HDFlow: A Multi-layer Design and Evaluation Framework for Hyperdimensional Computing

Jinghao Wen[1], Dongning Ma[1,2], Sizhe Zhang[1], Xun Jiao[1]
[1]Department of Electrical and Computer Engineering, *Villanova University*, Villanova, USA
[2]Department of Computer Science, Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, United Arab Emirates
Email: {jwen01, dma2, szhang6, xun.jiao}@villanova.edu, Dongning.Ma@mbzuai.ac.ae

***Abstract*—Brain-inspired hyperdimensional computing (HDC) is an emerging computing paradigm that imitates the deep behaviors of abstract brain circuit functions. HDC is a more memory-centric computing paradigm compared with existing machine learning algorithms. HDC has been increasingly applied to different domains as an energy-efficient and lightweight alternative to existing learning algorithms. Although promising, design and evaluation of HDC face significant obstacle due to a lack of a general user-friendly framework. Existing open source projects for HDC mostly focus on one or few applications and require notable manual effort for model configuration that become a bottleneck for the practical evaluation and deployment of HDC models. To address these challenges, this paper proposes a design and evaluation framework for HDC called HDFlow. It provides multi-layer access to the entire lifecycle of HDC processing, from the bottom hypervector to the top system layer, to enable exhaustive evaluations such as error injection and quantization. HDFlow is implemented entirely using python and only requires minimal libraries, allowing highly-flexible customization including encoding schemes, error models, quantization configurations, etc. HDFlow also provides flexible interface to read data from different applications to avoid building an entire HDC framework every time for a new application. HDFlow leverages Just-in-Time (JIT) compilation as acceleration to enhance the execution speed.**

I. Introduction

Machine learning algorithms have been increasingly applied to edge computing platforms such as Internet-of-Things devices and embedded systems. Although machine learning algorithms such as deep neural networks (DNN) are showing outstanding accuracy that can even surpass human in many sophisticated tasks, their execution usually require considerable resources such as storage, computation cost and power which may not be adequately available on edge computing devices. Recently, the brain-inspired Hyperdimensional Computing (HDC) emerges as an alternative to the existing machine learning algorithms in the edge computing platforms such as bio-signal processing [1], robotics, and natural language processing [2]. Compared with deep neural networks, HDC exhibits relatively smaller model size and lower power consumption as well as stronger support for emerging architectures such as in-memory computing [3], while still maintaining comparable accuracy with negligible loss.

Although HDC proves to perform outstandingly on various applications, design and evaluation on HDC have yet been sufficiently examined and thoroughly explored. As HDC has been increasingly applied into security-critical domains, it is of great significance to examine the robustness and reliability of HDC under the passive variability or the active adversarial attacks from malicious users. On the other hand, unlike servers or desktop computers that may reside in data centers with adequate climate control, the working environment of edge devices is also more inclement. The devices running HDC may be exposed to uncertainties or errors originated from factors ranging from external noises, extreme weathers or internal changes of physical conditions such as device ageing. Furthermore, recent works on testing HDC already reveals that HDC models may be vulnerable to uncertainty such as hardware errors [4] or malicious attacks [5]. However, those works are mainly focused on specific type of application such as image classification, thus, the frameworks used are hardly reusable and generalizable to enable large scale and cross-application design and evaluation on HDC.

Therefore, it is of high necessity to develop an easy-to-use and flexible evaluation framework for HDC, to enable its larger scale of practical deployment with broader applications. However, unlike DNN, developing such a framework faces several unique challenges specific to HDC. First, HDC is less application-agnostic compared with DNN. In DNN, types of layers, and the network structures are of high re-usability across different applications while the encoding schemes in HDC are highly specific to the applications. Second, evaluation on HDC requires rigorous and frequent access to different layers of HDC processing, ranging from the operand vectors to the entire system. Third, since HDC is highly memory-centric, evaluation on HDC requires diverse types of (customized) error models to realistically reflect the characteristics of diverse (emerging) memory architectures.

To overcome these challenges we propose **HDFlow**, a multi-layer design and evaluation framework for hyperdimensional computing. The contributions can be summarized as follows:

- **HDFlow** is a multi-layer design and evaluation framework for the emerging hyperdimensional computing. **HDFlow** has three key features: software-only and application-agnostic design, multi-layer access, and flexible and easy configuration.
- Based on the features, we implement three components of **HDFlow**: the *Front-end* which loads application data and specify user configurations, the *Back-end* which interacts with each layer in HDC, and the *Evaluation* which

TABLE I: **HDFlow** Features, Components and Use-cases

Sections	Section V	Section VI	Section VII
Features	SW-only & application-agnostic design	Multi-layer access	Flexible & easy configuration
Components	**HDFlow** Front-end	**HDFlow** Back-end	**HDFlow** Evaluation
Use-cases	**HDFlow** for model development	**HDFlow** for assessing robustness	**HDFlow** for design space exploration

implements and operates different evaluation tasks.

- We present three corresponding use-cases to highlight each of the three features: using **HDFlow** for model development, assessing robustness, and design space exploration. The use-cases reveal several properties of HDC such as performance differences between encoding mechanisms, robustness against bit errors, and resilience to quantization.

II. Related Work

Since Karneva's introduction of HDC into applications with learning tasks [6], related works on HDC mostly focuses on three directions:

Application-driven HDC: In biosignal processing, Rahimi et al. uses HDC on hand gesture recognition based on EMG and achieves 97.8% accuracy on average, which surpasses the SVM by 8.1% [7]. **NeuroHD-RA** [8] offers an efficient and scalable solution for edge-compatible lectrocardiogram (ECG) disease detection. HDC also shows promising potential in emerging applications, e.g., faster gene pattern matching [9] and blockchain [10].

Hardware acceleration and system frameworks: FPGA-based systems such as **F5-hd** provide fast and flexible deployment of HDC models. **QuantHD** also proposes a quantization framework for HDC to achieve smaller model size [11]. There is also enhancement of the memory structure of HDC, such as leveraging memristors in the emerging in-memory computing scheme [3]. These approaches demonstrate the strong synergy between HDC and non–von Neumann architectures.

Robustness, reliability, and security evaluation: Prior work has analyzed the impact of bit errors in associative memory and proposed differential fuzz testing frameworks to uncover vulnerabilities in HDC implementations [5]. Other studies have explored adversarial attacks on HDC-based voice recognition systems, revealing that HDC models are not inherently immune to malicious manipulation [4], [12].

Though various applications scenarios and acceleration frameworks of HDC have been proposed, an easy-to-use evaluation framework for HDC is still largely absent, creating an obstacle for practical and large-scale deployment of HDC. This paper presents a systemic effort to diminish the obstacle by designing a multi-layer framework dedicated for HDC design and evaluation.

III. HDC Preliminaries

A. Hypervectors and HDC operations

As the brain's neural circuits have a massive number of neurons and synapses, it is suggested that large circuits are fundamental and essential to the brain's processing. Based on the understanding that brains compute with patterns of neural activity that are not readily associated with numbers, HDC was founded on the mathematical properties of high-dimensional spaces which show remarkable agreement with behaviors controlled by the brain [6], [13]. Instead of computing with numbers, HDC computes with high-dimensional vectors (e.g., 10,000 Dimension) which are referred to as hypervectors (HV): $\vec{H} = \langle h_1, h_2, \ldots, h_d \rangle$.

$$\begin{aligned} \vec{H_p} + \vec{H_q} &= \langle h_{p1} + h_{q1}, h_{p2} + h_{q2}, \ldots, h_{pd} + h_{qd} \rangle \\ \vec{H_p} \times \vec{H_q} &= \langle h_{p1} \times h_{q1}, h_{p2} \times h_{q2}, \ldots, h_{pd} \times h_{qd} \rangle \\ \rho_1(\vec{H}) &= \langle h_d, h_1, h_2, \ldots, h_{d-1} \rangle \end{aligned} \quad (1)$$

In HDC, HVs are the fundamental blocks to represent "symbols" reflecting real-life matters such as pixel values, signal levels or language characters in the high-dimensional space. HVs are holographic, and (pseudo)random with independent and identically distributed (i.i.d.) components. That is, every piece of information contained in the HV is equally distributed over all its elements in a full holistic representation of the vector so that no element inside the HV is more responsible for storing one piece of information over another. The formation of HVs naturally pertains to arithmetic operations such as (element-wise) addition, (element-wise) multiplication, and permutation (vector rotation) as shown in Eq. 1 which can form an algebra over the vector space.

B. HDC Model Development

HDC model development is generally composed of three phases: training, retraining and inference. All the three phases feature a fundamental process of encoding.

Encoding uses item memory to project real-world feature vectors into their high-dimensional space representations using a set of HD operations Γ. Item memory $\mathbf{R}$ consists of k seed HVs, each representing a possible value of the discrete features. Let $\vec{F}$ be the real-word feature vector of a certain sample, the process of encoding $\vec{F}$ into its representing sample HV $\vec{S}$ can be noted as $\vec{S} = \Gamma(\mathbf{R}(\vec{F})) = \Gamma(\mathbf{R}(\{f_1, f_2, \ldots, f_m\}))$. HDC uses the value of discrete features to index the corresponding seed HVs and then apply HD operations to aggregate the seed HVs into one sample HV $\vec{S}$.

Training is the process of establishing the associative memory using the training set. Associative memory $\mathbf{A}$ is the trainable part of an HDC model. It consists of l class HVs, each representing a class in a classification task. For each training sample, HDC adds its sample HV to the corresponding class HV, i.e., $\vec{A_l} = \sum \vec{S_l}$. This process is to aggregate the information from sample HVs together into the AM.

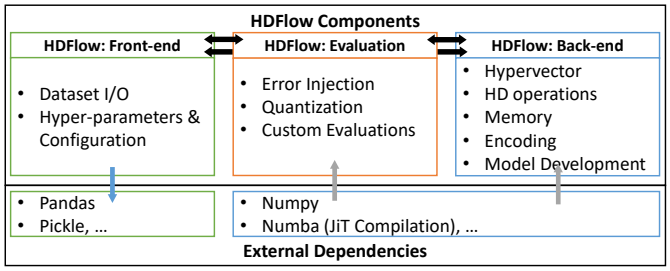


Fig. 1: **HDFlow** has 3 components: Front-end, Back-end and Evaluation, with dependencies for interface or acceleration.

Inference is the process of using unseen data from the inference set to evaluate the trained model's performance. First, the unseen sample with unknown class is encoded into its sample HV specifically referred to as query HV $\vec{Q_?}$. The HDC model then calculates the distance (δ) between this query HV and each class HV in the associative memory to obtain the distance metrics. The class with the smallest distance $p = argmin(\delta(\vec{Q_?}, \mathbf{A}))$ marks the prediction of the unseen sample.

IV. HDFlow Overview

HDFlow is a comprehensive framework for HDC performance that can perform diverse types of design and evaluation studies such as error injection, fault tolerance evaluation, quantization, etc., from operation to system level. **HDFlow** has three key features: software-only and application-agnostic design, diverse and multi-layer interface, and flexible and easy configuration. Each of the three features corresponds to the three components implemented of **HDFlow** as shown in Fig. 1. We also present one use-case for each of the features and components. A detailed explanation of the **HDFlow** features, components and use-cases can be found in Table I.

V. HDFlow Front-End

A. Feature: Software-only & application-agnostic design

Existing frameworks usually require extensive dependencies such as or hardware acceleration architectures such as FPGA or GPU [14]. Although efficient, such frameworks are usually difficult to use for the purpose of evaluation, particularly for machine learning engineers or data scientists who are familiar with Python but may not have intensive background in hardware acceleration or EDA. As a framework dedicated for design and evaluation, **HDFlow** is purely Python-based and requires minimal dependencies for easy customization. **HDFlow** though, still provides interface for acceleration through Just-in-Time (JiT) compilation. On the other hand, unless other machine learning algorithms that are less application-agnostic, HDC rely on application-specific encoding schemes. As HDC is an emerging computation scheme, the application data do not usually have a standardized format. Therefore, users are usually required to develop specific encoding frameworks for each application, adding difficulty to extensively evaluate HDC performance across different applications. **HDFlow** is flexible on defining input data by providing different I/O presets, thus data from different applications can be directly leveraged by the integrated encoding mechanisms inside **HDFlow**.

B. Component: *HDFlow Front-end*

The **HDC Front-end** of **HDFlow** performs two major tasks: 1) to load application datasets, and 2) to specify **HDFlow** with configurations and hyperparameters provided by user. As a design and evaluation framework, **HDFlow** should be able to load datasets from diverse application sources. **HDFlow** has a specific module to interact with external libraries such as Pandas and Pickle to conform with multiple dataset formats. **HDFlow** also defines a unified format of configuring an HDC model. As shown in Code 1, the configuration file allows user specifications that are parsed and taken by the front-end to operate the back-end and evaluation components.

User can specify diverse configurations corresponding to 1) the different phases in HDC model development (Line 3-20), and 2) the evaluation mechanisms and objectives (Line 21-26). For model development, parameters such as epoch, learning rate, HV dimension, datatypes and encoding schemes can be configured by the user. For evaluation mechanisms, parameters related to the evaluation tasks can be specified, for example in Code 1 (Line 21-26), the file specifies the evaluation task to be random error injection to the associative memory during inference when the memory is read.

```
1  version=0.3
2  [Dataset]
3  dataset_path=./dataset/isolet
4  [Hyperparameters]
5  train_test_split=0.2
6  retrain_epoch=10
7  learning_rate=1
8  num_class=10
9  [Hypervector]
10 hv_dimension=10000
11 hv_datatype=bipolar
12 distance_metric=cosine
13 [Memory]
14 item_memory_scheme={random}
15 item_memory_datatype=bipolar
16 items=25
17 associative_memory_scheme=zero
18 associative_memory_datatype=int16
19 [Encoding]
20 encoding_scheme=id
21 [Evaluation]
22 type=error_injection
23 error_func=random
24 error_location=associative_memory
25 error_phase=inference
26 error_type=memory_read
```

Code 1: **HDFlow** config file example for evaluating error injection to the associative memory with ISOLET dataset.

C. Use-case: *HDFlow for Model Development*

Encoding is one of the most important scheme in model development. In this use case as an example, we focus on comparing the accuracy of HDC model with different encoding schemes. We use spam text detection application for this

TABLE II: HDC accuracy with different encoding schemes

encoding	SMS dataset	YouTube dataset
id-based	0.963	0.908
permutation-based	0.971	0.922

evaluation. The spam text detection application features two datesets: SMS and YouTube comments, with a binary tasks for each dataset to classify a message or comment to be spam or not spam. As illustrated in Fig. 4,

We can apply different encoding schemes such as id-based and permutation-based encoding. Id-based encoding treats each feature inside the feature vector as a record assigned with a unique id to differentiate from other records. The value and the id of a feature are used to index the corresponding base or id HVs from the item memory and id memory. They are multiplied to aggregate the information to establish the feature HV. Permutation-based encoding only requires one item memory. Instead of using unique ids, it permutes the base for "id" times to differentiate from other features and establish the feature HV. All the features HVs are then added together to obtain the HV that represents the sample, i.e., the sample HV. Using **HDFlow**, switching between encoding schemes can be completed by changing the parameter at Code 1 (Line 20-21), without developing the entire encoding scheme repetitively.

Using **HDFlow**, we present the classification accuracy of HDC models with id-based and permutation-based encoding in table II. For both the SMS and Youtube datasets, permutation-based encoding exhibits higher classification accuracy, i.e., around 0.8% to 1.4% than id-based encoding. Such difference indicates that encoding scheme may affect the application performance, highlighting the importance of designing and evaluating diverse encoding schemes for an application. **HDFlow** can help designers to conveniently and efficiently search the optimal encoding schemes as well as other model development configurations without repetitive coding.

VI. HDFlow Back-End

A. Feature: Multi-layer access

HDC system is multi-layer, from the bottom hypervectors to the entire system. Although there are existing evaluation frameworks such as error injection frameworks, they are mostly targeted at a specific layer with static faults and based on system-wise naive error models due to lack of flexibility [5]. **HDFlow** grants multi-layer access from top to bottom with high granularity as users can perform various types of evaluations such as quantization and error injection, enabling more authentic and comprehensive evaluation and HDC testing.

B. Component: *HDFlow Backend*

To fulfill the diverse evaluation tasks as specified by the user, **HDFlow** features a back-end component as an engine that supports **HDFlow** with the access to different layers of HDC. In the stack of the HDC model as shown in Fig. 2, **HDFlow** can obtain access to 5 layers from the bottom

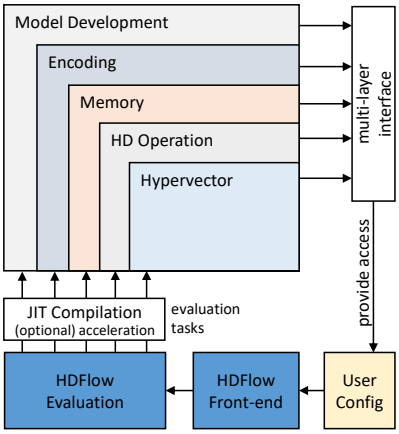


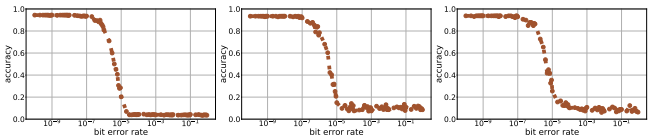
Fig. 2: **HDFlow** Back-end with access to multiple layers.

HV layer to the top layer on the entire model development. For example, to inject errors into the HD operations layer, **HDFlow** is able to get access of and modify the operands hypervectors during runtime, allowing error injection at the granularity of each individual operation.

HDC requires intensive high dimensional array operations, thus **HDFlow** leverages the NumPy library for efficient computing. We also leverage the (optional) Numba library for Just-in-Time (JIT) compilation, parallelism and interface to hardware accelerators such as GPU platforms. Enabling Numba is a one-line effort in the configuration file by adding a decorator [15], without the necessity on modifying **HDFlow** internally. This optional interface grant freedom to users that would like to leverage other hardware acceleration efforts for a faster or energy-efficient evaluation on the HDC model.

C. Use-case: *HDFlow for Assessing Robustness*

In the use case of evaluation on robustness, as an example, we use the "bit flip error" as our error type. When accessing the associative memory during each inference, every bit has a probability (bit error rate) to flip. This introduces errors in HDC inference and can subsequently result in incorrect classifications. Using **HDFlow** we could perform evaluations of HDC under different error rates by specifying error schemes as a Python decorator during configuration rather than hardcoding the error injection functions into the HDC system which could results in poor flexibility when we want to examine different error schemes and rates. Fig. 3 shows the classification accuracy of HDC model under different bit flip error rate of three datasets evaluated using **HDFlow**. We can observe that when the bit error rate increases from minimum ($< 10^{-9}$ to 10^{-7}), the accuracy does not suffer a significant degradation. However, when the bit error rate surpasses a threshold around 10^{-7}, the accuracy drastically decreases and collapses below 0.2. The accuracy curves illustrate that HDC can tolerate up to on average 10^{-7} bit error rate, advising designers to carefully configure their systems in the region before the collapse of accuracy. With **HDFlow**, such evaluation can be completed with just modifying the configuration file as suggested by



(a) ISOLET (b) CARDIO (c) HAR

Fig. 3: HDC accuracy with different bit error rates.

Code 1 instead of re-implementing different encoding and/or model development schemes.

VII. HDFlow Evaluation

A. Feature: Flexible & easy configuration

HDFlow already supports a diverse type of HDC processing. For example, **HDFlow** provides diverse types of encoding schemes such as record-based encoding and N-gram encoding; supports different types of hypervectors such as different datatypes like binary, bipolar and float and elaborations such as sparse vector; provides different quantization algorithms such as linear and exponential quantization; provides different error injection algorithms such as dynamic errors and stuck-at-fault errors. In addition, user can seamless integrate their custom encoding schemes, quantization algorithms and error models in **HDFlow**. By following the **HDFlow** developing principles, those customized features are of "substantial equivalence" to the canonical features initially available in **HDFlow**.

B. Component: *HDFlow Evaluation*

HDFlow already provides multiple in-house evaluations such as error injection and quantization. The evaluations are highly customizable since **HDFlow** provides *de facto* interface to modify the behavior of the functions in HDC model development. This is done by formulate the evaluation objectives as implementation using the Python decorator. Code 2 shows a typical decorator in **HDFlow** to perform error injection at the operation layer, i.e., to the addition operation.

The working mechanism of the injection involves three parts: First is the original HV operation, namely the *add*. It performs a simple element-wise addition using the two operand HVs. Second is the main error injection implemented using decorator, it takes the function to be injected *hdc_func* and check if it is the correct layer to inject by examining the *INJECTION_ENABLE* flag (Line 6–7). If true, the inner function *injection_wrapper* will take and execute the addition operation function, then modify the operation results *hv* using the error function *injection*. Third is the *injection* function that takes an HV and injects errors into them.

In the evaluation component of **HDFlow**, there are different decorators available targeting at different layers or phases of HDC according to Table III. In addition to the provided evaluations, user can define their own evaluations following its format. By using the decorator interface provided in **HDFlow**, user does not have to modify the internal codes and implementations of **HDFlow**, which significantly alleviates the effort of developing an entire evaluation components from bottom to top layer for each application or HDC configuration. This enhances the **HDFlow**'s scalability as users can now operate **HDFlow** for multiple configurations and sweep between diverse evaluation settings within one script.

```
1  @injection_operation
2  def add(hvA, hvB):
3      return np.add(hvA, hvB)
4
5  def injection_operation(hdc_func):
6      if not INJECTION_ENABLE:
7          return hdc_func
8      def injection_wrapper(*args, **kwargs):
9          hv = hdc_func(*args, **kwargs)
10         injection(hv)
11         return hv
12     return injection_wrapper
13
14 def injection(*args, **kwargs):
15     #PERFORM_INJECTION_HERE
```

Code 2: Example of a decorator available from **HDFlow** evaluation, for injecting error to the addition HD operation.

TABLE III: Available evaluation decorators by default from **HDFlow** evaluation. User can add their custom decorators.

decorator	description
@injection_HV	Inject error to one HV
@injection_operation	Inject error to one HD operation
@injection_AM	Inject error to an entire memory
@injection_encoding	Inject error for encoding
@injection_model	Inject error for the entire model (development)

TABLE IV: Available error functions by default from **HDFlow** evaluation. User can add their custom functions.

error function	error scheme
bit_flip	bit flip
bit_high	bit set to high level (one)
bit_low	bit set to low level (zero)
random	replace with random number
HV_fault	random number of elements inside HV are unavailable

C. Use-case: *HDFlow for Design Space Evaluation*

Related works have indicated that the HVs or memories inside an HDC model can be quantized for smaller memory overhead and energy-efficient execution [11]. However, the introduction of quantization usually incites model quality degradations since the quantization process brings error into the system. It is of critical importance to examine the impact of quantization so as to design and implement retraining mechanisms to potentially regain or recovery the degraded accuracy. In Fig. 5, we evaluate the accuracy across different bit widths (int32, int16, in8 and binary) for quantization on the associative memory with three datasets: ISOLET, CARDIO and HAR. We can observe that the HDC model accuracy of all the three datasets suffers from degradation when quantization is introduced. With more aggresive quantizations (fewer bit

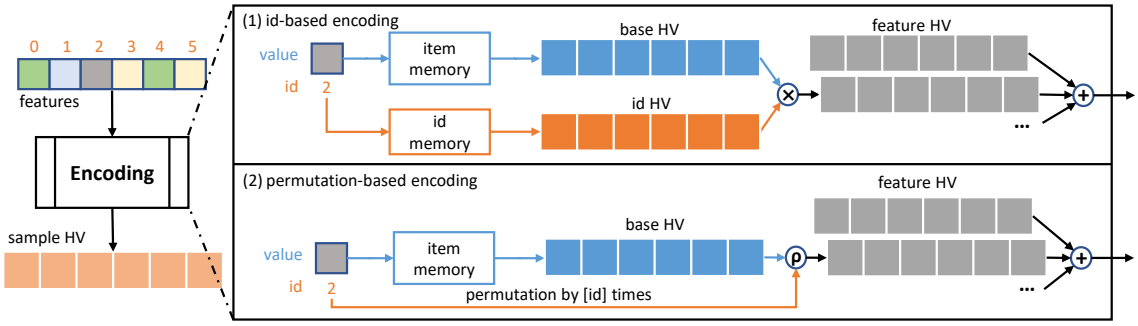


Fig. 4: Different encoding schemes: id-based encoding vs. permutation-based encoding.

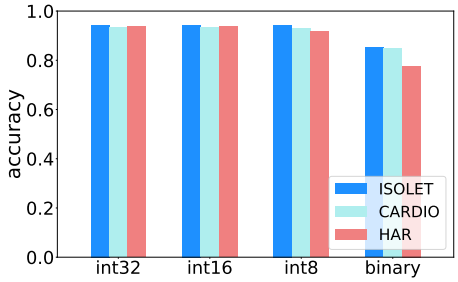


Fig. 5: HDC accuracy with different quantization bitwidths.

widths), the accuracy degradation increases. Moreover, different datasets exhibit different sensitivity against quantization, e.g., HAR suffers significantly higher accuracy drop than the other two datasets. This highlights the necessity to enable a case-by-case quantization for different applications. With **HDFlow**, evaluation of quantization can be done using one script and can be reused across different applications.

VIII. Conclusion

Hyperdimensional computing, as an emerging computing scheme with advantages such as energy efficiency and compact model size, is gaining increasing capability in various application domains. However, the absence of an easy-to-use design and evaluation framework is prohibiting its large-scale practical deployment. In this paper, we present **HDFlow**, a dedicated framework to facilitate and enhance the design and evaluation of HDC models. **HDFlow** has several advantages over existing HDC frameworks including simple implementation, multi-layer access to the HDC model, easy-to-use framework as well as intensive support for customization. Three use cases related to HDC presented in this paper such as model development, error injection, and quantization exhibit the flexibility and versatility of **HDFlow**, highlighting the diverse scenarios that **HDFlow** can help designers to efficiently evaluate their HDC system. To the best of our knowledge, **HDFlow** is the first flexible multi-layer design and evaluation framework for HDC.

Acknowledgment

This work was partially supported by NSF grant #2202310. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Burrello, K. Schindler *et al.*, "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, pp. 1–4.

[2] F. R. Najafabadi, A. Rahimi *et al.*, "Hyperdimensional computing for text classification," in *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, 2016, pp. 1–1.

[3] G. Karunaratne, M. Le Gallo *et al.*, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

[4] S. Zhang, R. Wang *et al.*, "Assessing robustness of hyperdimensional computing against errors in associative memory," in *IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2021.

[5] D. Ma, J. Guo *et al.*, "Hdtest: Differential fuzz testing of brain-inspired hyperdimensional computing," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[6] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.

[7] A. Rahimi, S. Benatti *et al.*, "Hyperdimensional biosignal processing: A case study for emg-based hand gesture recognition," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE, 2016, pp. 1–8.

[8] Z. He, J. Wen *et al.*, "Neurohd-ra: Neural-distilled hyperdimensional model with rhythm alignment," *arXiv preprint arXiv:2507.14184*, 2025.

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

[10] J. Wen, D. Ma *et al.*, "Proof-of-useful-work blockchain for trustworthy biomedical hyperdimensional computing," *arXiv e-prints*, pp. arXiv–2202, 2022.

[11] M. Imani, S. Bosch *et al.*, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2268–2278, 2019.

[12] R. Thapa, D. Ma, and X. Jiao, "Hdxplore: Automated blackbox testing of brain-inspired hyperdimensional computing," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 90–95.

[13] A. Thomas, S. Dasgupta, and T. Rosing, "Theoretical foundations of hyperdimensional computing," *arXiv preprint arXiv:2010.07426*, 2020.

[14] S. Salamat, M. Imani *et al.*, "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*, 2019, pp. 53–62.

[15] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.