Windfarm Forced Oscillation Detection Using Hyperdimensional Computing

Shyam Yathirajam*, Arash Peighambari*, Ruben Roberts†, Hamed Nademi†,
Sreedevi Gutta*, Justin Morris* and Ali Ahmadinia*
*California State University, San Marcos, CA 92096, USA, Computer Science and Information Systems Department
†California State University, San Marcos, CA 92096, USA, Physics Department
{yathi001,peigh001,rober331,hnademi,sgutta,justinmorris,aahmadinia}@csusm.edu

2023 IEEE International Conference on Big Data (BigData) | 979-8-3503-2445-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigData59044.2023.10386628

***Abstract*—Convolutional Neural Networks (CNNs) have been explored to detect forced oscillations in windfarm systems in the past. However, these CNNs require a significant amount of data samples between inference queries and a significant amount of computational power and time. This leads to systems that have a large delay between a forced oscillation occurring and detecting the forced oscillation. This paper presents a novel approach applying Hyperdimensional Computing (HDC) as an effective solution for the first time in forced oscillation detection to overcome the problems of CNNs. HDC is able to reduce the time to detect forced oscillations in two ways: First, by reducing the time needed to collect data to create a new inference sample by reducing the number of data points required. Second, by providing a significantly smaller, more energy efficient, and faster model for detection than current state-of-the-art. Our results show that HDC, with an FPGA implementation, is able to achieve $55\times$ faster detection of forced oscillations in windfarms while achieving the same accuracy as the best current CNN models using software solutions.**

I. Introduction

System Operators and utilities give higher priority to standardizing the connections and monitoring of Distributed Energy Resources (DERs). In recent years, DERs such as solar and wind power stations present havoc to the distribution portion of the grid. The intermittent fluctuations of power coming from the generation source creates power quality issues and instability challenges [1], [2], which make the possibility of system failures more frequent and causes expensive damage to the power system infrastructure. To tackle this issue, optimizing wind turbine performance is essential, and the analysis of current wave data is crucial. Accurate identification of oscillatory signals within this data is challenging but vital, as these signals indicate operational concerns. Swift classification of these oscillations can enhance maintenance efficiency and ensure uninterrupted clean energy generation. To address this complexity, methodologies like 2-Dimensional Convolutional Neural Networks (2D CNNs) and 1-Dimensional Convolutional Neural Networks (1D CNNs) have been explored [3]. While CNNs excel in capturing intricate time series patterns, 2D CNNs involve image transformation, introducing complexity, time delay, and cost. CNN architectures might not be optimal for wind farm operations, necessitating a specialized approach for classifying oscillatory signals within current wave data. Other more efficient and light-weight machine learning algorithms are more appropriate for this application in order to reduce complexity, energy consumption, time delay, and cost.

While CNNs have established themselves as potent tools for pattern recognition in various contexts, they come with computational demands that may be at odds with the resource constraints of IoT devices present in wind farms. In contrast, Hyperdimensional Computing (HDC) is a light-weight algorithm that works on Hypervectors (HVs) with dimensionality in the $10,000$s [4]. HDC first transforms the data into HVs through encoding yielding compact, yet information-rich, representations of intricate time series current wave data. Then, all subsequent operations in HDC are extremely efficient bitwise operations rather than computationally burdensome floating point operations seen in CNNs [5].

This paper introduces an innovative approach that builds upon the foundational concept of HDC, offering a unique perspective to address the oscillatory signal classification challenge in wind farm operations. To our knowledge, this is the first time HDC is being used to detect forced oscillations. Our approach with HDC reduces the total time to detect forced oscillations in wind farms in two ways. First, we reduce the number of data points required to create a new sample with a novel labeling strategy. We introduce the idea of NGrams along with a new aggressive labeling strategy and overlapping windows. Second, by utilizing a more efficient model in HDC, we reduce the computational time of the inference phase. Additionally, to achieve the best performance, we implement our HDC model in an FPGA architecture. A comparative analysis is undertaken, pitting the HDC approach against established 2D and 1D CNN techniques. Our results show that HDC, with an FPGA implementation, is able to achieve $55\times$ faster detection of forced oscillations in windfarms while achieving the same accuracy as the best current CNN models using software solutions.

II. Background and Related Work

A. Current State-of-the-Art

Figure 1 is the detailed model of the entire power system of interest based on wind turbine-generator type 3 is used to produce 100MW (deploying a cluster of 10 wind turbines) for the connected grid with operation voltage of 500kV. Each wind turbine has a rating of 10MW, and an output voltage of 575V, which steps up to 500kV using a grid transformer. The studied wind farm is interconnected to two power networks using different transmission line lengths (62 Miles and 242 Miles). A simulated system implemented in MATLAB/Simulink environment is used to create low-frequency oscillations and record synthesized data. Figure 1 also shows an example of the types of oscillations we are trying to detect in these systems. The blue portion of the waveform shows normal operation, while the red portion shows when an oscillation occurs. Additionally, at the bottom of the graph, we show the two key components of the time delay from the start of the oscillation, to the detection of the oscillation. The two key parts of the delay are: 1) Gathering the data necessary to perform inference on a new sample for

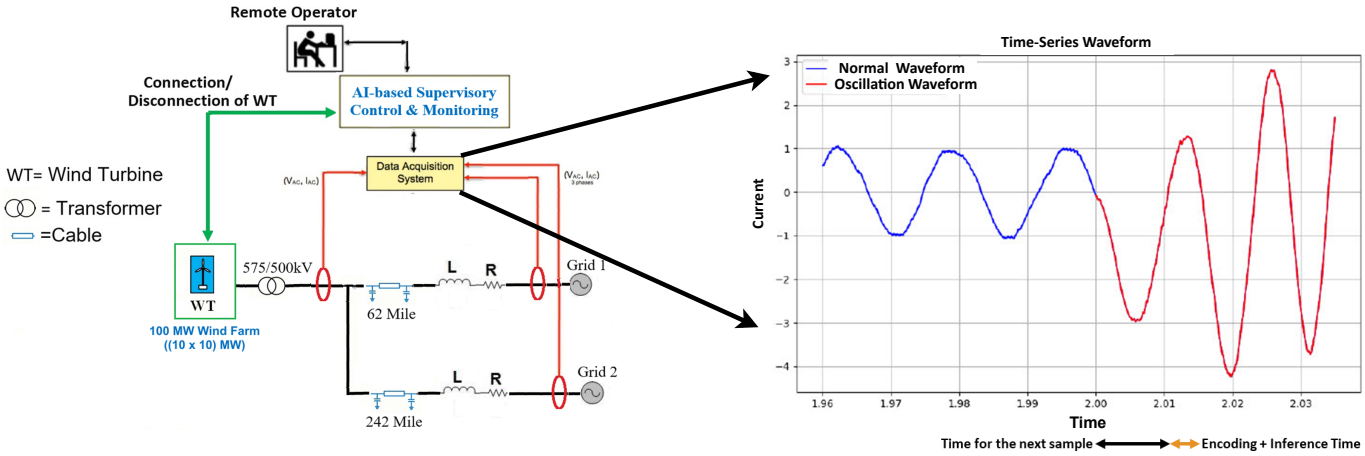

Fig. 1. Configuration of the studied wind farm based on wind turbine-Type 3 connected to the utility grids to detect forced oscillations based on Machine Learning schemes. Normal waveform is shown in blue and the forced oscillation is shown in red. The total time to detect the forced oscillation is depicted as: total time = time for the next sample + computation to classify the sample.

the model and 2) The time it takes for the model to process the inference sample. This is the case for all ML model-based solutions for oscillation detection.

There have been multiple reported studies and solutions to identify and detect forced oscillations that appeared in both land-based and offshore wind farms [6]–[9]. In recent years, several techniques have been proposed based on a field-programmable gate array (FPGA), or CPU-based monitoring systems to offer both low-frequency harmonics monitoring functions and recording the required data for the post-event evaluation [8], [9]. Data analytics based on time-series methods are introduced by the research community for anomaly detection to expedite decision-making in numerous industrial applications. For large-scale power systems, an efficient monitoring and control scheme is crucial when the number of required measurements and datasets is very large. In [10], Deep Learning algorithms are examined to obtain information about the working conditions of the conventional power grid without downgrading computational performance. Despite the superior performance shown by these techniques for traditional power systems, the observed fast dynamics of renewable energy generations and control system interactions demand different performance monitoring strategies that are not fully explored in real-time. It is understood that the state-of-the-art and existing methodologies proposed for wind farms are relying on converting field measurements to waveform images to determine considerable overhead for corresponding electric variables in real-time analysis and deployment at the edge device [11], [12].

2D CNN Approach:

In the context of wind farm oscillation detection, researchers have explored the use of 2D Convolutional Neural Networks (CNNs) [13]. This approach involves converting the time-series data representing wind farm sensor measurements into 2D image-like representations [14]. The raw wave data is converted into images by applying a fixed sampling rate per second. This transformation facilitates the creation of images with time as one dimension and signal intensity as the other. These images encapsulate the temporal patterns present within the raw data and are used as input for the 2D CNN model. The 2D CNNs are then applied to these waveform images to automatically learn spatial and temporal features that are relevant for detecting oscillations. However, these methods bring extra cost with the extra pre-processing techniques. For example, to create a new current image, the model needs to wait for 666 samples to be generated. Therefore, increasing the delay between the oscillation and detection with the requirement to gather more data. Additionally, 2D CNN models are computationally expensive and large models, once again increasing the total delay, but due to the computational portion.

1D CNN Approach:

Another approach involves using 1D Convolutional Neural Networks (CNNs) [15] directly on the time-series data. In this case, the data is not converted into 2D images, but instead, the 1D structure of the time-series is preserved, and 1D CNNs can learn patterns and features from the sequential data. This reduces the data gathering portion of the time delay. 1D CNNs are also smaller and more efficient than 2D CNNs, which also reduces the computational part of the delay. However, this approach is less accurate than the 2D CNN model.

B. State-of-the-Art Challenges and How HDC Address Them

CNN Challenges:

While CNN-based methods offer powerful tools for pattern recognition, they are not devoid of limitations. The majority of these solutions are suffering from practical deployment in real-world applications, considering the operational requirements and data acquisition limitations of wind energy systems. The main contribution of this study is to apply AI-based performance monitoring techniques that have not been considered for emerging wind power grids to establish a benchmark analysis of datasets, providing a real-time assessment of any low-frequency oscillations that appear in such systems. Our system provides a more responsive real-time analysis in two main ways: 1) Using a more light-weight and faster algorithm in HDC and 2) Classifying real time series data from the measurement system directly without converting the data into a 2D image. Our method aims to reduce the delay from both portions: reducing the number of samples needed by the model reduces the data gathering portion of the delay, and using a more efficient model than CNNs reduces the computational portion of the delay.

Improvements with HDC Approach:

Building upon the foundation of CNN-based approach and cognizant of their limitations, this paper introduces novel approach rooted in HDC. This approach offers a uniqu perspective on addressing the oscillatory signal classificatic challenge within wind farm operations. HDC encoding direct operates on raw time series data, without the need for in age conversion. The inherent characteristics of HDC enab computational efficiency, catering to the constrained compu environment of IoT devices while maintaining accuracy.

C. Hyperdimensional Computing Background

Encoding Data in Hyperspace.

At the core of our approach lies the transformation of data into a higher-dimensional space through a well-defined mathematical process [4], [16], [17]. This encoding process is crucial for capturing the inherent characteristics of the data and enabling effective analysis. For each sample, which represents a sequence of N sequential values from the original one-dimensional raw data, we utilize a dot product operation with the corresponding base vector, denoted as '$\mathcal{B}[i]$'. This operation quantifies the alignment of the sample with the specific base vector, effectively representing its underlying attributes and properties. To introduce variability and adaptability to the encoding process, a random component denoted as '$\mathcal{R}[i]$' is incorporated. This random value is generated with a Gaussian distribution, adding a dynamic aspect to the encoding [18]. The mathematical representation of this encoding process is as follows:

$$\mathrm{H}[i] = \mathrm{sign}(\cos(X \cdot \mathcal{B}[i] + \mathcal{R}[i]) \cdot \sin(X \cdot \mathcal{B}[i] + \mathcal{R}[i])) \quad (1)$$

Here, $'X'$ signifies the *sample*, while $\mathcal{B}[i]$ and $\mathcal{R}[i]$ represent the corresponding base vector and random value, respectively. The dot product operation captures the alignment between the sample and the base vector, while the sinusoidal functions introduce variability and distinctive patterns into the encoded representation. Finally, the sign function binarizes the encoded vector.

This encoding process serves as a pivotal step in our exploration of HDC, as it encapsulates the essence of transforming real-world data into a higher-dimensional space that can be harnessed for advanced analysis. Furthermore, the encoded Hypervectors are quantized to binary values $(-1, 1)$, and all subsequent operations in HD space are performed on binary values, improving the efficiency and performance of the algorithm on embedded systems [19].

HDC Training:

To train the Class HVs, we simply accumulate all of the encoded samples into one HV representing each class [5]. For all training samples $\mathcal{F}^j$ with label $y_j = \ell$, HDC produces and accumulates encoded vectors $\vec{\mathcal{H}}^j$ to create the class vector $\vec{\mathcal{C}}^\ell$:

$$\vec{\mathcal{C}}^\ell = \sum_{j \text{ s.t. } y_j = \ell} \vec{\mathcal{H}}^j$$

To boost the accuracy, the retraining epochs of HDC perform inference on the training samples and update the classes by adding the mispredicted vector $\vec{\mathcal{H}}$ to the right class $\vec{\mathcal{C}}^\ell$ (again) and subtracting it from the wrongly suggested $\vec{\mathcal{C}}^{\ell'}$.

$$\vec{\mathcal{C}}^\ell = \vec{\mathcal{C}}^\ell + \vec{\mathcal{H}} \qquad \vec{\mathcal{C}}^{\ell'} = \vec{\mathcal{C}}^{\ell'} - \vec{\mathcal{H}} \quad (2)$$

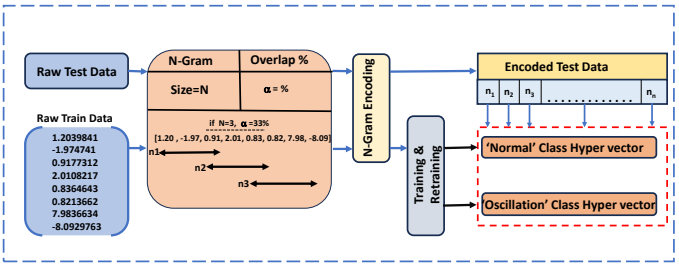

Fig. 2. HDC Flow representing the NGrams formation and their encoding to high dimensional space and then class vectors formation and improvement with retraining epochs.

After the retraining procedure, the Class HVs are quantized again into binary values [20]. This improves the efficiency and performance of the algorithm on embedded systems while not impacting the accuracy of the model.

Inference:

The inference of HDC is accomplished by comparing the encoded sample $\vec{\mathcal{H}}$ with the class vectors and returning the label with the highest score. As we are binarizing the vectors, we use hamming distance to find out the similarity between class vectors and data samples [21]. This allows for an efficient implementation of HDC in hardware, as we only need to count the number of mismatches between binary dimensions. This can be achieved with an array of XOR gates. The equation for this is shown below where $\ell^\star$ is the output label, $\vec{\mathcal{H}}$ is the query hypervector, and $\vec{\mathcal{C}}$ are the class hypervectors.

$$\ell^\star = \underset{j \in \mathcal{C}}{\mathrm{argmax}} \ \mathrm{Hamming}(\vec{\mathcal{H}}, \vec{\mathcal{C}}^j)$$

III. HDC for Oscillation Detection

As mentioned in Section II-A, CNNs are the current state-of-the-art models for oscillation detection in power systems. However, the conversion of time series data into images, as seen in the 2D CNN methodology, introduces complexity. Paramount to all of this, converting the raw time series data into the frequency domain and then creating the images for the CNN to process takes time. The time to detect the oscillations is critical in this application. Furthermore, the computational demands of CNNs, especially in resource-constrained environments, pose challenges for real-time implementation on IoT devices within wind farms. The intricate network architecture and computations involved can strain the capabilities of these devices, potentially delaying critical decision-making processes. In this work, we propose the use of HDC as an alternative and efficient approach. We propose two ways for HDC to improve the time delay of detecting the oscillations: 1) We introduce a new method of generating sample data with the raw data requiring significantly fewer data samples to create an inference sample (seen in Figure 2), and 2) We demonstrate that HDC is just as accurate as CNN based models at detecting the oscillations while being orders of magnitude faster.

A. Data Generation Strategies

Before discussing the changes our method makes in creating samples and labeling them, we will first cover how current state-of-the-art works. In prior work, labeling a sample as "oscillating" required that the entire data window exhibited oscillatory behavior [3]. This approach further adds delay in detecting the oscillation as the model needs to have a sample

fully oscillating before it can effectively classify the sample. Furthermore, for 2D CNNs, it is necessary to wait for a sufficient number of data points in order to create an image of the current waveform. This introduces even more delay. 1D CNNs alleviated this by working on the raw data with each new sample consisting of the next N data points, where N is much smaller than the number of data points to create a new image. We will call these *NGrams*. *NGrams* can be represented as follows:

$$\text{NGram}_i = [x_i, x_{i+1}, \ldots, x_{i+N-1}]$$

However, the downfall of 1D CNNs is that they are not as accurate in classifying as a 2D CNN. Our work utilized these *NGrams* as the basis for our samples, but introduces innovative labeling strategies to further reduce the delay.

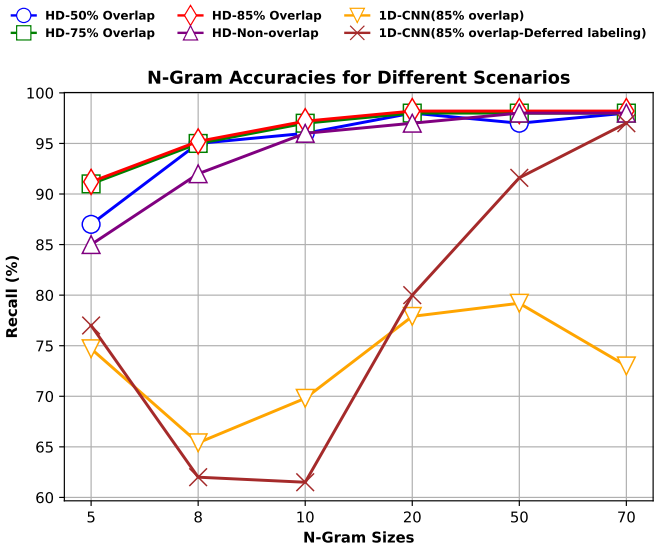

Fig. 3. Impact of N-gram window size, aggressive labeling, and overlapping windows on Recall

B. Aggressive Labeling Strategy

In our aggressive labeling strategy, each *NGram* undergoes rigorous scrutiny to determine its label – oscillated or normal. Specifically, we employ a stringent criterion: if any data point within an NGram coincides with a known fault event, the *NGram* is unambiguously labeled as oscillated. Conversely, *NGrams* devoid of such critical events are labeled as normal. This kind of aggressive labeling helps us identify the oscillatory signals faster than state-of-the-art models by ensuring samples are labeled as oscillating as early as possible. For example, with an *NGram* of 70 and sample rate of 50μs, this labeling strategy can reduce the time delay of gathering samples by up to $2\times$ if there is a portion of the sample that is part of an oscillation. However, this strategy includes even small evidence of an oscillation signal and labels it as an oscillatory data point, introducing a challenging environment for subsequent analysis. However, in Figure 3, we can see that even with this aggressive labeling strategy, HDC is able to classify the samples accurately. However, when we use this strategy with CNNs, they are not able to maintain accuracy. For instance, Figure 3 shows that for the same Ngram size of 70, the 1D CNN loses over 10% in recall rate when using the aggressive labeling strategy compared to needing the entire sample oscillating (Deferred Labeling). We further reduce the time delay of detecting oscillations by introducing overlapping windows.

C. Window Overlap

We introduce the concept of overlap to further reduce the time delay of detecting oscillations. Overlapping scenarios in the context of *NGrams* refer to the way these windows overlap in time. By shifting NGrams with controlled overlaps – such as 50%, 75%, and 85% – our model requires fewer data points to create a new sample. For overlapping scenarios with a given overlap percentage α, the starting point of the next NGram is determined by the overlap window size $\beta = (1-\alpha) \times N$. Thus, the NGrams in the overlapping scenario can be expressed as:

$$\text{NGram}_i = [x_i, x_{i+1}, \ldots, x_{i+N-1}]$$

$$\text{NGram}_{i+\beta} = [x_{i+\beta}, x_{i+\beta+1}, \ldots, x_{i+\beta+N-1}]$$

In Figure 3, we show the impact of overlapping as well as different *NGram* sizes. We have observed that overlapping experiments were able to maintain a constant recall rate. Moreover, our experiments have revealed a remarkable finding: the use of overlapping scenarios with HDC significantly brings up the choice to detect oscillations in a faster way. This, in combination with aggressive labeling, allows a swifter identification of operational concerns that are crucial in wind farms. This enables the HDC model to achieve faster oscillation detection without an impact on recall. Furthermore, we conduct a comparative analysis with CNN-based methods as well. A key advantage of HDC becomes apparent: while CNNs require the necessity of labeling delay, HDC does not require this delay. This factor significantly contributes to the effectiveness of HDC in being able to detect oscillatory signals in a faster way than CNNs. In summary, our comparative analysis suggests that while 1D and 2D CNNs have demonstrated prowess in capturing intricate patterns, HDC presents a viable alternative, potentially offering a more focused and efficient solution for the classification of oscillatory and normal signals in wind turbine data. For example, our findings show that HDC with an *NGram* size of 10 is able to maintain the same accuracy as state-of-the-art CNNs, which require an NGram of 70 and deferred labeling. Comparing HDC with 1D CNNs that require delayed labeling and a minimum *NGram* size of 70, just by reducing the delay of gathering data, HDC shows an improvement of up to $35\times$ as HDC only needs two new data points to create a new sample, where the CNN needs to sample 70 new data points to create a new sample.

Figure 3 demonstrates that HDC is successful in maintaining a high and constant recall rate at different scenarios. Furthermore, due to the light-weight nature of HDC, the encoding and processing steps used in HDC can be efficiently mapped to the Field-Programmable Gate Arrays (FPGAs), a hardware architecture that is commonly used in IoT devices. Using different methodologies mentioned in the next subsections, we are able to further reduce the delay of detecting oscillations by speeding up HDC inference delay, or the second component of the total delay. This makes HDC an efficient alternative approach to be deployed in wind farms for oscillation detection.

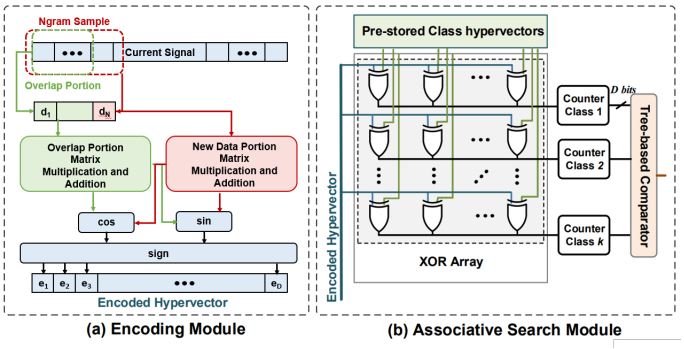

Fig. 4. FPGA implementation of the encoding and associative search block.

IV. Efficient Hardware Implementation

HDC can be accelerated on different platforms such as Central Processing Unit (CPU), Graphics Processing Unit(GPU), Field-Programmable Gate Array(FPGA), or Application-Specific Integrated Circuit(ASIC) [5], [22]–[24]. FPGA is one of the best options as HDC computation involves bitwise operations among long vector sizes [25]–[27]. For example, during inference, we need to calculate hamming distance, which simply counts the number of mismatching binary dimensions. Additionally, unlike ASICs or Program In Memory (PIM) implementations, FPGAs offer reprogramability and faster design times [28], [29]. General strategies for optimizing the performance of HDC are (i) using a pipeline and partial unrolling on a low level (dimension level) to speed up each task and (ii) using dataflow design on a high level (task level) to build a stream processing architecture that lets different tasks run concurrently. In the following, we explain the functionality of HDC in encoding, inference phases, training, and retraining.

A. Encoding Implementation

As we explained in Section II, we use a non-linear encoding to implement the encoding module. Due to the sequential and predictable memory access patterns, this encoding approach can be implemented efficiently on an FPGA. Figure 4a shows the hardware implementation of the HDC encoding module. The core part of the encoding process is the matrix multiplication between the non-linear matrix and the sample to be encoded. This matrix multiplication is unfolded and pipelined for an efficient implementation. However, due to overlapping samples, our design can also precompute the overlapping part of the sample for the next input. This matrix multiplication and the subsequent addition is implemented in Digital Signal Processors(DSPs). However, for a more efficient implementation, the sin and cos applied at the end are implemented in Look Up Tables(LUTs).

B. Inference Implementation

In the hardware implementation, we represent all $\{-1, +1\}$ values with $\{0, 1\}$ respectively. This enables us to represent each element of the hypervectors using a single bit. After the retraining, the HDC model is a stable model that can be used in the inference phase. The encoding module is integrated with the similarity check module as the entire inference part. Each test data point is first encoded to high-dimensional space using the same encoding block explained in Section IV-A. Next, the design checks the Hamming distance similarity of the data point with all pre-stored class HVs, in order to find the class with the highest similarity. This is done efficiently with an array of XOR gates and a tree-based comparator. This design is also unfolded and pipelined to improve performance during training and retraining. However, the pipelined architecture is unable to be fully utilized during inference due to the sequential nature of the data.

C. Training Implementation

The initial training and training blocks utilize the existing encoding and inference blocks. For initial training, the samples first need to be encoded by the encoder block, then they are simply accumulated into Flip Flips storing the class HVs. Then, during retraining, the sample is again re-encoded and sent to the inference block. The inference block result then controls if the sample is subtracted and added to the Flip Flops based on if the sample was labeled correctly or not.

V. Experiments and Results

A. Experimental Setup

We have implemented HDC training, retraining, and inference on both software and Hardware. For software, we used Python. For the CPU architecture, we used an Intel(R) Xeon(R) Gold 6144 CPU @ 3.50GHz with a total RAM of 64GB. For GPU, we utilized NVIDIA Quadro P5000 which has 16GB of VRAM. For FPGA, we have implemented our design in Verilog. We verified the timing and functionality of the models by synthesizing them using Xilinx Vivado Design Suite [30]. The synthesis code has been implemented on a Kintex-7 FPGA KC705 Evaluation Kit.

We have evaluated the performance of our design and tested it on 3 different datasets. All of these datasets are generated from silulation data using MATLAB and Simulink. The 3 datasets have different characteristics listed below: Windfarm Dataset-1 (sampled at $50\mu s$ with 1 oscillation included), Windfarm Dataset-2 (sampled at $10\mu s$ which includes no oscillation data), and Windfarm Dataset-3 (sampled at $10\mu s$ that includes 3 oscillation waves in it). In this section, we present the results of our method based on HDC and compare them with the 1D and 2D CNN approaches. All datasets are split into train and test sets. The train set is simulated for 15 seconds and the test set is simulated for 6 seconds.

B. Multiple Oscillations and Faster Sampling

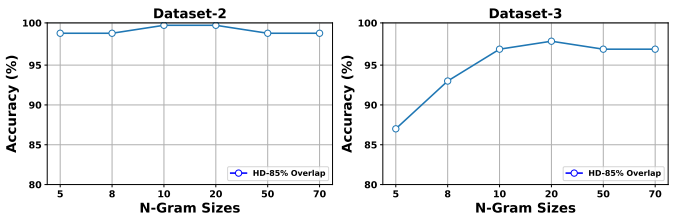

Fig. 5. Performance of HDC on Dataset-2 with no oscillations included and Dataset-3 with multiple oscillation included.

The results in Section III were based on dataset-1, which is sampled at a rate of $50\mu s$; this dataset contained a single oscillation pattern. However, In real-world systems, samples are drawn at a faster rate, around $10\mu s$ as well as containing multiple oscillations. To address this, we evaluate HDC on two additional datasets.

Dataset 2:

Sampled at a rate of 10μs, this dataset presented a scenario with no distinct oscillation patterns embedded within the data. This dataset is used as a control for normal operation to ensure our model does not identify any false positives as normal operation and this is a much more common case compared to oscillating.

Dataset 3:

Sampled at a rate of 10μs, this dataset presents a more complex scenario with three distinct oscillation patterns embedded within the data.

Figure 5 shows the accuracy of our HDC model with different NGram sizes on the additional datasets. The results on dataset 2 demonstrate that HDC with an NGram of 10 does not indicate any false positives. This is critical to maintain optimal performance of the system as any false positive would unnecessarily halt the system. Our results also show that HDC maintained performance on dataset 3, which has multiple oscillations, achieving an accuracy rate of 97% across various NGrams and only dropping after lower than 10 NGrams. This underscores its ability to accurately identify oscillations even in the presence of multiple occurrences. Furthermore, we can say that our HDC model works and achieves the best performance when the NGram is 10 and the overlap is 85%. Additionally, we can see that HDC is able to accurately identify normal operation with no false positives on dataset 2.

By demonstrating that the HDC model is able to accurately identify oscillations with a dataset sampled at 10μs, HDC demonstrates a further advantage in terms of time to detect the oscillation. With a sample rate of 10μs vs 50μs, our model sees data points at a rate $5\times$ faster. This means that the HDC model will not only generate a sample faster by requiring fewer data points, but also, by collecting data points faster. This also demonstrates that HDC can be deployed in real systems where the sample rate is closer to 10μs.

C. HD Performance and Energy Efficiency

In the context of real-world applications, where efficiency, speed, and energy consumption play pivotal roles, we assessed our HDC methodology in diverse scenarios. These evaluations were conducted on three platforms: a server (CPU), a Raspberry Pi, and an FPGA implementation. The aim was to understand the efficiency of our approach across a spectrum of computational environments.

Figure 6(A) indicates training time taken by HDC on different hardware platforms that include the time taken for encoding the windfarm data and also the time taken for 20 retraining epochs. The results show that HDC performance on the FPGA platform is boosted by *23,100*$\times$ compared to a CPU server and by *111,100*$\times$ compared to a Raspberry Pi. Figure 6(B) indicates the energy taken by HDC on different Hardware platforms during the training and retraining process. The results show that HDC performance on the FPGA platform is *138,800*$\times$ more energy efficient compared to a CPU server and by *117,200*$\times$ compared to a Raspberry Pi. Although FPGA is clearly a superior hardware platform for HDC training, it is important to note that training is one offline and only run once. Therefore, it is still feasible to run HDC training on a CPU server as training only takes approximately 2 hours on a CPU. However, training takes approximately 2 days to run on a Raspberry Pi and is not feasible.

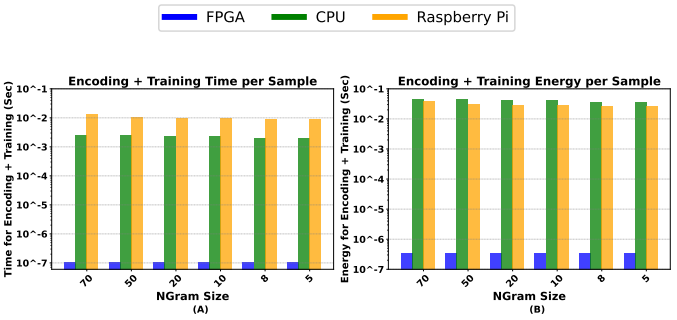

Fig. 6. (A) Representation of Encoding + Training Time per sample on different Hardware (B) Representation of Encoding + Training Energy per sample on different Hardware

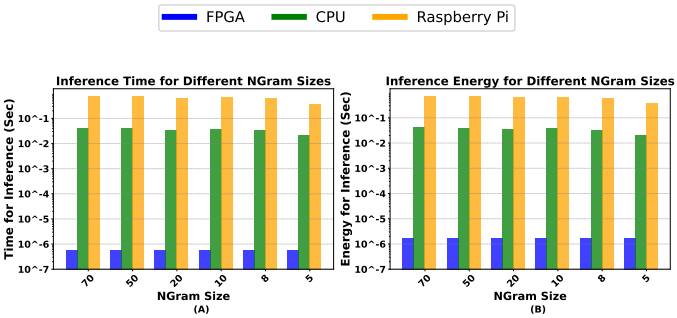

Fig. 7. (A) Representation of inference time per sample on different hardware (B) Representation of inference energy per sample on different hardware.

In the context of windfarm data, it is also important to observe the time HDC takes to detect an oscillation, which plays an important role in determining the effectiveness of the algorithm, and Figure 7(A) above showcases the inference time of HDC on different hardware platforms. For the following comparisons, we utilize an NGram of 10 and overlap of 85%. Our results indicate that our HDC model on FPGA is, *83,300*$\times$ faster than the CPU server and *459,200*$\times$ faster than Raspberry Pi. However, it is important to note that the time to detect oscillations does not only depend on the inference time, but also the time to collect a full sample for inference.

When we consider the entire process to detect an oscillation, utilizing an FPGA for acceleration is *111*$\times$ faster than the CPU server and *460*$\times$ faster than Raspberry Pi. This is because a large portion of the time for detection is waiting for the next sample for inference. For example, HDC with an NGram of 10 and an overlapping scenario of 85% requires 2 data points to create a new sample. This means that there is a delay of 20μs between each new sample. Another noteworthy aspect about running HDC inference on CPUs and Raspberry Pis is that inference time is longer than the time it takes to collect the next sample. Therefore, these systems are not feasible for deployment of the HDC model in a real system.

Figure 7(B) shows the energy consumption of the HDC model on the different hardware platforms. The results indicate that not only is FPGA faster at detection, but it is also significantly more energy efficient. Our HDC model on FPGA is, *25,300*$\times$ more energy efficient compared to the CPU server and by *456,700*$\times$ compared to a Raspberry Pi. Therefore, it is also not feasible to utilize CPUs or Raspberry Pis for oscillation detection in terms of energy efficiency either. FPGAs are notably faster at detection and also orders of magnitude more energy efficient.

TABLE I
HDC VS STATE-OF-THE-ART CNNS COMPARISON.

Model	Recall Rate	Data Collection Time	Inference Time	Time to Detect Oscillation	Inference Energy	Training Time	Training Energy	Model Size
2D CNN [3]	98%	33.3ms	20ms(CPU) 21ms(GPU)	53.3ms(CPU) 54.3ms(GPU)	3.1J(CPU)	1ms(CPU) 0.6ms(GPU)	0.15J(CPU)	2.7MB
1D CNN [3]	94%	110μ	2ms(CPU) 3ms(GPU)	2.11ms(CPU) 3.11ms(GPU)	0.29J(CPU)	0.8μs(CPU) 0.46μs(GPU)	12mJ(CPU)	0.6MB
HDC	97%	20μs	41ms(CPU) 0.22μs(FPGA)	41.02ms(CPU) 20.22μs(FPGA)	740mJ(CPU) 0.65μJ(FPGA)	2.3ms(CPU) 0.04μs(FPGA)	41mJ(CPU) 0.13μJ(FPGA)	3.3KB

D. Further improvements with Dimensionality Reduction

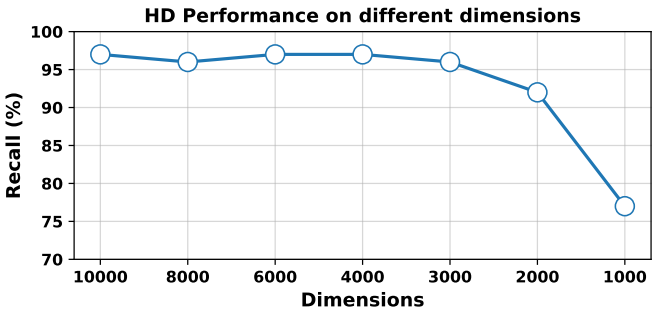

Fig. 8. HDC NGram-10 and 85% Overlap Performance for Different Dimensions.

Figure 8 shows the impact of model dimensionality on the accuracy of the HDC model. The results indicate that, HDC is able to maintain high accuracy through different dimensions. Particularly, HDC was able to maintain a constant recall rate with dimensionality as low as $4,000$. This is noteworthy as our FPGA implementation scales by the dimensionality of the HDC model. Therefore, by demonstrating that the HDC model is able to maintain accuracy up to 4,000 dimensions, our FPGA implementation with 4,000 lower dimensions is $2.5\times$ faster and more energy efficient compared to the full 10,000 dimensional model.

E. Comparison with State of the Art

Table I compares the HDC model with 1D and 2D-CNNs in terms of different performance metrics such as recall rate, time that each model takes to detect oscillations, energy efficiency, and model sizes. In this table, we report the metrics for the best parameters for HDC and CNNs. The parameters for HDC were explored in the Section V-C and Section V-D. We found that HDC can achieve a balance of recall, energy efficiency, and execution time when running on an FPGA, with a dimensionality of $4,000$, an NGram of 10, and an overlap percentage of 85%.

Table I also focuses on 1D CNN and 2D CNN where 1D CNN performance is recorded for 70 NGram with an overlapping window of 85% whereas the 2D CNN performance was obtained from the process where the time series data was converted into images and then passed through a 2D CNN for classification. From the table, we can clearly see that the 1D CNN is the better model for deployment as it is able to detect oscillations $25\times$ faster than the 2D CNN, trains the model $1,300\times$ faster, has a $4.5\times$ smaller model size, with the cost of 4% in recall rate.

For the CNN models we also included a comparison running on CPU vs GPU. It is noteworthy that for inference, GPU is actually worse for these models. This is because in the oscillation detection application, only one sample is available at a time. Therefore, the GPU resources are significantly underutilized during the inference process and the overhead of moving the data from CPU memory to GPU memory is not overcome by the computational speedup. However, utilizing GPU for training is approximately $2\times$ faster. Therefore, for the best comparison against out HDC model, we use the 1D CNN running on a CPU for inference comparisons and the 1D CNN running on GPU for training comparisons.

For 1D CNN, the best recall with smallest NGram and highest overlap is with an NGram of 70 and overlap of 85% as with less than 70 NGrams recall is significantly lower. For an application like oscillation detection, recall is paramount to all other metrics. This significantly impacts the total time to detect and oscillation as the 1D CNN needs to wait for $5.5\times$ more samples to create a new NGram compared to HDC. Due to this difference and the performance difference of computing the inference phase, HDC is $55\times$ faster at detecting oscillations than the 1D CNN. Furthermore, the HDC model is trained $7.6\times$ faster than the 1D CNN model. In terms of energy efficiency, the HDC model is 5 orders of magnitude more energy efficient than the 1D CNN model during inference and 4 orders of magnitude more energy efficient during training. The HDC model is also $182\times$ smaller than the 1D CNN model. HDC achieves this while maintaining the same recall rate as 2D CNNs, without the need for deferred labeling. Finally, HDC is the only model, besides the significantly slower 2D CNN, that is able to perform inference faster than the time it takes to collect data for the next sample. Therefore, in terms of feasible deployment, HDC is the only choice.

F. Future Work

The results presented in this paper are initial findings using HDC for forced oscillations detection in benchmarked wind farms. There are multiple ways we plan to extend this work. For instance, in this work our models only detect oscillations appear at the interconnection to the grid. In real wind energy plants, there are many monitoring supervisory schemes used in different locations. For example, a wind farm comprises of multiple turbines generation units, each of which need to be equipped for this type of oscillation. There are multiple aspects to explore with more nodes. Such as how we showed with one node to monitor, GPUs are underutilized for CNN architectures and result in worse performance. However, with more data available, the GPU can be better utilized. The same is true for the HDC FPGA architecture. During inference, the pipeline is not fully utilized. This is why training is approximately $5\times$ faster than inference as all of the training samples are available and can fill up and fully utilize the pipelined architecture in

the FPGA. With more nodes added during inference, more samples are needed to be tested and the pipeline can be filled properly, leading to more throughput. Additionally, with the FPGA implementation of HDC, the data collection phase is now the longer phase. We can further optimize the time to detect by increasing the sampling rate further considering the practical limitations. Furthermore, this work can be extended to detect oscillations in other renewable energy systems besides wind farms like solar farms. Finally, the datasets tested in this paper are purely simulation data. We have plans to extend this work to test on real world measurements as the data becomes available.

VI. Conclusion

In this study, we tackled the crucial problem of detecting oscillations in the power grid systems with a focus on highlighting the limitations of Deep Convolutional Neural Networks and eliminating the pre-processing steps that are mandatory in CNNs. This is the first work to utilize a more light-weight and efficient model in HDC to improve upon the existing state-of-the-art CNNs. Our approach with HDC reduces the total time to detect forced oscillations in wind farms in two ways. First, we reduce the number of data points required to create a new sample, which in turn reduces the delay to detect an oscillation. Second, by utilizing a more efficient model in HDC, we reduce the computational time of the inference phase on the sample. Additionally, to achieve the best performance, we implemented our HDC model in an FPGA architecture. Our results show that HDC, with an FPGA implementation, is able to achieve $55\times$ faster detection of forced oscillations in windfarms while achieving the same accuracy as the best current CNN models using software solutions.

Acknowledgements

This work was supported by NSF grants #2334256.

References

[1] Q. Lin, B. Wen, R. Burgos, X. Li, Q. Wang, and X. Li, "Dq impedance modeling and stability analysis of a three-phase four-wire system with single-phase loads," *IEEE Transactions on Power Electronics*, 2023.

[2] T. Heins, M. Joševski, S. Gurumurthy, and A. Monti, "Centralized model predictive control for transient frequency control in islanded inverter-based microgrids," *IEEE Transactions on Power Systems*, 2022.

[3] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[4] 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.

[5] M. Imani, A. Rahimi, D. Kong, T. Rosing, and J. M. Rabaey, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 445–456, IEEE, 2017.

[6] F. Salehi, I. B. M. Matsuo, A. Brahman, M. A. Tabrizi, and W.-J. Lee, "Sub-synchronous control interaction detection: A real-time application," *IEEE Transactions on Power Delivery*, vol. 35, no. 1, pp. 106–116, 2019.

[7] B. Zhang and H. Nademi, "Modeling and harmonic stability of mmc-hvdc with passive circulating current filters," *Ieee Access*, vol. 8, pp. 129372–129386, 2020.

[8] L. Zhao, Y. Zhou, I. B. Matsuo, S. K. Korkua, and W.-J. Lee, "The design of a remote online holistic monitoring system for a wind turbine," *IEEE Transactions on Industry Applications*, vol. 56, no. 1, pp. 14–21, 2019.

[9] Z. Zheng, S. Huang, Q. Bu, C. Shen, and J. Yan, "Detection of nonlinear behavior induced by hard limiting in voltage source converters in wind farms based on higher-order spectral analysis," *Journal of Modern Power Systems and Clean Energy*, 2023.

[10] S. A. Dorado-Rojas, T. Bogodorova, and L. Vanfretti, "Time series-based small-signal stability assessment using deep learning," in *2021 North American power symposium (NAPS)*, pp. 1–6, IEEE, 2021.

[11] G. R. Garcia, G. Michau, M. Ducoffe, J. S. Gupta, and O. Fink, "Temporal signals to images: Monitoring the condition of industrial assets with deep learning image processing algorithms," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 236, no. 4, pp. 617–627, 2022.

[12] S. A. Dorado-Rojas, S. Xu, L. Vanfretti, M. I. I. Ayachi, and S. Ahmed, "Ml-based edge application for detection of forced oscillations in power grids," in *2022 IEEE Power & Energy Society General Meeting (PESGM)*, pp. 1–5, IEEE, 2022.

[13] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, pp. 1–6, Ieee, 2017.

[14] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525–538, 2018.

[15] W. Tang, G. Long, L. Liu, T. Zhou, J. Jiang, and M. Blumenstein, "Rethinking 1d-cnn for time series classification: A stronger baseline," *arXiv preprint arXiv:2002.10061*, pp. 1–7, 2020.

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

[17] A. Thomas, S. Dasgupta, and T. Rosing, "Theoretical foundations of hyperdimensional computing," *Journal of Artificial Intelligence Research*, vol. 72, pp. 215–249, 2021.

[18] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahim, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *arXiv preprint arXiv:2112.15424*, 2021.

[19] M. Imani, S. Bosch, S. Datta, S. Ramakrishna, S. Salamat, J. M. Rabaey, and T. Rosing, "Quanthd: A quantization framework for hyperdimensional computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2268–2278, 2019.

[20] M. Imani, J. Morris, J. Messerly, H. Shu, Y. Deng, and T. Rosing, "Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing," in *Proceedings of the 56th Annual Design Automation Conference*, pp. 1–6, 2019.

[21] M. Imani, Y. Kim, S. Riazi, J. Messerly, P. Liu, F. Koushanfar, and T. Rosing, "A framework for collaborative learning in secure high-dimensional space," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 435–446, IEEE, 2019.

[22] J. Kang, B. Khaleghi, T. Rosing, and Y. Kim, "Openhd: A gpu-powered framework for hyperdimensional computing," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2753–2765, 2022.

[23] J. Morris, H. W. Lui, K. Stewart, B. Khaleghi, A. Thomas, T. Marback, B. Aksanli, E. Neftci, and T. Rosing, "Hyperspike: hyperdimensional computing for more efficient and robust spiking neural networks," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 664–669, IEEE, 2022.

[24] T. Zhang, J. Morris, K. Stewart, H. W. Lui, B. Khaleghi, A. Thomas, T. Goncalves-Marback, B. Aksanli, E. O. Neftci, and T. Rosing, ": Accelerating event-based workloads with hyperdimensional computing and spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

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

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

[27] J. Morris, S. T. K. Set, G. Rosen, M. Imani, B. Aksanli, and T. Rosing, "Adaptbit-hd: Adaptive model bitwidth for hyperdimensional computing," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pp. 93–100, IEEE, 2021.

[28] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, vol. 3, no. 6, pp. 327–337, 2020.

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

[30] T. Feist, "Vivado design suite," *White Paper*, vol. 5, 2012.