Low-Power Hyperdimensional Computing Processors for Real-Time Wearable Sensor Fusion and Keyword Classification

Daniel Sun

Acknowledgement

I would like to thank Alisha Menon for seeing potential in a wide-eyed undergraduate, and shaping my research experience at Berkeley with invaluable support in each project I have undertaken. I'd also like to thank Melvin, who's remarkable ideas during the first semester helped kickstart a journey of astounding discoveries down the line. I would like to thank Kyoungtae for his collaboration with the SVM project, Dennis and Anirudh for their ideas in TinyML, Sarina for her help in the TBiocas project, as well as Professor Jan Rabaey for the opportunity to be a part of this research space in the first place. Finally, I would like to thank my family and friends for their support at each step, and defining my time during the 5th year master's program with each experience along the way.

Low-Power Hyperdimensional Computing Processors for Real-Time Wearable Sensor Fusion and Keyword Classification

by Daniel Sun

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Jan Rabaey
Research Advisor

5/13/2022

(Date)

* * * * * * *

[illegible]

Professor Sophia Shao
Second Reader

5/11/2022

(Date)

Abstract

Low-Power Hyperdimensional Computing Processors for Real-Time Wearable Sensor Fusion and Keyword Classification

by

Daniel Sun

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Jan Rabaey, Chair

Low-power machine learning techniques on hardware are increasingly vital and important for a wide range of applications ranging from wearables such as smartwatches to devices on the edge. Hyperdimensional Computing (HDC) is one proposed algorithm that has been consistently proven to demonstrate high accuracy with minimal power consumption, across diverse classification tasks such as gesture recognition [20] or speech recognition [12]. HDC processors have been implemented in the past, however their energy-efficiency has largely been limited by the costly hypervector memory storage, which grows linearly with the number of input features or sensors. In this work, we propose a novel method of combining HDC Sensor Fusion with CA *rule 90* in conjunction with vector folding, to reduce the memory requirement of this processor to near 0, reaching an energy efficiency of 39.1 nJ/prediction; a 4.9x energy efficiency improvement, 9.5x per channel, over the state-of-the-art HDC processor. This processor is also compared with an analogous SVM implementation, demonstrating a 9.5x energy efficiency improvement over SVM when scaling to a large number of 214 channels. The energy-efficiency and scalability of HDC is testament to its applicability on a broad range of low-power machine learning tasks today. As such, exploration of HDC on other classification tasks such as TinyML Keyword Spotting is also performed, as part of an ongoing effort to pave the way for HDC to become the paradigm of choice for high-accuracy, low-power, and real-time classification tasks.

Contents

Acknowledgments

There are many people for whom this research would not have been the same without. I would like to thank Alisha Menon for seeing potential in a wide-eyed undergraduate student, and shaping my research experience at Berkeley with invaluable support and advice in every one of the projects I have undertaken. I'd also like to thank Melvin, who's remarkable ideas and work ethic during the first semester helped kickstart a journey of astounding discoveries and findings down the line. I would like to thank Kyoungtae for his collaboration with the SVM project, Dennis and Anirudh for their ideas in TinyML, Sarina for her help in the TBiocas project, as well as Professor Jan Rabaey for the opportunity to be a part of this groundbreaking research space in the first place.

Finally, I would like to thank my family and friends for their support at each step, and defining my time during the 5th year master's program with each experience along the way.

Chapter 1

Introduction

With the rising popularity of machine learning, many efforts in training and inference have shifted towards investigating how traditionally large machine learning classifiers can be optimized to execute on more power-constrained and energy-efficient devices. Across a wide range of applications, the need for such low-power optimizations is increasingly pertinent. These include applications ranging from keyword spotting in edge devices such as Amazon Echo, to seizure detection implemented in wearable devices such as smartwatches.

The benefit is not just power efficiency; the ability to perform classification real-time, on-device, and near-sensor allows for a reduced overhead in transferring data across storage or the cloud, as well as improving latency, autonomy, and security of the task [4]. In cases such as Keyword Spotting, the always-on requirement of the application means that the device must be energy-efficient while idly waiting to hear a matching keyword, and also have a low-latency response whenever a valid keyword is detected. In other cases such as with biosignal monitoring devices, on-board classification of sensor information eliminates the costly transmission of raw data streams, reducing the wireless transmission power consumption to just a classification, regardless of the number of sensors.

One emerging algorithm that bears promises of combining competitive classification performance with minimal energy consumption is Hyperdimensional Computing (HDC) [14]. The inspiration for HDC lies in the idea that human brains do not detect patterns using scalar arithmetic, but instead, through processing large patterns of neural activity. Analogously, HDC maps input information into very wide pseudo-random binary hypervectors (HVs), with dimensions often ranging into the thousands. These hypervectors are then spatially and temporally encoded into a large hyperdimensional space for classification [14].

The encoding process of HDC is based on three key binary operations: bundling (bitwise majority count), binding (bitwise XOR), and permutation (cyclical shift) [14]. The simplicity of these operations, in addition to its inherent binary representation, is suggestive of the potential power-efficiency of HDC.

HDC has been proven to demonstrate superior classification performance over other state-of-the-art algorithms on a variety of applications, ranging from seizure detection [5], gesture recognition [20], language classification [17], and speech recognition [12]. HDC has also performed with high classification accuracy under constraints such as limited training data [6], one-shot learning [21], and rapid model updates [20]. These properties are useful for physiological sensors where the training data is user-specific and the sensors are vulnerable to displacement between each use or during extended usage.

One particular property that HDC adapts very well to, is maintaining energy efficiency as the number of input features scale to large quantities. When performing biosensor classification with signals such as EMG and EEG, arrays of sensors are used to collect spatial information [20]. Then, the data from each sensor undergoes feature extraction, resulting in several different features per individual sensor. This results in a large set of features, or channels, over which classification occurs in order to recognize a physiological behavior. In this work, the scalability and flexibility of HDC is demonstrated on emotion recognition with the AMIGOS dataset, containing a grand total of 214 input channels.

Needless to say, HDC is not the only machine learning algorithm capable of performing classification on a wide variety of datasets. Algorithms such as SVM and XGB have also been demonstrated on tasks such as emotion recognition in the past [18]. However, as the number of input channels increase, HDC remains more efficient than other algorithms such as SVM, when implemented on hardware [19]. In this work, an analogous SVM emotion recognition classifier was implemented, and its energy-efficiency compared with the HDC implementation across a range of input channels. At 214 channels, HDC is 9.5x more energy-efficient per channel than SVM for the same emotion recognition task.

Lastly, this work also strives to expand the traditional realm of HDC classification into other popular low-power machine learning communities today. In particular, we attempt to apply HDC towards the Keyword Spotting benchmark, one of the 4 TinyML machine learning benchmarks [1]. There is a lot of space to experiment with different ways to perform feature exploration for this application, as this can heavily impact the resulting accuracy of the classifier. Various tradeoffs in feature exploration and optimization are discussed and implemented in this work as well.

1.1 Dataset and Application

In this work, we demonstrate the energy-efficiency of HDC on multi-modal biosensor emotion classification with the AMIGOS dataset developed by [24]. In this dataset, 32 GSR (Galvanic Skin Response), 77 ECG (Electrocardiogram), and 105 EEG (Electroencephalogram) features were extracted [28]. Algorithmically, these signals were measured for 33 subjects as they watched 16 videos. For each video, the 214 channels of physiological features were measured to classify the subject's reaction into one of four corners on the valence (positive vs. negative emotion) and arousal (alertness) axes [22]. A single 1-bit valence and arousal signal, either high or low, is generated per set of 214 features, representing the resulting emotion classification.

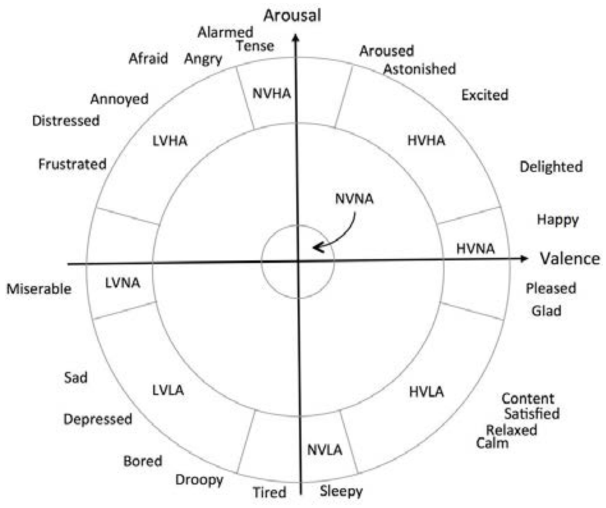


Figure 1.1: Emotion Circumplex Model [3]

Previous work has been done on the AMIGOS dataset using other machine learning algorithms such as Fisher's linear discriminant with Gaussian Naive Bayes [7], extreme learning machines [23], and SVM [28]. However in all cases, HDC achieves the highest arousal and valence accuracy (83.8% average) compared to aforementioned algorithms such as SVM (67.15% average), and XGB (74.25% average) [18]. Given its telling performance, an HDC sensor fusion processor was implemented on an ASIC based on the algorithms in this section, and its details and optimizations are described in the following chapter.

The HDC architecture was specifically based upon many of the HDC algorithmic optimization described in [18]. For starters, the hypervector dimension D could be as small as 2000 bits with little degradation in accuracy (typical HDC applications can have datapath widths up to 10000 bits). The ngram N in the temporal encoder was also tuned to an optimal of 3. Input GSR, ECG, and EEG signals only need to be quantized to 3 discrete values {-1, 0, 1}, significantly simplifying the amount of CiM vector storage required. In fact, the same set of feature HVs (CiMs) can be used across all channels and sensor modalities, making the total number of hypervectors necessary for storing feature representation constant regardless of channel count. This is possible because pseudo-orthogonality is maintained through the identification HVs (iMs) which are unique for each channel. Lastly, CA *rule 90* can be used to generate each of the iM vectors in real time, reducing the hardware storage of these vectors to near 0.

Chapter 2

HDC Sensor Fusion ASIC

HDC lends well to hardware implementations given its fundamental binary representation and simplistic operations. Several implementations of HDC accelerators and processors have been demonstrated in the past, such as those from Datta et al. [8] and Eggiman et al. [9]. However, the energy efficiency of the processor from [8] is limited when scaling to a large number of channels. In the Hyper-Dimensional Processor designed by [8], each input data sensor channel requires a unique pseudo-orthogonal channel identification HV (iM). These iMs are stored in 256 instances of 1024×8 ROMs, which account for 42% of the total power utilization. This energy footprint can be largely eradicated with the use of real-time iM generation techniques.

Eggiman et al. from [9] alleviates this problem by generating the channel identification HVs and the adjustably pseudo-orthogonal feature vectors with a similarity manipulator during the encoding process, leading to a 3x energy-efficiency improvement in post-layout simulation over [8]. However, the pseudo-random vector generation process is complex, and still requires iterative permutation with N 4-input and N 2-input multiplexers, where N is the datapath width.

This work presents an HDC architecture implemented with CA *rule 90*, a form of hypervector generation that requires virtually no memory or logic at all. The high level diagram of this proposed processor is shown in Figure 2.1.

2.1 CA *rule 90*

CA *rule 90* is a method for generating hypervectors on the fly [15]. The CA, or Cellular Automaton, consists of a grid of cells arranged in a specific pattern, over which a set of

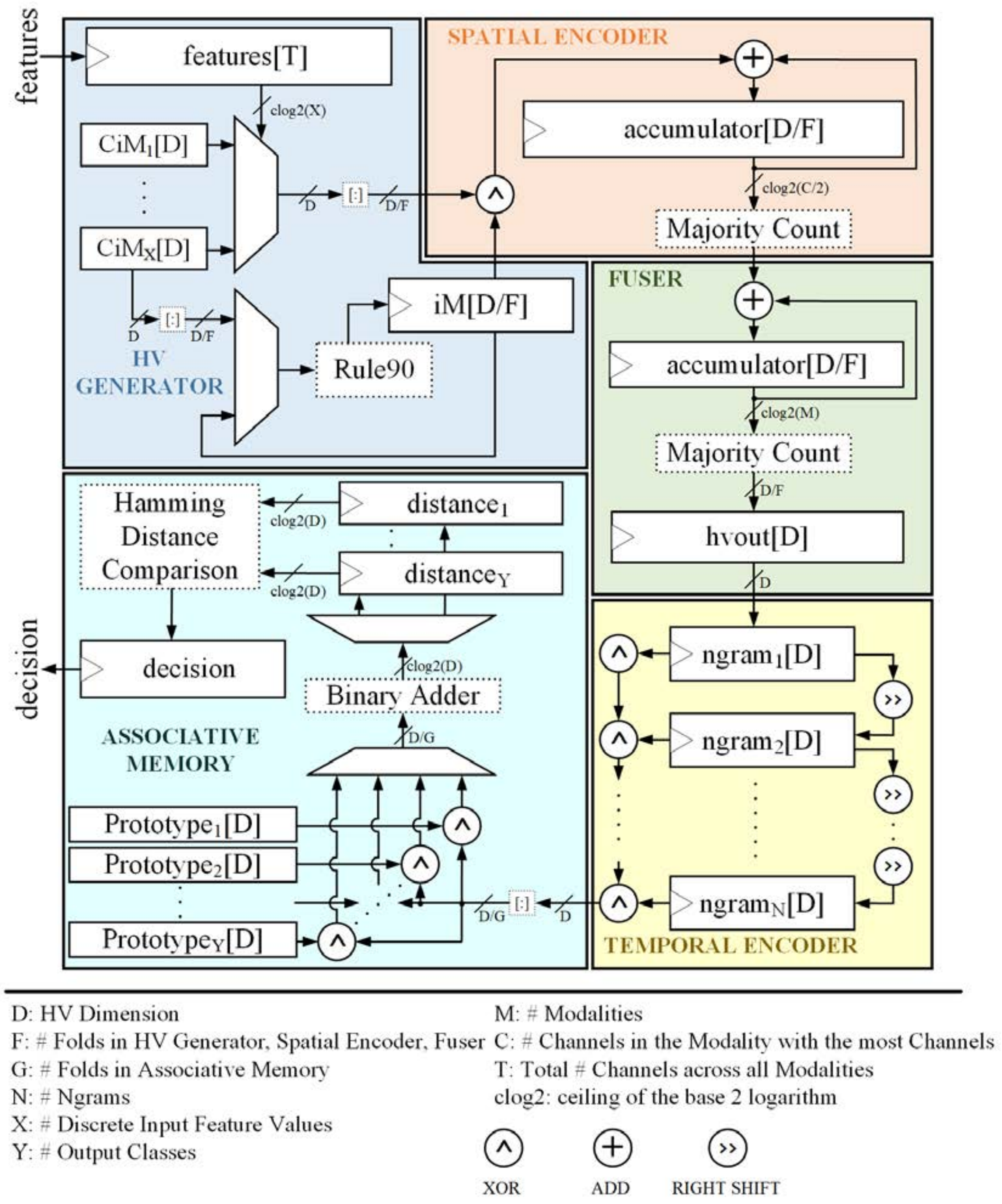


Figure 2.1: HDC Processor with CA *rule 90* hypervector generation

discrete computations update each cell using the state of the current and nearby cells [16]. In CA *rule 90*, the formula for updating the cells is the simple definition of an XOR of the two nearest cells. In the HDC implementation, a single seed vector is needed. By permuting the

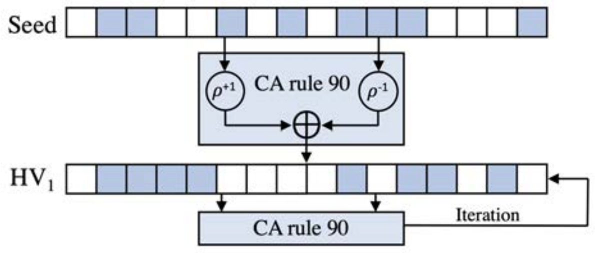


Figure 2.2: CA *rule 90*

seed vector with a left and right shift, and then an XOR across the resulting vectors, a new pseudo-orthogonal vector can be generated. With a seed vector of just 25 bits, new degrees of freedom, or new pseudo-orthogonal vectors, can be continuously generated for more than 10^3 iterations before saturating. For the majority of HDC applications, this would be sufficient.

2.2 Block by Block

The HDC Sensor Fusion ASIC consists of 5 blocks: *HV Generator*, *Spatial Encoder*, *Fuser*, *Temporal Encoder*, and *Associative Memory*.

HV Generator

This block maps the 214 input GSR, ECG, and EEG features into the hyperdimensional space. Each of the 214 channels is assigned a unique pseudo-random item memory (iM) vector.

For mapping discrete features, a continuous item memory (CiM) is used. Typically, the CiM is comprised of a collection of hypervectors that gradually increase in hamming distance from one end of the discretization range to the other. In this application however, each feature is only discretized to {-1, 0, 1}, so only 3 CiM vectors are needed. These are stored as tie-high/tie-low cells.

In this architecture, the iM HVs are generated on-the-fly instead of being pre-generated or stored in memory. This is done using CA *rule 90*, where the CiM vector corresponding to feature value {-1} is used as the seed vector to generate all subsequent iM vectors for each channel. Only a single iM vector is stored for the current channel and a new one is generated every cycle. As a result, increasing the number of input channels only affects the latency of the *HV Generator* computation, and not the amount of storage required. For sequential channel access patterns, each additional channel adds a cycle to the computation, while random channel access patterns will require reinitializing from the seed vector and iterating until the specified channel is reached.

Finally, each channel's iM vector is binded with the corresponding CiM vector to map the feature into hyperdimensional space.

Spatial Encoder

The *Spatial Encoder* (SE) accumulates the mapped hypervectors from the *HV Generator* across all channels. To minimize leakage power, only one channel is accumulated per cycle. Once accumulation of all channels in a sensor modality is completed, a majority count is performed to generate a single HV for the modality. Given the largest modality EEG of 105 features, a saturating counter of 6 bits is set within the accumulator.

Fuser

The *Fuser* bundles the HV from each modality to provide an equal contribution from each of the modalities: GSR, ECG, and EEG. The resulting output HV is an observation across all channels.

Temporal Encoder

The *Temporal Encoder* (TE) takes in the *Fuser* output HV as an ngram. A sequence of ngrams are stored, where each ngram is a permutation of the previous one. The ngrams are binded to encode and recognize temporal patterns in the time-series data. In this implementation, an *ngram* of three is used to track changes in features over every three time samples.

Associative Memory

The *Associative Memory* (AM) stores prototype vectors for each class, in this case, high and low valence and arousal. The *Temporal Encoder* output HV is compared against each prototype vector using Hamming Distance to generate a decision. The class with the least distance is predicted.

2.3 Vector Folding

One major component of HDC architecture design space exploration is evaluating the effect of vector folding. The HDC algorithm inherently involves processing vectors of very large dimensions. As a result, control signals managing these wide hypervectors will often have a large fanout, which can lead to a large amount of buffer insertion during place & route in order to meet timing requirements. The presence of many buffers can also lead to more leakage power than desired.

To alleviate this situation, the HDC datapath can be folded to a fraction of the hypervector dimension, reducing the number of registers in the datapath, and decreasing buffer insertion. This will result in a decrease in both leakage power and area, while introducing a cost in latency. Given the general assumption that biosignals only need to be sampled once every few milliseconds, the cost in latency is not of great concern.

In the proposed architecture, a hypervector of dimension D is divided into F folds in the $HV\ Generator$, $Spatial\ Encoder$, and $Fuser$, and G folds in the $Associative\ Memory$.

The clock frequency can be adjusted depending on the fold factor, in order to allow a classification to finish within a target latency. For example, Table 2.1 shows the required clock period for a classification target of 1 ms. The effect is that as the number of folds increases, the dynamic power increases from a higher switching frequency, while the leakage power decreases from the use of a smaller datapath width. The theoretical optimal fold factor where power dissipation is minimized can be found where the switching and leakage power values cross. This optimal depends on the target classification latency, as seen in Figure 2.4.

Accuracy

When implementing vector folding, the three CiM vectors remain the same, but each iM vector is now generated one fold at a time, as follows. The first D/F section of the {-1} CiM vector is used to generate the initial fold F_1 of the first iM. The first fold of each subsequent iM is then generated by simply applying CA *rule 90* iteratively. This process is then repeated for the subsequent folds of each iM, one after another. Ultimately, only a single iM fold of dimension D/F needs to be stored in hardware, minimizing the footprint of iM generation even further. However as the width of the CA reduces, the number of possible pseudo-random vectors the CA can generate also scales linearly downwards. As seen in 2.3, for a fold width of less than 20 bits, equivalent to 100 folds, the classification accuracy degrades to a random coin flip. This is consistent with the findings of [15], which found that the number of degrees of freedom sharply decreases with a CA grid size below 20; 20 bits provides fewer than 10 pseudo-random iterations, while even 25 bits provides more than 1000. The optimal fold factor is typically well above a fold width of 20, as such, the accuracy is not affected for the power-optimized design.

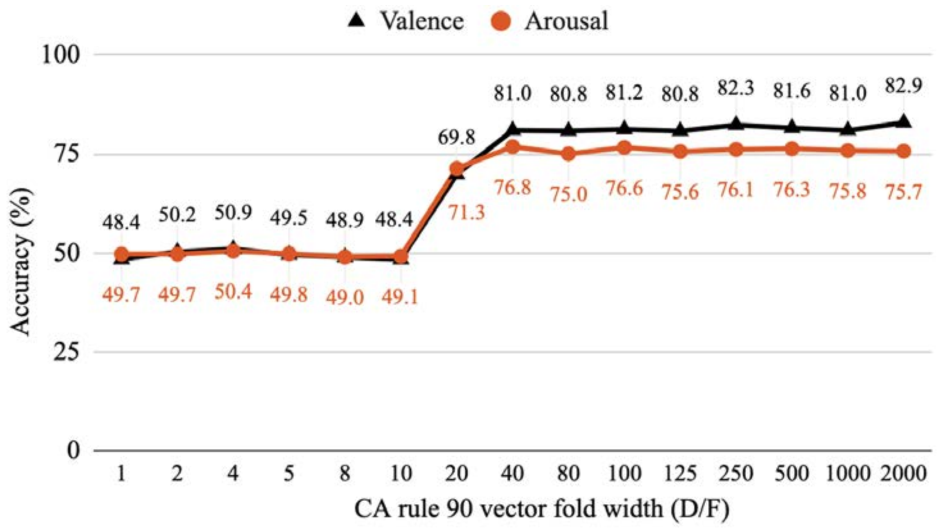


Figure 2.3: Vector folding of the HDC classifier + CA *rule 90* maintains great accuracy until a fold width of 20

Folding by Block

The benefit of vector folding is most prominent in the *Spatial Encoder* and *Fuser*, where the accumulator registers within each block can be decreased by a factor of F, as seen in Figure 2.1. A large unfolded vector *hvout* of size D sits at the end of the *Fuser*, collecting the data from each fold. This is the critical path of the design, as it is the destination register with the most fanout. Nonetheless, vector folding allows fanout to be reduced for all the accumulator registers upstream.

The *Associative Memory* has its own fold factor G. This is because in most applications, the number of channels is greater than the number of output classes. Hence, the *Associative Memory* typically takes fewer cycles to execute than the *Spatial Encoder*. As a result, the AM can operate with considerably more folds than the SE, and be executed in parallel with the rest of the HDC datapath. G is chosen to allow the AM to take the same amount of cycles as the SE, as the AM can perform distance computation on the current classification while the SE processes the next classification. AM vector folding reduces the footprint of the hamming distance computations.

The *Temporal Encoder* is not folded in the final architecture. In theory, the ngrams can be folded to a width of D/F. Permutations of each D/F section of the ngrams can be binded and sent to the associative memory for comparison. However, for n ngrams, n full spatial and fuser encodings need to be performed in order to regenerate the fold sections of the previous $n-1$ ngrams, increasing latency by a factor of n cycles. In addition, the *HV Generator* would need to buffer n times the amount of features, since n sets of features would be needed at a time per classification. The costliness of this operation is described in the following section.

2.4 Results

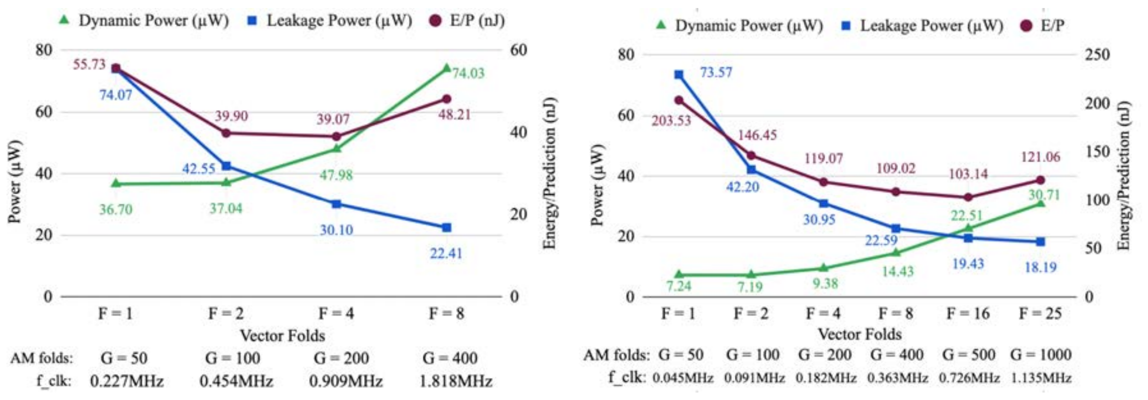


Figure 2.4: The optimal fold factor varies depending on target classification latency

Left: 1ms target, Right: 5ms target

The HDC and SVM ASIC designs in this work are all implemented in the TSMC 28nm HPM technology. Development was facilitated with the HAMMER framework [27]. VDD was lowered and VTH was optimized. For an unfolded HDC Sensor Fusion design at a frequency of 222 MHz, running at 0.9 VDD yielded an energy per prediction of 69 nJ. The same design at 0.8 VDD had an energy per prediction of 61 nJ. Removing LVT cells improved this further to 43 nJ. As a result, CA *rule 90* designs used the lowest available 0.8V VDD supply. ROM designs used the lowest characterized 0.9V VDD supply. Numbers

Table 2.1: Fold Factors and Required Clock Frequency for 1ms Target

# Folds (F)	Accumulator Width (D/F)	Clock Period (ns)
1	2000	4400
2	1000	2200
4	500	1100
8	250	550
16	125	275
50	40	88
100	20	44
1000	2	4.4

were reported at the TT corner of 25°C. For minimal leakage power given the low-frequency use-case, only RVT/HVT cells were utilized during synthesis with Cadence Genus. An area utilization of 70% was targeted for place & route with Cadence Innovus. Power measurements were obtained using Cadence Voltus using post place & route Synopsys VCS gate-level simulation, back-annotated SDF delays, and SPEF parasitic resistances and capacitances at the TT corner.

As seen in Figure 2.4, the optimal fold factor minimizing energy per prediction varies depending on the target classification latency desired. For a 1ms target with required clock frequencies shown in Table 2.1, the optimal fold factor is at $F = 4$, yielding an energy per prediction of 39.07 nJ. For a smaller number of folds, the leakage power of the *Spatial Encoder* and *Fuser* accumulators dominate, while at more folds, the energy is dominated by the processor increasing its frequency to complete the classification task in time. In all folding cases, the *Associative Memory* is folded separately with a factor of G, and remains very small in comparison to the rest of the datapath.

Figure 2.4 also shows that for a 5ms target, the optimal fold factor shifts to around 16 folds, for an energy per prediction of 103.14 nJ. This can be seen because the dynamic power curve shift downwards from a lower frequency requirement across the board, while the leakage power curve remains roughly the same. As different applications have varying sensor sampling rates, adjusting the HDC processor fold factor is useful for targeting the least energy footprint for a given use case.

Figure 2.5 demonstrates the effect on energy if the *Temporal Encoder* was also folded. This is a costly maneuver. The leakage power decreases more significantly as the fold factor increases, but the dynamic power also increases much more dramatically. In all folding cases, the energy per prediction ends up being higher than if the TE was not folded. The optimal ends up being the case where folding isn't performed at all ($F = 1$), indicating that folding the *Temporal Encoder* is counterproductive in reducing energy per classification.

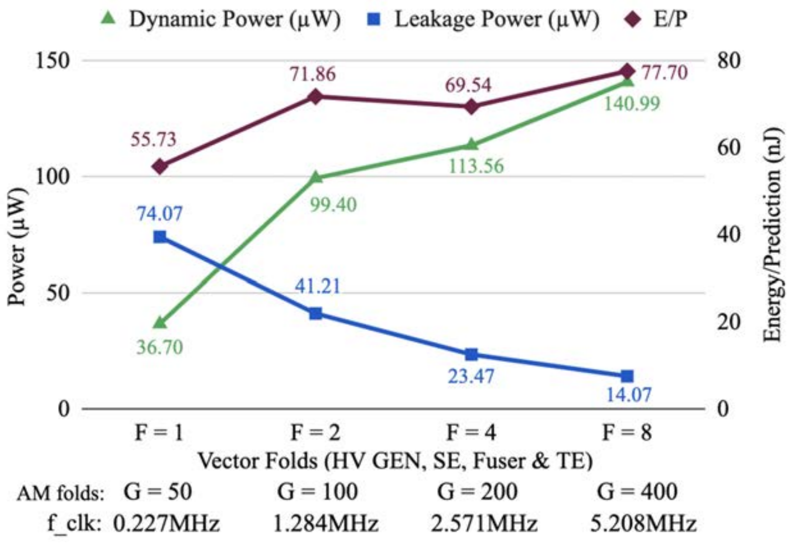


Figure 2.5: Folding the *Temporal Encoder* is costly

Figure 2.6 demonstrates an alternate paradigm of executing the HDC Sensor Fusion processor: removing the classification latency target, and running at maximum frequency. This paradigm targets batch processing or post-processing instead of real-time continuous classification. In this case, the classification latency is no longer constant since the goal is to maximize throughput. As a result, the switching power is greatly increased, but the energy per prediction remains small as the operations can be performed in very short bursts. In this case, the critical path decreases from reduced fanout at higher F, decreasing both switching and leakage power. However, high fold factors become less profitable because the latency increases linearly by fold factor, while the savings in total power are not enough to offset. Additionally in these designs, leakage power occupies only 0.1% of the total power consumption, so the effect of reducing hardware utilization is also less visible. A fold factor of $F = 2$ is narrowly the optimal choice, at 17.82 nJ energy per prediction.

Future work can be done to determine the energy efficiency for a specific target latency when operating at maximum frequency, but power gating until the next sensor signal arrives. There may be overhead of power gating and possible increase in virtual VDD, but for long target classification latencies > 1ms, this could be a competitive option.

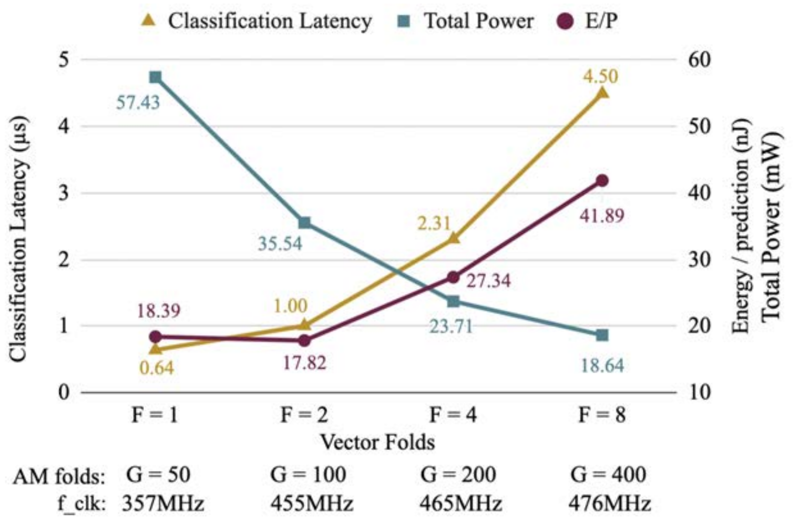


Figure 2.6: A fold factor of $F = 2$ is narrowly the optimal choice when running at maximum frequency

Chapter 3

SVM Sensor Fusion ASIC

The hallmark of the proposed HDC Sensor Fusion architecture is its ability to maintain energy efficiency as the number of input channels scale. To demonstrate this capability, an SVM Sensor Fusion processor was also implemented for direct comparison. The SVM architecture was designed based on the algorithm described in [28], also targetting the AMIGOS dataset.

The goal of the SVM is to find a hyper plane in N-dimensional space that distinctively separates data points into their respective classes. In this work, two separate hyperplaces for valence and arousal classification are needed. The classification decision can be performed using the equation below.

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b \tag{3.1}$$

α_i represents the support vector weights, y^i are the corresponding labels of the support vector, K is the kernel function, $x^{(i)}$ are the support vectors, x is the new data point, and b is the intercept. Training was done in python using the SVC function in sklearn package. Linear kernel with regularization parameter of 0.2 and 0.75 were used for valence and arousal, respectively.

Depending on the number of input features, a different amount of valence and arousal support and alpha vectors are required to split the data into a sufficiently separable N-dimensional space. This is determined during training. For 214 total features as per the AMIGOS dataset, a total of 120 support vectors for the valence classification and 155 support vectors for the arousal classification are needed, listed in Table 3.2. Each support vector consists of 214 elements, resulting in memory storage of a 120×214 valence support matrix

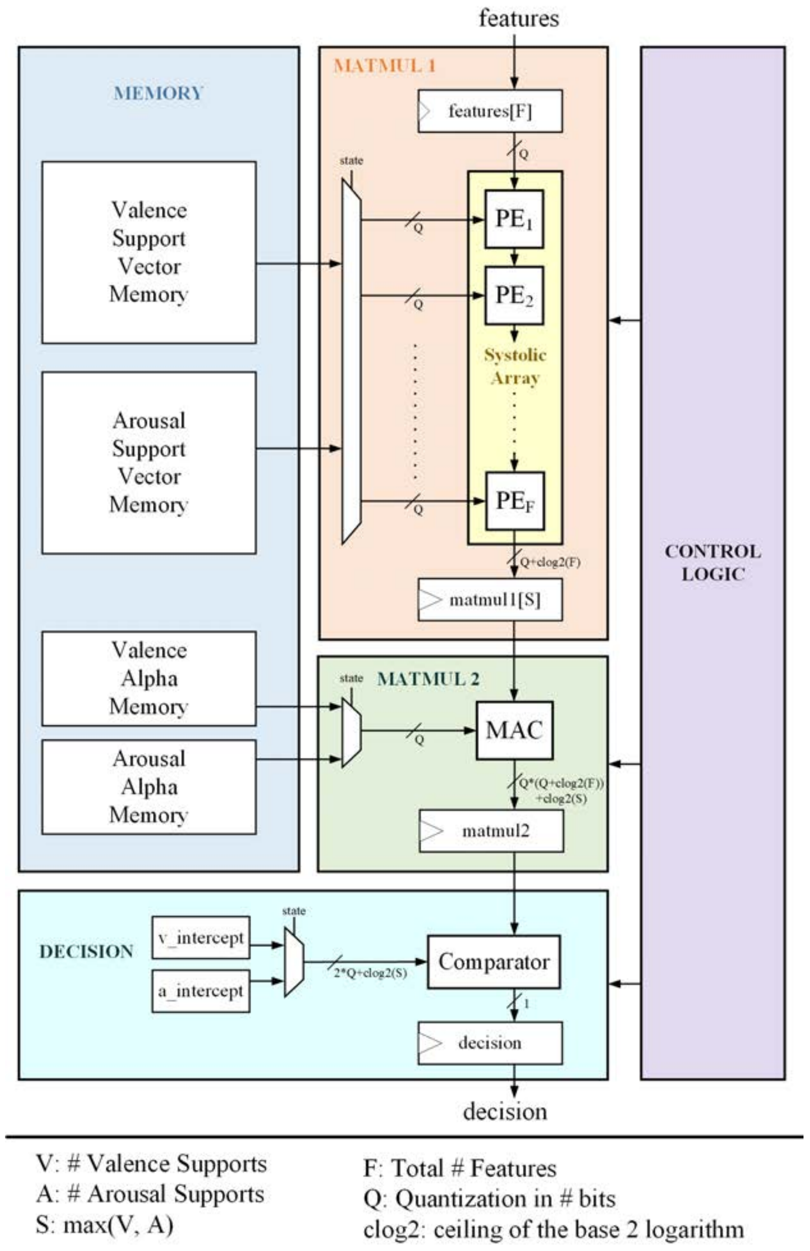


Figure 3.1: SVM Sensor Fusion Architecture

and a 155 × 214 arousal support matrix. A set of weights α_i for each support vector and a single intercept b are also generated during training.

Work from [28] demonstrated that SVM achieves an accuracy of 68% for valence and 66.3% for arousal. However, recent local experimentation of the SVM algorithm has revealed that SVM can achieve an accuracy as high as 94.34% for valence and 86.79% for arousal with recursive feature elimination.

3.1 Quantization

Optimizing the number of bits to quantize is pivotal, as small changes in quantization can have a sizeable effect in both hardware efficiency and algorithmic accuracy. As shown in Figure 3.2, for SVM with 214 features, the optimal quantization was $Q = 9$ bits, although accuracy fluctuates quite arbitrarily around this point. With recursive feature elimination (RFE), input features only need to be quantized to $Q = 5$ bits for accuracy degradation $< 5\%$.

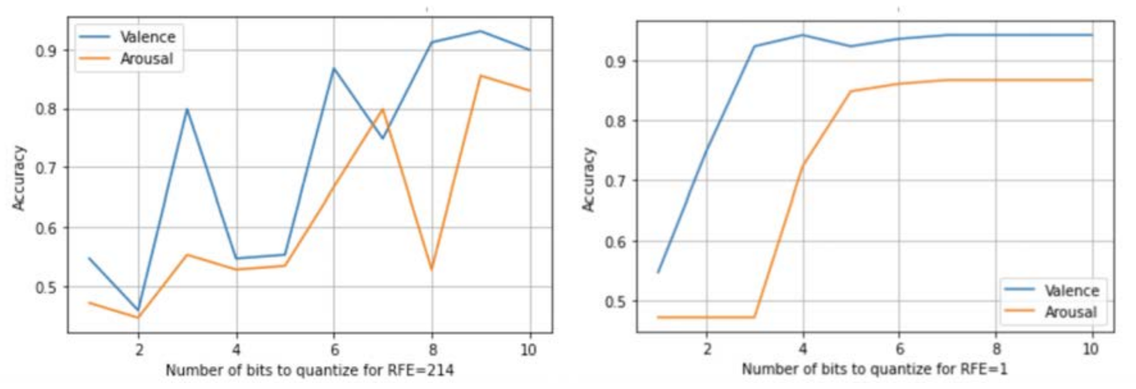


Figure 3.2: Quantization of 9 bits is optimal for 214 features - With RFE feature reduction, a quantization of just 5 bits is necessary

Another design decision is within the PE of the systolic array shown in Figure 3.1. Each PE takes in two inputs, multiplies them, and accumulates them to the next PE. Within each PE however, an additional quantization unit brings the output of the multiplication back to Q bits with minimal accuracy degradation. This allows each element in the $matmul1$ register to remain a comparable size to the input quantization Q. For $matmul2$, this quantization was not done as this step is directly followed by the final decision making, where the result should not lose any important information.

3.2 Block by Block

Figure 3.1 depicts the overall system diagram of the SVM architecture. Note that this following architecture description does not include design modifications that would follow from recursive feature elimination. All input data is quantized to a minimum datawidth of 9-bits. The classification process is divided into 3 steps.

- *MATMUL 1*: The first step is a matrix multiply between the support vector matrix and a set of 214 incoming features. To optimize cycle count, a systolic array with weight-stationary dataflow is implemented, consisting of 214 total processing elements (PEs). An entire set of features are first loaded into the systolic array, after which support vectors can then flow into each PE and propagate downwards. Each PE is also double buffered, allowing a new set of features to be sent in while the current classification is still running. Within each PE in the systolic array, input support values are multiplied with the stored feature, quantized back to 9-bits with minimal accuracy degradation, then accumulated with the partial sum from the PE above. The result of this matrix multiplication is an $S \times 1$ vector, where S is the number of support vectors.
- *MATMUL 2*: The second matrix multiplication is a single MAC, performing the multiplication between the $1 \times S$ α vector, and the $S \times 1$ result from *MATMUL 1*.
- *Decision*: The result of *MATMUL 2* is compared with the intercept b to generate the final classification. The classification is 1 if the result of *MATMUL 2* is greater than $-b$, and 0 otherwise.

Valence and arousal classification share the hardware for the $MATMUL$ and *Decision* blocks, so a state machine controls muxes and registers to compute the two classifications sequentially.

To store the large set of support and alpha vectors, this SVM architecture contains a memory block, implemented as a set of ROMs. The data storage is optimized for minimal retrieval time and minimal control logic overhead. The entire set of required supports per cycle at the west face of the systolic array are stored as a single row in memory, resulting in single-cycle data retrieval from memory. The intercept value for the two valence and arousal classifications are stored as constant tie cells.

3.3 HDC vs SVM

To holistically compare the HDC and SVM Sensor Fusion Processors, energy per prediction numbers were sweeped across a range of input channel counts from 3 to 214 channels.

Additionally, an HDC ROM implementation was created to gain insight on what the energy consumption of HDC would be if all the item memories were stored in ROMs instead of being generated on the fly. This parallels earlier HDC work, where iMs were mostly stored in memory. The HDC ROM architecture is minimally altered from the baseline CA *rule 90* design; only the CA *rule 90* unit was replaced with an item memory ROM containing $214 \times F$ rows and D/F columns, each row containing one fold of a pre-computed item memory vector.

As shown in Figure 3.3, for very small number of channels, HDC and SVM perform similarly, but as the number of channels exceed around 25, the energy utilization of the SVM processor skyrockets.

This is because for HDC with CA *rule 90*, the main hardware cost of increasing channel counts is the accumulator, for which the bit width increases marginally in order to count up to the total number of channels in the largest modality. CA *rule 90* generates a new iM vector per cycle, so no additional iM or CiM vector storage is required for additional channels.

Meanwhile, the HDC ROM implementation shows the impact of increasing the ROM for additional iM vector storage.

Lastly, the SVM processor incurs a colossal cost for large channel counts. The support and alpha memory size, the number of PE units in the systolic array, and the size of the resulting $matmul2$ register, all increase as the number of input features goes up.

Ultimately, with the 214 features of the AMIGOS dataset, the HDC CA *rule 90* implementation uses only 39.07 nJ per prediction, an improvement of 9.5x over the SVM architecture at 372.17 nJ, and 3x better than a traditional HDC implementation with ROMs.

3.4 SVM Recursive Feature Elimination

As mentioned in the previous section, SVM energy efficiency degrades significantly if there are a large number of features. One interesting method to mitigate this is through the use of Recursive Feature Elimination (RFE) [28]. RFE is a feature selection technique that recursively discards the weakest features until the number of remaining features becomes a pre-defined value set by the user. This technique is useful in hardware design as it can

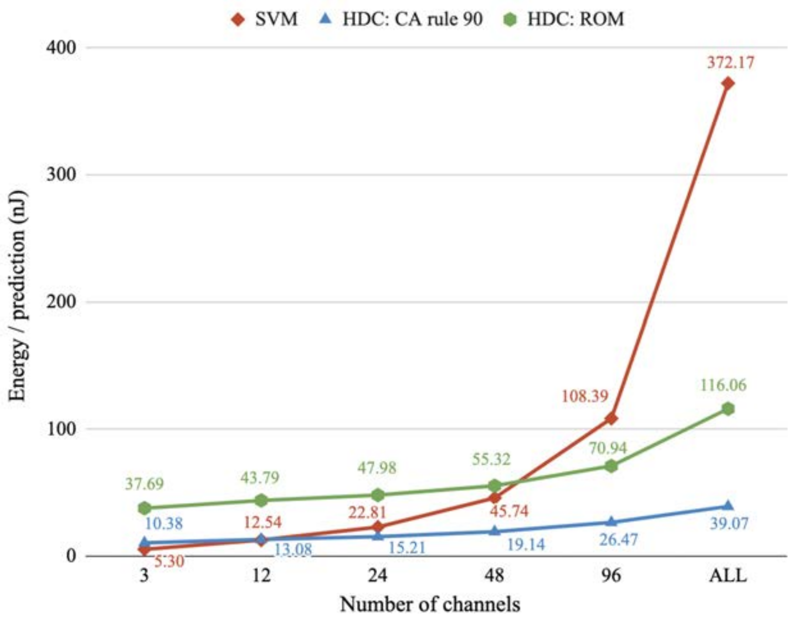
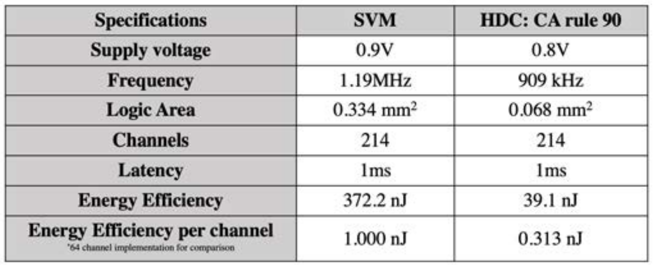


Figure 3.3: Energy/prediction comparison as the number of channels scales between SVM, HDC with CA *rule 90* and HDC with ROM



Specifications	SVM	HDC: CA rule 90
Supply voltage	0.9V	0.8V
Frequency	1.19MHz	909 kHz
Logic Area	0.334 mm^2	0.068 mm^2
Channels	214	214
Latency	1ms	1ms
Energy Efficiency	372.2 nJ	39.1 nJ
Energy Efficiency per channel *64 channel implementation for comparison	1.000 nJ	0.313 nJ

Table 3.1: HDC Sensor Fusion using CA *rule 90* with vector folding vs SVM Sensor Fusion, both targeting physiological signal classification

reduce memory requirements, register dimensions, and lead to significant decreases in power consumption, area, and complexity.

For the AMIGOS dataset emotion recognition applications, the number of features was successfully eliminated from 214 down to just 1, without affecting the testing accuracy at all. Table 3.2 shows the dimensions of the support, alpha, and test vectors with and without RFE, as well as the resulting accuracies. Interestingly, SVM RFE down to 1 feature has an even higher accuracy than classifying with the original full set of features.

When specifying a feature count of 1, the RFE algorithm always chooses the 214th feature

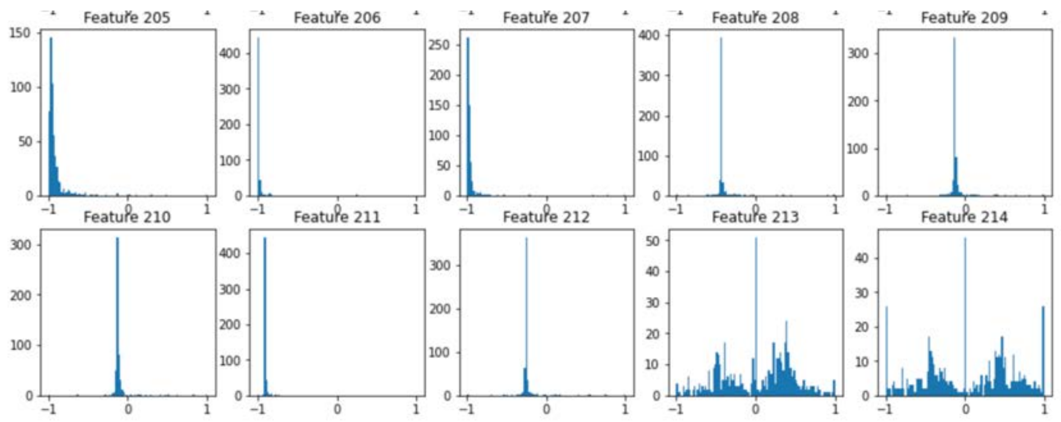


Figure 3.4: Only features #213 and #214 of the 214 total features are actually needed to obtain adequate SVM accuracy

for valence and the 213th feature for arousal. After plotting the distribution of each of the 214 features, as seen in Figure 3.4, we discover that the 213th and 214th features are both distributed more widely across the entire normalized {-1,1} range, compared to many of the other features in the dataset. This is the case for the majority of the features, and provides an explanation as to why RFE was able to narrow down the useful features to such a narrow set. A sample of the distributions of features 205-214 can be seen in Figure 3.4.

Once RFE reduces the number of features down to just 1, the systolic array shown in Figure 3.1 can be rendered entirely unnecessary, and thus removed. For the SVM RFE=1

Table 3.2: Vector Dimensions and Testing Accuracy with and without RFE

	Without RFE	With RFE
Valence alpha vector	1×120	1×118
Valence support vectors	120×214	118×1
Valence test vectors	159×214	159×1
Arousal alpha vector	1×155	1×164
Arousal support vectors	155×214	164×1
Arousal test vectors	159×214	159×1
Valence testing accuracy	90.57 %	94.34 %
Arousal testing accuracy	85.53 %	86.79 %

hardware implementation however, one can still exploit parallelism, in a design exploration manner analogous to vector folding in HDC.

In particular, it is possible to tune the amount of parallelism during the first matrix multiplication between the support and test vectors. For example in the valence case with RFE=1, the first matrix multiplication would involve multiplying a 118×1 support vector with a 1×1 test vector. This can be done in one cycle in maximal parallelism with 118 simultaneous MACs, entirely sequential with 1 MAC, or somewhere in between.

Analysis from figure 3.5 with 1ms target shows that the SVM RFE=1 implementation has the most energy efficiency when run entirely sequentially. P2_4 represents performing 2 valence support $\times$ test vector multiplications in parallel, and 4 arousal support $\times$ test vector multiplications in parallel.

The baseline hardware cost of SVM with RFE is very small, requiring only a few registers to store intermediate matrix results and a minimized memory due to RFE and shared valence/arousal resources. As a result, the switching and leakage power increase by a large factor when fully parallelized. The reduced latency from parallelism is also not nearly as significant in SVM as compared to HDC; a fully parallel SVM classification requires 287 cycles, while an entirely sequential classification requires 567 cycles; this is only an increase of 2x for the sequential case. These factors combined make the sequential SVM the most favorable configuration.

SVM with RFE demonstrates the lowest energy per prediction at 1ms target seen yet, at a record of only 3.85 nJ energy per classification.

Ultimately as it currently stands, HDC is the preferred choice over SVM for cases where there are a large number of channels, and feature elimination is not an option. In such cases, HDC is shown to have 9.5x lower energy per prediction than SVM.

However, recursive feature elimination has proven to be very powerful in selecting the specific features in the dataset that are actually algorithmically decisive in achieving the desired accuracy. With RFE, SVM can reach a low of 3.85 nJ per prediction, lower than the HDC folded CA *rule 90* benchmark of 39.07 nJ. Future work can also be done to incorporate feature reduction techniques with HDC, or evaluate the accuracy if just a few features were fed to the HDC classifier.

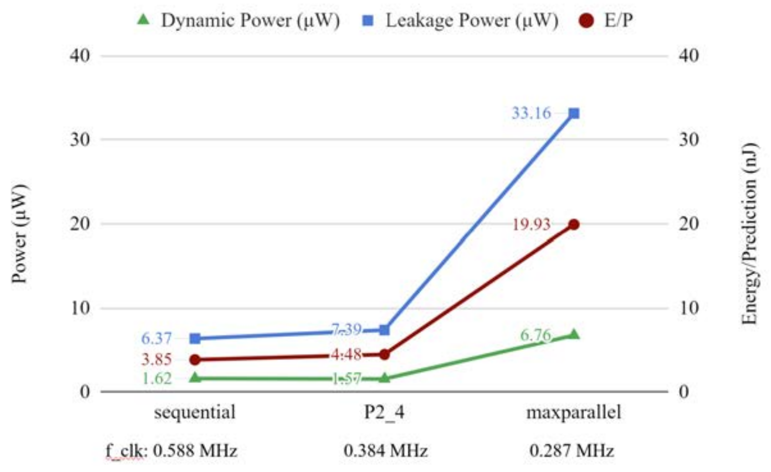


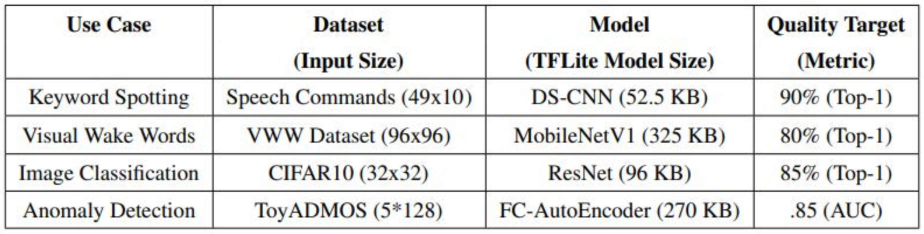
Figure 3.5: For SVM implementations with feature elimination at 1ms target, an entirely sequential run is the most optimal

Chapter 4

TinyML Keyword Spotting

In the previous two chapters, a direct comparison was made between HDC and SVM on the same dataset, classification task, and underlying hardware technology. However in the larger hardware for machine learning ecosystem, the ability to compare two computing paradigms in a fair and unbiased manner is often hindered by the diverse range of devices that machine learning tasks can be implemented on. Methods of pre-processing and feature extraction can be very different between designs, and these operations are often inconsistently included across energy consumption reports. There is also significant variation depending on if the hardware was evaluated at a low level or at the application level. Even chip peripherals and software deployment stacks can impact the ultimate results [1].

Part of the existence of TinyML [1] is to address these many concerns mentioned above. The goal is to foster innovation in this space by establishing a systematic framework for fair, replicable, and robust machine learning hardware evaluation.



Use Case	Dataset (Input Size)	Model (TFLite Model Size)	Quality Target (Metric)
Keyword Spotting	Speech Commands (49x10)	DS-CNN (52.5 KB)	90% (Top-1)
Visual Wake Words	VWW Dataset (96x96)	MobileNetV1 (325 KB)	80% (Top-1)
Image Classification	CIFAR10 (32x32)	ResNet (96 KB)	85% (Top-1)
Anomaly Detection	ToyADMOS (5*128)	FC-AutoEncoder (270 KB)	.85 (AUC)

Table 4.1: MLPerf Tiny v0.5 Inference Benchmarks [1]

To accomplish this, four benchmarks constitute the MLPerf Tiny benchmark suite. These are Keyword Spotting, Visual Wake Words, Image Classification, and Anomaly Detection, representing archetypical machine learning tasks. Each of the benchmarks is accompanied by a specific dataset, model, and target accuracy. Submissions are also divided into a closed and open category. Closed submissions must adhere to the same models and datasets as the reference implementation and meet the accuracy target provided, while open submissions allow for the use of any model and dataset, such that tradeoffs between different models and their accuracy, latency, and energy consumption can be demonstrated.

In this work, we explore how Hyperdimensional Computing can be used to perform on the keyword spotting benchmark. As previously mentioned, HDC has been proven consistently with high accuracy for a variety of tasks including biosensor classification such as seizure [5] and gesture recognition [20], and recognizing individual sounds with VoiceHD [12]. However, HDC has not been demonstrated yet on the particular TinyML benchmarks shown above, and as such, this work attempts to take the first steps in connecting HDC directly with TinyML tasks.

The keyword spotting task involves recognizing specific predetermined words or basic phrases, and is commonly used in human-machine interaction such as to trigger devices to begin speech recognition as with Siri, or in voice-controlled robots. This suggests an always online system that wastes little power when off, yet responds with low-latency once triggered. For machine learning algorithms designed around this task, an initial state also cannot be known, and the algorithm should not be able to assume when a keyword will be presented [2].

In the TinyML benchmark, the keyword spotting task is evaluated based on its ability to distinguish between ten key words: ’on,’ ’off,’ ’stop,’ ’go,’ ’left,’ ’right,’ ’yes,’ ’no,’ ’up,’ ’down.’ Additionally, distinguishing these ten words against random noise and silence is necessary; this was not yet done in this work. Audio for these words come in one second recordings, from the Google Speech Commands database [29], with speakers of varying ages, gender, and ambient environments.

4.1 Feature Extraction

In order for HDC to be able to accurately classify incoming audio samples, a comparatively elaborate system of feature extraction was necessary. Various feature extraction techniques were attempted, many of which were inspired by speech processing techniques mentioned in Acoustic and Auditory Phonetics [13], as well as in experimentation with Praat, a software for performing phonetics analysis [26].

First, audio data provided by the Google Speech Commands dataset was given to be sampled at 16 kHz, with a total audio length of one second. Then, spectrograms were generated by taking individual windows across the audio time series, and performing Hamming windowed short-time Fourier transforms on each window. Broadband spectrograms were created using an 80 sample (5 ms) window with a 16 sample time step between each window. Samples were zero-padded to a total length of 30 ms, accentuating the formant structure of each spectrum. Narrowband spectrograms were created with a much wider 480 sample window (30 ms) with the same time step, providing more information on the voicing and harmonics of the speech. Both spectrograms were obtained at a range from 0 to 8 kHz, within the range of typical human speech.

Various methods for feature extraction were employed to maximize the accuracy of HDC on the TinyML Keyword Spotting benchmark. In the beginning, we used MFCCs and Mel Spectrograms, the typical features that are used for Neural Network algorithms such as DS-CNNs or RNNs. However, it was found that HDC struggles to perform well when classifying two-dimensional image data. As a result, a more verbose feature extraction method using various signal processing and linguistic techniques was used in anticipation that the accuracy for HDC would be improved with largely one-dimensional feature data, as is typically the case in fields where HDC shines, such as in biosensor classification. These techniques are elaborated upon in the following sections.

Spectrogram Based Features

The process of determining the best features to send into the HDC classifier was a largely iterative and exploratory task. In essence, the possibilities are endless. Column-wise spectral means, standard deviations, derivatives, means of derivatives, and presence of voicing are all possible features that can be extracted from the baseline broadband and narrowband spectrograms. For example, a generic amplitude feature was extracted, which was the sum of each column in the spectrogram. This simple feature could highlight an overall profile of a specific word; 'off' would often have two peaks, one for the vowel and a wider plateau for the /f/ sound.

More granular features can also be extracted to focus on specific phonemes that distinguish words in the target word bank. For example, an /f/ indicating feature was collected by finding the mean of all the values in each column spectrum, then calculating how far on average each value in the column is from the column mean. In regions of the spectrogram where the /f/ sound is produced, the column mean is relatively high, but the values are spread relatively evenly from low to high frequencies. Since the /f/ sound is also not voiced, a voicing mask could be applied on top of the feature, to reduce false positive in /f/ detection. This feature can also be generalized to other fricatives that occupy a similar space in the spectrogram.

[11] also presents various strategies for detecting alveolar fricatives such as /s/ and /z/. These alveolar fricatives are found when there is a high normalized amplitude of frequency components above 2 kHz. The equation for this, equation 4.1, is derived from [11], with f^{F}_{high} set to 2.5 kHz in this experiment.

$$F(t) = \frac{1}{f^{\mathrm{F}}_{\mathrm{high}} - f^{\mathrm{F}}_{\mathrm{low}} + 1} \sum_{i=f^{\mathrm{F}}_{\mathrm{low}}}^{f^{\mathrm{F}}_{\mathrm{high}}} \log|S(t,\ i)| \tag{4.1}$$

The voicing mask is useful for masking certain features that are known to only be expected in voiced or unvoiced regions. These can be taken from the narrowband spectrogram, for which regions that are voiced have harmonic ripples at equal intervals that extend across the spectrum, with peaks at multiples of the fundamental frequency. A voicing detector can search within this fundamental range around 150 Hz to 800 Hz to determine if a large amplitude of recurring peaks exists.

Lastly, a slope profile was also taken to observe regions in the spectrogram with sudden increases in amplitude, sometimes indicating stops such as /p/, /t/, or /k/. This feature was subject to some noise, however, and was sometimes confused with glottal stops that often precede words starting with vowels.

Formant Extraction

To more accurately predict words, it was also necessary to extract features that could reveal the vowel phonemes present in the speech. Traditionally, this is done using Linear Predictive Coding (LPC). There are many variants of formant extraction techniques such as those mentioned in [25]; in this work, we used a method based on multiple linear regression derived from [10].

LPC is a technique to estimate the vocal tract transfer function, from which the poles, the formants, can be calculated. Arithmetically, this is done by first taking a single sample window, then performing multiple linear regression between this sample and the previous M sample windows before it. This is captured by the equation below.

$$Y(k) = a(1)Y(k-1) + a(2)Y(k-2) + ... + a(M)Y(k-M) + \epsilon \tag{4.2}$$

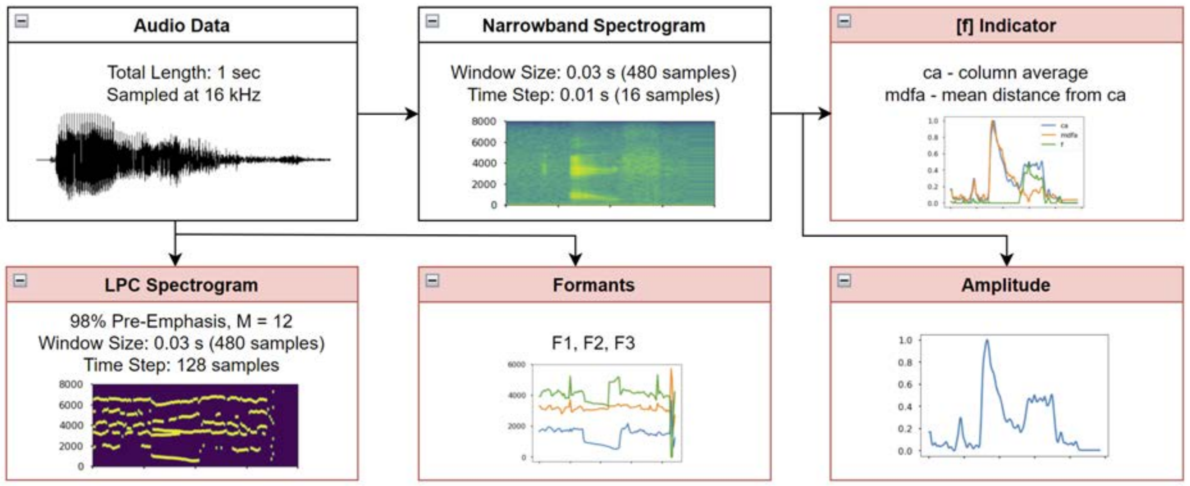


Figure 4.1: Noteworthy portions of the feature extraction pipeline are shown here. These diagrams show an example of the word 'off'

LPC works by taking advantage of the repeating patterns of similar-looking waveform chunks [13] present in voiced speech. These patterns can be auto-correlated at regular intervals. The fundamental frequency can then be extracted by determining the period of the repeating waveform. Repeating patterns at higher frequencies can be found as formants.

The audio time signal is first passed through pre-emphasis to increase the amplitude of high frequency components with $y_n = x_n - px_{n-1}$ where $p = 0.98$.

For Equation 4.2, $M = 12$, which would correspond to calculating a maximum of $M/2 = 6$ total formants, a typical value for speech recognition applications. The a coefficients are fitted to minimize error, for an optimal prediction of sample $Y(k)$. These are the LPC coefficients.

A frequency response is then computed on the padded LPC coefficients, and the absolute value + logarithm is computed around them to form a spectrogram of dimensions 512×125, corresponding to a maximum frequency of 8 kHz and spectrums over 1 second. The first three formants are usually below 4.5 KHz, so the higher frequencies were discarded, reducing the dimension to 300×125. Peak detection is then done, and the spectrogram discretized, where all points in each column within 10 values of a peak were set to $\{1\}$, $\{0\}$ otherwise. Finally to reduce redundancy, the y-axis of the spectrogram is downsampled by 10x to form the LPC Spectrogram shown in Figure 4.1.

A similar process of peak detection can be done to create the formants, with the first three shown in Figure 4.1.

4.2 HDC

In the latest version, six groups of features are fed into the HDC classifier. These are the amplitude, /f/ indicator, /s/ indicator, slope profile, and 30 features each being a row from the LPC Spectrogram. Figure 4.1 shows an example of these features for a sample of the word 'off.' To improve model accuracy and anticipated hardware performance, these features were quantized to between 11-20 discrete values. Values in the LPC Spectrogram are already discretized to either $\{0\}$ or $\{1\}$.

Features are only sent into the HDC core when there is voice activity, determined by the amplitude of the spectrogram.

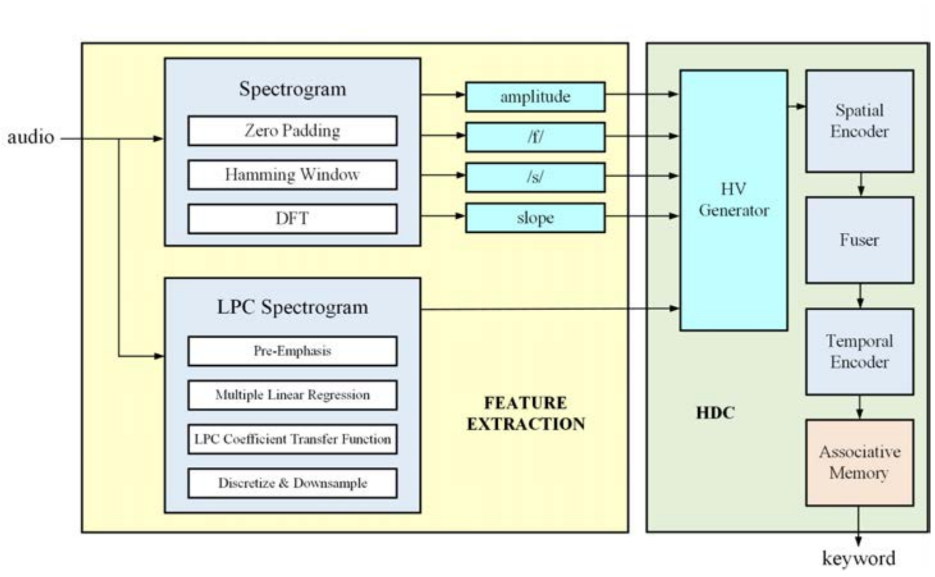


Figure 4.2: Feature extraction is a major component of the HDC classification pipeline

More work is needed to tune HDC hyperparameters in order to optimize accuracy. Currently, an HV dimension of $D = 10000$ is used, but if HDC can maintain accuracy with a smaller dimension, this would be reduced. An $ngram = 20$ is used to track 20 samples of features; a minimum of $ngram = 10$ is likely necessary since LPC spectrogram features are interpolated over 10 points to match the sample rate of the other features. This hyperparameter should also be tuned in future work.

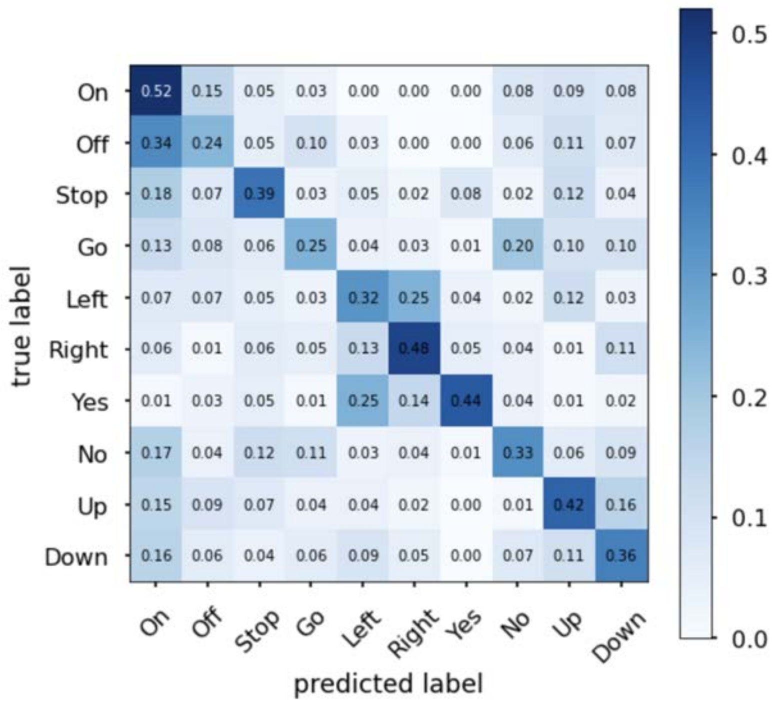
Similar to HDC work described in Section 2, a CiM memory is needed to map discretized values into the hyperdimensional space. Only two CiM vectors are needed for the LPC spectrogram features since they are discrete to $\{0\}$ or $\{1\}$, but other features require around

11-20 CiM vectors each. This would form the minimum memory requirement of the HDC processor. Lastly, iMs are generated with CA *rule 90*.

Features are split into two modalities, the 4 'consonant features,' and the 30 LPC features. These are encoded into their independent vectors during spatial encoding, then the two output modality vectors are fused in the *Fuser*. The *Temporal Encoder* tracks 20 ngrams. Lastly, the *Associative Memory* stores the 10 prototype vectors for each keyword, and a hamming distance is performed to determine a final classification.

The TinyML Keyword Spotting classification endeavour is still a work in progress. Currently, accuracy results are not comparable to those found in conventional neural network algorithms, with averages in the 40% accuracy range. Nonetheless, future work can feature experimentation with applying a neural network frontend to capture more relevant time-series features, which can also be more scalable then performing granular feature extraction specific to speech recognition as done in this work. As seen above, the process of extracting the right features for maximum HDC accuracy can be a time-consuming process, and this time could be significantly reduced if a correctly tuned neural network can generate necessary features automatically.

Future comparisons in accuracy and hardware efficiency can be made between full HDC, hybrid HDC + NN for feature extraction, and full neural network implementations. A neural network may require more hardware resources than manual feature extraction shown above, but can likely also boost accuracy significantly, since they have long been proven for image processing tasks such as extracting patterns from spectrograms or MFCCs. This direction may be necessary for HDC to perform in tasks involving image classification in general. A tradeoff can be made between incorporating hybrid models for increased accuracy and power efficiency with the HDC base implementation. Ultimately, the success of an HDC core relies heavily on a reliable feature extraction process and extensive HDC hyperparameter tuning across the entire system.



	On	Off	Stop	Go	Left	Right	Yes	No	Up	Down
On	0.52	0.15	0.05	0.03	0.00	0.00	0.00	0.08	0.09	0.08
Off	0.34	0.24	0.05	0.10	0.03	0.00	0.00	0.06	0.11	0.07
Stop	0.18	0.07	0.39	0.03	0.05	0.02	0.08	0.02	0.12	0.04
Go	0.13	0.08	0.06	0.25	0.04	0.03	0.01	0.20	0.10	0.10
Left	0.07	0.07	0.05	0.03	0.32	0.25	0.04	0.02	0.12	0.03
Right	0.06	0.01	0.06	0.05	0.13	0.48	0.05	0.04	0.01	0.11
Yes	0.01	0.03	0.05	0.01	0.25	0.14	0.44	0.04	0.01	0.02
No	0.17	0.04	0.12	0.11	0.03	0.04	0.01	0.33	0.06	0.09
Up	0.15	0.09	0.07	0.04	0.04	0.02	0.00	0.01	0.42	0.16
Down	0.16	0.06	0.04	0.06	0.09	0.05	0.00	0.07	0.11	0.36

Figure 4.3: An ongoing effort is being made to improve HDC accuracy for this particular TinyML Keyword Spotting endeavour

Chapter 5

Conclusions and Future Work

In this work, the advantage of low-power hardware efficiency with HDC is cemented with a series of optimizations including CA *rule 90* and vector folding.

By reducing memory storage requirements to only a handful of CiM and iM vectors, and sweeping for an optimal vector fold, an energy efficiency of 39.1 nJ per prediction is achieved at typical biosensor application latencies. At the 4-fold factor, the HDC sensor fusion classifier uses a logic area of $0.06794mm^2$, with no designated memory unit at all.

Since this processor was designed specifically targeting sensor arrays where the channel count is large, a 64 channel implementation of the HDC sensor fusion ASIC was also created to compare against previous state of the art in an equitable manner by the metric of energy efficiency per channel. The 64 channel version was evaluated at 1ms latency, and found to have an energy consumption of 0.313 nJ per channel. The 64 channel implementation by Eggiman et al. [9] consumed 2.98 nJ per channel. This is a 9.5x improvement over the previous state of the art.

An analogous SVM processor was also implemented and demonstrated on the same dataset, classification task, and underlying hardware characteristics. For applications with more than a few channels, HDC achieves better energy efficiency than SVM, while still maintaining high classification accuracies of 82.3% valence and 76.1% arousal. At 214 channels, HDC is also 9.5x more energy efficient than the SVM implementation.

Nonetheless, the race between algorithms to determine which one reigns supreme is by nature a contentious one. SVM confirms this yet again by the reality that coupled with recursive feature elimination, SVM can reduce the number of input channels all the way from 214 down to just one, as least for this dataset. With this newfound power, SVM was capable of classifying emotions with the AMIGOS dataset using only 3.85 nJ per classification.

Additional work in software modelling of the SVM classifier also showed that SVM accuracies are not as subpar as initially determined in [28], only one feature was enough to provide 94.34% valence and 86.79% arousal accuracy.

Future work in the HDC sensor fusion classifier can be done to explore the application of CA *rule 90* and vector folding on different benchmarks with various throughput requirements, particularly non-physiological time-series tasks such as anomaly detection, where feature resolution and hence the CiM may need to be scaled to larger sizes. As more classes are added, the AM storage will also run into the same scaling issue as the iM. HDC vector compression techniques can be explored to address this. Now that the memory requirement in the *HV Generator* is nullified, the accumulator in the *Spatial Encoder* is now the largest block. This could imply employing feature reduction or elimination for HDC, since this has already been shown to work very well with SVM. This would reduce the overall number of channels and therefore spatial encoder cycles and register size.

Work in HDC vectorization has also been done in the past, and more work in integrating HDC workloads onto Hwacha and RISC-V vector instructions can be done. Significant speedups can be made simply with proper parallelism, dataflow optimizations, and exploiting L1 and L2 cache sizes and dimensions.

Lastly, a detour for HDC was made into a research space that it has largely not been exposed to yet, that being TinyML. HDC was used to perform Keyword Spotting, and a variety of feature extraction techniques were experimented to gradually improve the HDC classification accuracy. When it was clear that HDC struggles in classifying with two dimensional features such as MFCCs, a series of feature extraction strategies were drawn from signal processing and linguistics, in an attempt to spoon-feed HDC the information it needs. More HDC hyperparameter tuning is also needed to confirm the highest accuracy that the current feature extraction methodologies can achieve. At an average of 40% per word, there is a still much that can be done to improve the accuracy.

Much future work can be done in this space, including replacing the granular and largely manual feature extraction task with a small neural network unit, to not only improve the accuracy, but also portability to other classification tasks. Since conventional DNNs and RNNs have been proven for image processing for many applications, a hybrid HDC+NN approach will likely be useful. Lastly, comparing HDC+NN vs a full NN solution in hardware should be done to highlight to what extent HDC plays in optimizing the total energy-efficiency of the classifier.

In a world where low-power machine learning is the new paradigm, there are many roles where HDC can fit in with stellar performance and energy-efficiency. This work has shown that high accuracies can be achieved with very little hardware footprint on HDC's behalf. This work has also propelled HDC's journey onto new paths, such as with the Keyword Spotting benchmark from TinyML. There are many directions that future work can take, and it is of no doubt that in each one, HDC can shine. The end goal is applicability within any low-power device: smartphones, iot devices, and wearables all performing at state of the art efficiency thanks to none other than HDC running seamlessly under the hood.

Bibliography

[1] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. "MLPerf Tiny Benchmark". In: (2021). DOI: `10.48550/ARXIV.2106.07597`. URL: `https://arxiv.org/abs/2106.07597`.

[2] Peter Blouw, Gurshaant Malik, Benjamin Morcos, Aaron R. Voelker, and Chris Eliasmith. "Hardware Aware Training for Efficient Keyword Spotting on General Purpose and Specialized Hardware". In: (2020). DOI: `10.48550/ARXIV.2009.04465`. URL: `https://arxiv.org/abs/2009.04465`.

[3] Patricia Bota, Chen Wang, Ana Fred, and Hugo Plácido da Silva. "A Review, Current Challenges, and Future Possibilities on Emotion Recognition Using Machine Learning and Physiological Signals". In: *IEEE Access* PP (Sept. 2019), pp. 1–1. DOI: `10.1109/ACCESS.2019.2944001`.

[4] T. Bouguera, J.-F. Diouris, J.-J. Chaillout, R. Jaouadi, and G. Andrieux. "Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN". In: *sensors* (2018).

[5] Alessio Burrello, Lukas Cavigelli, Kaspar Schindler, Luca Benini, and Abbas 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)*. IEEE. 2019, pp. 752–757.

[6] En-Jui Chang, Abbas Rahimi, et al. "Hyperdimensional Computing-based Multimodality Emotion Recognition with Physiological Signals". In: *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2019, pp. 137–141.

[7] Juan Abdon Miranda Correa, Mojtaba Khomami Abadi, Niculae Sebe, and Ioannis Patras. "Amigos: A dataset for affect, personality and mood research on individuals and groups". In: *IEEE Transactions on Affective Computing* (2018).

[8] Sohum Datta, Ryan AG Antonio, Aldrin RS Ison, and Jan M Rabaey. "A programmable hyper-dimensional processor architecture for human-centric IoT". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.3 (2019), pp. 439–452.

[9] Manuel Eggimann, Abbas Rahimi, and Luca Benini. "A 5 μW Standard Cell Memory-Based Configurable Hyperdimensional Computing Accelerator for Always-on Smart Sensing". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.10 (2021), pp. 4116–4128.

[10] *Formant Analysis using LPC.* `https://sail.usc.edu/~lgoldste/Ling582/Week\%209/LPC\%20Analysis.pdf`.

[11] Kouki Furuya, Arata Kawamura, and Youji Iiguni. "Alveolar fricative consonants detection with easily interpretable feature for speech training". In: *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS).* 2019, pp. 1–2. DOI: `10.1109/ISPACS48206.2019.8986317`.

[12] Mohsen Imani et al. "Voicehd: Hyperdimensional computing for efficient speech recognition." In: *2017 IEEE international conference on rebooting computing (ICRC).* IEEE. 2017, pp. 1–8.

[13] Keith Johnson. "Acoustic and auditory phonetics". In: *Phonetica* 61.1 (2004), pp. 56–58.

[14] P. Kanerva. "Hyperdimensional computing: An introduction to com- puting in distributed representation with high-dimensional random vectors". In: *Cognitive computation* 1.2 (2009), pp. 139–159.

[15] Denis Kleyko, Edward Paxon Frady, and Friedrich T Sommer. "Cellular automata can reduce memory requirements of collective-state computing". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[16] Denis Kleyko and Evgeny Osipov. "No two brains are alike: Cloning a hyperdimensional associative memory using cellular automata computations". In: *First International Early Research Career Enhancement School on Biologically Inspired Cognitive Architectures.* Springer. 2017, pp. 91–100.

[17] Abhijith M and Shekhar G. “Language Classification Technique Using In-memory High Dimensional Computing (HDC)”. In: *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT).* Vol. 1. 2019, pp. 293–298. DOI: 10.1109/ICICICT46008.2019.8993227.

[18] Alisha Menon, Anirudh Natarajan, Reva Agashe, Daniel Sun, Melvin Aristio, Harrison Liew, Yakun Sophia Shao, and Jan M Rabaey. “Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata”. In: *arXiv preprint arXiv:2104.02804* (2021).

[19] Alisha Menon, Daniel Sun, Melvin Aristio, Harrison Liew, Kyoungtae Lee, and Jan M Rabaey. “A Highly Energy-Efficient Hyperdimensional Computing Processor for Wearable Multi-modal Classification”. In: *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS).* IEEE. 2021, pp. 1–4.

[20] Ali Moin, Andy Zhou, et al. “A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition”. In: *Nature Electronics* (2020), pp. 1–10.

[21] Ali Moin, Andy Zhou, 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).* IEEE. 2018, pp. 1–5.

[22] Jonathan Posner, James A Russell, and Bradley S Peterson. “The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology”. In: *Development and psychopathology* 17.3 (2005), p. 715.

[23] Siddharth Siddharth, Tzyy-Ping Jung, and Terrence J Sejnowski. “Utilizing deep learning towards multi-modal bio-sensing and vision-based affective computing”. In: *IEEE Transactions on Affective Computing* (2019).

[24] Siddharth Siddharth, Tzyy-Ping Jung, and Terrence J. Sejnowski. *Multi-modal Approach for Affective Computing.* 2018. DOI: 10.48550/ARXIV.1804.09452. URL: https://arxiv.org/abs/1804.09452.

[25] Thorsten Smit, Friedrich Türckheim, and Robert Mores. “Fast and robust formant detection from LP data”. In: *Speech Communication* 54.7 (2012), pp. 893–902. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2012.03.002. URL: https://www.sciencedirect.com/science/article/pii/S0167639312000313.

[26] Will Styler. “Using Praat for linguistic research”. In: *University of Colorado at Boulder Phonetics Lab* (2013).

[27] Edward Wang et al. “A methodology for reusable physical design”. In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2020, pp. 243–249.

[28] Sheng-Hui Wang, Huai-Ting Li, En-Jui Chang, and An-Yeu Andy Wu. “Entropy-assisted emotion recognition of valence and arousal using XGBoost classifier”. In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2018, pp. 249–260.

[29] Pete Warden. *Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition*. 2018. DOI: `10.48550/ARXIV.1804.03209`. URL: `https://arxiv.org/abs/1804.03209`.

Appendix A

Images

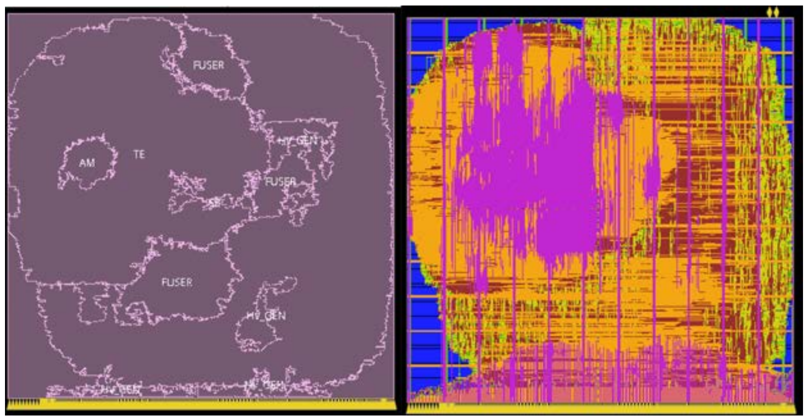


Figure A.1: Layout of optimized HDC Sensor Fusion with CA *rule 90* and 4 vector folds placed on a $310um \times 310um$ square

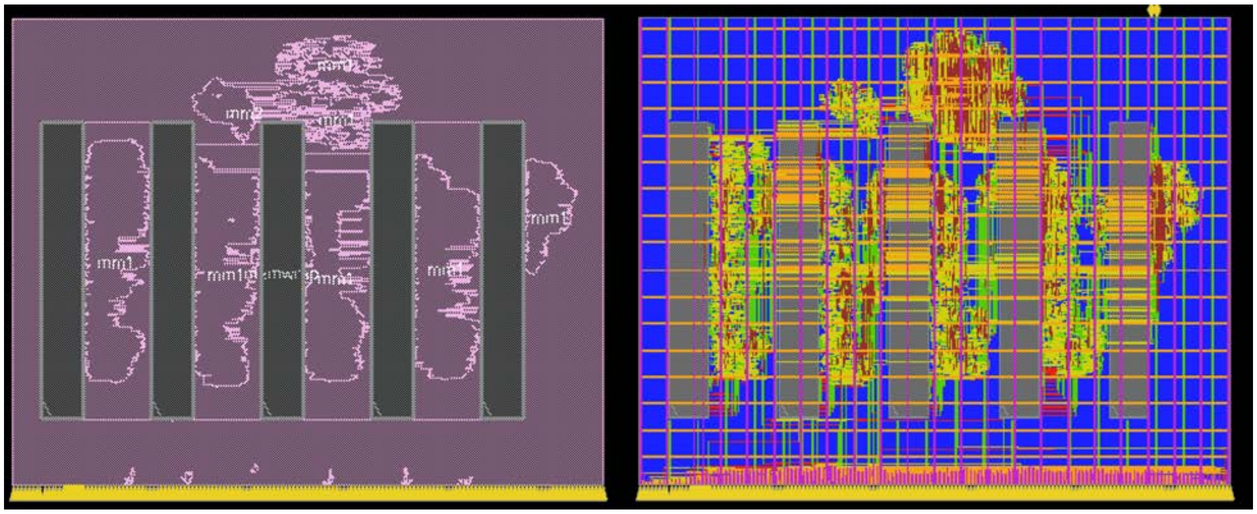


Figure A.2: Layout of optimized SVM Sensor Fusion with 64 channels and no RFE placed on a $620um \times 490um$ rectangle

Appendix B

Code Repository

Code for this project is publically available at
`https://github.com/Calculasians/HDC-Sensor-Fusion-Research`.
Final optimized HDC designs are located in
`Verilog Source Code/FoldedRule90/HDC_Sensor_Fusion_SEFUAMFoldedRule90`.
Final optimized SVM designs can be found in
`SVM/hw/SVM_ROM_Systolic`.
Finally, Python models for verification can be found in `Software Model - Python` and
`SVM/sw`.