU. Acceleration for training on both classical and online HD, and acceleration for inference (equivalent for both approaches) is illustrated. Comparison with XCelHD [20] speedup was not possible as the code is not publicly available. As can be seen, both HDTorch and HDTorch+ greatly reduce benchmark runtime: HDTorch provides up to (12.7x/6x)/10.7x (training)/inference speedup for classical/online runtime, respectively, while HDTorch+ improves these gains to (139x/9.5x)/130x. Note that online HD speedup is significantly lower as a result of the necessity to recalculate the class vectors after every datapoint, a challenge addressed in Section 6.4.2. Finally, the observed differences in speedup for different datasets is due to the different dataset feature counts; for example, PAMAP has the smallest number of features and thus the smallest possible speedup due to more limited parallelization opportunities.

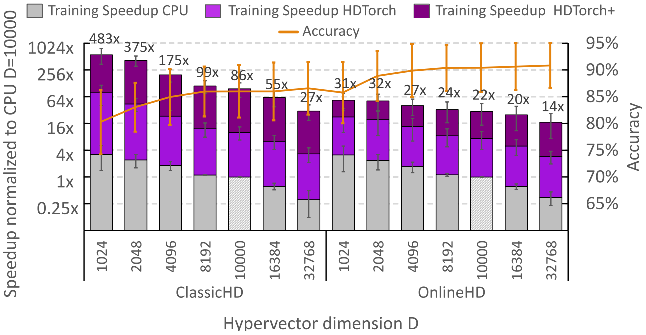


Figure 5. Average speedup for HDTorch-CPU, HDTorch and HDTorch+ for different dimensions of HD vectors on 4 benchmark datasets. Values are normalized to the runtime of the HDTorch model running on CPU with the typical D=10000.

6.3. Memory Consumption Analysis

Figure 4 also illustrates trends in memory consumption across the datasets. Memory values are normalized to the memory usage of running training and inference on the CPU. It can be seen that utilizing HDTorch without HDTorch+ extensions does not reduce memory usage, as the tensors used to store hypervectors are identically sized on either the CPU or GPU. Introducing HDTorch+ extensions, on the other hand, reduces memory consumption by approximately 10x for both training and inference in the aforementioned benchmark datasets. This is due to reducing the memory footprint for hypervectors by 8x by bit packing, and utilizing the more memory efficient HDTorch+ bit-array summation functions, which instantiate fewer intermediate tensors during computation. Once again, PAMAP is an outlier on memory consumption reduction as it has an order of magnitude fewer samples compared to the other datasets.

6.4. Parameters Influence

We also evaluate the impact of hyperparameter variance on accuracy and runtime for the four benchmarks. We find a trade-off between model complexity/size, accuracy, and runtime. Measured runtime and accuracy values are averaged across the four datasets, and error bars indicate the standard deviation of the averaged values.

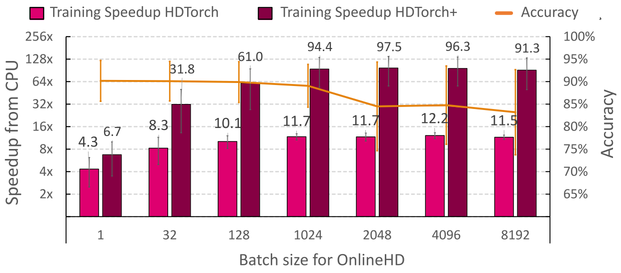


Figure 6. Acceleration with respect to HDTorch-CPU, compared for HDTorch and HDTorch+ implementations. Comparison is given for training using different batch sizes.

6.4.1. Hypervector Dimension. Figure 5 illustrates the impact on model training time when varying the width D of the class and feature hypervectors between 1024 to 32768 bits. The runtime values are normalized to the runtime of the HDTorch model running on CPU with the typical D=10000. As can be seen, reducing D drastically reduces runtime, up to 483x/31x for classical/online learning with HDTorch+, at the cost of an average 6% decrease in accuracy. On the other hand, increasing D past 10000 provides little improvement to accuracy. However, our results show that even a model with a D=32768 runs 27x/14x faster over a CPU model of D=10000 for classical/online HD, indicating that if a user would like to test a higher dimensional model on their dataset, HDTorch makes this feasible in a reasonable amount of time.

6.4.2. Batch Size. Online HD benefits less from HDTorch acceleration due to the fact that online HD is trained sequentially, with one data sample accumulated into the class vector at a time, and thus cannot be highly parallelized. There is, however, the possibility to improve runtime by only updating class vectors after a batch size of n new datapoints are received. Figure 6 illustrates batching effects on model runtime and accuracy. It can be seen that batching data significantly decreases training time, especially for HDTorch+, with an average of 68x performance gain achieved across all batch sizes and datasets. This comes at a cost of an average accuracy drop of 7% at a batch size of 8192. This drop will vary according to dataset complexity; hence, batch size should be tuned to the particular use case of the model.

6.5. Epilepsy Detection Use Case

Finally, we test classical HD and online HD implementations on a real-life dataset for epilepsy detection. Figure 7 illustrates HDTorch and HDTorch+ speedup with respect to CPU-HDTorch for classical/online HD. Accelerations are shown for the encoding, training, and inference stages. For classical HD, HDTorch achieves a 7x speedup for all stages, with HDTorch+ providing an additional 10x gain for each stage, for total speedups of 79x, 78x, and 70x for encoding, training, and inference, respectively. For online HD, performance gains are similar for encoding and inference, whereas for training, gains are significantly reduced, to 2.1x and 2.7x for HDTorch and HDTorch+. This is due to the parallelization constraints described in Section 6.1.

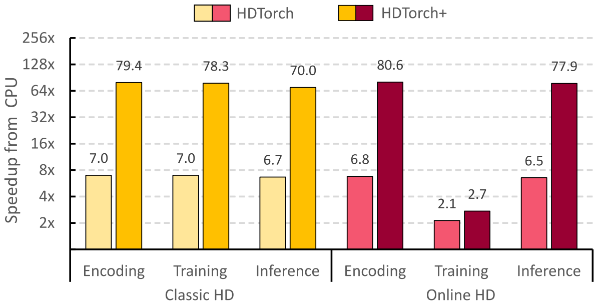


Figure 7. Acceleration in respect to HDTorch-CPU, compared for HDTorch and HDTorch+ implementations, for encoding stage, training and interference on CHB-MIT dataset.

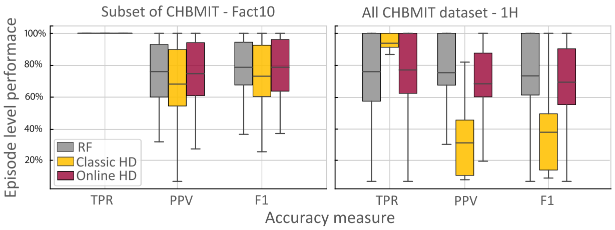


Figure 8. Epilepsy detection performance comparing training on a subset of data (typical in previous literature) against using the whole CHB-MIT dataset. Boxplots represent performance distribution for all 24 subjects, with mean performance marked as a horizontal line.

The speedup achieved by using the HDTorch library enables us to evaluate HDC performance on the entirety of the CHB-MIT database. Evaluation of HDC performance on such a big database enables better insights into performance of HDC algorithms for real-life applications on wearable devices. Figure 8 illustrates the detection of epileptic episodes for the two dataset organizations described in Section 5.1.2, namely, *Fact10* and *1HSeg*. For *Fact10*, there is no significant difference between performance of random forest, classical HD and online HD, with all of them detecting almost all seizures with some amount of false positives. On the other hand, training and predicting on the *1HSeg* dataset organization shows a significant decrease in performance for classical HD. While it still detects almost all seizures, it also has significantly more false positives, thus reducing the F1 score. While random forest and online HD miss some seizures, they also avoid such a significant drop in precision and F1 score. Even so, random forest slightly outperforms online HD performance.

These results indicate two key takeaways. First, analyzing a subset of data may not generalize to performance on the entirety of a dataset, especially when the dataset is large and highly unbalanced, as is the case of the CHB-MIT database. Second, new training strategies are constantly improving HD computing's accuracy, bringing it closer to that of other standard ML models, such as random forest. Both takeaways motivate the necessity for HDTorch, which enables new HDC strategies to be explored on realistic datasets, paving the way for future breakthroughs in the field.

7. Conclusion

In order to bring HDC performance in line with state-of-the-art ML algorithms and position it as an algorithm for continuous online monitoring for wearables in healthcare, algorithm optimization and design space exploration are necessary. Thus, in this work we have presented HDTorch, the first open-source, PyTorch-based library for HD computing with CUDA extensions for hypervector operations.

On four HDC benchmark datasets, we demonstrated an average 111x/68x training speedup for classical/online HD, respectively, and an average 87x speedup for inference. In addition, we have shown that HDTorch's CUDA extensions for HDC operations reduce training/inference memory consumption by up to 10x. HDTorch's utility and flexibility have been demonstrated by analyzing the effects of hypervector dimension and batch size on model accuracy and runtime.

Finally, the speedup achieved by using the HDTorch library enables us to evaluate HDC performance on the large, highly unbalanced CHB-MIT epilepsy dataset, where we demonstrate that the performance resulting from typical approach of training on a subset of the data does not necessarily generalize to training on the entire dataset. This important observation must be carefully considered when developing future HD models for medical wearable devices.

References

[1] A. Mitrokhin *et al.*, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, 2019.

[2] Y. Guo *et al.*, "Hyperrec: Efficient recommender systems with hyperdimensional computing," *ASPDAC*, 2021.

[3] G. Karunaratne *et al.*, "Real-time language recognition using hyperdimensional computing on phase-change memory array," *AICAS*, 2021.

[4] K.-H. Yu *et al.*, "Artificial intelligence in healthcare," *Nature Biomedical Engineering*, 2018.

[5] A. V. L. N. Sujith *et al.*, "Systematic review of smart health monitoring using deep learning and artificial intelligence," *Neuroscience Informatics*, 2022.

[6] E.-J. Chang *et al.*, "Hyperdimensional computing-based multimodality emotion recognition with physiological signals," *AICAS*, 2019.

[7] A. Rahimi *et al.*, "Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition," *ICRC*, 2016.

[8] A. Burrello *et al.*, "An ensemble of hyperdimensional classifiers: Hardware-friendly short-latency seizure detection with automatic ieeg electrode selection," *J-BHI*, 2021.

[9] U. Pale *et al.*, "Multi-centroid hyperdimensional computing approach for epileptic seizure detection," *Frontiers in Neurology*, 2022.

[10] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, 2009.

[11] A. Moin *et al.*, "A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition," *Nature Electronics*, 2021.

[12] S. Benatti *et al.*, "Online learning and classification of emg-based gestures on a parallel ultra-low power platform using hyperdimensional computing," *TBioCAS*, 2019.

[13] M. Imani *et al.*, "Semihd: Semi-supervised learning using hyperdimensional computing," *ICCAD*, 2019.

[14] M. Imani *et al.*, "A framework for collaborative learning in secure high-dimensional space," *CLOUD*, 2019.

[15] M. Imani *et al.*, "Exploring hyperdimensional associative memory," *HPCA*, 2017.

[16] U. Pale *et al.*, "Hyperdimensional computing encoding for feature selection on the use case of epileptic seizure detection," *arXiv:2205.07654*, 2022.

[17] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," *ICRC*,

[18] U. Pale *et al.*, "Systematic assessment of hyperdimensional computing for epileptic seizure detection," *EMBC*, 2021.

[19] A. Hernandez-Cane *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," *DATE*, 2021.

[20] J. Kang *et al.*, "Xcelhd: An efficient gpu-powered hyperdimensional computing with parallelized training," *ASP-DAC*, 2022.

[21] A. Rahimi *et al.*, "Hyperdimensional computing for blind and one-shot classification of eeg error-related potentials," *Mobile Networks and Applications*, 2020.

[22] F. Asgarinejad *et al.*, "Detection of epileptic seizures from surface eeg using hyperdimensional computing," *EMBC*, 2020.

[23] A. Burrello *et al.*, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," *DATE*, 2019.

[24] U. Pale *et al.*, "Exploration of hyperdimensional computing strategies for enhanced learning on epileptic seizure detection," *arXiv:2201.09759*, 2022.

[25] M. Imani *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," *HPCA*, 2021.

[26] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," *DAC*, 2018.

[27] T. F. Wu *et al.*, "Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: Hyperdimensional computing case study," *ISSCC*, 2018.

[28] A. Rahimi *et al.*, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," *ISLPED*, 2016.

[29] S. Gupta *et al.*, "Felix: Fast and energy-efficient logic in memory," *ICCAD*, 2018.

[30] G. Karunaratne *et al.*, "In-memory hyperdimensional computing," *Nature Electronics*, 2020.

[31] G. Karunaratne *et al.*, "Energy efficient in-memory hyperdimensional encoding for spatio-temporal signal processing," *TCAS-II: Express Briefs*, 2021.

[32] H. He, "The state of machine learning frameworks in 2019," *The Gradient*, 2019.

[33] D. Dua and C. Graff, *UCI machine learning repository*, 2017.

[34] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," Thesis, Massachusetts Institute of Technology, 2009.

[35] D. Sopic *et al.*, "E-glass: A wearable system for real-time detection of epileptic seizures," *AISCAS*, 2018.

[36] R. Zanetti *et al.*, "Robust epileptic seizure detection on wearable systems with reduced false-alarm rate," *EMBC*, 2020.

[37] Goldberger Ary L. *et al.*, "Physiobank physiotoolkit and physionet," *Circulation*, 2000.

[38] A. Mohammed *et al.*, "Time-series cross-validation parallel programming using mpi," *ICDABI*, 2021.