Holistic Design towards Resource-Stringent Binary Vector Symbolic Architecture

Shijin Duan[1], Nuntipat Narkthong[1], Yukui Luo[2], Shaolei Ren[3], Xiaolin Xu[1]
[1]*Northeastern University*, [2]*Binghamton University*, [3]*University of California, Riverside*
{duan.s, narkthong.n, x.xu}@northeastern.edu, yluo11@binghamton.edu, shaolei@ucr.edu

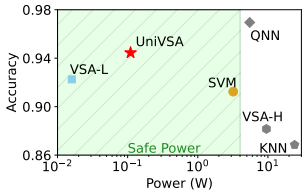
***Abstract*—Classification tasks on ultra-lightweight devices demand devices that are resource-constrained and deliver swift responses. Binary Vector Symbolic Architecture (VSA) is a promising approach due to its minimal memory requirements and fast execution times compared to traditional machine learning (ML) methods. Nonetheless, binary VSA's practicality is limited by its inferior inference performance and a design that prioritizes algorithmic over hardware optimization. This paper introduces UniVSA, a co-optimized binary VSA framework for both algorithm and hardware. UniVSA not only significantly enhances inference accuracy beyond current state-of-the-art binary VSA models but also reduces memory footprints. It incorporates novel, lightweight modules and design flow tailored for optimal hardware performance. Experimental results show that UniVSA surpasses traditional ML methods in terms of performance on resource-limited devices, achieving smaller memory usage, lower latency, reduced resource demand, and decreased power consumption.**

I. Introduction

TinyML frameworks are essential for edge devices, aiming for low latency, minimal hardware overhead, and high throughput [1]–[3], particularly in brain-computer interfaces (BCIs). Traditional ML like SVMs [4] and linear classifiers [5] are still favored for BCI due to their lightweight implementations, while in modern ML, even the binary neural networks (BNNs) fail to meet the power constraints of implanted BCI devices [4], [6]. Thus, there's a vital need for ML models that are both **lightweight** and **performant** for advancing tinyML applications on resource-stringent devices. We contend that the emerging binary vector symbolic architecture (VSA) possesses these characteristics. Binary VSA encodes objects and values as binary vectors and executes operations through logical operations, significantly enhancing performance on resource-limited devices [7]–[10].

Currently, the state-of-the-art (SOTA) training approach for lightweight binary VSA is low-dimensional computing (LDC) [11]. Binary VSA models optimized by LDC strategy can have vector dimension $D \approx 100$, with only minimal loss in inference accuracy compared with SOTA high-dimensional models with $D = 10,000$ [12]. The low-dimension binary VSA results in kilobyte-scale model sizes and power consumption within safe limits on BCIs, laying the groundwork for applying binary VSA models in resource-stringent environments.

However, current binary VSA implementations still face other challenges in resource-constrained devices such as BCIs. Specifically, binary VSA models could underperform compared to traditional ML approaches in certain BCI tasks. For instance, an LDC-trained binary VSA model achieves an accuracy 5% lower than that of an SVM for the EEGMMI task (Table II). Additionally, there is a notable saturation in accuracy with increasing vector dimensions, such as around $D = 128$ in [11]. Innovations in binary VSA design are essential to overcome these accuracy constraints within power limitations. Furthermore, past binary VSA hardware implementations are often developed separately from the model design, as independent accelerators [9], [11], without considering the hardware overhead and its suitability for the intended application. A binary VSA designed solely with performance under consideration may be suboptimal or even unsuitable for specific hardware, especially with severe resource constraints.



	Memory(KB)	Latency(ms)
SVM	$\times 10^3 - 10^4$	$\times 10^2$
KNN	–	$\times 10^2$
QNN	$\times 10^3 - 10^4$	$\times 10^2$
VSA-H	$\times 10^3 - 10^4$	$\times 10^{-2}$
VSA-L	$\times 1 - 10$	$\times 10^{-3}$
UniVSA	$\times \mathbf{1 - 10}$	$\times \mathbf{10^{-2}}$

Fig. 1: Comparison between UniVSA, high-dimensional VSA (noted as VSA-H [9], [12]), low-dimensional VSA (LDC [11]), and other lightweight ML models (including QNN [13], BNN [14], SVM [15], and KNN [16]).

In this work, we address crucial challenges by proposing a universal binary VSA framework, namely *UniVSA*, highlighting its underlying algorithm-hardware co-optimization mechanisms. Importantly, our approach does not only involve algorithmic optimization to define an improved binary VSA model, but also specifies the hardware implementation based on the well-trained model. On the algorithm side, we emphasize a previously overlooked issue in binary VSA — the absence of interaction between features when generating vector representations. To remedy this, we introduce a binary feature extraction module that establishes relationships between features. Additionally, we enrich the binary VSA models with other efficient designs. In terms of hardware, we propose modules for UniVSA which consist of essential primitives and a controller for execution scheduling. In Fig. 1, we highlight the general comparison between UniVSA and other lightweight ML methods.

Our main contributions are three-fold as summarized below:

- **Algorithm-optimization: Model Design.** We propose an improved binary VSA model in our UniVSA framework. This model has more capability and flexibility in the

configuration to improve the inference accuracy.

- **Hardware-optimization: Implementation.** We specify modules to accelerate UniVSA. Implementing UniVSA employs parallelization and pipelining within critical modules to minimize hardware overhead. Meanwhile, resource usage is carefully managed to facilitate parallelism for reduced latency while preventing excessive power consumption.
- **Evaluation:** We evaluate UniVSA compared with other lightweight ML methods under resource-stringent scenarios. These results validate the algorithmic optimization of UniVSA, and demonstrate the lightweight and high efficiency of UniVSA hardware implementation.

II. Preliminaries

A. Binary Vector Symbolic Architecture

For a sample x with N features and M available values for each feature, binary VSA generates bipolar vector sets $\mathbf{F} = \{\boldsymbol{f}_i\} \in \{-1, 1\}^{N \times D}, i = [1, N]$ and $\mathbf{V} = \{\boldsymbol{v}_m\} \in \{-1, 1\}^{M \times D}$, $m = [1, M]$ to represent feature positions and values, respectively. Note for continuous values, they are discretized into M intervals to suit the VSA computing. The sample x is encoded as a bipolar vector $\boldsymbol{s}$ as

$$\boldsymbol{s} = \text{sgn}\left(\sum_{i=1}^{N} \boldsymbol{f}_i \circ \boldsymbol{v}_{x_i}\right) \tag{1}$$

where $\text{sgn}(\cdot)$ is the sign function to binarize the accumulation result. We set $\text{sgn}(0) = 1$ as a tiebreaker. A toy example of binary VSA encoding is illustrated in the upper part of Fig. 2, explaining how a sample is encoded as a bipolar vector.

B. Binary VSA on Classification Tasks

For a classification task, binary VSA generates a bipolar vector for each category (i.e., class). Assuming that there are C classes, the training of a binary VSA model is to derive a class vector set $\mathbf{C} = \{\boldsymbol{c}\} \in \{-1, 1\}^{C \times D}$. During inference, a query sample is encoded as binary vector $\boldsymbol{s}$ using Eq. 1. Then, the query sample vector will be compared with all class vectors, for which the one with the highest similarity will be determined as the predicted label:

$$\text{label} = \arg\max_{j} \ \text{SIM}(\boldsymbol{s}, \boldsymbol{c}_j) \tag{2}$$

$\text{SIM}(\cdot)$ can be any similarity measurements, where Hamming distance and dot-product are commonly used in binary VSA. In summary, binary VSA behaves as follows for a query input x:

$$x \xrightarrow{x_i \to \boldsymbol{v}_{x_i}} [\boldsymbol{v}_{x_1}, ..., \boldsymbol{v}_{x_N}] \xrightarrow{\text{sgn}(\sum \boldsymbol{f} \circ \boldsymbol{v})} \boldsymbol{s} \xrightarrow{\arg\max \mathbf{C}\boldsymbol{s}} \text{label} \tag{3}$$

The lower part of Fig. 2 demonstrates an example of the dot-product metric as the similarity measurement.

C. Low-Dimensional Training on Binary VSA

Considering the necessity of low dimension to implement binary VSA on resource-constrained devices, we explain the low-dimensional computing (LDC) strategy [11] for binary VSA training. LDC maps the binary VSA model (i.e., Eq. 1 and Eq. 2) onto a partial BNN, which is then optimized for vector generation:

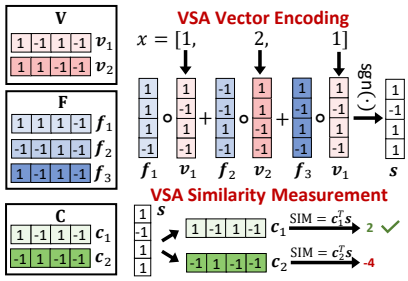


Fig. 2: Encoding x using binary VSA (upper) and measuring similarity between x and stored class vectors $\mathbf{C}$ (lower). Assume x has three features and two available values, thus $N = 3, M = 2$; there are two categories $C = 2$.

Value Projection. A *unique* neural network, namely *ValueBox* (VB), is proposed to project an feature value x_i to a bipolar vector $\boldsymbol{v}_{x_i}$, mimicking the look-up step "$x_i \to \boldsymbol{v}_{x_i}$" in Eq. 3. VB is composed of multi-layer perceptron (MLP) followed by binarization, i.e., $\boldsymbol{v}_x = \text{VB}(x) = \text{sgn}(\text{MLP}(x))$. During binary VSA implementation, $\mathbf{V} = \{\boldsymbol{v}_m\}$ is collected by evaluating all available values $m = [1, M]$ through $\text{VB}(m)$.

Vector Encoding. LDC interprets the vector encoding (i.e., the second step in Eq. 3) as a specially structured binary layer. In this view, $\boldsymbol{v}_x$ are regarded as the layer input, and $\mathbf{F} = \{\boldsymbol{f}_i\}$ are the binary weights.

Similarity Measurement. The equivalence of the Hamming distance and the dot-product is proved in [11]. Therefore, the similarity measurement can be achieved with a binary dense layer (at the third step in Eq. 3), where the binary weights correspond to the class vector set $\mathbf{C}$.

After optimizing this partial BNN, the feature vector set $\mathbf{F}$ and class vector set $\mathbf{C}$ are directly extracted from the binary layers, while the value vector set $\mathbf{V}$ is derived from VB. During inference, only $\mathbf{V}$, $\mathbf{F}$, and $\mathbf{C}$ are required in VSA process (Eq. 1 and 2); partial BNN is only utilized for training. Although LDC offers a strategy to reduce vector dimensions, binary VSA still falls short in resource-constrained applications due to *performance constraints* and *undeveloped design flow*.

III. UniVSA: Model Design

For resource-stringent scenarios, binary VSA should meet two critical requirements: ***ultra-lightweight*** (i.e., low dimension) and ***high accuracy***. We extend the binary VSA model in terms of these goals, with three-fold enhancements: ① Discriminated Value Projection – the features with less importance for classification can be represented with fewer elements; ② Binary Feature Extraction – a binary convolutional layer is introduced to construct interactions between features; ③ Soft Voting – ensemble strategy is embedded. We demonstrate the overview of our UniVSA model in Fig. 3.

A. Binary VSA Model under UniVSA Framework

We take the electrocorticography (ECoG) signal [17] as an example for the model input, which is one typical format

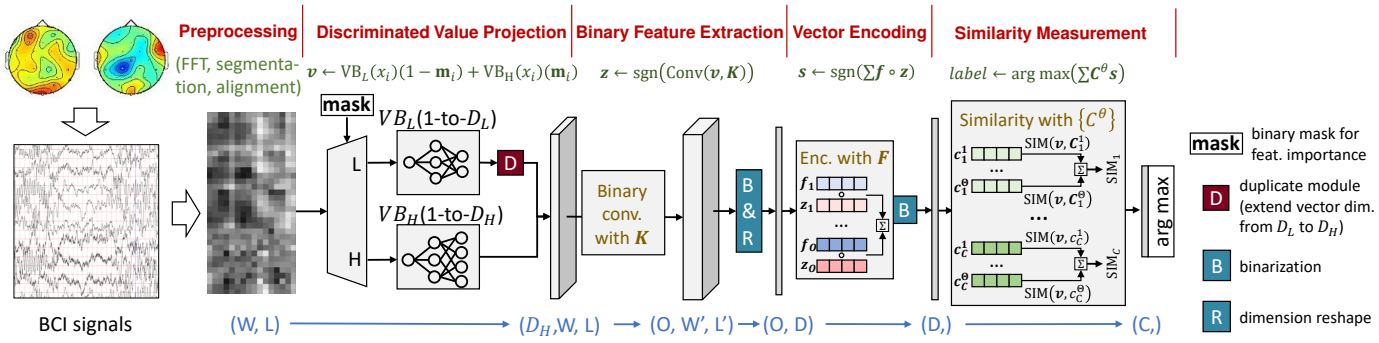


Fig. 3: The UniVSA model design. Symbols in parentheses show the vector/matrix dimensions for each stage.

of implanted BCI signals. Specifically, it is preprocessed and evenly divided into W sliding windows with overlap, where each window contains a signal snippet of length L [4], i.e., the input is shaped as (W, L), in total $N = W \times L$ features.

1) Discriminated Value Projection: Current binary VSA models consider all input features with the same importance. However, some time or frequency intervals of an ECoG signal are simply irrelevant or noisy information. We define a feature's importance as whether it has obvious impacts on accuracy for classification. Specifically, an input-wise binary **mask** is generated through the feature subset selection strategy [18]. The features corresponding to 1 in **mask** represent high importance, while others with 0 are low-importance. The input passes through a discriminator defined by **mask**, where values under high-importance features go to VB_H and under low-importance ones go to VB_L (in Fig. 3). Specifically, VB_L produces bipolar vectors of lower dimension than those from VB_H.

2) Binary Feature Interaction by Convolution: The features are independently encoded in current binary VSA models (Eq. 1), while interactions between features are overlooked; accordingly, we propose to use convolution to extract relationships between features. Given the strict resource and power constraints in resource-stringent scenarios, we employ binary convolution for the feature extraction on binary value vectors.

3) Vector Encoding: We follow the LDC training to apply the encoding procedure (Eq. 1). As aforementioned in Sec. II-C, the binary weights in this layer correspond to feature vector set $\mathbf{F}$. Yet, unlike other binary VSA models where $\boldsymbol{f}_i \in \mathbf{F}$ is generated for each feature position, the $\boldsymbol{f}_i$ in our UniVSA model represents the channel position of the binary convolution output. Consequently, the output of encoding $\boldsymbol{s}$ is the binary vector representation of the input ECoG signal.

4) Similarity Measurement with Soft Voting: In LDC training [11], the similarity measurement layer generates low-dimension class vectors, by optimizing a binary dense layer. However, more bias (also known as underfitting) will be introduced as the vector dimension decreases [19]. We upgrade this design, by using multiple binary dense layers in parallel, as an ensemble layer for classification. Therefore, a sample is evaluated by multiple similarity layers with average probabilities. Assuming Θ similarity layers, we have Θ sets of class vectors $\mathbf{C}$ generated. Eq.2 can be rewritten as

$$\text{label} = \arg\max_j \frac{1}{\Theta} \sum_{\theta=1}^{\Theta} \text{SIM}(\boldsymbol{s}, \boldsymbol{c}_j^\theta) = \arg\max_j \sum_{\theta=1}^{\Theta} \mathbf{C}^\theta \boldsymbol{s} \quad (4)$$

By jointly considering the similarity results of every dense layer, the output produces the category prediction on the class with the highest similarity.

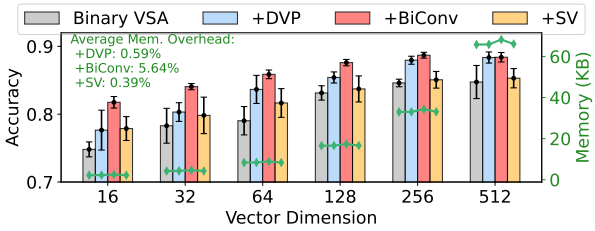


Fig. 4: The ablation study of the three-fold enhancements over binary VSA. The inference accuracy (bar) of binary VSA is significantly improved with little memory footprint (line).

B. Preliminary Assessment on UniVSA Model

We validate the positive impact of discriminated value projection (DVP), binary feature extraction (BiConv), and soft voting (SV) on inference accuracy and their memory footprint for resource-stringent scenarios, in Fig. 4. The evaluation is based on the EEGMMI dataset [20]. BiConv consistently improves the accuracy of binary VSA across all vector dimensions and help ensures a stable training process, evidenced by low accuracy deviations. This underscores *the necessity to incorporate feature interaction* in binary VSA models. DVP is less stable and initially underperforms BiConv when vector dimensions are low, but performs better as the vector dimension increases, even comparable to BiConv. SV underperforms the other two methods, yet still positively impacts binary VSA accuracy at lower vector dimensions, helping *alleviate underfitting issues* in low-capacity binary VSA models. By harnessing the unique strengths of these methods, UniVSA achieves a notable enhancement in binary VSA inference. In particular, the result of the memory footprint shows that these extensions have tiny overhead for the model size, i.e., $+0.59\%$ on DVP, $+5.64\%$ on BiConv, and $+0.39\%$ on SV, considering the overall kilobyte-scale memory requirement.

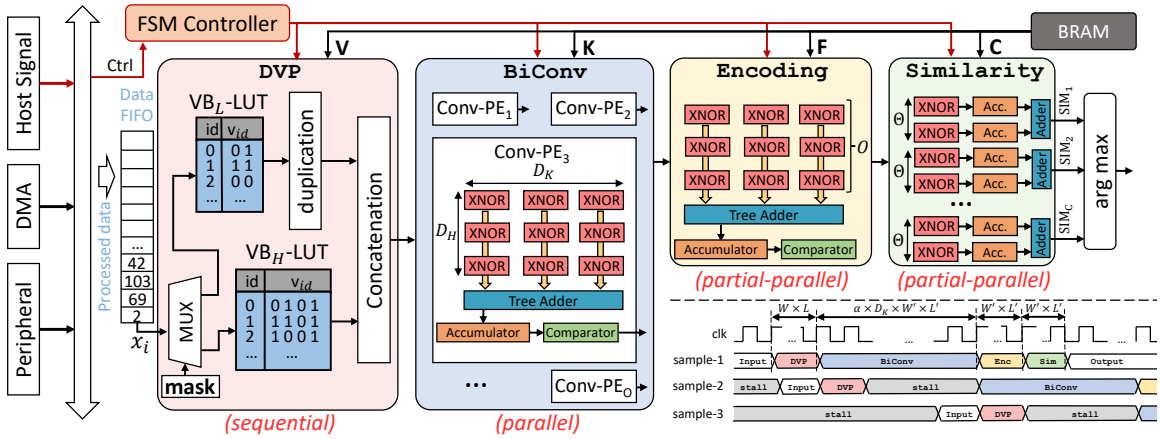


Fig. 5: The hardware overview of UniVSA. We indicate the execution scheduling under each module in parentheses, and demonstrate the overall pipelined scheduling of UniVSA for streaming inputs at the bottom right, where $\alpha = \max\{D_K, log_2 D_H\}$ is the execution time for a single convolution iteration.

IV. UniVSA: Hardware Implementation

A. Module Design

The hardware implementation of UniVSA contains four modules: **binary convolution, encoding, similarity measurement**, and **central controller**. Besides, we employ auxiliary circuits for **discriminated value projection** (DVP) and intermediate data buffering. We illustrate the hardware architecture in Fig. 5.

Discriminated Value Projection. The input features go through VB$_H$/VB$_L$ (decided by their feature importance **mask**) to derive the value vector for each feature. Since `DVP` operates significantly faster than `BiConv` (bottom right of Fig. 5). Parallelism on `DVP` would increase hardware overhead without reducing the latency of UniVSA. Thus, we do sequential execution on `DVP` to minimize resource usage and power consumption. A data FIFO is integrated at the `DVP` input to sequentially feed data elements.

Binary Convolution. The binary convolution layer extracts implicit features in the binary value vectors. As notation, value vectors have size (D_H, W, L) and the binary kernel $\mathbf{K}$ has size (O, D_H, D_K, D_K), where O is the output channel number and D_K is the kernel size. The binary convolution is accelerated by a *double buffering* strategy [21] to preload the next data block, without waiting for the accomplishment of current convolution. During convolution, we parallelize the computation by splitting kernels into O parts with size (D_H, D_K, D_K). Each part participates in computing to determine the binary value for one channel. Therefore, considering the output feature map with size (O, W', L'), $W' \times L' \times D_K$ iterations are required.

Encoding and Similarity Measurement. We flatten the output of binary convolution into shape $(O, W' \times L')$, to prepare for encoding. The encoding input is XNORed with feature vectors $\mathbf{F}$ and summed along the O-axis through an adder tree. The binarization is then performed to produce a sample vector $\boldsymbol{s}$ of dimension $W' \times L'$. Similarly, $\boldsymbol{s}$ is XNORed with Θ sets of class vectors $\mathbf{C}$ for the similarity measurement. For each category, the XNORed results are summed up to derive the final similarity. Only partial parallelism is applied on `Encoding` and `Similarity` along dimension O and Θ, to balance the vector operation and resource usage. Akin to `DVP`, their latency can be covered by `BiConv` under streaming inputs.

B. Hardware Consideration

Given the limited resource in tiny devices, we consider the **Memory** and **Resource** to quantify the hardware efficiency. The memory is determined by vector groups **V**, **F**, **K**, and **C**. With the input size (W and L), number of available values (M), number of categories (C), value vector dimension (D_H/D_L), kernel size (D_K) and output channel (O) in convolution, and number of voters (Θ), the required memory is

$$\begin{aligned}\textbf{Memory} = {} & M \times (D_H + D_L) + O \times D_H \times D_K^2 \\ & + W \times L \times O + W \times L \times \Theta \times C\end{aligned} \tag{5}$$

Since `BiConv` dominate resource usage (in Fig. 6), we mainly consider their resource for configuration. Without loss of generality, we define the resource overhead as

$$\textbf{Resource} \approx \beta \times D_K \times O \times D_H \tag{6}$$

where β is a coefficient indicating that the resource usage increment is approximately proportional to the kernel size and input/output channel of binary convolution.

We treat the memory footprint and resource usage as a penalty for UniVSA configurations, to balance the accuracy and the hardware overhead in the model design. The hardware overhead is normalized by taking the basis $\mathbf{M}_0$ and $\mathbf{R}_0$ with configuration $(D_H, D_L, D_K, O, \Theta, M) = (4, 2, 3, 64, 1, 256)$:

$$\mathcal{L}_{HW} = \lambda_1 \frac{\textbf{Memory}}{\mathbf{M}_0} + \lambda_2 \frac{\textbf{Resource}}{\mathbf{R}_0} \tag{7}$$

Here, λ_1 and λ_2 are scaling factors to align with the inference accuracy and differentiate these two hardware overheads.

V. Evaluation

A. Experiment Setup

Datasets. We evaluate the UniVSA framework on real-world BCI classification tasks and VSA tasks (in Table I):

TABLE I: Benchmark configurations.

	Domain	# of Classes	Input Size (W, L)	Model Config. $(D_H, D_L, D_K, O, \Theta)$
EEGMMI	Time	2	(16,64)	(8, 2, 3, 95, 1)
BCI-III-V	Frequency	3	(16,6)	(8, 1, 3, 151, 3)
CHB-B	Frequency	2	(23,64)	(8, 2, 3, 16, 3)
CHB-IB	Frequency	2	(23,64)	(4, 1, 5, 16, 1)
ISOLET	Time	26	(16,40)	(4, 4, 3, 22, 3)
HAR	Time	6	(16,36)	(8, 4, 3, 18, 3)

EEGMMI [20], BCI-III-V [22], CHB-B [23], CHB-IB [23], ISOLET [24], HAR [25]. The first four datasets are collected in electroencephalography (EEG) signals for BCI devices. EEGMMI records volunteers' activities from 64 channels, and BCI-III-V is collected for mental imagery activities. We follow the preprocessing in [26] and [22] for each dataset, respectively. The CHB (balanced and imbalanced version) dataset is for seizure detection. In addition, we include two tasks commonly evaluated for VSA models. ISOLET is a voice recording of 26-letter speaking, and HAR is an activity gesture recording from accelerometers and gyroscopes. Inputs for UniVSA are discretized to 256 levels in advance [11], i.e., $M = 256$, and shaped as 2-D of size (W, L).

Model Design. We determine the configuration of UniVSA model by considering both the accuracy and hardware overhead, i.e., $obj = (Acc - \mathcal{L}_{HW})$ with $\lambda_1 = \lambda_2 = 0.005$. The optimization procedure on configuration can also be referred to [27]. To search for the optimal configuration for each task, we utilize the evolutionary search with elitist preservation [28]. In Table I, we demonstrate the optimal searched configurations to all benchmarks. The UniVSA models are trained on Python 3.7 with NVIDIA A10 GPU acceleration.

Hardware Implementation. We construct the UniVSA on Ultra96-V2 SoC equipped with a ZU3EG FPGA [29]. Communication of control signals, data input, and final output between the CPU and the FPGA is conducted through `AXI_HPM_LPD` [30] interface. To optimize resource usage and scheduling, we develop the hardware implementation in Verilog using Vivado 2022.2.

B. Performance of Software Design

In Table II, we present the comparison of the accuracy and memory footprint between UniVSA and other efficient methods, including SOTA high-dimensional binary VSA training strategies (i.e., LeHDC [12] with $D = 10,000$), low-dimensional binary VSA training (i.e., LDC with $D = 128$ [11]), LDA (32-bit float), SVM (16-bit float with RBF kernel), and KNN (K=5). Although deep learning approaches were also explored, we exclude dense comparisons with them because of their failure on resource and power budget [2].

Among traditional ML models, SVM generally has better performance, notwithstanding its significantly larger model size ($\times 255$ on LDA). Although SVM is a favored method for BCIs [4], it could perform poorly on certain tasks, such as BCI-III-V and HAR, according to our evaluations. On the other hand, binary VSA models trained by LDC can have

TABLE II: Model comparison between UniVSA and other lightweight methods on *accuracy* and *memory*. Memory footprint (KB) is shown in parentheses.

	LDA	**KNN**	**SVM**	**LeHDC [12]**	**LDC [11]**	**UniVSA**
EEGMMI	0.7004 (8.19)	0.8262 (−)	0.8766 (11223.04)	0.7980 (1602.50)	0.8279 (16.54)	0.8971 (13.59)
BCI-III-V	0.8599 (1.15)	0.9888 (−)	0.8971 (510.22)	0.8235 (443.75)	0.9370 (1.71)	0.9545 (3.57)
CHB-B	0.9067 (11.78)	0.9744 (−)	0.9819 (1990.14)	0.8992 (2162.50)	0.9669 (23.71)	0.9774 (4.51)
CHB-IB	0.9142 (11.78)	0.9488 (−)	0.9729 (3612.29)	0.8675 (2162.50)	0.9639 (23.71)	0.9684 (3.67)
ISOLET	0.9410 (66.56)	0.9140 (−)	0.9602 (5048.32)	0.9489 (1152.50)	0.9133 (10.78)	0.9282 (8.36)
HAR	0.7625 (13.82)	0.5582 (−)	0.7852 (6743.81)	0.9523 (1047.50)	0.9256 (9.44)	0.9338 (3.14)
average	0.8475 (18.88 KB)	0.8685 (−)	0.9124 (4.24 MB)	0.8816 (1.29 MB)	0.9225 (15.05 KB)	0.9445 (8.31 KB)

comparable SVM inference performance on average while consuming significantly less memory, i.e., 0.5% of the memory required by SVM. However, LDC underperforms SVM on some tasks, e.g., 4.87% lower on EEGMMI and 4.69% lower on ISOLET, echoing the accuracy challenges in Sec. I. In addition, high-dimensional binary VSA models (LeHDC) achieve better accuracy on certain tasks, yet require memory on the MB scale.

Our proposed UniVSA framework on average outperforms other ML methods in terms of accuracy. When compared to LDC, UniVSA shows superior accuracy across all tasks, underscoring its improved model design and inference performance. Additionally, UniVSA's design strategy, which considers hardware overhead, results in a smaller memory footprint than LDC for most evaluated tasks. Notably, the memory footprint of UniVSA on CHB tasks is much lower than that of LDC. This is because the memory overhead (Eq.5) of binary VSA is dominated by the size of feature vectors $\mathbf{F}$, i.e., $(O, W \times L)$ for UniVSA and $(D, W \times L)$ for LDC, where the searched $O = 16$ in UniVSA is much smaller than $D = 128$ in binary VSA trained from LDC. While UniVSA does not always outperform all other ML methods, with SVM frequently delivering the highest performance, it does offer consistently high accuracy. The performance of SVM, however, is task-dependent and it underperforms on certain tasks. In contrast, UniVSA generally delivers close-to-best results, except for the ISOLET task, demonstrating its generality in lightweight classification tasks.

C. Performance of Hardware Implementation

Framework Comparison. We compare the hardware implementation of UniVSA with ML methods including SVM [32] and KNN [16], and SOTA binary VSA hardware implementations, LookHD [9] with $D = 2000$ and LDC [11] with $D = 64$. Additionally, we consider lightweight NN approaches, i.e., BNN [14]/QNN [13]. We choose the comparison by the closest FPGA architecture in the previous work so that the lowest variations in resource allocation and power consumption are induced during implementation. The results are summarized in Table III.

① Compared with conventional ML methods and NN models, UniVSA can achieve much lower hardware overhead, with only

TABLE III: The hardware performance of UniVSA against other models. Values in parentheses are estimated. ISOLET is selected for UniVSA since it has the closest input size to other binary VSA models.

	FPGA Arch.	Input Size / Classes	Frequency (MHz)	Memory (KB)	Latency (ms)	Power (W)	LUTs ($\times 10^3$)	BRAMs	DSPs
SVM [31]	Virtex-5	(20,20) / –*	84	(406)	14.29	3.2	31.85	131	59
KNN [16]	Stratix IV	64 / 2	131.42	–	69.12	24	135	–	80
BNN [14]	Zynq-ZU3EG	(3,32,32) / 10	250	–	(0.36)	4.1	51.44	212	126
QNN [13]	Zynq-ZU3EG	(3,224,224) / 1000	250	(1450)	(24.33)	5.5	51.78	159	360
LookHD [9]	Kintex-7	617 / 26	200	(165)	–	(9.52)	165	175	807
LDC [11]	Zynq-ZU3EG	784 / 10	200	6.48	0.004	(0.016)	0.75	5	1
UniVSA	Zynq-ZU3EG	(16,40) / 26	250	8.36	0.044	0.11	7.92	1	0

* The SVM implementation is evaluated on a detection task, not classification.

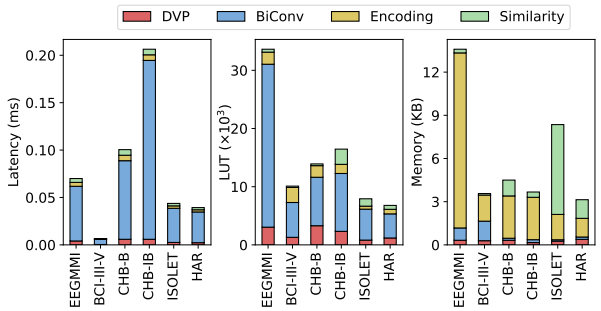


Fig. 6: The hardware overhead of each stage in UniVSA.

TABLE IV: The hardware performance of UniVSA on all tasks. The throughput is estimated for streaming inputs to UniVSA.

Benchmark	Latency (ms)	Power (W)	LUTs ($\times 10^3$)	BRAMs	DSPs	Throughput ($\times 10^3$)
EEGMMI	0.070	0.45	33.62	3	0	17.34
BCI-III-V	0.007	0.18	10.10	1	0	184.84
CHB-B	0.100	0.34	13.92	1	0	12.06
CHB-IB	0.206	0.21	16.46	1	0	5.30
ISOLET	0.044	0.11	7.92	1	0	27.78
HAR	0.039	0.10	6.78	1	0	30.85

0.1∼0.5× resource usage. UniVSA also consumes order-of-magnitude lower power and latency than these ML methods. On the other hand, as indicated in an FPGA-based SVM survey [15], all the collected SVMs have power consumption greater than **1.5W**. Taking into account this solid line, except for low-dimensional binary VSA (i.e., LDC) and UniVSA, other models in Table III are not suitable for BCI tasks. Note that BNN and QNN possibly have better inference performance than binary VSA models, especially on complex classification tasks. However, we propose UniVSA as a binary VSA framework to conduct easy classification tasks while requiring ultra-high efficiency, where UniVSA can achieve comparable inference accuracy and a minimal hardware budget.

② Compared with other binary VSA work, such as LookHD [9] and LDC [11], UniVSA shows comprehensive superiority over LookHD, yet consumes more power and resources than LDC. Nevertheless, considering that UniVSA can achieve better accuracy and lower memory footprint (Table II), and the resource usage is still acceptable (less than SVM [32]) by BCIs, we account for this as a feasible trade-off to better support classification tasks on resource-stringent devices.

Ablation Study. We further demonstrate the hardware overhead of the computing stages in UniVSA, in Fig. 6. The `BiConv` layer consumes the most resources and execution time in all tasks, far more than other stages. This supports our previous discussion that `BiConv` dominates the UniVSA execution, which is the motivation that we sequentialize the `DVP`, `Encoding`, and `Similarity` stages to minimize the resource usage, rather than optimize their latency using parallelism. Moreover, the memory footprint of `BiConv` is very low, since only the kernel $\mathbf{K}$ is stored with a small dimension (O, D_H, K, K); yet, $\mathbf{F}$ with size $(O, W \times L)$ or $\mathbf{C}$ with size $(\Theta, C, W \times L)$ occupies most of the memory footprint when the input size or classes is large.

Hardware Performance on Benchmark. We provide the hardware performance for all tasks in Table IV. Overall, all tasks demonstrate power consumption of less than 0.5W and latency under 0.2ms on the FPGA. The throughput analysis under streaming inputs demonstrates the efficiency of UniVSA, that all tasks can achieve over 5,000 throughput, sufficient for BCI applications. With pipeline, the execution time is close to the `BiConv` latency. Comparison between tasks reveals a strong correlation between hardware overhead and factors such as data input size and convolution channels. This finding bolsters our co-optimization strategy for UniVSA model design, taking hardware into consideration.

VI. Conclusion

In this paper, we present an end-to-end algorithm-hardware co-optimization framework, UniVSA, solving the challenges in current binary VSA design on resource-stringent devices. UniVSA advances the binary VSA by model designing and hardware acceleration with a completed process. Our evaluation demonstrates the effectiveness of UniVSA on benchmarks selected under resource-stringent scenarios, achieving the best inference accuracy on average and lowest memory footprint against other ML models. Further evaluation also demonstrates the efficiency and lightweight of UniVSA.

Acknowledgment

Shijin Duan, Nuntipat Narkthong, and Xiaolin Xu were supported in part by the US NSF under Grants CNS-2326597, CNS-2239672, and a Cisco Research Award. Shaolei Ren was supported in part by the US NSF under CNS-2326598.

REFERENCES

[1] L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 2021.

[2] S. Aggarwal and N. Chugh, "Review of machine learning techniques for eeg based brain computer interface," *Archives of Computational Methods in Engineering*, vol. 29, no. 5, pp. 3001–3020, 2022.

[3] Z. Lv, L. Qiao, Q. Wang, and F. Piccialli, "Advanced machine-learning methods for brain-computer interfacing," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 5, pp. 1688–1698, 2020.

[4] I. Karageorgos, K. Sriram, J. Veselỳ, M. Wu, M. Powell, D. Borton, R. Manohar, and A. Bhattacharjee, "Hardware-software co-design for brain-computer interfaces," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 391–404.

[5] S. Kumar, A. Sharma, and T. Tsunoda, "An improved discriminative filter bank selection approach for motor imagery eeg signal classification using mutual information," *BMC bioinformatics*, vol. 18, pp. 125–137, 2017.

[6] G. Qiao, S. Hu, T. Chen, L. Rong, N. Ning, Q. Yu, and Y. Liu, "Stbnn: Hardware-friendly spatio-temporal binary neural network with high pattern recognition accuracy," *Neurocomputing*, vol. 409, pp. 351–360, 2020.

[7] 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.

[8] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, pp. 1–11, 2020.

[9] 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)*. IEEE, 2021, pp. 221–234.

[10] 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.

[11] S. Duan, X. Xu, and S. Ren, "A brain-inspired low-dimensional computing classifier for inference on tiny devices," in *tinyML Research Symposium 2022*, 2022.

[12] S. Duan, Y. Liu, S. Ren, and X. Xu, "Lehdc: Learning-based hyperdimensional computing classifier," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1111–1116. [Online]. Available: https://doi.org/10.1145/3489517.3530593

[13] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek *et al.*, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 23–32.

[14] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, "Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 171–182.

[15] S. Afifi, H. GholamHosseini, and R. Sinha, "Fpga implementations of svm classifiers: A review," *SN Computer Science*, vol. 1, pp. 1–17, 2020.

[16] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 167–170.

[17] A. H. Shoeb and J. V. Guttag, "Application of machine learning to epileptic seizure detection," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 975–982.

[18] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[19] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[20] P. PhysioBank, "Physionet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000, https://archive.physionet.org/pn4/eegmmidb/.

[21] X. Tan, X.-W. Shen, X.-C. Ye, D. Wang, D.-R. Fan, L. Zhang, W.-M. Li, Z.-M. Zhang, and Z.-M. Tang, "A non-stop double buffering mechanism for dataflow architecture," *Journal of Computer Science and Technology*, vol. 33, pp. 145–157, 2018.

[22] J. R. Millan, "On the need for on-line learning in brain-computer interfaces," in *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, vol. 4. IEEE, 2004, pp. 2877–2882, https://www.bbci.de/competition/iii/desc_V.html.

[23] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," 2009, https://archive.physionet.org/pn6/chbmit/.

[24] D. Dua and C. Graff, "UCI machine learning repository," 2017, https://archive.ics.uci.edu/ml/datasets/isolet.

[25] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." p. 3, 2013, https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones#.

[26] X. Zhang, L. Yao, X. Wang, J. Monaghan, D. Mcalpine, and Y. Zhang, "A survey on deep learning-based non-invasive brain signals: recent advances and new frontiers," *Journal of neural engineering*, vol. 18, no. 3, p. 031002, 2021. [Online]. Available: https://github.com/xiangzhang1015/Deep-Learning-for-BCI

[27] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[28] D.-C. Dang, T. Friedrich, T. Kötzing, M. S. Krejca, P. K. Lehre, P. S. Oliveto, D. Sudholt, and A. M. Sutton, "Escaping local optima using crossover with emergent diversity," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 484–497, 2017.

[29] Xilinx, "Zynq ultrascale+ mpsoc data sheet: Overview," 2021. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[30] XILINX. (2017) Vivado axi reference guide. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide

[31] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2013, pp. 129–136.

[32] M. Qasaimeh, A. Sagahyroon, and T. Shanableh, "Fpga-based parallel hardware architecture for real-time image classification," *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56–70, 2015.