AeneasHDC: An Automatic Framework for Deploying Hyperdimensional Computing Models on FPGAs

Marco Angioli, Saeid Jamili, Marcello Barbirotta, Abdallah Cheikh,
Antonio Mastrandrea, Francesco Menichelli, Antonello Rosato, Mauro Olivieri
Dept. of Information Engineering, Electronics and Telecommunications, Sapienza University of Rome, Italy,
Email: {name.surname}@uniroma1.it

***Abstract*—Hyperdimensional Computing (HDC) is a bio-inspired learning paradigm, that models neural pattern activities using high-dimensional distributed representations. HDC leverages parallel and simple vector arithmetic operations to combine and compare different concepts, enabling cognitive and reasoning tasks. The computational efficiency and parallelism of this approach make it particularly suited for hardware implementations, especially as a lightweight, energy-efficient solution for performing learning tasks on resource-constrained edge devices. The HDC pipeline, including encoding, training, and comparison stages, has been extensively explored with various approaches in the literature. However, while these techniques are mainly oriented to improve the model accuracy, their influence on hardware parameters remains largely unexplored. This work presents AeneasHDC, an automatic and open-source platform for the streamlined deployment of HDC models in both software and hardware for classification, regression and clustering tasks. AeneasHDC supports an extensive range of techniques commonly adopted in literature, automates the design of flexible hardware accelerators for HDC, and empowers users to easily assess the impact of different design choices on model accuracy, memory usage, execution time, power consumption, and area requirements.**

***Index Terms*—Hyperdimensional Computing, Automatic Hardware Acceleration, FPGA, Classification, Clustering, Regression, Edge AI**

I. Introduction

Hyperdimensional Computing (HDC) or Vector Symbolic Architecture (VSA) mimics the cognitive and reasoning processes of the human brain by employing high-dimensional distributed representations called hypervectors (HVs). HDC relies on the mathematical properties of high-dimensional spaces and leverages a well-defined set of simple vector arithmetic operations, to combine and compare HVs, enabling logical and reasoning tasks. In recent years, HDC has been applied across diverse domains, such as robotics [1], recommendation systems [2], healthcare [3]–[6], natural language processing [7], [8], DNA sequencing [9], and speech recognition [10], for modelling and solving classification [11], clustering [12], [13] and regression [14] problems. In these applications, HDC has demonstrated performance comparable to traditional Artificial Intelligence (AI) approaches while exhibiting several important additional properties, including computational efficiency, scalability, one-shot learning, resilience against noise and hardware faults, human-interpretable models, energy efficiency, and extreme parallelism [15]–[17]. These characteristics position HDC as an ideal choice for addressing cognitive tasks in resource-constrained contexts, such as IoT applications on the edge [15], [18] and align perfectly with the capabilities of Field-Programmable Gate Arrays (FPGAs), presenting significant opportunities for hardware acceleration [16].

In HDC, learning processes are more straightforward than in traditional AI, as they involve specific operations on HVs that are encoded from data and then combined, aggregated, and compared according to the specific problem. The literature offers various design choices that can be adopted to enhance model accuracy or reduce its complexity. These configurations encompass aspects of the high-dimensional space characteristics such as size, density, data type of the elements, and the adopted similarity measure, as well as the techniques used to quantize scalar values, encode feature vectors and options for clipping and retraining the model.

However, while the impact of these configurations on accuracy has been extensively studied, in the existing literature, there is a notable gap concerning how different choices impact power consumption, area occupation, and execution time when translated into hardware. As a result, hardware implementations frequently rely on ad-hoc decisions inspired by state-of-the-art approaches without a proper justification or consideration of the specific problem and the available resources, possibly resulting in heavy suboptimal solutions.

In this work, we present AeneasHDC, an automatic, open-source platform that enables easy and fast deployment of HDC models in software and hardware. The framework offers extensive customization options for the model, covers various learning tasks, such as classification, clustering, and regression and adapts to different hardware requirements, resulting suitable for both high-power and limited-resource systems. This is facilitated through a graphical user interface and a fully automated process that abstracts the complexity of custom hardware designs. In this way, AeneasHDC is the first automatic framework for HDC in the literature that allows users to explore and assess the impact of different design choices on accuracy and hardware parameters.

The rest of the work is organized as follows: Section II introduces the basics of HDC, describes the main design choices that can be adopted for customizing the models for learning tasks, reviews the previous works in the literature and highlights the need for a flexible automatic hardware generation environment. Section III presents the AeneasHDC platform, describing the entire automatic workflow in detail and discussing the supported flexibility in the software and hardware models. In Section IV, we validate the proposed environment conducting a series of tests varying the problem characteristics, the required task and the model configuration and we discuss the results in terms of perfomance and hardware requirements. Finally, Section V summarizes the main outcomes of the work.

II. Background and Related Works

In the intricate workings of the human brain, information is linked with patterns of neural activity distributed across an extensive network of neurons, which activations form the basis of our cognition and reasoning processes. HDC diverges from conventional AI approaches, which rely on scalar numbers, and mimics the brain's behaviour using high-dimensional holographic representations, where the information is distributed across the elements.

The HV *size* (D) determines the number of independent elements that can be stored in the hyperspace, also called capacity, following an exponential relationship [17]. Often, D is set to a high value, such as thousands of dimensions (e.g., $D = 10,000$), but in resource-constrained contexts, selecting the appropriate size is crucial, since it directly impacts computational complexity, memory usage, area and energy consumption.

The elements of the HVs can belong to various *data types*, including binary {0,1}, bipolar {-1,1}, ternary {-1,0,1}, quantized, integers or real [18]. The wider the range of data, the better the model's performance, but this improvement comes at the cost of increased complexity.

Another important parameter is the *density*. In a dense HV, all the elements have an equal distribution probability, while in a sparse model, the zero value predominates, with the occurrences of other values comparatively infrequent. In [19], the impact of different densities has been investigated on the model's computational complexity and accuracy.

In an HDC problem, fundamental concepts, elements, or symbols are mapped to random HVs that, thanks to the mathematical properties of the hyperspace, result orthogonal and thus, non-similar. The HVs are then manipulated and combined using three vector operations, Bundling, Binding and Permutation, which form the core computational structure of HDC models. Bundling ($\oplus$) combines two HVs into a single one, which results similar to both inputs; it is associative and commutative. Binding ($\otimes$) associates two different concepts, resulting in a non-similar HV; this operation is associative, commutative, self-invertible, and can distribute over bundling. Permutation (ρ) reorders the elements of the HV, creating a dissimilar one; it's invertible and can distribute over both binding and bundling. The specific implementation of these three operations depends on the characteristics of the hyperspace [20]. Lastly, a similarity measurement assesses the distance and angle between HVs in the high-dimensional space. Common methods for this include, but are not limited to, Hamming Distance (for binary HVs), Dot Product, and Cosine Similarity.

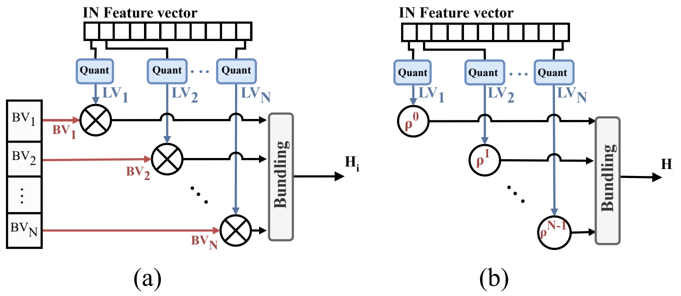


Fig. 1: Visualization of encoding techniques for feature vectors: (a) Record-based (b) N-gram based

The rest of this Section is devoted to outline the basic theoretical concepts behind the learning tasks in HDC (classification, regression and clustering), both in the training and in the inference stages. Additionally, it provides insights into the primary design choices available for customizing the HDC model and analyzes previous related works in the literature, highlighting the urgent need for an automatic environment.

A. Encoding

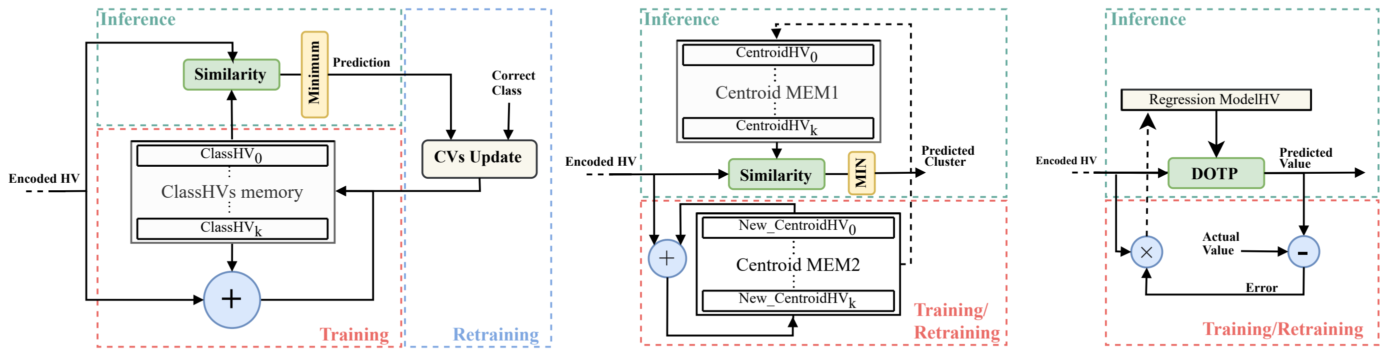
The first fundamental step always performed in an HDC problem, both in training and in inference, is the encoding or mapping of the input data to HVs. Encoding can be implemented using different approaches that, in this Section, are explored and explained.

Consider a typical scenario, where the i^{th} input data is a feature vector comprising N features: $F_i = [f_1, f_2, ..., f_N]$, with $f_j \in \mathbb{R}$ for $\forall j \in [1, N]$. To encode F_i into a target HV, H_i, the value and the index (or ID) of each feature must be represented in the high-dimensional space and linked in a value-ID pair.

For the values, it is crucial to establish a mapping strategy that ensures close scalars in the original space are projected in similar HVs. This is done by discretizing the data range into q levels and assigning each of them to a distinct HV, denoted as LV. In the existing literature, three distinct techniques are commonly employed to create the LVs: *Linear Mapping* [21], *Approximately Linear* [19], and *Thermometer* [22], each possessing unique advantages and disadvantages.

For the indexes of features, instead, two main approaches are adopted in the literature. In the *record-based encoding*, each index i is associated with a randomly generated base-vector, denoted as BV_i, that is stored in memory and binded with the corresponding LV_i, as shown in Fig. 1(a) and (1).

$$H_i = \sum_{i=1}^{N} (BV_i \otimes LV_i) \tag{1}$$



(a) (b) (c)

Fig. 2: Visualization of typical HDC functional schemes for different problems: (a) Classification, (b) Regression, (c) Clustering

In the *N-gram-based encoding*, the feature positions are encoded through permutations, leveraging the inherent property of this operation to produce orthogonal HVs. Each ID is associated with a different fixed number of permutations, as shown in Fig. 1(b) and (2). This technique removes the need for N BV in memory but, as pointed out in [11], achieves lower accuracy in some classification problems, such as speech recognition.

$$H_i = \sum_{i=1}^{N} \rho^i(LV_i) \tag{2}$$

Clipping techniques can be applied to adjust the resulting HVs elements in different ranges, such as: binary, bipolar, tripolar, or quantized. For instance, for obtaining bipolar HVs, the sign function is applied to each element, while for binary HVs, the majority rule shown in (3) is used, where $H_{i,d}$ is the d^{th} dimension of the encoded HV. Clipping allows for a trade-off between accuracy and hardware complexity.

$$Clip\_H_{i,d} = \begin{cases} 0 & H_{i,d} < \frac{\text{n.bundled HVs}}{2} \\ \text{rand} & H_{i,d} = \frac{\text{n.bundled HVs}}{2} \\ 1 & otherwise \end{cases} \tag{3}$$

B. Classification in HDC

Classification is a supervised learning problem in which the model must predict categorical class labels of new input patterns based on past observations. Fig. 2(a) shows a general overview of the structure of an HDC classification model, including training, retraining and inference phases.

During *training* all the encoded H_i belonging to the same label are aggregated, as shown in (4), to build a representative prototype CV_j for each class j.

$$CV_j = \sum_{\forall i \,\in\, class_j} (H_i) \tag{4}$$

After training, *inference* is performed by encoding each input feature vector into a query HV (Q_i), and searching for the most similar CV in memory. It's important to note that while the encoding step typically consumes the most time during the training phase, the inference stage often places a heavier computational load on the associative search, accounting for up to 83% of the total execution time [23].

To improve the HDC model's accuracy, after the initial training and inference steps, an iterative process of *retrain* can be performed. During this stage, the estimated CVs are calibrated using new data or by repeating the training for multiple iterations on the same dataset. Retraining is performed by removing Q_i from the mispredicted class CV_{wrong} and adding it to the correct one, $C_{correct}$, as shown in (5). Notably, in the case of retraining, the clipping is only applied at the last step.

$$\begin{cases} C_{wrong} = C_{wrong} - Q_i \\ C_{correct} = C_{correct} + Q_i \end{cases} \tag{5}$$

A lack of control in the CV update can easily lead the model to slow convergence or divergence [11]. In [24], the widely known concept of learning rate in AI is extended to HDC, proposing two different adaptive training techniques: iteration-dependent and data-dependent. In the iteration-dependent approach, the learning rate linearly decays as a function of the error rate. In the data-dependent approach, the adopted learning rate depends on how far the data was misclassified.

C. Regression in HDC

Regression, a core task in supervised learning, involves predicting the relationship between an input feature vector and its corresponding output y. This process utilizes a loss function to quantify the error between the predicted value $\hat{y}$ and the actual one y and update the model.

In the context of HDC, the regression procedure is detailed in Fig. 2(b), [14] and begins with initializing a Model HV (MV) to zero. Each dataset pattern i is first encoded in H_i using the techniques described in Section II-A, and then it is dot-multiplied with the MV to obtain the predicted value $\hat{y}_i$. $\hat{y}_i$ is compared with the actual value y_i, to compute the error $E_i = y_i - \hat{y}_i$ that it is used to updated the MV as shown in (6). A learning rate α can be applied to modulate the update step. This process can be iterated over several epochs to adjust the MV towards minimizing the prediction error.

$$MV = MV + \alpha(E \times H_i) \tag{6}$$

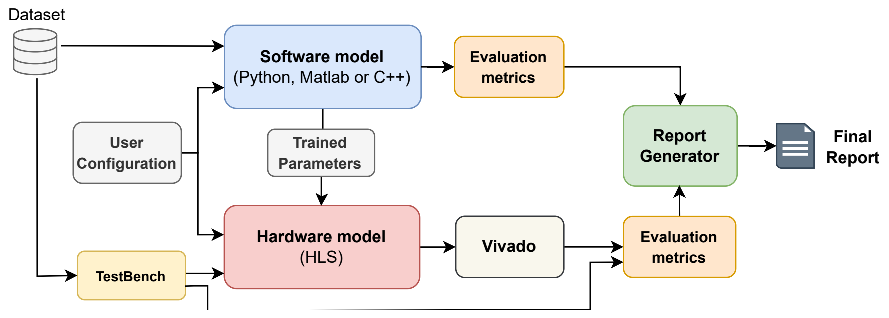


Fig. 3: Overview of the Aeneas Hypercompute framework

D. Clustering in HDC

Clustering is an unsupervised learning task that involves organizing input patterns into K distinct clusters based on their similarity. This process in HDC is straightforward and can be implemented as illustrated in Fig. 2(c) [12], [13].

Each cluster is represented by a Centroid HV (KV) that is stored in the Centroid memory and it is initialized with the HV of a pattern randomly chosen from the dataset. Patterns are encoded into HVs using the method detailed in Section II-A and, at each iteration, the model performs clustering by computing the similarity between H_i and each KV, assigning the pattern to the closest one.

During training, a second Centroid memory is used. When the cluster j is selected for the given input pattern, the corresponding KV_j in the second memory is updated by adding H_i. After processing all the patterns in the training dataset, the first Centroid Memory is replaced with the updated one. This procedure can be repeated over several epochs.

E. Related Works

HDC has recently gained substantial attention for its ability to address and solve reasoning problems across different domains, offering robust, fast, low-power, and lightweight solutions [15]–[17]. HDC is particularly suitable for custom hardware designs on FPGAs and Application-Specific Integrated Circuits (ASICs) thanks to the extremely parallel nature of all the involved arithmetic operations. Many works in the literature have attempted to propose hardware solutions and optimizations to accelerate HDC tasks [13], [21], [25], [26].

However, finding the ideal solution for a specific application adopting a custom design approach can be complex and time-consuming. To alleviate this challenge, the authors in [27] introduced F5-HD, a comprehensive framework designed to automate the deployment of HDC accelerators on FPGAs. F5-HD empowers users to specify problem characteristics, including dataset details, target FPGA, power constraints, and desired trade-offs in accuracy. The framework then automatically configures the architecture by exploiting pre-defined processing elements.

Similarly, the work in [28] introduces HD2FPGA, an automated tool for generating HDC hardware accelerators connected to a host CPU. In addition to what is described for [27], this framework supports clustering problems, different HV sizes, and automatically calibrates hardware parallelism to match user-specific requirements.

However, some limitations persist within these automated models. Neither of these frameworks allows users to customize the details of the HDC model, as they adhere to fixed and predetermined design choices. The chosen encoding technique does not allow for the generation of accelerators that can work with temporal sequences or different input data, nor is it flexible enough to support a variety of applications. The hardware is consistently instantiated for training, inference, and retraining without the flexibility to synthesize only a specific block or explore the effect of hardware parallelism at different levels of the learning process.

On the algorithmic side, part of the design choices available for HDC models [17]–[19], [24] has been explored in [29] and [30]. The work in [29], presents HDTorch, a software library with CUDA extensions for testing the impact of different strategies on the model accuracy. In HwAwHDC [30], a framework based on Reinforcement Learning optimization is used to search for the optimal trade-off between accuracy, model size, execution time, memory usage, and energy consumption of HDC software models implemented on embedded devices.

Nonetheless, there is a noticeable gap in the literature. Existing works do not provide a straightforward method for systematically testing and exploring the influence of different model design choices on hardware implementations alongside their impact on accuracy. This paper tries to fill this gap by presenting a fully automated and extremely flexible environment for the deployment of HDC models in hardware.

III. AeneasHDC

This work presents AeneasHDC, an automatic, open-source platform for simple and fast deployment of HDC models in both software and hardware. The framework is extremely customizable, supporting a wide range of the most common techniques adopted in the literature for learning problems, enabling users to easily assess the impact of different design choices on model accuracy, memory usage, execution time, power consumption, and area requirements.

Fig. 3 illustrates the workflow of AeneasHDC. An intuitive Graphic User Interface (GUI) allows to specify the problem characteristics, including dataset details, the number of features and classes, the learning task, the desired programming language, the target FPGA and a comprehensive set of configuration options for the model. An interactive mechanism lets the user decide how to distribute the workload between the software and the hardware implementations. The hardware design, in fact, is highly customizable as will be discussed in Section III-B and can be generated to accelerate the entire process (train, inference, retrain) or just certain stages. For example, users might choose to execute the training and retraining of the model in software and instantiate hardware only for inference, as is commonly done in embedded applications.

Given these input parameters, the system automatically generates all the configuration files required by the following entities and instantiates the model in software and hardware. From the software side, the model can be produced in Python, Matlab or C++, thanks to the versatile AeneasHDC library. The model is executed, and the results in terms of accuracy metrics are collected. Additionally, the data required for inference are extracted and passed to the hardware model.

The hardware architecture is generated using the Xilinx Vitis High-Level Synthesis (HLS) tool, and it is then synthesized, simulated and tested using Vivado 2023. Data on power consumption, execution time, and resource allocation is gathered from the design.

The report-generation module generates HTML summaries from software and hardware results, detailing model configuration, datasets, programming language, performance metrics, execution time, memory usage, power consumption, and resource utilization. These reports are stored locally and are easily accessible through a user-friendly GUI for reviewing and comparing different test runs.

The entire described process is engineered to be fully automated, abstracting complex or time-consuming tasks associated with coding models or designing custom hardware architectures. The GUI is an efficient singular point of control that enables users to manage the model at every stage.

AeneasHDC is completely open-source, with all its libraries and scripts available on GitHub [31]. We offer comprehensive documentation that explains basic usage, theoretical concepts behind parameter selections, and detailed insights into the code and library functions. This encourages community collaboration to enhance and evolve the framework over time. Additionally, our GitHub repository includes a suite of tests for assessing the effects of various parameters on hardware implementation in generic learning problems, facilitating practical application and continuous improvement.

A. Explorable Design Space

AeneasHDC provides a comprehensive range of configuration options for developing the HDC model, encompassing most techniques commonly found in the literature, as detailed in Section II. Table I lists all the currently supported features.

The hyperspace can be configured to support binary, bipolar, tripolar and integer HVs in any dimension and density. This allows to evaluate the impact of HV size and sparsity on computational complexity and model accuracy.

Three techniques are available for generating LevelVectors and representing scalar numbers: Linear, Approximately Linear and Thermometer. The Linear method offers the highest accuracy at the expense of high memory usage. Conversely, the Approximately Linear and Thermometer methods reduce memory needs and power consumption, respectively.

AeneasHDC offers robust support for both spatial and temporal sequence encoding. For spatial encoding, users can choose from two distinct techniques: record-based and n-gram-based, allowing customization to specific problem requirements. For temporal encoding, the framework provides the option to use custom window sizes or n-gram configurations, enhancing its suitability for a diverse range of challenges.

Clipping techniques can be optionally applied post-encoding, post-training, or at both stages. These include standard binary, bipolar, and ternary clipping, along with quantized formats optimized for hardware implementations. Of particular note is the power-of-two quantization, which allows multiplication and division operations required for encoding and cosine similarity to be replaced by simpler arithmetic shifts in hardware.

Our platform facilitates retraining, supporting multiple epochs and state-of-the-art learning rate strategies. It can be conducted on the same training dataset, as is common in literature, or on any tailored one.

Finally, the framework allows users to select from various similarity measures during the inference stage.

AeneasHDC currently supports clustering, regression and classification problems, but it can be easily extended to other applications modifying the open-source code.

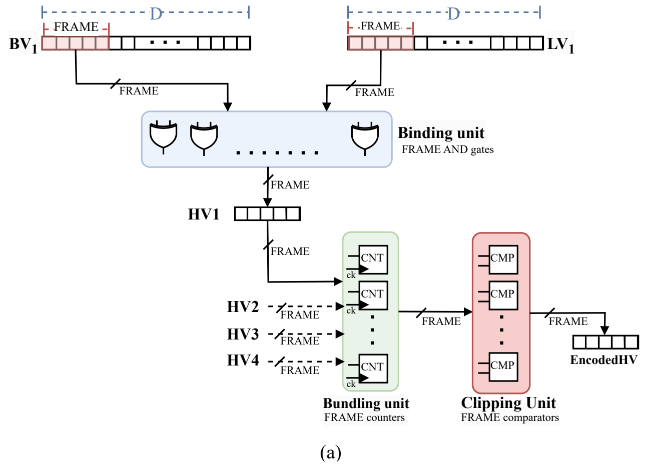
B. Hardware Architecture

The AeneasHDC platform is designed to transition from software-defined models to practical hardware implementations seamlessly. All the configurations discussed in Section III-A can be implemented in hardware, on any target FPGA, empowering users to directly assess how different design choices impact hardware performance and efficiency.

The accelerators synthesized and implemented via AeneasHDC feature a standard AXI (Advanced eXtensible Interface) connection with the processor and the dedicated

TABLE I: Explorable Design Space in the software library

Parameters	**Explorable space**
Learning Problem	[Classification, Clustering, Regression]
HV Dimension	Any, ex:$[200, 500, 1.000, 2.000, 10.000]$
HV DataType	[Binary, Bipolar, Tripolar, Integers]
HV Density	[Dense, Sparse]
HV Sparsity	Any, ex: $[0.01, 0.02, 0.05, 0.1, 0.2]$
Similarity Measure	[Hamming, Cosine, Dot Product]
LevelHVs Mode	[Linear, Approx. Linear, Thermometer]
Number of LevelHVs	Any, ex: $[5, 10, 30, 50, ...]$
Spatial Encoding Mode	[Record-based, N-gram based]
Temporal Encoding Mode	[Yes, No]
Temporal Encoding Window	Any, ex: [2,3,5]
Clipping Technique	[Binary, Ternary, PowerOfTwo, Quantized, Integer]
Retraining	[Yes, No]
Retrain Epochs	Any, ex: [5, 10, 20, 50]
Learning Rate Mode	[Data-dependent, Iteration Dependent]



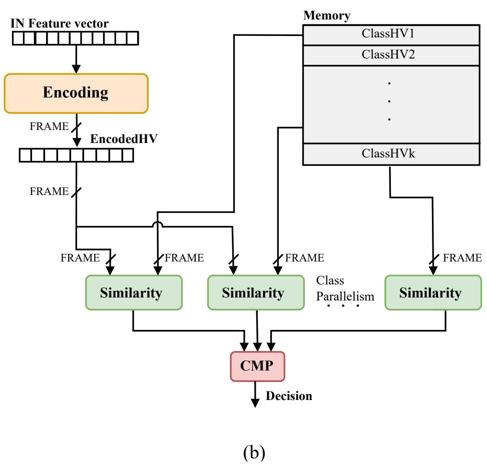


Fig. 4: Reconfigurable Hardware parallelism applied to a Record-based encoding example case. (a) Element parallelism, FRAME elements are computed in parallel. (b) Class Parallelism.

TABLE II: Hardware Parallelism Degrees

Parameters	Explorable space
Element Parallelism (*FRAME*)	Any in range $[1, D]$
Feature Parallelism	Any in range $[1, N]$
Class Parallelism	Any in range $[1, K]$

memories that ensures straightforward integration and broad compatibility. In these designs, the encoding process follows a pipeline approach that enables simultaneous processing of various input elements at different stages, optimizing hardware resource utilization, operating frequency, and throughput. Additionally, our framework introduces two pivotal features that add significant flexibility to the hardware architectures, ensuring efficient solutions in diverse environments.

- **Selective Model Instantiation:** synthesis of specific operational stages in the HDC learning process. The architecture can be finely tuned for accelerating the training process on high-resource FPGAs or for performing only the inference on resource-constrained systems;
- **Hardware Parallelism:** *partial processing* mechanism to control the degree of hardware parallelism at different stages of the learning process, allowing trade-offs between processing speed, hardware requirements and power consumption. This control can be applied at three different levels, summarized in Table II.
  1) **Element parallelism:** the *FRAME* parameter controls the number of HV elements processed in parallel. Fig. 4(a) demonstrates element parallelism applied to a record-based encoding technique: all the hardware is replicated *FRAME* times, and at each processing step, *FRAME* elements are processed in parallel, resulting in a total of $D/FRAME$ iteration for each HV;
  2) **Feature Parallelism:** reconfigurable level of parallelism that enables concurrent processing of multiple features, in the range $[1, N]$. This approach is particularly effective for elaborating complex datasets with numerous features;
  3) **Class Parallelism:** reconfigurable hardware parallelism for the similarity search depicted in Fig. 4(b). The user can decide to compare the encoded HV with each class sequentially or to enable parallel computation of $k \in [1, K]$ classes. This is instrumental in elaborating complex multi-class datasets and can be combined with the previous two levels.

IV. Experiments

To validate the AeneasHDC platform and demonstrate its versatility in generating HDC models in software and hardware, we conducted a series of tests varying the problem characteristics, the required task and the model configuration. These tests have been designed to showcase the capability of the platform and its applicability in varied scenarios and do not represent the search for the optimal configuration of the model, which will be the subject of future research. In these experiments, the HDC models have been first trained and retrained over 50 iterations using the software model. Then, we deployed the hardware for the inference stage on the Zynq UltraScale+ ZCU102 board, using a target frequency of 100 MHz and a 64-element frame size. For each experiment, we evaluated the software and hardware accuracy, the area occupation, power consumption, and execution time.

Table III provides a detailed summary of the configuration chosen for the HDC model in each test, while Table IV shows the corresponding outcomes. For classification, we used two datasets. The first, *CARDIO* [32], consists of 3 classes and spatial sequences described by 21 features. For this test, we employed 512-dimensional binary dense HVs that have been encoded using the n-gram technique, linear levels, no class clipping, and cosine similarity. The obtained results confirm the capability of the platform, showing an accuracy of 88.78% for both the software and the hardware models generated by AeneasHDC. In terms of hardware requirements,

TABLE III: Summary of HDC Model Configurations and Datasets

Test-id	Dataset	Task	HV Dimension	Data Type	Encoding Technique	Temporal Encoding	LV Technique	#LVs	Data Clipping	Similarity Measure
Test 1	CARDIO	Classification	512	Binary	N-gram	No	Linear	10	No	Cosine
Test 2	EMG	Classification	256	Binary	Record	Yes (n-gram=4)	Approx.	21	Binary	Hamming
Test 3	Hepta	Clustering	1024	Bipolar	Record	No	Thermometer	20	Quantized (6 bits)	Cosine
Test 4	Boston	Regression	512	Binary	Record	No	Linear	20	No	Dot Product

the synthesized architecture necessitated 12766 Lookup Tables (LUTs), 19744 Flip-Flops (FFs), and 80 Digital Signal Processors (DSPs). The power consumption was measured at 206 mW for dynamic power, with a processing speed of 5542.92 classifications per second. Notably, the high hardware demand for this accelerator is attributed to the complex cosine similarity and the absence of clipping, requiring 17 bits for each element of the CVs. Clipping or quantizing the CVs would be a key strategy for minimizing the area requirements of the architecture. It should be noted that this test employed the n-gram encoding technique that eliminates the need to store BVs in memory but, as described in Section II is not universally effective and, when implemented in hardware with a partial element-parallelism, requires processing the first frame elements twice for correctly performing the right-cyclic rotation, leading to additional execution time.

TABLE IV: Performance Metrics of HDC Models

Test-id	Software Metrics	Hardware Metric	#LUTs	#FFs	#DSPs	Dynamic Power	Task/s
Test 1	88.78% Acc.	88.78% Acc.	12766	19744	80	206	5542.92
Test 2	94.79% Acc.	94.79% Acc.	2996	6616	0	6	3219.22
Test 3	100% NMI	100% NMI	14090	19536	90	161	80042.68
Test 4	28.79 MSE	28.8 MSE	4330	4800	0	30	5543.85

The second dataset we examined for the classification task was *EMG* [33], a temporal encoding problem involving four features distributed across five classes. For this, we employed 256-dimensional binary dense HVs, and we used the record-based encoding in conjunction with the n-gram one to efficiently capture information about temporal windows of 4 elements. We also used approximately linear levels and applied binary clipping to the trained CVs, using the Hamming distance as the similarity measure. This configuration achieved 94.79% of accuracy, both for the software and the hardware models. The hardware architecture synthesized for this configuration is notably efficient, comprising 2996 LUTs, 6616 FFs, and no DSPs, and consuming only 6 mW of dynamic power while producing 3219.22 classifications per second. This remarkable efficiency in hardware resources and power consumption, despite the model's complexity, can be attributed to the binary clipping technique, a state-of-the-art solution for implementing HDC in resource-constrained systems.

For the clustering task, we employed the *Hepta* [34] dataset, which comprises 3 features and 7 clusters. The HDC model was set up with 1024-dimensional bipolar HVs, using Record-based encoding and the thermometer technique to generate levels. We applied quantized clipping with 6-bit precision to the trained ClassHVs and cosine similarity. This approach was chosen to balance hardware simplicity with effective performance. Remarkably, both the software and hardware models achieved a Normalized Mutual Information (NMI) score of 1.00, indicating perfect correlation. The hardware configuration required 14090 LUTs, 19536 FFs, and 90 DSP, while consuming 161 mW of dynamic power. Similar to the first test, the use of complex cosine similarity led to substantial hardware overhead, but employing a quantized format significantly simplified the logic, reducing the bit requirement for each element of the CVs from eighteen to just six. Additionally, the small amount of features in this dataset allows for clustering 80042.68 samples per second.

Finally, for regression, we analyzed the *Boston Housing* dataset [35] comprising 12 features. The model was configured with a 512-dimensional binary HV, Record-based encoding and dot product. For this test, we used the basic regression procedure described in Section II, but AeneasHDC supports the multi-model technique described in [12] too. In this problem, both the software and hardware model achieved a Mean Squared Error (MSE) equal to 33.45. The synthesized accelerator required 4330 LUTS, 4800 FFs and zero DSPs, while consuming 30 mW and estimating 5543 samples per second.

V. Conclusions

In this paper, we introduced AeneasHDC, a comprehensive and open-source framework designed for the automated generation of HDC learning models in software and hardware. AeneasHDC represents a versatile solution that supports a broad spectrum of design choices available in the literature, enabling rapid and straightforward exploration of the impact of different configurations on accuracy, power consumption, memory requirements, hardware overhead and execution time.

The proposed framework integrates an intuitive GUI that streamlines the specification of problem parameters (dataset, number of input features, classes and target FPGA), the required task and the model configuration. The hardware accelerators generated by AeneasHDC can be reconfigured to support different operational stages and levels of parallelism, allowing trade-offs between execution time and hardware resource allocation.

We validated the environment by conducting a series of tests varying the problem characteristics and the model configuration. For each experiment, we evaluated the performance of the software and hardware models in terms of accuracy and hardware parameters, demonstrating the platform's capability and efficiency to generate flexible models across different learning tasks.

This environment provides a straightforward method for exploring the influence of different design choices on hardware, representing a new milestone in automatic hardware generation for HDC models. In our next study, we will use the proposed platform to explore the impact of specific configurations on balancing accuracy and hardware efficiency and provide guidelines for optimizing the model setup across different datasets and application requirements.

ACKNOWLEDGMENT

This work was partially supported by the Technopole Flagship Project 6 under the Italian PNRR program.

REFERENCES

[1] A. Mitrokhin, P. Sutor, C. Fermüller, and Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, vol. 4, no. 30, p. eaaw6736, 2019.

[2] Y. Guo, M. Imani, J. Kang, S. Salamat, J. Morris, B. Aksanli, Y. Kim, and T. Rosing, "Hyperrec: Efficient recommender systems with hyperdimensional computing," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 384–389, 2021.

[3] N. Watkinson, D. Devineni, V. Joe, T. Givargis, A. Nicolau, and A. Veidenbaum, "Using hyperdimensional computing to extract features for the detection of type 2 diabetes," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 149–156, IEEE, 2023.

[4] A. Burrello, L. Cavigelli, K. Schindler, L. Benini, and A. Rahimi, "Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 752–757, IEEE, 2019.

[5] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, *et al.*, "An emg gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[6] A. Burrello, K. Schindler, L. Benini, and A. Rahimi, "One-shot learning for ieeg seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing," in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, IEEE, 2018.

[7] F. R. Najafabadi, A. Rahimi, P. Kanerva, and J. M. Rabaey, "Hyperdimensional computing for text classification," in *Design, automation test in Europe conference exhibition (DATE), University Booth*, pp. 1–1, 2016.

[8] A. Joshi, J. T. Halseth, and P. Kanerva, "Language geometry using random indexing," in *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers 10*, pp. 265–274, Springer, 2017.

[9] Y. Kim, M. Imani, N. Moshiri, and T. Rosing, "Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 115–120, IEEE, 2020.

[10] M. Imani, D. Kong, A. Rahimi, and T. Rosing, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *2017 IEEE international conference on rebooting computing (ICRC)*, pp. 1–8, IEEE, 2017.

[11] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[12] M. Imani, Y. Kim, T. Worley, S. Gupta, and T. Rosing, "Hdcluster: An accurate clustering using brain-inspired high-dimensional computing," in *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1591–1594, 2019.

[13] L. Ge and K. K. Parhi, "Robust clustering using hyperdimensional computing," *arXiv preprint arXiv:2312.02407*, 2023.

[14] A. Hernandez-Cano, C. Zhuo, X. Yin, and M. Imani, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 7–12, IEEE, 2021.

[15] C.-Y. Chang, Y.-C. Chuang, C.-T. Huang, and A.-Y. Wu, "Recent progress and development of hyperdimensional computing (hdc) for edge intelligence," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2023.

[16] D. Kleyko, M. Davies, E. P. Frady, P. Kanerva, S. J. Kent, B. A. Olshausen, E. Osipov, J. M. Rabaey, D. A. Rachkovskij, A. Rahimi, *et al.*, "Vector symbolic architectures as a computing framework for emerging hardware," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1538–1571, 2022.

[17] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–52, 2023.

[18] E. Hassan, Y. Halawani, B. Mohammad, and H. Saleh, "Hyper-dimensional computing challenges and opportunities for ai applications," *IEEE Access*, vol. 10, pp. 97651–97664, 2021.

[19] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey, "Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 5880–5898, 2018.

[20] K. Schlegel, P. Neubert, and P. Protzel, "A comparison of vector symbolic architectures," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4523–4555, 2022.

[21] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 4, pp. 1–25, 2019.

[22] D. A. Rachkovskiy, S. V. Slipchenko, E. M. Kussul, and T. N. Baidyk, "Sparse binary distributed encoding of scalars," *Journal of Automation and Information Sciences*, vol. 37, no. 6, pp. 12–23, 2005.

[23] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 221–234, IEEE, 2021.

[24] M. Imani, J. Morris, S. Bosch, H. Shu, G. De Micheli, and T. Rosing, "Adapthd: Adaptive efficient training for brain-inspired hyperdimensional computing," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1–4, IEEE, 2019.

[25] M. Imani, S. Salamat, S. Gupta, J. Huang, and T. Rosing, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pp. 493–498, 2019.

[26] S. Salamat, M. Imani, and T. Rosing, "Accelerating hyperdimensional computing on fpgas by exploiting computational reuse," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1159–1171, 2020.

[27] S. Salamat, M. Imani, B. Khaleghi, and T. Rosing, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 53–62, 2019.

[28] T. Zhang, S. Salamat, B. Khaleghi, J. Morris, B. Aksanli, and T. S. Rosing, "Hd2fpga: Automated framework for accelerating hyperdimensional computing on fpgas," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pp. 1–9, IEEE, 2023.

[29] W. A. Simon, U. Pale, T. Teijeiro, and D. Atienza, "Hdtorch: Accelerating hyperdimensional computing with gp-gpus for design space exploration," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2022.

[30] J. Yang, V. K. R. Yasa, Y. Sheng, D. Reis, X. Jiao, W. Jiang, and L. Yang, "Hardware-aware automated architecture search for brain-inspired hyperdimensional computing," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 352–357, IEEE, 2022.

[31] "Aeneashdc." https://github.com/AeneasHDC/, Jan. 2024.

[32] D. Campos and J. Bernardes, "Cardiotocography." UCI Machine Learning Repository, 2010. DOI: https://doi.org/10.24432/C51S4N.

[33] A. Rahimi, "Emg dataset." https://github.com/abbas-rahimi/HDC-EMG/blob/master/dataset.mat, Jan. 2024.

[34] A. Ultsch and J. Lötsch, "The fundamental clustering and projection suite (fcps): A dataset collection to test the performance of clustering and data projection algorithms," *Data*, vol. 5, no. 1, 2020.

[35] "Boston housing dataset." https://www.kaggle.com/datasets/vikrishnan/boston-house-prices, 2024.