Brain-inspired Trustworthy Hyperdimensional Computing with Efficient Uncertainty Quantification

Yang Ni[1], Hanning Chen[1], Prathyush Poduval[2], Zhuowen Zou[1], Pietro Mercati[3], and Mohsen Imani[1*]
[1]University of California Irvine, [1,2]University of Maryland, [3]Intel Labs
[*]Corresponding author: m.imani@uci.edu

2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD) | 979-8-3503-2225-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCAD57390.2023.10323657

***Abstract*—Recent advancement in emerging brain-inspired computing has pointed out a promising path to Machine Learning (ML) algorithms with high efficiency. Particularly, research in the field of HyperDimensional Computing (HDC) brings orders of magnitude speedup to both ML model training and inference compared to their deep learning counterparts. However, current HDC-based ML algorithms generally lack uncertainty estimation, despite having shown good results in various practical applications and outstanding energy efficiency. On the other hand, existing solutions such as the Bayesian Neural Networks (BNN) are generally much slower than regular neural networks and lead to high energy consumption. In this paper, we propose a hyperdimensional Bayesian framework called DiceHD, which enables uncertainty estimation for the HDC-based regression algorithm. The core of our framework is a specially designed HDC encoder that maps input features to the high dimensional space with an extra layer of randomness, i.e., a small number of dimensions are randomly dropped for each input. Our key insight is that by using this encoder, DiceHD implements Bayesian inference while maintaining the efficiency advantage of HDC. We verify our framework with both toy regression tasks and real-world datasets. We compare our DiceHD to several widely-used BNN baselines in terms of performance and efficiency. The results on CPU show that DiceHD provides comparable uncertainty estimations while achieving significant speedup compared to the BNN baseline. We also deploy DiceHD on two FPGA platforms with different acceleration capabilities, showing that DiceHD provides up to 84× (3740×) better energy efficiency for training (inference).**

I. Introduction

In the past ten years, research in the area of deep learning observed the fast growth of Deep Neural Network (DNN) based algorithms. We have seen that DNNs fundamentally change how machine learning interacts with our daily life through their advancements in natural language processing, object detection, and reinforcement learning. However, the complexity of DNNs and the computation cost of using such networks have also been increasing significantly. This inevitably leads to a surge of power consumption for training and inference, which essentially contrasts with the power limit and efficiency requirements of edge computing. Compared to the human brain, DNN-based algorithms are surprisingly inefficient, albeit the fact that neural networks are bio-inspired to start with.

Therefore, novel brain-inspired computing methods such as Spiking Neural Networks (SNN) and HyperDimensional Computing (HDC) are gaining traction because of their better efficiency [1], [2] and robustness against hardware noise [3]. In particular, HDC mimics human brain functionalities by learning and reasoning in high-dimensional spaces with lightweight operations [4], [5]. This is supported by the finding that information on sensory inputs is stored in the cerebellum cortex using high-dimensional neural activity patterns [6]. To enable HDC operations, inputs from the original low-dimensional space are encoded to vectors with thousands of dimensions, i.e., hypervectors. HDC-based algorithms, equipped with lightweight computations, are usually easily parallelizable using off-the-shelf hardware accelerators so that the efficiency is further improved [7]. Prior works [1] show that HDC provides a significant efficiency boost over other widely-deployed ML algorithms such as DNN and Support Vector Machine (SVM). Recent research brings this advantage of HDC to different kinds of learning tasks like classification/clustering [8]–[10], regression [11] and reinforcement learning [12]–[14], and it enables low-latency training and inference with less power consumption. However, current HDC algorithms are not without limitations.

We observed that HDC-based ML algorithms still lack the ability to provide uncertainty along with regular prediction. This ability is a must for safety-critical tasks where the importance of model trustworthiness and robustness are particularly emphasized [15]. For example, self-driving cars should make conservative decisions with high confidence. Predictions without uncertainty can lead to catastrophic consequences. This is not only a challenge for HDC but also for DNN because both mainly evolved without Bayesian statistics. Different from regular ML, Bayesian inference parameters have a probability distribution instead of a single value. This key difference enables the analytical expression of the posterior distribution and predictive distribution through Bayes' Theorem. The advantage of Bayesian statistics is that the posterior predictive distribution accounts for the noise of observation, model stochasticity, and prior knowledge about the task. Prior research works try to incorporate this advantage into the DNN learning process and propose several Bayesian Neural Networks (BNN) algorithms [16]–[19]. However, to approximate Bayesian statistics, expensive modifications are necessary for the original network structure or the training and inference processes. Unfortunately, existing BNN algorithms bring more computations and larger energy costs in the learning, compared to already complex DNNs. We believe the lightweight HDC with uncertainty estimation is a more

efficient alternative to existing BNN algorithms. We find that introducing random noise to the HDC encoding process effectively approximates the posterior distribution. This functions as the key to Bayesian inference while keeping the whole framework as lightweight as possible.

In this paper, we propose DiceHD, a hyperdimensional Bayesian framework that enables efficient uncertainty estimation for HDC-based regression algorithm. Our contributions are summarized as follows:

- Through DiceHD, we overcome a major limitation in existing HDC-based ML methods, i.e., the inability to provide uncertainty estimation. Previously, without model confidence, the usability of HDC regression algorithms is limited in safety-critical tasks. DiceHD, as it incorporates Bayesian statistics, opens up new opportunities such as more efficient exploration in optimization.
- We revisit the recent HDC regression algorithm and draw a connection to the more general Vector Function Architecture [20]. In essence, we encode the input to a high dimensional space and we construct the model hypervector that holographically represents the function to approximate.
- We propose a novel HDC encoder that includes perturbations via randomly dropped dimensions to propagate the uncertainty estimation in our DiceHD framework. Our solution, unlike existing BNN methods, introduces few complications to the original regression, simplifies the training, and enables computational reuse in the inference. In Section III-B, we show how this noisy mapping to hyperdimensional space effectively approximates the posterior distribution.
- Our design is evaluated on both CPU and FPGA platforms. We verify DiceHD using several toy regression tasks and multiple real-world datasets. Through visualization, we show that DiceHD is able to generate meaningful uncertainty estimation. Results on real data show that our framework significantly improves the training and inference efficiency, compared to BNN baselines. Compared to BNN on FPGA (or CPU), our design shows a noticeable speedup of up to 2.5× (17×) for training and 8×(748×) for inference. DiceHD provides 84× (3740×) better energy efficiency for training (inference).

II. Related Work

Bayesian Inference: The Bayesian paradigm utilizes probability instead of point estimates to represent the belief of models. This probability is updated as the model observes more training data points. In the past few years, there has been a resurgence of interest in Bayesian statistics due to the need for more informative ML models. There are multiple challenges in making modern ML algorithms Bayesian, especially if they are deep. Most of the existing works take the path of approximating the posterior distribution, which is often intractable. Markov Chain Monte Carlo (MCMC) methods have been used to generate samples from desired posterior distributions [21]–[23]. However, MCMC methods are hardly scalable, memory-hungry, and time-consuming. Therefore, methods in the family of Stochastic Variational Inference (SVI) are considered more suitable for the task. SVI methods learn a tractable variational distribution that is as close as possible to the original posterior. Nevertheless, SVI methods require significant training time and large computational costs due to the more complex networks [16], [17]. Some prior works enable Bayesian inference through ensemble methods [18], [19], [24]–[27]. Many of them approximate the posterior distribution by training multiple models and capture the model uncertainty through model averaging. However, training multiple models inevitably increases the runtime and energy costs. MC-Dropout [18] alleviates this significant overhead by leveraging neural network dropout layers. It enables uncertainty estimation without training multiple models. However, the computationally heavy DNN training process and the multi-layer deep structure significantly increase its energy consumption. This shows the need for a more efficient alternative ML algorithm.

Hyperdimensional Computing: For machine learning in resource-limited environments, HDC is a more efficient alternative computing paradigm compared to DNN. Previous works have successfully applied HDC to ML tasks of various natures. Considering supervised training as an example, prior works propose HDC-based algorithms for real-world regression [11], [28], bio-signal processing [29], [30], genome sequencing [31], [32], drug discovery [33], outlier detection [34], and spam detection [35]. These works have shown that HDC-based ML achieves notable energy savings and speedups in both training and testing, making HDC suitable for machine learning on CPUs even with tight power budgets. In addition, researchers have explored various hardware acceleration strategies to further enhance efficiency [7], [8], [11], [36], [37]. For example, the recent work [11] accelerates the HDC-based regression using FPGA and outperforms several baselines on runtime and energy costs. However, existing HDC algorithms failed to include uncertainty while doing predictions; and our goal is to fix this shortcoming while maintaining the high efficiency of HDC.

III. DiceHD: Enabling Efficient Bayesian HDC

Fig. 1 presents an overview of our DiceHD, and compares it with the non-Bayesian hyperdimensional regression algorithm. In Section III-A, we briefly introduce the regular algorithm that only provides point estimates. In Section III-B, we propose to incorporate uncertainty estimation into HDC-based regression.

A. VFA & Hyperdimensional Regression

In this section, we provide the intuition behind the HDC-based regression algorithm from the angle of Vector Function Architecture (VFA) [20]. VFA is regarded as an extension of the hypervector-based representation in HDC mathematics. One key insight of VFA is that we can define a function space where functions can be represented using high-dimensional vectors. More importantly, HDC operations are compatible and meaningful in this function space. To begin with, we define a function in the form of a weighted kernel sum: $f(x) =$

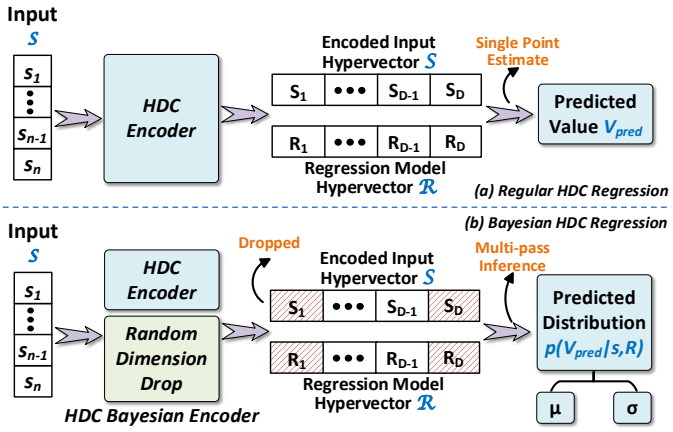

Fig. 1. Comparison between DiceHD and non-Bayesian HDC-based regression algorithm: (a) Basic structure of regular regression that gives point estimates. (b) Overview of our proposed DiceHD, which approximates predictive distribution through a noisy HDC Bayesian encoder.

$\sum_k \alpha_k K(x-x_k)$. When K is a universal, translation-invariant, and positive-definite kernel, we can represent any continuous function through this representation. α_ks are model parameters learned via supervised training. The function representation is obtained in VFA as follows:

$$f(x) = \sum_k \alpha_k K(x - x_k) = \sum_k \alpha_k \phi(x_k)\overline{\phi(x)} = y_k^T \overline{\phi(x)} \quad (1)$$

where $y_k = \sum_k \alpha_k \phi(x_k)$ is the vector representation of $f(\cdot)$ and $\phi(\cdot)$ is the mapping defined by the kernel K.

However, the exact mapping ϕ is often intractable if the kernel implicitly maps inputs to an infinite-dimensional space like the Radial basis function (RBF). There are also unknown kernels that do not have explicit mapping. On the other hand, functions represented using kernels need to accumulate through all training samples for each prediction, which is not scalable. To solve these challenges, prior work in [38] proposes that with a large but finite dimensional mapping Z, the shift-invariant kernel K as the one defined above can be approximated using inner-products:

$$K(x_m - x_n) \approx Z_D(x_m)^T Z_D(x_n) \quad (2)$$

where D is the dimensionality of the mapping. Authors in [38] provide several practical measures to design the mapping Z that corresponds to known kernels. In this paper, we focus on one of them that approximates the RBF kernel and it is defined as follows:

$$Z_D(x) = \sqrt{\frac{2}{D}} cos(\vec{H}x + \vec{B}) \quad (3)$$

$\vec{H}$ is a vector of dimension D with its elements randomly sampled from standard Gaussian distribution $\mathcal{N}(0,1)$ and $\vec{B}$ functions as a bias vector with elements sampled from uniform distribution $\mathcal{U}(0,2\pi)$. Once they are randomly generated, we keep them fixed during the later learning and inference.

In HDC-based regression, by using Equation (1), (2), and (3), we can construct a hyperdimensional representation of function, similar to y_k, with the mapping Z_D: $\vec{R} = \sum_k \alpha_k Z_D(x_k)$. We refer to this mapping Z_D as an HDC encoder that outputs encoded hypervectors $Z_D(x)$. The representation $\vec{R}$ shows that we can approximate the function through a weighted sum of encoded training samples, which makes itself also a hypervector. In addition, we refer to $\vec{R}$ as the model hypervector, and the inference is simply the inner-product between the model and encoded hypervector: $f(x) = \vec{R}^T Z_D(x)$. Notice that the complex conjugate is omitted because $Z_D(x)$ has only real components. To update the model hypervector $\vec{R}$, we feedback the prediction error as the weight for the corresponding encoded input. Assume a true value V_{true} and a predicted value $V_{pred} = \vec{R}^T Z_D(x_k)$, the update step for the model is: $\vec{R} = \vec{R} + (V_{true} - V_{pred}) Z_D(x_k)$. This update process is essentially tuning the parameter α_k for a particular training sample x_k through the hypervector element-wise add/subtract operation, which is highly parallelizable and lightweight.

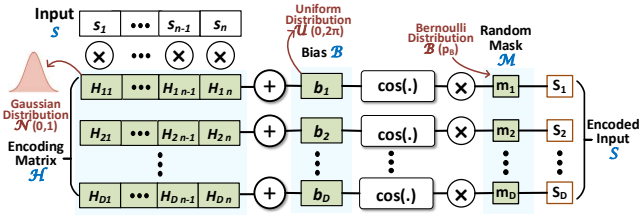

Fig. 2. Overview of DiceHD HDC Bayesian encoder: input $\vec{s}$ is mapped to hypervector $\vec{S}$ through a stochastic encoding process.

B. Hyperdimensional Regression with Uncertainty Estimation

In this section, we propose a noisy HDC encoder and describe how it contributes to an HDC-based regression with Bayesian uncertainty included. The regression mentioned above with VFA provides only point estimates with a deterministic HDC encoder, which is unable to inject the uncertainty during training. We found that it is effective to randomly drop dimensions in the HDC encoder to implement stochastic perturbations.

1) HDC Bayesian Encoder: Fig. 2 shows the structure of the HDC Bayesian encoder with randomly dropped dimensions. Our design follows the VFA mathematics as introduced in Section III-A. In this figure, we assume a multi-feature input vector $\vec{s} = \{s_1, s_2, \ldots, s_n\}$ instead of a single value x. We define an encoder matrix $\mathcal{H} = \{\vec{H_1}, \vec{H_2}, \ldots, \vec{H_n}\}$ with size $n \times D$, of which the elements are randomly generated: $\vec{H_n} \in \mathcal{N}^D(0,1)$. The bias is defined as: $\vec{B} \in \mathcal{U}^D(0,2\pi)$. The main difference in this encoder is that some of the dimensions in the encoded hypervector $\vec{S}$ are set to zeros or dropped. We show this modification in Fig. 2 as a randomly generated mask $\vec{M}$ with its elements $m \in Bernoulli(p_B)$. For performance considerations on hardware platforms, the encoded inputs without dropped dimensions may be saved for computational reuse in the iterative model update as well as for inference. As shown in the figure, the encoded hypervectors

are represented as:

$$\vec{S} = Z_D(\vec{s}) = \sqrt{\frac{2}{D}} cos(\mathcal{H}^T \vec{s} + \vec{B}) \circ \vec{M} \tag{4}$$

2) Approximate Posterior Distribution: We show next how DiceHD enables Bayesian inference through this noisy encoder. Fig. 3 presents the model update process in DiceHD. If we recall the regular HDC regression inference equation from Section III-A, we get the new single-pass inference equation:

$$V_{pred} = \vec{R}^T \vec{S} = \vec{R}^T \left(\sqrt{\frac{2}{D}} cos(\mathcal{H}^T \vec{s} + \vec{B}) \circ \vec{M} \right) \tag{5}$$

To model the uncertainty, it is crucial to learn the posterior distribution conditioned on all training samples. It is defined using the Bayes' theorem as follows:

$$p(\vec{R}|\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}}) = \frac{p(\boldsymbol{\mathcal{V}}|\boldsymbol{\mathcal{S}}, \vec{R}) p(\vec{R})}{\int p(\boldsymbol{\mathcal{V}}|\boldsymbol{\mathcal{S}}, \vec{R}) p(\vec{R}) d\vec{R}} \tag{6}$$

where $\{\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}}\}$ refers to the training dataset. The posterior is focused on $\vec{R}$, since the training process only updates the model hypervector $\vec{R}$ instead of the HDC encoder $\mathcal{H}$ and $\vec{B}$. The prediction with uncertainty is obtained through the predictive distribution:

$$p(V_{pred}|\vec{s}, \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}}) = \int p(V_{pred}|\vec{s}, \vec{R}) p(\vec{R}|\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}}) d\vec{R} \tag{7}$$

The main difficulty in calculating the accurate predictive distribution is that the Equation (6) is intractable, and more specifically, the integration over the model $\vec{R}$. Therefore, the only way to deal with it is by approximating the intractable posterior distribution. One standard method is Variational Inference as mentioned in Section II, where a surrogate variational distribution $q(\vec{R})$ is used in place of the real posterior $p(\vec{R}|\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}})$. To ensure that $q(\vec{R})$ is a good approximation, we can minimize the Kullback–Leibler (KL) divergence between these two distributions: $\min_{q(\vec{R})} \text{KL}\left(q(\vec{R})||p(\vec{R}|\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{V}})\right)$. In most cases, KL divergence is not minimized in its original form because this requires again the exact posterior. Instead, we rearrange the terms and maximize the log evidence lower bound [39]:

$$\mathcal{J} = \int q(\vec{R}) \log\, p(\boldsymbol{\mathcal{V}}|\boldsymbol{\mathcal{S}}, \vec{R}) d\vec{R} - \text{KL}(q(\vec{R})||p(\vec{R})) \tag{8}$$

The first term in this objective can be factorized as a sum in terms of each training sample $\{\vec{s}_k, V_k\}$, assuming K is the total number of training data points: $\sum_{k=1}^{K} \int q(\vec{R}) \log\, p(\hat{V}_k|\vec{s}_k, \vec{R}) d\vec{R}$. Then we apply Monte Carlo integration for each term in this sum to avoid the exact computation of integral, i.e., we sample a realization $\hat{R} \sim q(\vec{R})$. In addition, we assume that $p(\hat{V}_k|\vec{s}_k, \vec{R})$ follows a Gaussian distribution with precision τ and mean V_k. Finally, we get the corresponding loss function:

$$\mathcal{L}_{MC} = -\sum_{k=1}^{K} \log\, p_{\mathcal{N}(V_k, \tau^{-1})}(\hat{V}_k|\vec{s}_k, \vec{R}) + \text{KL}(q(\vec{R})||p(\vec{R})) \tag{9}$$

To support the variational inference as formulated above, we need to construct a proper variational distribution $q(\vec{R})$. In DiceHD, this is achieved through adding a random mask in the HDC Bayesian encoder. Equation (5) can be rearranged as: $V_{pred} = (\vec{R} \circ \vec{M})^T (\sqrt{\frac{2}{D}} cos(\mathcal{H}^T \vec{s} + \vec{B}))$. This allows us to define the variational distribution on the model hypervector $\vec{R}$ as: $q(\vec{R}) = \prod_{d=1}^{D} q(r_m)$, where $q(r_m)$ is assumed to be a Gaussain mixture model with a Bernoulli mask embedded:

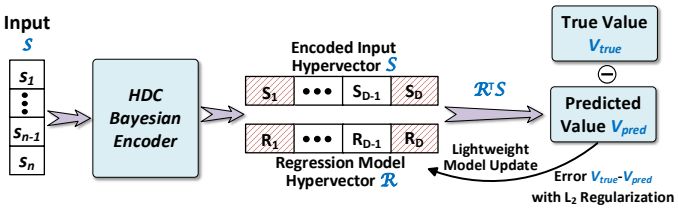

Fig. 3. Model hypervector update process in DiceHD: minimize the KL divergence through lightweight hypervector operations.

$$q(r_m) = p_B \mathcal{N}(r, \sigma^2) + (1 - p_B) \mathcal{N}(0, \sigma^2) \tag{10}$$

In this equation, p_B is the probability for the Bernoulli mask to be 1, i.e., the corresponding dimension is retained. r is the expected mean value of the model hypervector element and σ is a positive standard deviation of the Gaussian model. From Equation (10), we observe that a noisy HDC encoder not only perturbs the encoded results but also can be equivalently added onto the variational distribution $q(\vec{R})$ as element dropping. In this case, the KL divergence term in Equation (9) can be further approximated by following [18]. It provides a way to approximate the KL divergence between a Gaussian mixture and a single Gaussian, especially when the dimensionality is high and therefore applies to our case:

$$\mathcal{L} \propto \sum_{k=1}^{K} \frac{1}{2K} (V_k - \hat{V}_k)^2 + \frac{p_B}{2\tau K} ||\vec{R}||_2^2 \tag{11}$$

This can also be intuitively understood as a likelihood function plus an extra regularization term. They ensure that the regression will converge to the true values, and prevent overfitting and deviating too much from the prior distribution through the KL divergence.

One advantage of regular HDC-based algorithms lies in the efficient training process. Our DiceHD maintains a lightweight hypervector update process compared to the one defined in Section III-A. As presented in Fig. 3, the model hypervector is updated with feedback from the prediction error $V_{true} - V_{pred}$. The loss function in Equation (11) can be minimized through simple element-wise operations with learning rate γ:

$$\vec{R} = \left(1 - \frac{\gamma p_B}{\tau}\right) \vec{R} + \gamma (V_{true} - V_{pred}) \vec{S} \tag{12}$$

3) Bayesian Inference: In inference, we apply model averaging by having testing samples evaluated for several iterations, and then obtain the mean predicted value $\mu_{V_{pred}}$ and standard deviation $\sigma_{V_{pred}}$. During each iteration, the random mask is regenerated for $\vec{R}$. We can also compute the log

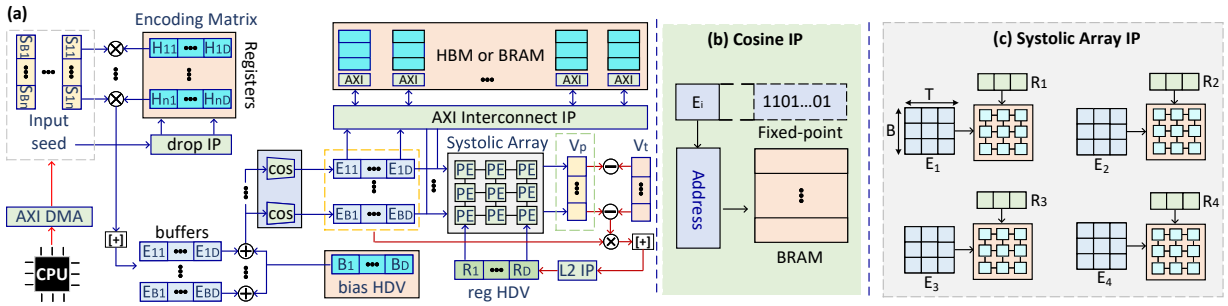

Fig. 4. (a) Online bayesian learning FPGA acceleration architecture. V_p is the batch prediction value vector, and V_t is the true value vector. (b) The cosine kernel function IP implementation. (c) The systolic array IP architecture design.

predictive distribution, i.e., log-likelihood, by a Monte Carlo integration of Equation (7):

$$\begin{aligned}\log\, p(V_{pred}|\vec{s},\vec{s}_k,V_k) &\approx \int p(V_{pred}|\vec{s},\vec{R})q(\vec{R})d\vec{R}\\ &\approx \log\left(\frac{1}{T}\sum_{T} p(V_{pred}|\vec{s},\vec{R})\right)\\ &= \text{logsumexp}\left[-\frac{1}{2}\tau(V_{true}-V_{pred})^2\right]-C\end{aligned} \tag{13}$$

where the constant $C = \log T - \frac{1}{2}\log 2\pi - \frac{1}{2}\log\tau^{-1}$. T is the number of inference passes and we use the variational distribution $q(\vec{R})$ to replace the intractable $p(\vec{R})$ before Monte Carlo integration.

To summarize, the noisy DiceHD HDC encoder contributes to a practical variational distribution that can be efficiently optimized, leading to a posterior predictive distribution for regression with uncertainty estimation. Compare to the regular HDC-based regression, the whole process especially the training still remains lightweight as defined in Section III-A.

C. FPGA acceleration for DiceHD

Fig. 4(a) shows the acceleration architecture of DiceHD on FPGA. The host CPU will load input data and random seed into the kernel FPGA via Xilinx AXI DMA IP. The random seed is mainly used for the drop IP to generate random numbers. For each training or inference iteration, the drop IP will decide each dimension of the hypervector to be 0 (drop) or the original value (not drop). As is shown in Figure 4(b), for hardware cosine function implementation, we adopt the triangle codebook method to efficiently implement cosine encoder IP for kernel encoding [7], [40]. Compared to Taylor expansion or Xilinx CORDIC IP, using on-chip BRAM to store pre-compute cos value and treating each fixed-point number as memory access address are much faster and more efficient on FPGA. Specifically, since each hypervector element's precision is fixed-point, we can use it as an input address to access on-chip storage (such as BRAM) where we pre-store all possible cosine values. Since for cosine function, its valid input is only in the range $[-\pi,\pi]$, we will first quantize the input element into this range and then access BRAM to get the corresponding cosine value.

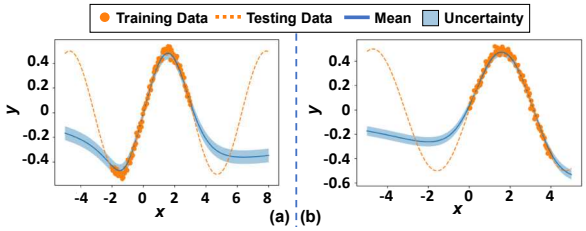

Fig. 5. Visualizations for the DiceHD regression with uncertainty estimation

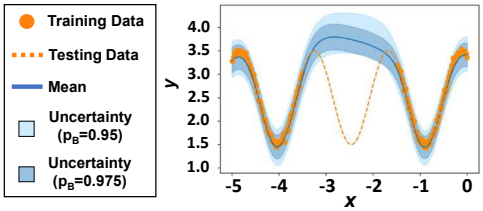

Fig. 6. DiceHD uncertainty estimation with different Bernoulli probabilities

After finishing the encoding process, the encoded hypervector will simultaneously forward to systolic array IP and AXI Interconnect IP, which means we do not need to conduct the rest training of inferring epochs encoding operation anymore. We use a Systolic Array IP to perform regression operations and generate the predicted value. As is shown in Figure 4, to simplify the FPGA synthesize and placement difficulty, instead of using a single large dimension systolic array, we cut the hypervector regression operation into small chunks [41] and assign each small chunk computation to a small systolic array IP. After all small systolic array finish computation, we perform a concatenation operation to generate the predicted value V_p. Compared to traditional DNN training [42], an HDC-based FPGA accelerator is much easier to support on-chip online learning, which means our platform can support both training and inference. The first advantage is that HDC-based model training does not need too much power-hungry DSP. The second advantage is that we only need to update a single hypervector instead of multiple layers.

IV. Experimental Results

A. Experimental setup

To evaluate our Bayesian HDC framework, we implement DiceHD on both CPU and FPGA platforms. The CPU

TABLE I
REGRESSION QUALITY COMPARISON ON REAL-WORLD TASKS ACROSS DIFFERENT BAYESIAN INFERENCE ALGORITHMS

Dataset	R^2					Log-likelihood				
	Variational Inference	PBP	Stein VI	MC Dropout	Our Design	Variational Inference	PBP	Stein VI	MC Dropout	Our Design
Boston Housing	0.86	0.89	0.88	0.89	0.89	-2.90	-2.57	-2.68	-2.40	-2.52
Propulsion Plant	0.87	0.87	0.84	0.99	0.98	3.73	3.73	3.68	4.38	4.12
Power Plant	0.95	0.95	0.94	0.95	0.94	-2.89	-2.84	-3.17	-2.80	-2.84
Wine Quality Red	0.39	0.40	0.41	0.42	0.39	-0.98	-0.97	-0.97	-0.92	-0.97
Computer Activity	-	-	0.97	0.97	0.96	-	-	-3.15	-2.49	-2.68
Stock	-	-	0.98	0.98	0.98	-	-	-2.14	-1.26	-1.33
Geographical Analysis	-	-	0.68	0.68	0.68	-	-	0.76	0.77	0.76

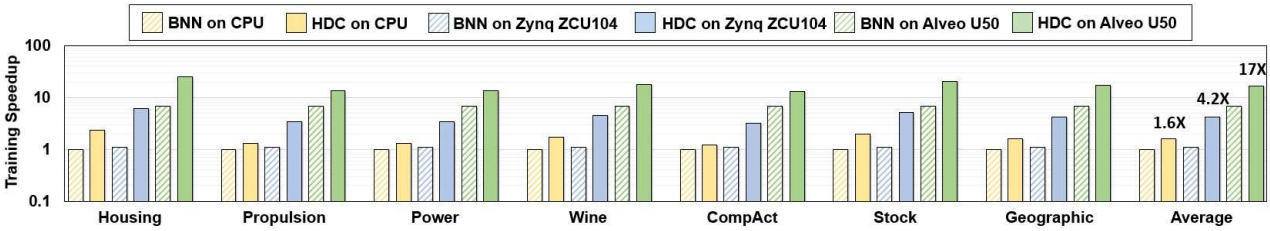

Fig. 7. Training runtime speedup comparison on CPU and two FPGAs. The speedup is normalized to the BNN (MC-Dropout) runtime on the CPU.

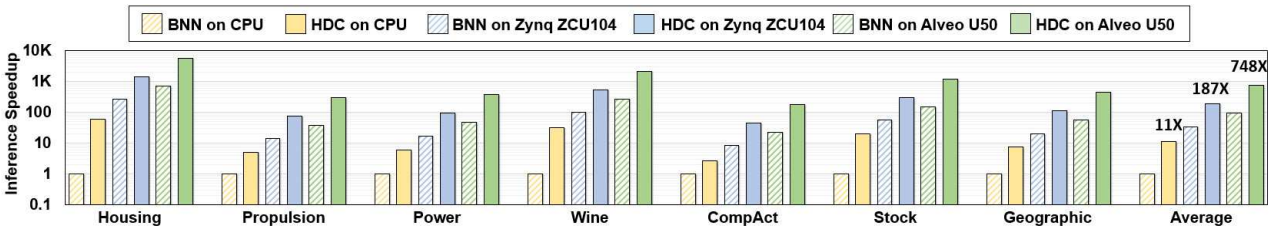

Fig. 8. Inference runtime speedup comparison. The speedup is normalized to the BNN (MC-Dropout) runtime on the CPU.

is Intel Core i7-10700; and for FPGA, we choose Xilinx Zynq ZCU104 and Alveo U50. Our CPU implementation uses Python with Scikit-learn. We also used the Xilinx Vitis framework to conduct the communication between CPU and FPGA via PCIe [43]. We select multiple regression workloads that are comprised of toy datasets as well as real-world regression tasks. We provide visualizations for the DiceHD regression and uncertainty estimation on 1-D toy datasets. As for multi-dimensional practical regression tasks, we select several publicly available datasets from UCI Machine Learning Repository [44] and OpenML [45]. Our baselines include several widely-deployed BNN algorithms such as direct variational inference (VI), probabilistic backpropagation (PBP), and dropout-based approximation [18], [25], [27], [46]–[48]. The neural network used in BNN baselines has two hidden layers and each has 50 neurons. For DiceHD, we use hypervectors with dimensionality $D = 2000$. Similar to other dropout-based BNNs, we use grid search to find the suitable setting of Bernoulli probability p_B (in the range of 0.95 to 0.995) and estimate the precision τ. We set the training iterations to 200 and the number of inference iterations to $T = 300$. For each practical regression task, we use 20 random splits and average the prediction, training runtime, and testing runtime.

B. Toy Workloads Visualization

In Fig. 5. we show the visualization of DiceHD Bayesian regression on a noisy and partially observable sine function. For (a) and (b), the parts of available training data points are different. The mean value prediction is shown in a solid curve and the uncertainty ($\pm 3\sigma$) is shown as the shady area. We notice that DiceHD predicts the testing data with lower accuracy where training data is not presented, however, it gives a significantly higher uncertainty in those areas. This shows that the DiceHD model is not confident about the prediction.

In Fig. 6, we visualize the results of DiceHD with a slightly more complex example where training data points are separated into two clusters. We show the effect of different Bernoulli probabilities on uncertainty estimation: the range of uncertainty increases when we tune down p_B and vice versa. As shown in this figure, the model is highly unsure about the prediction in $-3 < x < -2$ due to the lack of training data, and the prediction converges with the presence of training data points. Notice that the uncertainty is not zero even with training data, and this is because the data contains noise during training. It is expected during Bayesian inference since the model uncertainty should also account for the observation noise.

TABLE II

DESIGN ACCELERATION ON XILINX ALVEO U50 WHEN BATCH SIZE IS 8, FREQUENCU IS 200 MHZ AND THE FPGA POWER CONSUMPTION IS 22W. AT ZYNQ ULTRASCALE+ ZCU104, THE BATCH SIZE IS 1, FREQUENCY IS 200MHZ, AND THE FPGA POWER CONSUMPTION IS 7.7W.

	Xilinx Alveo U50					ZCU 104				
Name	LUT	FF	DSP	BRAM	URAM	LUT	FF	DSP	BRAM	URAM
Total	797K	1458K	3	806	0	167.6K	227.5K	3	132	0
Available	872K	1743K	5952	1344	640	230K	460.8K	1728	312	96
Utilization	91.3%	83.6%	∼0%	59.9%	0%	72.8%	49.3%	∼0%	42.3%	0%

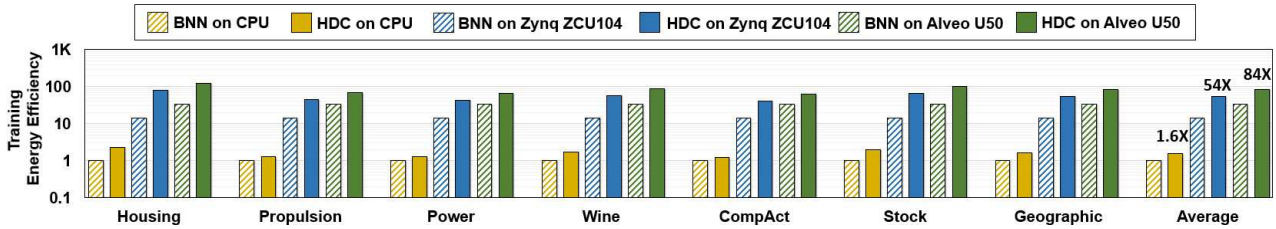

Fig. 9. Training energy efficiency comparison across hardware platforms. The results are normalized to BNN (MC-Dropout) energy consumption on CPU.

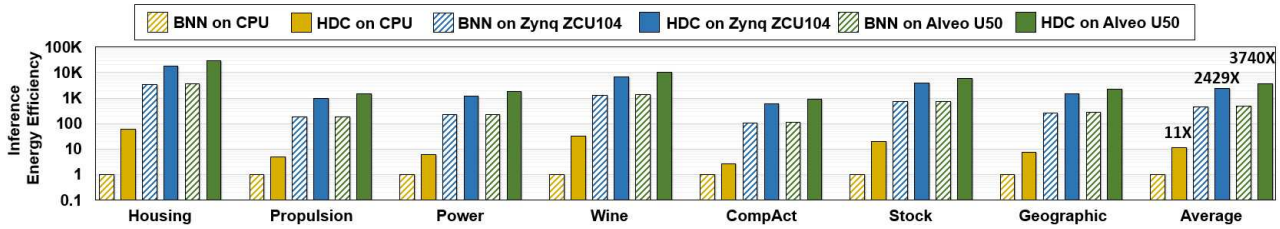

Fig. 10. Cross-platform inference energy efficiency comparison. The results are normalized to BNN (MC-Dropout) energy consumption on CPU.

C. Regression Accuracy & Uncertainty Estimation

Table I compares the Bayesian inference quality of DiceHD with baseline BNN algorithms. We implement MC-Dropout using the open-sourced code provided by the original authors and Stein-VI using the Pyro framework [49]. We report the results for 7 different datasets that focus on different regression tasks. The results of some datasets are omitted for variational inference and PBP since they are not reported in original papers. We select two metrics, coefficient of determination (R^2) and log-likelihood, to evaluate the regression quality and the uncertainty estimation respectively. The log-likelihood, as defined in Section III-B3, represents the posterior probability density function for the prediction. For both metrics, the higher value means better quality. In terms of the regression quality across all tested datasets, DiceHD achieves comparable R^2 value and log-likelihood to the best algorithm among baselines.

We also observe that, on average, MC-Dropout achieves the highest R^2 and log-likelihood when compared to other BNNs. In addition, it also has a smaller computational overhead than variational inference and a relatively more lightweight structure than the probabilistic network in PBP, making itself a strong baseline. In the following sections, we will focus on comparing MC-Dropout with our DiceHD in terms of Bayesian inference efficiency on various hardware platforms.

D. Runtime & Energy Efficiency

This parameter tuning overhead is not included in the training runtime for both DiceHD and MC-dropout, however, our method has a much smaller overhead thanks to the fast learning and inference.

Table II presents the resource utilization of DiceHD acceleration on Xilinx Alveo U50 and Zynq Ultrascale+ ZCU104. We suppose the batch size and frequency for both training and inferring are 8 and 200 MHz on Alveo U50. For ZCU104, due to limited resources, we set the batch size to 1 and keep the FPGA frequency still as 200 MHz. As the baseline to our FPGA acceleration, we also implement MC-Dropout on these two FPGAs mentioned above. We use Xilinx deep-learning processor unit (DPU) [50] for efficient implementation of the BNN inference. We perform the training acceleration on FPGA boards based on previous DNN FPGA training frameworks [42], [51].

Fig. 7 and Fig. 8 compare the training and inference runtime of DiceHD with BNN (MC-Dropout) when running both algorithms on CPU (Intel Core i7-10700) and FPGAs (Zynq ZCU104 and Alveo U50). The results are normalized as speedup when compared to MC-Dropout runtime on the CPU. We compare the training runtime for each regression task in Fig. 7 and compute the geometric average over all the tasks. DiceHD is, on average, 60% faster than the baseline on CPU and 17× faster after our FPGA acceleration on Alveo U50. In contrast, MC-Dropout on Alveo U50 shows a smaller

speedup of 6.7×, compared to its CPU implementation. As for results collected on the Zynq FPGA, the speedup values are generally lower (about 4.2× for HDC and 1.1× for MC-Dropout) mainly because of its low power consumption. Fig. 8 compares the inference runtime. On average, DiceHD achieves an 11× speedup over the MC-dropout baseline on CPU; the speedup is 187× and 748× if with FPGA acceleration on Zynq ZCU104 and Alveo U50.

We also compare the energy efficiency between DiceHD and the baseline BNN (MC-Dropout) in terms of the training cost (Fig. 9 and inference cost (Fig. 10. On CPU, thanks to the smaller runtime costs, our HDC-based method provides significantly better energy efficiency than the baseline, i.e., 1.6× (11×) improvement for training (inference). With FPGA acceleration, DiceHD achieves up to 84× and 3740× better energy efficiency on Alveo U50 for training and inference, respectively. Compared to the CPU, the FPGA acceleration shows advantages in both power consumption and runtime. The power consumption of DiceHD on Alveo U50 is only 22W and 7.7w for Zynq ZCU104, which are much smaller than the CPU (around 100W).

V. Conclusion

In this paper, we propose DiceHD, a hyperdimensional Bayesian framework that enables efficient uncertainty estimation for HDC-based regression algorithm. We propose a noisy HDC encoder that leads to an approximation of the true posterior distribution. DiceHD provides meaningful uncertainty estimations while also achieving significant speedup in both training and inference compared to the BNN baseline.

Acknowledgements

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780 and #2312517, and #2319198, Semiconductor Research Corporation (SRC), Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research, grants #FA9550-22-1-0253, and generous gifts from Xilinx and Cisco.

References

[1] A. Hernandez-Cane, N. Matsumoto *et al.*, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 DATE*. IEEE, 2021, pp. 56–61.

[2] H. Fang, B. Taylor *et al.*, "Neuromorphic algorithm-hardware codesign for temporal pattern learning," in *DAC*. IEEE, 2021, pp. 361–366.

[3] P. Poduval, Y. Ni *et al.*, "Adaptive neural recovery for highly robust brain-like representation," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 367–372.

[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] P. Poduval, H. Alimohamadi *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, vol. 16, p. 757125, 2022.

[6] C. J. Stoodley, "The cerebellum and cognition: evidence from functional imaging studies," *The Cerebellum*, vol. 11, no. 2, pp. 352–365, 2012.

[7] M. Imani, Z. Zou *et al.*, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE HPCA*. IEEE, 2021, pp. 221–234.

[8] J. Morris, K. Ergun, B. Khaleghi *et al.*, "Hydrea: Towards more robust and efficient machine learning systems with hyperdimensional computing," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 723–728.

[9] Y. Ni, Y. Kim, T. Rosing, and M. Imani, "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 292–297.

[10] M. Imani, A. Zakeri *et al.*, "Neural computation for robust and holographic face detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 31–36.

[11] A. Hernández-Cano, C. Zhuo *et al.*, "Reghd: Robust and efficient regression in hyper-dimensional learning system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 7–12.

[12] Y. Ni, M. Issa *et al.*, "Hdpg: hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1141–1146.

[13] Y. Ni, D. Abraham, M. Issa, Y. Kim, P. Mercati, and M. Imani, "Efficient off-policy reinforcement learning via brain-inspired computing," *arXiv preprint arXiv:2205.06978*, 2022.

[14] M. Issa, S. Shahhosseini *et al.*, "Hyperdimensional hybrid learning on end-edge-cloud networks," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 652–655.

[15] H. Kumar *et al.*, "Towards improving the trustworthiness of hardware based malware detector using online uncertainty estimation," in *DAC*. IEEE, 2021, pp. 961–966.

[16] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *International conference on machine learning*. PMLR, 2015, pp. 1613–1622.

[17] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *Journal of Machine Learning Research*, 2013.

[18] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[19] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, 2017.

[20] E. P. Frady, D. Kleyko, C. J. Kymn, B. A. Olshausen, and F. T. Sommer, "Computing on functions using randomized vector representations," *arXiv preprint arXiv:2109.03429*, 2021.

[21] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid monte carlo," *Physics letters B*, vol. 195, no. 2, pp. 216–222, 1987.

[22] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[23] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[24] T. Fushiki, "Bootstrap prediction and bayesian prediction under misspecified models," *Bernoulli*, vol. 11, no. 4, pp. 747–758, 2005.

[25] Y. Gal, J. Hron, and A. Kendall, "Concrete dropout," *Advances in neural information processing systems*, vol. 30, 2017.

[26] I. Osband, C. Blundell *et al.*, "Deep exploration via bootstrapped dqn," *Advances in neural information processing systems*, vol. 29, 2016.

[27] X. Fan, S. Zhang, K. Tanwisuth *et al.*, "Contextual dropout: An efficient sample-dependent dropout module," *arXiv preprint arXiv:2103.04181*, 2021.
[28] M. Hersche *et al.*, "Integrating event-based dynamic vision sensors with sparse hyperdimensional computing: a low-power accelerator with online learning capability," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 169–174.
[29] A. Burrello *et al.*, "Hyperdimensional computing with local binary patterns: One-shot learning of seizure onset and identification of ictogenic brain regions using short-time ieeg recordings," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 2, pp. 601–613, 2019.
[30] Y. Ni, N. Lesica, F.-G. Zeng, and M. Imani, "Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.
[31] Z. Zou, H. Chen *et al.*, "Biohd: an efficient genome sequence search platform using hyperdimensional memorization," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 656–669.
[32] H. E. Barkam, S. Yun, P. R. Genssler *et al.*, "Hdgim: Hyperdimensional genome sequence matching on unreliable highly scaled fefet," in *DATE*. IEEE, 2023, pp. 1–6.
[33] D. Ma, R. Thapa, and X. Jiao, "Molehd: Drug discovery using brain-inspired hyperdimensional computing," *arXiv preprint arXiv:2106.02894*, 2021.
[34] R. Wang, X. Jiao, and X. S. Hu, "Odhd: one-class brain-inspired hyperdimensional computing for outlier detection," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 43–48.
[35] R. Thapa, B. Lamichhane, D. Ma, and X. Jiao, "Spamhd: Memory-efficient text spam detection using brain-inspired hyperdimensional computing," in *ISVLSI*. IEEE, 2021, pp. 84–89.
[36] B. Khaleghi *et al.*, "tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications," in *DATE*. IEEE, 2021, pp. 408–413.
[37] A. Kazemi, F. Müller, M. M. Sharifi, H. Errahmouni *et al.*, "Achieving software-equivalent accuracy for hyperdimensional computing with ferroelectric-based in-memory computing," *Scientific reports*, vol. 12, no. 1, p. 19201, 2022.
[38] A. Rahimi and B. Recht, "Random features for large-scale kernel machines," *NIPS*, vol. 20, 2007.
[39] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.
[40] H. Chen, M. Issa *et al.*, "Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning," in *Proceedings of the 41st IEEE/ACM ICCAD*, 2022, pp. 1–9.
[41] H. Chen, M. H. Najafi *et al.*, "Full stack parallel online hyperdimensional regression on fpga," in *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 2022, pp. 517–524.
[42] T. Wang, T. Geng, A. Li, X. Jin, and M. Herbordt, "Fpdeep: Scalable acceleration of cnn training on deeply-pipelined fpga clusters," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1143–1158, 2020.
[43] V. Kathail, "Xilinx vitis unified software platform," in *ACM/SIGDA FPGA 2020*, 2020, pp. 173–174.
[44] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[45] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 2, pp. 49–60, 2014.
[46] A. Graves, "Practical variational inference for neural networks," *Advances in neural information processing systems*, vol. 24, 2011.
[47] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of bayesian neural networks," in *International conference on machine learning*. PMLR, 2015, pp. 1861–1869.
[48] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose bayesian inference algorithm," *Advances in neural information processing systems*, vol. 29, 2016.
[49] E. Bingham, J. P. Chen, M. Jankowiak *et al.*, "Pyro: Deep Universal Probabilistic Programming," *JMLR*, 2018.
[50] X. Zhang *et al.*, "Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *ICCAD*, 2020, pp. 1–9.
[51] Y. Tang, X. Zhang, P. Zhou, and J. Hu, "Ef-train: Enable efficient on-device cnn training on fpga through data reshaping for online adaptation or personalization," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 5, pp. 1–36, 2022.