ULEEN: A Novel Architecture for Ultra-low-energy Edge Neural Networks

ZACHARY SUSSKIND and AMAN ARORA, The University of Texas at Austin, USA
IGOR D. S. MIRANDA, Federal University of Recôncavo da Bahia, Brazil
ALAN T. L. BACELLAR, LUIS A. Q. VILLON, and RAFAEL F. KATOPODIS, Federal University of Rio de Janeiro, Brazil
LEANDRO S. DE ARAÚJO, Universidade Federal Fluminense, Brazil
DIEGO L. C. DUTRA and PRISCILA M. V. LIMA, Federal University of Rio de Janeiro, Brazil
FELIPE M. G. FRANÇA, Instituto de Telecomunicações, Portugal and Federal University of Rio de Janeiro, Brazil
MAURICIO BRETERNITZ JR., ISCTE–Instituto Universitario de Lisboa, Portugal
LIZY K. JOHN, The University of Texas at Austin, USA

"Extreme edge"[1] devices, such as smart sensors, are a uniquely challenging environment for the deployment of machine learning. The tiny energy budgets of these devices lie beyond what is feasible for conventional deep neural networks, particularly in high-throughput scenarios, requiring us to rethink how we approach edge inference. In this work, we propose ULEEN, a model and FPGA-based accelerator architecture based on weightless neural networks (WNNs). WNNs eliminate energy-intensive arithmetic operations, instead using

[1]Extension of Conference Paper: The model discussed in this work was originally proposed in "Pruning Weightless Neural Networks" [58]. We make the following major extensions: (1) We demonstrate an FPGA implementation of the proposed design rather than only a software model; (2) We add comparisons against binary neural networks in software and hardware (FINN); (3) We perform a sensitivity analysis to examine the impacts of our multi-pass learning and ensemble techniques.

This research was supported in part by Semiconductor Research Corporation (SRC) Tasks No. 3015.001 and No. 3148.001, National Science Foundation (NSF) Grant No. 2326894, CAPES and CNPq, Brazil, FCT/COMPETE/FEDER, FCT/CMU IT Project FLOYD: POCI-01-0247-FEDER-045912 and FCT/MCTES, and ISTAR Projects No. UIDB/04466/2020, No. UIDP/04466/2020, and No. DSAIPA/AI/0122/2020 Aim Health Portugal, through national funds and when applicable co-funded EU funds under Project UIDB/50008/2020. Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies.
Authors' addresses: Z. Susskind and L. K. John, Department of Electrical and Computer Engineering, The University of Texas at Austin, 2501 Speedway, Austin, Texas 78712, USA; e-mails: zsusskind@utexas.edu, ljohn@ece.utexas.edu; A. Arora, Arizona State University, 660 South Mill Avenue 203-09, Tempe, Arizona, 85281, USA; e-mail: aman.kbm@asu.edu; I. D. S. Miranda, Centro de Ciências Exatas e Tecnológicas, Universidade Federal do Recôncavo da Bahia, Rua Rui Barbosa, 710 - Centro - Cruz das Almas/BA - 44.380-000, Brazil; e-mail: igordantas@ufrb.edu.br; A. T. L. Bacellar, L. A. Q. Villon, R. F. Katopodis, D. L. C. Dutra, and P. M. V. Lima, Cidade Universitária da Universidade Federal do Rio de Janeiro, Avenue Horácio Macedo, 2030, Rio de Janeiro/RJ - 21941-598, Brazil; e-mails: alanbacellar@poli.ufrj.br, {lvillon, rkatopodis, ddutra, priscilamvl}@cos.ufrj.br; L. S. de Araújo, Campus da Praia Vermelha, Universidade Federal Fluminense, Rua Passo da Pátria, 156, Niterói/RJ - 24210-253, Brazil; e-mail: leandro@ic.uff.br; F. M. G. França, Faculdade de Ciências, Dep. de Ciências e Computadores, Instituto de Telecomunicações, Rua Campo Alegre, 1021/1055, 4169-007, Porto, Portugal; e-mail: felipe@ieee.org; M. Breternitz Jr., ISCTE - Instituto Universitário de Lisboa, Av. das Forças Armadas, 1649-026 Lisboa, Portugal; e-mail: Mauricio.Breternitz.Jr@iscte-iul.pt.

1544-3566/2023/12-ART61
https://doi.org/10.1145/3629522

table lookups to perform computation, which makes them theoretically well-suited for edge inference. However, WNNs have historically suffered from poor accuracy and excessive memory usage. ULEEN incorporates algorithmic improvements and a novel training strategy inspired by binary neural networks (BNNs) to make significant strides in addressing these issues. We compare ULEEN against BNNs in software and hardware using the four MLPerf Tiny datasets and MNIST. Our FPGA implementations of ULEEN accomplish classification at 4.0–14.3 million inferences per second, improving area-normalized throughput by an average of 3.6× and steady-state energy efficiency by an average of 7.1× compared to the FPGA-based Xilinx FINN BNN inference platform. While ULEEN is not a universally applicable machine learning model, we demonstrate that it can be an excellent choice for certain applications in energy- and latency-critical edge environments.

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**; • **Hardware** → **Hardware accelerators**; • **Computer systems organization** → *Special purpose systems;*

Additional Key Words and Phrases: Weightless neural networks, WiSARD, neural networks, inference, edge computing, MLPerf tiny, high throughput computing

ACM Reference format:
Zachary Susskind, Aman Arora, Igor D. S. Miranda, Alan T. L. Bacellar, Luis A. Q. Villon, Rafael F. Katopodis, Leandro S. de Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França, Mauricio Breternitz Jr., and Lizy K. John. 2023. ULEEN: A Novel Architecture for Ultra-low-energy Edge Neural Networks. *ACM Trans. Arch. Code Optim.* 20, 4, Article 61 (December 2023), 24 pages.
https://doi.org/10.1145/3629522

1 INTRODUCTION

Recent advancements in **deep neural networks (DNNs)** have driven rapid progress in a wide range of problem domains. However, this has come at the cost of an exponential increase in the size and complexity of models. Consequently, there has been a complementary effort to increase the efficiency of machine learning. Optimizations such as pruning, compression, low-precision quantization [25], and sub-network selection (e.g., Once-for-all [10]) reduce the memory requirements and computational demands of large models, enabling deployment on resource-constrained "edge" devices. However, deploying machine learning on the very smallest of devices—the so-called "extreme edge"—remains a major challenge. This domain includes devices that perform computation adjacent to physical sensors [18], where energy might be provided by the sensor itself (such as energy-harvesting image sensors [37]), or by a small battery that is expected to last for years without replacement [51]. Aggressive approaches are needed to meet the stringent energy budgets of these devices.

Binary neural networks (BNNs) [16, 17, 27, 52, 59] are an approach to deploying machine learning on the extreme edge that have received considerable prior interest. BNNs take quantization to its limit by reducing network weights and activations to single-bit values. By replacing energy-intensive multiplication with XNOR operations, BNNs achieve energy efficiency orders of magnitude better than conventional DNNs. However, like DNNs, BNNs must propagate activations through many layers of computation. This may still result in an unacceptably high latency for real-time applications.

Weightless Neural Networks (WNNs) are an approach to machine learning fundamentally distinct from DNNs and BNNs. WNNs perform computation primarily using lookup tables, as opposed to the arithmetic or fixed logical functions that dominate other approaches [3]. Individual weightless neurons, referred to as *RAM nodes*, concatenate binary inputs to form an address into an internal lookup table and output the (binary) entry accessed. Unlike DNN (multiply-accumulate) or BNN (XNOR-popcount) neurons, RAM nodes are capable of learning nonlinear functions of their inputs. This enables single-layer WNNs to learn complex behaviors that would require multiple layers with non-linear activations in a DNN.

The concept of WNNs is not new - in fact, the earliest implementations date back to the 1950s [43]. Their simple structure was well-suited to early IC manufacturing techniques, and they achieved modest commercial success in the 1980s. However, these early WNNs were surpassed by DNNs in both accuracy and memory efficiency by the 1990s. Recent work [14, 46, 55, 57] has narrowed this gap, demonstrating that WNNs are comparable or superior to quantized DNNs on certain datasets, and can be implemented efficiently using modern FPGA and ASIC synthesis techniques.

In this article, we demonstrate an approach to further improve the efficiency of WNNs for edge applications, building on our recent model, BTHOWeN [57]. We present a weightless neural architecture we call **ULEEN (Ultra-low-energy Edge Networks**) for inference under extreme energy constraints. ULEEN incorporates submodel ensembles and replaces the single-pass training rule used in prior WNNs with a novel multi-pass gradient-based learning rule. We present an FPGA-based inference accelerator architecture for this model and compare it against a state-of-the-art efficient BNN inference platform (Xilinx FINN [59]), as well as the earlier BTHOWeN and Bloom WiSARD [55] models.

Our specific contributions in this article are as follows:

(1) ULEEN, a novel weightless neural model that enhances prior WNNs by introducing a multi-pass feedback-based learning rule, additive submodel ensembles, and RAM node pruning: We compare ULEEN to fully connected binary neural networks on the four MLPerf Tiny datasets and MNIST. ULEEN reduces parameter size by a geometric average of 1.9× compared to comparably accurate BNNs and increases accuracy by an average of 3.1% over similarly sized BNNs.
(2) A fast, energy-efficient FPGA-based inference accelerator architecture for ULEEN: We compare our accelerator against the Xilinx FINN [59] platform for optimized BNN deployment on a Xilinx Zynq Z-7045 FPGA. Our FPGA implementation demonstrates superior performance versus similarly accurate BNNs, including a 6.1× decrease in **area-delay product (ADP)** with an 8.9× increase in energy efficiency on the KWS dataset, and a 5.0× decrease in ADP with a 3.8× increase in efficiency on ToyADMOS/car.
(3) A comparison of ULEEN against two prior memory-efficient WNNs, Bloom WiSARD [55] and BTHOWeN [57]: We show that our optimizations in ULEEN can reduce inference error by up to 5.3× and model parameter size by up to 5.5× compared to Bloom WiSARD across the KWS, ToyADMOS, and MNIST datasets. We demonstrate that our multi-pass learning and ensemble techniques provide significant accuracy improvements over BTHOWeN, reducing inference error by an average of 2.3×. Pruning enables a 27% reduction in model size with minimal accuracy impact, giving a final result comparable in size to BTHOWeN with superior accuracy.

Source code for this work is available at https://github.com/ZSusskind/ULEEN. The remainder of this article is organized as follows: In Section 2, we discuss prior WNNs, including BTHOWeN, our previous architecture for efficient inference. In Section 3, we discuss the ULEEN model and inference accelerator architecture. In Section 4, we provide additional information on datasets and implementation of ULEEN. In Section 5, we compare ULEEN against FINN, a state-of-the-art FPGA inference platform based on BNNs. We also provide results comparing the performance of ULEEN against prior WNNs. In Section 6, we give some additional context into the prior work in this domain. Last, in Section 7, we discuss potential future work and conclude.

2 BACKGROUND

2.1 Weightless Neural Networks

WNNs are neural models that perform computation using lookup tables. The basic computational unit in WNNs, the *RAM node*, is an n-input **lookup table (LUT)** with 2^n learnable single-bit entries.

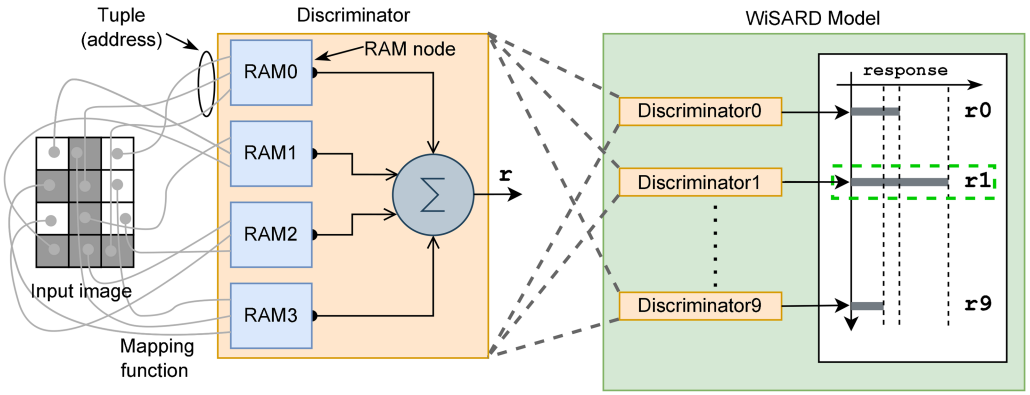

Fig. 1. WiSARD, a simple WNN model.

Each permutation of the contents of this LUT represents a unique Boolean function, meaning there are 2^{2^n} possible functions for a single RAM node. By contrast, a single XNOR-and-popcount neuron in a BNN is restricted to a set of just $n2^n+2$ learnable functions.[2] This nonlinearity allows individual RAM nodes to capture complex behaviors: for instance, it is impossible for a single neuron in a DNN or BNN to learn the XOR function, but this is trivial for a RAM node.

The downside of this expressiveness is that the size of a RAM node grows exponentially with its number of inputs, quickly becoming intractable. Therefore, WNNs are typically sparsely connected, containing many RAM nodes that are each only sensitive to a small subset of the available inputs.

The process of training a WNN involves determining which Boolean functions its component RAM nodes should represent. Many approaches have been explored for this, including both supervised [54] and unsupervised [61] techniques. Most commonly, WNNs are trained using a supervised one-pass learning rule. In this approach, all RAM nodes are initially filled with zeros. Binarized inputs are then sequentially presented to the network and formed into addresses to the RAM nodes. When a RAM node sees an address during training, it sets the corresponding location in its LUT to 1. Presenting the same pattern to a node again has no further effect; therefore, there is no need for multiple epochs of training. Thanks to this single-pass training approach, WNNs can be trained up to four orders of magnitude more rapidly than DNNs and support vector machines [11].

Since the RAM nodes in a WNN can only memorize patterns they were exposed to during training, one might expect them to generalize poorly to new data. However, although an inference sample may not be identical to any training sample, many of the "subpattern" addresses seen by individual RAM nodes will have also been present in training data. Therefore, as long as an inference sample is not too different from any training sample, the network can still effectively generalize.

2.2 WiSARD

WiSARD (Wilkie, Stonham, and Aleksander's Recognition Device) [4] is a WNN model developed in the early 1980s, notable as the first WNN architecture to see commercial usage via the WISARD/CRS1000 image processing neurocomputer. WiSARD also serves as a baseline model on which many subsequent WNNs have been built. As depicted in Figure 1, WiSARD is composed of

[2]To see this, we observe that there are 2^n possible Boolean weight vectors for n inputs, n non-trivial choices for the threshold, and 2 trivial cases: threshold ≤ 0 (constant 1), and threshold $\geq n+1$ (constant 0).

submodels, known as *discriminators*, which are each specialized for one of M output classes. Each discriminator is in turn composed of a set of RAM nodes. For an I-input model with n-input RAM nodes, there are $N \equiv I/n$ RAM nodes per discriminator, and therefore a total of $MN2^n$ learned parameters. Input features are assigned to RAM nodes using a pseudo-random mapping, which is shared between discriminators.

During training, input samples are sequentially presented to the discriminator corresponding to their output class. The single-pass learning rule described previously is used to update the RAM nodes in the indicated discriminator only. During inference, samples are instead presented to all discriminators. The outputs of the RAM nodes in each discriminator are then summed to produce *response scores*, and the class corresponding to the discriminator with the strongest response is taken to be the prediction of the network. In the example shown in Figure 1, the response from Discriminator 1 is the strongest, since the input image contains the digit "1."

In the unlikely event that an inference sample is exactly identical to a training sample, all RAM nodes in the discriminator corresponding to the correct output will have a 1 in their accessed location. In the more typical case, where the inference sample is not identical to any entry in the training data, it is still likely that some of the RAM node input patterns will be shared. Therefore, while many RAM nodes will produce a 0, some should still output a 1. As long as the number of RAM nodes that output a 1 in the correct discriminator is still larger than the number in any other discriminator, the network will output a correct prediction. This mechanism is what allows WiSARD to generalize to new data.

The value of n is a crucial hyperparameter that must typically be chosen by trial and error. Small values of n restrict the complexity of the patterns the model can learn, which improves the ability of the model to generalize but may harm overall accuracy. Large values of n produce more specialized behavior, but may also result in overfitting to training data as the model memorizes data artifacts rather than useful behaviors (as an extreme example, a single LUT with $n = I$ can perfectly memorize any input pattern but clearly can not generalize) [4]. The exponential relation between n and the total model parameter size also restricts the feasible range of choices.

WiSARD also has an uncommon property that makes it of theoretical interest: the Vapnik–Chervonenkis[3] dimension of a WiSARD model is large and can be computed exactly [12], while for most other complex ML models it must be approximated using statistical methods. Informally, this means that WiSARD has a large capacity to learn patterns, and a probabilistic upper bound on its generalization error can be easily computed.

2.3 BTHOWeN

BTHOWeN [57] (Bleached, Thermometer-Encoded, Hashed-input Optimized Weightless Neural Network), our prior work, enhanced the WiSARD model to improve model accuracy and memory efficiency and proposed an inference accelerator architecture. BTHOWeN outperformed inference accelerators for equally accurate quantized MLPs in throughput, area, and energy efficiency across a range of tabular datasets. BTHOWeN also outperformed the earlier Bloom WiSARD WNN [55] on these same datasets, establishing a new state of the art for WNNs. With ULEEN, we further improve on BTHOWeN to achieve higher accuracies on larger datasets. We provide a brief description of BTHOWeN here to explain the foundation on which ULEEN is built.

2.3.1 Counting Bloom Filters. A major challenge impacting WiSARD is the exponential growth in model size as the number of inputs to each RAM node, n, increases. As discussed previously,

[3]The **Vapnik–Chervonenkis (VC)** dimension of a binary classifier represents the complexity of the knowledge it can represent. Specifically, the VC dimension of a model is the largest value k for which it can correctly classify some set of k points no matter how elements of the set are labeled.

extremely large values of n result in overfitting. However, the value of n that gives the best model accuracy is still often too large to be practically implementable. In Reference [55], the authors observe that these very large RAM nodes are highly sparse: effectively, they are learning Boolean functions with many inputs but few minterms. They propose the Bloom WiSARD model, which uses Bloom filters[4] in RAM nodes instead of simple LUTs and can greatly reduce the size of a model with minimal impact on accuracy.

In both the conventional WiSARD model and Bloom WiSARD, RAM nodes become sensitive to an input pattern the first time it is seen during training. This means that rare patterns are treated with equal importance to common ones, which can result in the model learning spurious behaviors. Bleaching [14] is a technique in which RAM nodes count how many times each pattern was seen during training, and patterns seen fewer than some threshold b times are set to 0 before inference. This is accomplished by replacing the single-bit LUT entries in a WiSARD model with multi-bit counters.

BTHOWeN combines these two ideas by using *counting* Bloom filters to leverage both bleaching and model compression. Counting Bloom filters provide an *upper bound* on how many times patterns were seen. During training, multiple hash functions are used to index a small table of counters, and the counter with the least value is incremented. Bleaching is then performed, with counter values less than b replaced with 0 and larger values replaced with 1. Even though the upper bounds recorded by the counting Bloom filters are not tight, and therefore bleaching may not be strictly applied, this approach yields good accuracy. Additionally, the filters can be statically binarized after bleaching, meaning conventional (single-bit) Bloom filters can be used for inference.

2.3.2 Nonlinear Thermometer Encoding. Input features to WNNs are traditionally represented using a single bit: 1 if the feature is greater than its mean value in the training data and 0 otherwise. However, the granularity that can be provided using this encoding scheme is clearly limited. Binary integer encodings like those used in quantized DNNs are a poor alternative, since all bits are treated with equal importance when forming addresses. For instance, the least significant bit of an integer, taken in isolation, is essentially meaningless noise and therefore should not be used as an address bit.

Thermometer encoding is a *unary* coding in which a value is instead compared against a series of increasing thresholds. This encoding is named for its similarity to a mercury thermometer: as the input value (analogous to the mercury) increases and passes thresholds (the lines on the glass), bits become set from least to most significant. Thermometer encodings are the preferred multi-bit encoding for WNNs in prior work [31].

Most prior work uses equal intervals between the encoding thresholds, spacing them evenly between the minimum and maximum values of each feature. However, if a feature has outlying values, then this may result in poor resolution near the center of its range while an excessive number of bits are used only to represent these outliers. BTHOWeN instead assumes that features follow a roughly normal distribution. For each feature, the mean μ and variance σ^2 in the training data are used to specify the Gaussian curve $\mathcal{N}(\mu, \sigma^2)$. Next, the encoding thresholds are chosen to divide this distribution into regions of equal probability. This approach was first explored in Reference [62] in a narrow context, but BTHOWeN demonstrates that it is applicable to a wide range of classification tasks, even if input features do not actually follow a normal distribution.

[4]A Bloom filter consists of a small binary-valued lookup table and a set of independent hash functions. An input to the filter is hashed with each of the functions, and the results are used as addresses to index the LUT. The minimum of the accessed entries is the filter output.

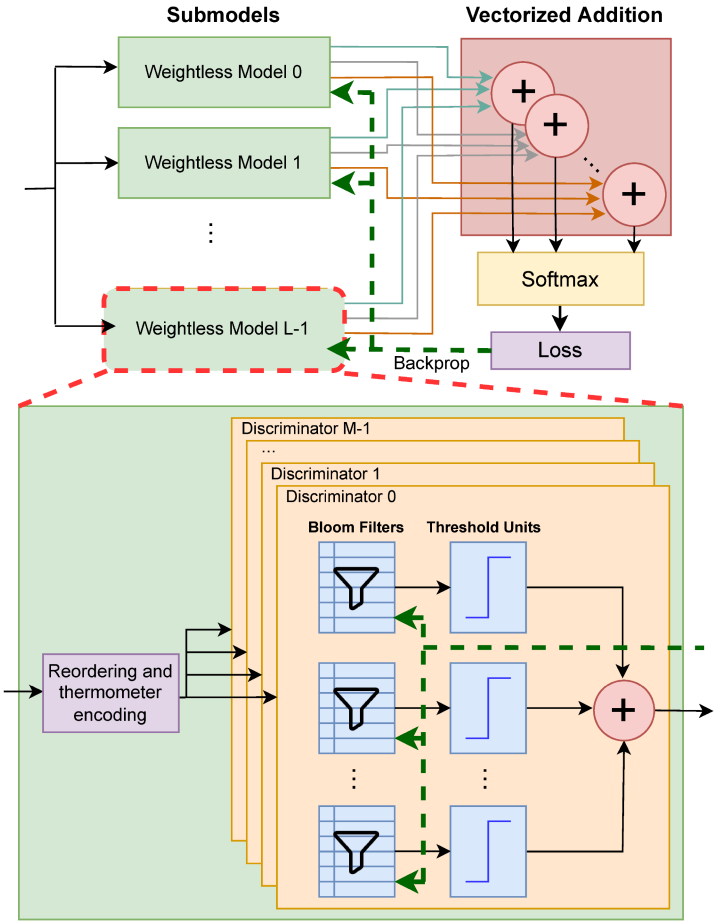

Fig. 2. ULEEN is composed of an ensemble of independently trained submodels, each of which is a WNN. Each submodel is composed of discriminators. Discriminators use continuous Bloom filters during training to allow for gradient-based weight updates. This figure shows the multi-pass training process, which uses backpropagation based on the straight-through estimator.

3 PROPOSED DESIGN: ULEEN

The ULEEN model includes three notable enhancements over our prior BTHOWeN model: (i) continuous Bloom filters, (ii) ensembles, and (iii) pruning. Furthermore, we explore a novel multi-pass training technique that gives improved accuracy over the conventional single-pass learning rule.

3.1 Model Overview

Figure 2 shows the ULEEN model at a high level, including training with our multi-pass technique. Inputs are fed to an ensemble of smaller weightless submodels, which are trained independently (i.e., an independent loss value is computed for each submodel). Submodel results are aggregated by adding the response scores for the corresponding discriminators in each submodel, shown in the "Vectorized Addition" block. During training, we take a softmax of the response scores for each submodel and compute cross-entropy loss. ULEEN uses continuous Bloom filters during training, which internally hold floating-point values. After training, the filters are binarized, converting them into conventional Bloom filters.

Our multi-pass training technique for ULEEN is a modified version of backpropagation based on the straight-through estimator [63]. The flow of gradients during backpropagation is shown with the green dotted arrow in the figure. Pruning, which eliminates RAM nodes that contribute least to overall accuracy, is a post-training technique not shown in Figure 2.

3.1.1 Continuous Bloom Filters. ULEEN replaces the counting Bloom filters used in BTHOWeN with continuous-valued Bloom filters, which store entries as floating-point numbers between −1 and 1. As shown in Figure 3, during inference, continuous Bloom filters output −1 if the least accessed entry is negative and +1 otherwise.

We use continuous Bloom filters to enable gradient-based updates to filter entries via backpropagation. Conventional Bloom filters with binary entries do not have enough granularity to represent the fine updates required for gradient-based learning. However, there is a challenge in using backpropagation for continuous Bloom filters: as demonstrated in Equation (1), the derivative of the sign function used for output binarization is not well-behaved:

$$\text{sign}(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}, \qquad \text{sign}'(x) = \begin{cases} +\infty & x = 0 \\ 0 & x \neq 0 \end{cases}. \tag{1}$$

If we use the sign function without modification, then during backpropagation, gradients will either be canceled to 0 (if $x \neq 0$) or explode to infinity (if $x = 0$). This issue also occurs in the domains of quantized and binary neural networks, where weights are commonly stored internally as floating-point values during training. A popular solution for these networks is to use the ***straight-through estimator* (STE)** function [17, 27, 63]. The STE behaves identically to the sign function during the forward training pass but computes a "proxy gradient" during the backward pass instead of its actual derivative. There are several different formulations of the STE in literature; for the continuous Bloom filters in ULEEN, we use the version given in Equation (2), which was also used in Reference [17]:

$$\text{STE}(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}, \qquad \text{STE}'(x) = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases}. \tag{2}$$

The hash functions used for the continuous Bloom filters in ULEEN do not need to be cryptographically secure and have constant-length input, so we can use very simple arithmetic-free approaches. In particular, we use functions drawn randomly from the H3 family [13], which consists only of AND and XOR operations. This same approach to hashing is also used in BTHOWeN [57].

Even though Bloom filters decouple the size of a RAM node from its number of inputs, it is neither practical nor desirable to provide all inputs to each filter. This would result in every filter learning the same patterns, which would negate the ability of ULEEN to generalize, and would also require very large hash tables to avoid excessive aliasing of different inputs. Therefore, like earlier approaches, we assign non-overlapping random subsets of the model inputs to each filter.

3.1.2 Efficient Ensembles. Ensembles, like the one shown in Figure 4, combine multiple weak classifiers into a single strong classifier. Ensembles have been extensively studied in other areas of machine learning, and are the driving concept behind techniques such as Bayesian model averaging, boosting, and bagging [19].

In ULEEN, we leverage ensembles by independently training several WNN submodels on the same training data. We then sum the response scores for each discriminator across the submodels before performing the final prediction. In other words, if a submodel i produces a response score R_{ij} for class j, then the final response score for this class will be $\sum_i R_{ij}$.

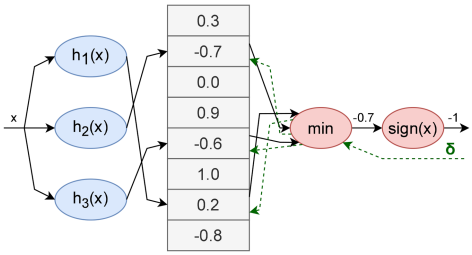

Fig. 3. Example of a continuous-valued Bloom filter with sign-based (+1/−1) binarization.

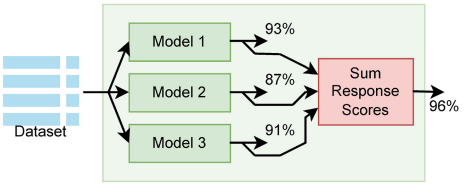

Fig. 4. Simplified view of an ensemble model for ULEEN. Summing the response scores of independently trained WNNs is more accurate than any of the individual submodels.

This ensemble technique is similar to but distinct from bagging. In bagging, submodels are trained using random subsets of the training data to influence them to learn different patterns and behaviors. However, in ULEEN, all submodels see the same training data, but the connections from model inputs to RAM nodes are different. This sparse connectivity forces RAM nodes in different submodels to behave distinctly.

One might reasonably expect that using ensembles of submodels would increase the size of a ULEEN model, since there are more RAM nodes in total. However, we have found that in practice this is frequently not the case. The individual submodels of an ensemble can be made much smaller (and therefore individually less accurate) than a monolithic model without significantly degrading ensemble accuracy. Since the amount of hash computation required for inference increases with the number of submodels, we avoid using ensembles with excessively many submodels. ULEEN ensembles of numerous tiny submodels can give excellent accuracy, but they are impractical to implement in hardware.

The idea of ensembles of WiSARD models was previously explored in Reference [44]. However, this work was based on the single-pass training technique and did not yield good results: the ensembles in this work were far larger than monolithic WiSARD models with only a marginal improvement in accuracy.

3.1.3 Pruning. Pruning removes parameters and connections from a model to reduce its size and complexity. Optimized DNNs [10, 26, 47, 64, 66] have applied pruning techniques with excellent results, but similar concepts have not been used in prior WNNs.

We introduce a novel technique for pruning WNNs based on identifying and eliminating RAM nodes that are irrelevant or harmful to model accuracy. After training, we evaluate how useful each RAM node is for predicting the model output. In particular, for each RAM node N_{ij} in discriminator d_i, we observe whether N_{ij} outputs a 1 when the correct class label is i. We then compute true and false positive and negative rates for this test. For a dataset with M classes, the utility score for a RAM node is given by $(M-1)(TPR-FNR)+(TNR-FPR)$.

Next, a fixed fraction of the RAM nodes in each discriminator with the lowest utility scores are removed from the model. Pruning in this way reduces the maximum possible response of a discriminator, since it now has fewer RAM nodes. However, the exact impact may vary between discriminators—for instance, a RAM node that always outputs 0 and a RAM node that always outputs 1 are equally useless, but removing them will have different impacts on a discriminator's response score. To compensate for this, we learn an integer bias for each discriminator, which is added to their outputs. Finally, we fine-tune the remaining RAM nodes with an additional brief training pass.

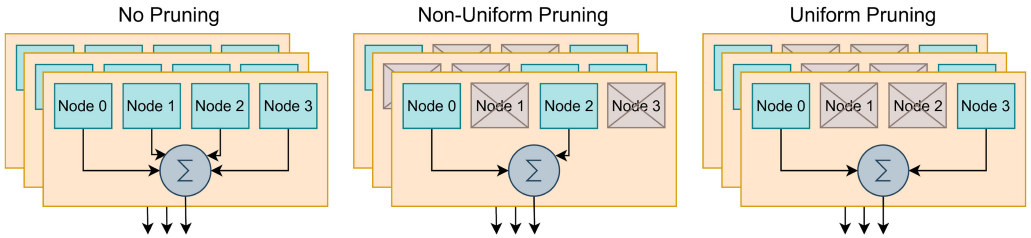

Fig. 5. Simplified views of pruned WNNs. Non-uniform pruning requires the same number of RAM nodes to be pruned in each discriminator but otherwise allows for arbitrary pruning behavior. Uniform pruning requires RAM nodes at the same index to be pruned in all discriminators.

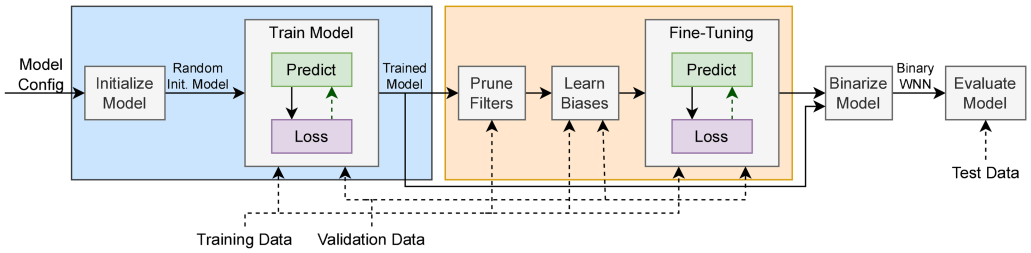

Fig. 6. ULEEN introduces a multi-pass WNN training rule based on backpropagation. After training, the filters that contribute least to overall model accuracy are pruned and replaced with a constant bias, and the remaining filters are fine-tuned.

As shown in Figure 5, we explored two different variants of pruning. Non-uniform pruning places no special requirements on which RAM nodes are eliminated, only enforcing that an equal number are pruned in each discriminator. Uniform pruning, by contrast, requires RAM nodes to be pruned at the same indices in all discriminators. Uniform pruning generally has a larger impact on accuracy than non-uniform pruning, since a RAM node with low utility in one discriminator may have high utility in another. However, uniform pruning is more effective in reducing the amount of hash computation needed for inference.

In all datasets explored in this article, we can prune at least 30% of RAM nodes from a ULEEN model with <1% reduction in accuracy. Though this does not approach the degree of pruning that is frequently possible for DNNs, it is still a notable reduction in memory requirements and circuit area. When used with an ensemble, bias terms can be summed across submodels, meaning the only inference overhead incurred by pruning is a single bias addition for each output class.

3.2 Training ULEEN

Figure 6 summarizes our multi-pass training process. Continuous Bloom filter entries are randomly initialized between -1 and 1 and iteratively updated using backpropagation with the straight-through estimator. Note that we do not backpropagate gradients through the hash functions, meaning thermometer encoding thresholds are not updated during training.

During training, we also use dropout ($p = 0.5$) to randomly set the outputs of RAM nodes to 0. Consequently, RAM nodes have three possible outputs during training (-1, 0, or 1) but only two during inference (-1 or 1). Without this regularization, we found that larger ULEEN models tended to overfit, in some cases perfectly memorizing the training data at the cost of generalization accuracy. Dropout effectively mitigates this issue.

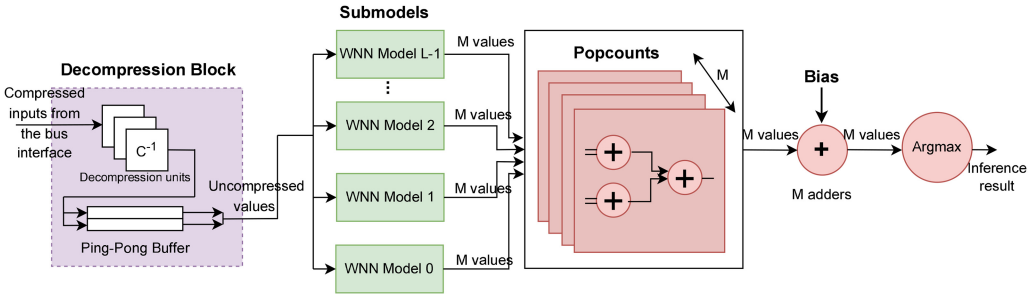

Fig. 7. Diagram of the ULEEN inference accelerator architecture. Input is deserialized and, if needed, decompressed, before being passed to an ensemble of submodels. The outputs of the submodels are summed and biased to get per-class responses, and the index of the strongest response is taken as the prediction.

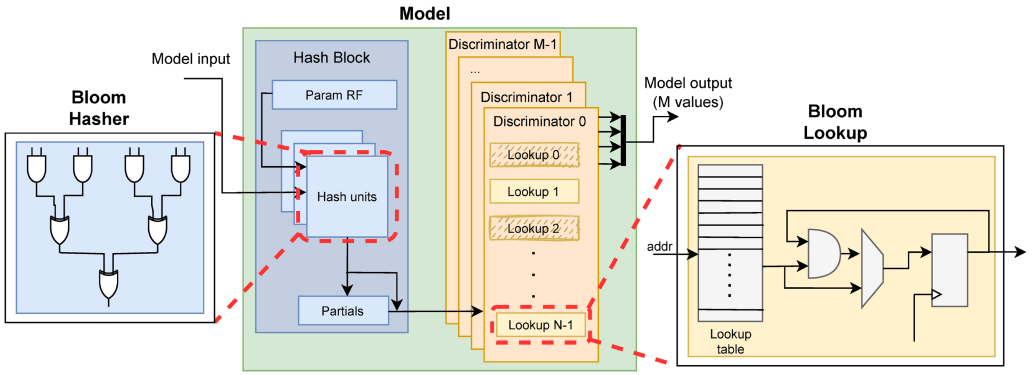

Fig. 8. Details of a submodel in the ULEEN inference accelerator. Each submodel contains a hardware block for computing hash functions and a set of hardware units for performing lookups. These blocks collectively compose the Bloom filters, which are divided into separate units to eliminate redundant computation.

Models are trained using the Adam optimizer [32] with a base learning rate of 10^{-3}. Initial filter entry values are uniformly sampled (i.e., drawn from $\mathcal{U}(-1, 1)$). We also experiment with a simple form of data augmentation for the MNIST dataset: We make nine copies of each image in the training set, shifted horizontally and vertically between -1 and 1 pixel.

After training, models are pruned and fine-tuned according to the method we proposed in Reference [58] and discussed previously in Section 3.1.3. After pruning, the continuous Bloom filters are statically binarized by applying the unit step function, converting them into conventional Bloom filters.

Unlike the single-pass learning rule used in prior work, ULEEN presents inputs to all discriminators during training, rather than just the one corresponding to the correct output class. The use of gradient-based learning causes discriminators corresponding to incorrect outputs to be inhibited in addition to the correct discriminator being reinforced, which is not possible with the single-pass rule.

3.3 Inference Accelerator Architecture

Figures 7 and 8 show the block diagram of our pipelined ULEEN inference accelerator. To simplify control logic, units on the chip operate in lockstep to the greatest extent possible. This means

that an entire input sample must be read in before computation can begin. This deserialization is performed using a double-buffered bus interface (not shown). Input data may optionally be compressed by replacing unary thermometer-encoded values with binary values representing how many bits are set. This reduces data movement from off-chip but requires a decompression unit to recover the thermometer encoding, shown at the left of Figure 7. This unit is eliminated from the design if input data is not compressed.

The discriminators in Figure 8 exhibit several optimizations. Though each H3 hash function in a Bloom filter requires a different random parameter, there is no disadvantage to sharing these parameters between all Bloom filters in a submodel. Therefore, a central register file (shown as "Param RF" in the figure) is used to store hash parameters. Since input order is shared between discriminators in a submodel, when hash parameters are also shared, all discriminators will receive the same hashed values. Therefore, it is redundant to calculate hashes for each discriminator separately; instead, we use a single central hashing block. The hash block is itself composed of many pipelined hash units that process input sequences with a throughput of 1 hash/cycle. As shown in the left part of Figure 8, these hash units perform only `AND` and `XOR` operations, with no arithmetic component. If hashing is fully parallelized, then the performance of the design will be heavily bottlenecked by off-chip bandwidth. Therefore, we can multiplex the hash units, using each one to compute multiple hashes, without impacting circuit throughput. This reduces the number of hash units and thus the circuit area. We accumulate partial hash results in an intermediate buffer. Once the buffer is full and the last partial hash is available, all Bloom filters perform a lookup in lockstep.

Since hashing is moved into a central block, discriminators themselves contain only lookup units. The lookup unit, shown at the right of Figure 8, consists of a lookup table and the hardware to perform an `AND` reduction. A 1-bit accumulator in each lookup unit can take as input either the output of the LUT or the `AND` of that output and its current contents. Once all hash lookups have been performed, the outputs of the lookup units are marked as valid.

Each submodel in an ensemble must compute its own hashes, since input orders and hash input and output widths vary. Since different submodels have different table contents, sizes, and pruning, they also have their own sets of filter units.

Popcounts and submodel response accumulation are performed using a vector of adder trees, shown in the right of Figure 7. Bias values are added to these results (Section 3.1.3), and the index of the strongest response is computed to produce a final inference result.

4 EVALUATION METHODOLOGY

We compare ULEEN against FINN [59], a framework from Xilinx Research Labs for BNN inference on FPGAs. FINN does not itself propose a novel BNN algorithm but is instead a tool for generating hardware accelerators for pretrained BNNs. Therefore, in addition to hardware results, comparing against FINN gives us an idea of the performance of ULEEN relative to the broader domain of BNN literature. We use fully connected, multilayer perceptron-style FINN models for our comparison. While FINN supports generating both fully connected and convolutional models, we focus solely on the former in this work, since ULEEN is not a convolutional architecture.

FINN only provides fully connected model results for MNIST. They propose three network topologies, SFC, MFC, and LFC, which each contain three hidden layers with 256, 512, and 1024 neurons per layer, respectively. Throughput-optimized "max" and area-optimized "fix" FPGA implementations are proposed for each of these models. We compare against the "max" implementations in this work, since ULEEN is also intended to achieve a high peak throughput. To the best of our knowledge, these FINN models represent the most efficient FPGA implementations of BNNs for MNIST. For the MLPerf Tiny datasets, we train BNN models for FINN using the Xilinx

Brevitas [48] low-precision machine learning library. We use three hidden layers for these models as well, with differing numbers of neurons per hidden layer.

4.1 Datasets

We consider the four datasets in MLPerf Tiny [7] and the MNIST [36] dataset in this work:

(1) **Keyword spotting (KWS):** This dataset is extracted from Speech Commands v2 [60], which consists of 105,829 utterances from 2,618 speakers. It consists of spectrograms representing ten different keywords, plus an "unknown word" category.
(2) **ToyADMOS/car:** This dataset consists of audio recordings of seven different toy cars [34]. The objective is to identify "anomalous" samples collected from deliberately damaged cars.
(3) **Visual Wake Words (VWW):** This dataset consists of 96×96 grayscale images extracted from the MSCOCO 2014 dataset [40]. The objective is to determine whether an image contains at least one person.
(4) **CIFAR-10:** An image classification dataset consisting of 32×32 RGB images in 10 classes [35].
(5) **MNIST:** Image classification, with 28×28 grayscale images of the digits 0–9 [36].

Not all of these datasets are well-suited to MLPs or ULEEN. In particular, CIFAR-10 and VWW exhibit high degrees of positional independence, where image features may be present in different locations and at different scales. Therefore, our primary goal with these two datasets is to show that ULEEN is superior to binary MLPs—additional algorithmic improvements, including multi-layer convolutional WNNs, will need to be explored in future work to approach state-of-the-art accuracies.

4.2 Software Implementation

While the traditional single-pass learning rule for WNNs is computationally inexpensive and easily run on a CPU, our multi-pass training flow is considerably more complex, as it requires gradient computation in addition to multiple epochs of training. This is acceptable for our purposes, since our focus in this work is on optimizing inference. We implement multi-pass training using custom extensions to the PyTorch [50] machine learning library, including C++ extensions leveraging the LibTorch API. This allows us to run GPU-based training, significantly improving training speed. Forward and backward passes for Bloom filters are implemented as a single multi-dimensional gather/scatter operation, which enables efficient memory-parallel computation.

FINN is designed to leverage Brevitas [48], an open-source library by Xilinx for creating low-precision and binary neural networks. We use Brevitas to train new binary MLP models for MNIST and the four datasets in MLPerf Tiny [7].

4.3 Hardware Implementation

We use the Mako [8] templating library to generate SystemVerilog source for the ULEEN accelerator. This flow automatically extracts parameters from trained models, constructs state machines, and determines how many functional units to instantiate to hit throughput targets while minimizing area. This templating allows us to generate new accelerators just by changing command-line parameters. We simulate our designs using Synopsys VCS 2018 to ensure functional correctness and to collect latencies and throughputs.

For our FPGA implementation and comparison, we use the Zynq Z7045 SoC platform, which is also used for FINN. Our design has the same I/O interface width as FINN (112 bits). We use Xilinx Vivado 2019.2 to synthesize and implement our designs. We target the same frequency of operation (200 MHz) as FINN, though we are unable to achieve this frequency in all cases due to FPGA

Table 1. Comparison between ULEEN and Prior Work (FINN) for MNIST and MLPerf Tiny

Dataset	ULEEN		FINN BNN Iso-accuracy			FINN BNN Iso-size		
	Size (KiB)	Test Acc.%	Size (KiB)	Test Acc.%	Hidden layer size	Size (KiB)	Test Acc.%	Hidden layer size
MNIST-S	16.9	96.2	40.8	95.8	256 × 3 (SFC)	16.4	95.2	128 × 3
MNIST-M	101	97.8	114	97.7	512 × 3 (MFC)	103	97.7	480 × 3
MNIST-L	262	98.5	355	98.4	1,024 × 3 (LFC)	283	98.0	896 × 3
KWS	101	70.3	324	70.6	1,024 × 3	101	67.0	524 × 3
ToyADMOS	16.6	86.3	36.1	86.1	256 × 3	16.4	85.5	144 × 3
VWW	251	61.8	3,329*	57.1*	2,048 × 3*	264	55.7	224 × 3
CIFAR-10	1,379	54.2	19,466*	45.7*	8,192 × 3*	1,345	44.4	1,700 × 3

ULEEN is smaller than similar-accuracy FINN models, and more accurate than similar-size FINN models. *FINN is unable to achieve ULEEN's accuracy for VWW or CIFAR-10.

routing congestion. Resource usage and power consumption are obtained from post-implementation Vivado reports.

We leverage the Xilinx FINN HLS flow to create optimized FPGA implementations for the BNNs we trained using Brevitas. The FINN compiler applies a series of transformations to convert the network's nodes to layers, which invoke functions in a highly optimized `finn-hls` library, using time-multiplexing (referred to in the FINN documentation as "folding") to reduce execution resources. The resultant C++ code is then passed through Xilinx Vivado HLS to generate the RTL. We also attempted to replicate the hardware generation for the SFC, MFC, and LFC models. However, our FINN implementations were significantly less efficient than Xilinx's published results. This suggests that these models were originally hand-tuned to an extent that we are not able to replicate. Therefore, we use the original values from Xilinx for these three models in our comparisons.

5 RESULTS

5.1 Software Comparison of ULEEN with BNNs

We begin our analysis by comparing ULEEN with the Xilinx FINN [59] platform for FPGA-based BNN inference. We present seven ULEEN models in Table 1: three for the MNIST dataset with varying model sizes and accuracies and four for the MLPerf Tiny datasets. We also present two FINN models for each ULEEN model: one to match ULEEN's accuracy ("FINN BNN Iso-accuracy" in Table 1), and one to target ULEEN's parameter size ("FINN BNN Iso-size"). The iso-accuracy FINN models for MNIST (SFC, MFC, and LFC) are from the original FINN paper; the other 11 models, we trained ourselves using the Xilinx Brevitas [48] library. Our FINN models use three hidden layers of equal size, which is the same approach used for the prior FINN models. Note that FINN is unable to match ULEEN's accuracy on VWW and CIFAR-10 even with model parameter sizes an order of magnitude larger.

Overall, ULEEN is consistently able to match the accuracy of binary MLPs with a smaller parameter size or match their parameter size with a higher accuracy. In particular, compared to iso-accuracy FINN models, ULEEN has 1.1× to 3.2× fewer parameters, with a geometric mean reduction of 1.9×[5]. Compared to iso-parameter-size FINN models, ULEEN is 0.1% to 9.8% more accurate, with a mean improvement of 3.1%.

[5]These figures for parameter size reduction exclude the iso-accuracy FINN models for the VWW and CIFAR-10 datasets, since they could not match the accuracy of ULEEN. If we include these two models, then the maximum improvement of ULEEN over FINN increases to 14.1× and the geometric mean to 3.4×.

Table 2. Details of the Selected ULEEN Models

Dataset	Model	Submodel	Bits/Inp	Inputs/Filter	Entries/Filter	Size (KiB)	Test Acc.%
MNIST	ULN-S	Ensemble	2	—	—	16.9	96.20
		SM0	2	12	64	7.19	92.91
		SM1	2	16	64	5.39	90.25
		SM2	2	20	64	4.38	86.16
	ULN-M	Ensemble	3	—	—	101	97.79
		SM0	3	12	64	10.9	83.54
		SM1	3	16	128	16.0	90.93
		SM2	3	20	256	26.0	92.92
		SM3	3	28	256	18.44	87.05
		SM4	3	36	512	29.38	80.93
	ULN-L	Ensemble	7	—	—	262	98.46
		SM0	7	12	64	25.0	88.78
		SM1	7	16	128	37.7	93.24
		SM2	7	20	128	30.2	92.44
		SM3	7	24	256	50.3	93.92
		SM4	7	28	256	43.1	90.47
		SM5	7	32	512	75.6	90.44
KWS		Ensemble	12	—	—	101	70.34
		SM0	12	5	8	9.62	56.93
		SM1	12	6	16	16.1	59.32
		SM2	12	7	32	27.5	59.94
		SM3	12	8	64	48.12	61.01
ToyADMOS		Ensemble	6	—	—	16.6	86.33
		SM0	6	7	64	6.88	83.61
		SM1	6	9	64	5.34	82.32
		SM2	6	11	64	4.38	79.85
VWW		Ensemble	12	—	—	251	61.76
		SM0	12	5	8	30.2	59.07
		SM1	12	7	16	43.2	57.78
		SM2	12	9	32	67.2	59.20
		SM3	12	11	64	110	58.96
CIFAR-10		Ensemble	8	—	—	1379	54.21
		SM0	8	6	32	112	49.12
		SM1	8	8	64	168	49.53
		SM2	8	12	128	224	46.39
		SM3	8	16	256	336	42.23
		SM4	8	20	512	538	38.27

SMx refers to individual submodels comprising the ensemble model.

Detailed information on the seven ULEEN models is shown in Table 2. We report accuracies for both the ensembles as a whole and their component submodels. We observe that in most cases, the ensemble is more than 4% more accurate than the best single submodel. This demonstrates the effectiveness of our ensemble technique.

As mentioned in Section 4, both ULEEN and FINN have poor accuracy on the VWW and CIFAR-10 datasets. This is an unsurprising result, since these two datasets contain images with a high degree of positional variance (e.g., a person could appear in the top left or bottom right of an image). Since ULEEN and the FINN models we use for comparison do not incorporate convolution, they struggle to learn position-independent features. While ULEEN is well-suited for applications with little positional variance, such as tabular datasets, we do not claim it as a universally applicable machine learning model. Since both ULEEN and FINN perform poorly on these two datasets, we do not consider them in our hardware implementation analysis.

5.2 Hardware Comparison of ULEEN with FINN

Detailed comparisons between our FPGA implementations of ULEEN and the iso-accuracy FINN models are shown in Table 3. We report dynamic energy for a single inference in isolation (batch size $b = 1$) and steady-state inference ($b = \infty$). The results reported for FINN SFC, MFC, and LFC in Reference [59] include total power but not dynamic power. Since we do not have the RTL to replicate these experiments directly, we use a constant 0.3 W static power for the three models. This value is an estimate based on the synthesis results of other FINN models; all experiments show a very similar static power consumption.

Overall, ULEEN reduces energy per inference by an average of 8.3× (5.5–11.7×; geometric mean) for a single inference, and by 7.1× (3.8–9.1×) for steady-state inference. While ULEEN also has superior latency and throughput to FINN (by 5.3× and 3.7×, respectively), this is more difficult to compare directly due to the different areas of the two models. We use LUT utilization as a proxy for total circuit area, noting that this comparison is to FINN's advantage as it also uses BRAMs while ULEEN does not. ULEEN's area-delay product is on average 4.5× (1.7–7.8×) lower than FINN's, with 3.6× (1.7–6.1×) higher throughput per unit area.[6] Therefore, ULEEN is substantially superior to the iso-accuracy FINN models in energy efficiency, latency, and throughput, even after adjusting for differences in model areas.

Figures 9 and 10 compare ULEEN's energy efficiency and throughput per unit area against the iso-accuracy and iso-size FINN models. ULEEN's advantage over the iso-size models is smaller than with the iso-accuracy models, but it still outperforms even these less accurate BNNs. ULEEN's steady-state energy efficiency is on average 3.8× (2.2–6.2×; geometric mean) better than the iso-size FINN models, and its throughput per unit area is 2.7× (1.4–4.8×) higher.

To summarize, ULEEN is faster and more efficient than even less accurate MLP-style BNNs in an optimized FPGA implementation. While it is not suitable for all workloads (particularly large image datasets), it is a strong option for applications with less positional variance and tabular datasets.

5.3 Sensitivity Analysis

Figure 11 shows the performance of prior work and the impact of different WNN model optimizations on the MNIST, KWS, and ToyADMOS datasets. The models in this figure demonstrate the improvements to WNN accuracies and parameter sizes in the last four years. The seven models evaluated are, in order, (1) Bloom WiSARD [55], (2) a variant of Bloom WiSARD that incorporates counting Bloom filters during training, (3) a further variant that also incorporates a simple linear thermometer encoding, (4) BTHOWeN [57], which instead uses a Gaussian thermometer encoding, (5) a variant of BTHOWeN, which uses our multi-pass learning technique, (6) an ensemble of such models, and (7) the full ULEEN model, including pruning.

[6]We normalize for circuit area, since in some cases ULEEN's circuit area is significantly larger than FINN's. Using unnormalized throughputs, ULEEN is 1.2–15.0× faster, with a geometric mean of 3.7×.

Table 3. Comparison of ULEEN against Iso-accuracy FINN Models for MNIST, KWS, and ToyADMOS

Model	Latency (μs)	Xput (kIPS)	Dynamic μJ/Inf. $b = 1$	Dynamic μJ/Inf. $b = \infty$	LUT	BRAM	Test Acc.%
ULN-S	0.21	14,286	0.191	0.062	17,319	0	96.20
FINN-SFC	0.31	12,361	2.170	0.566	91,131	4.5	95.83
ULN-M	0.29	14,286	0.823	0.199	49,445	0	97.79
FINN-MFC	N/D	6,238	N/D	1.763	N/D	N/D	97.69
ULN-L	0.94	4,048	3.137	0.823	123,102	0	98.46
FINN-LFC	2.44	1,561	20.74	5.445	82,988	396	98.40
ULN-KWS	0.39	10,000	2.536	0.642	141,074	0	70.3
FINN-KWS	7.78	668	29.72	5.716	42,847	151.5	70.6
ULN-ToyADMOS	0.34	11,111	0.549	0.143	29,404	0	86.3
FINN-ToyADMOS	3.52	1,568	3.022	0.547	14,100	34.5	86.1

FINN rows are shaded. Note that some results are not available for the MFC model, as they were not included in the publication that proposed the model (N/D: No data available).

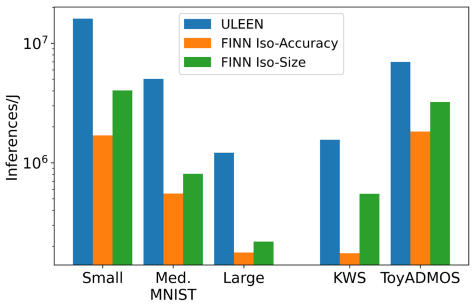

Fig. 9. Energy efficiency (steady-state inferences per Joule) comparison between FINN and ULEEN.

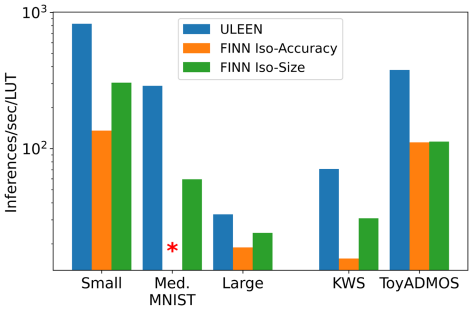

Fig. 10. Area efficiency (inferences per second per LUT) comparison between FINN and ULEEN. (*LUT usage is not provided for the medium MNIST iso-accuracy FINN model in Reference [59].)

ULEEN is significantly more accurate than our prior work, BTHOWeN, which in turn is both more accurate and smaller than the earlier Bloom WiSARD model. The BTHOWeN models shown in Figure 11 have on average 4.2× fewer parameters than the corresponding Bloom WiSARD models and reduce average test error by an average factor of 1.5×. The ULEEN models have approximately the same parameter sizes as the BTHOWeN models (a deliberate choice when selecting comparison models) but reduce test error by an additional geomean factor of 2.3×. Overall, ULEEN models are 3.1–5.5× smaller than Bloom WiSARD, with 2.3–5.3× lower error.

Comparing BTHOWeN to Bloom WiSARD, it is evident that the ability to reject rare input patterns (using counting Bloom filters in BTHOWeN, and continuous Bloom filters in ULEEN) is critical for model accuracy and efficiency. This change alone reduces test error by 1.3× and model size by 4.9× compared to the Bloom WiSARD baseline. Using a linear or Gaussian thermometer encoding provides some additional reduction in model error, with the Gaussian encoding providing a greater reduction than the linear approach. However, thermometer encoding also increases the model size. To counter this, we decrease the size of the Bloom filters, which negates some of the improvement in accuracy.

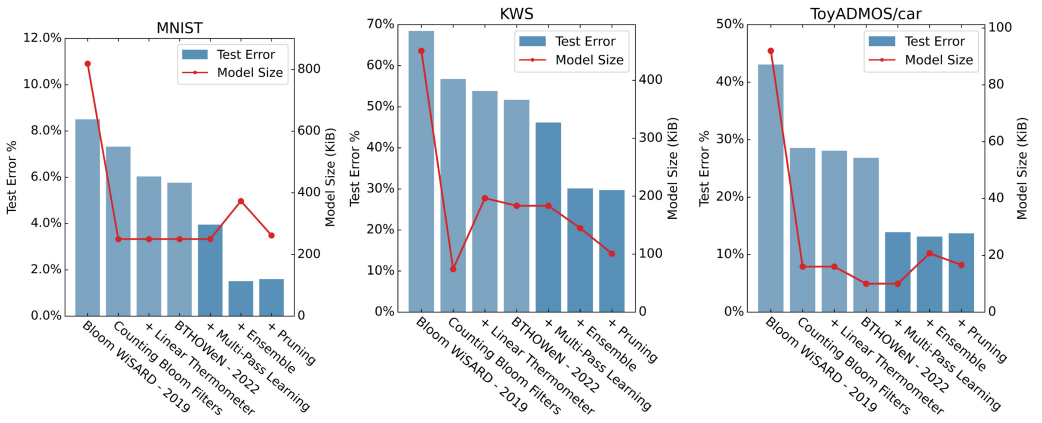

Fig. 11. Accuracy and model size of WNNs as progressive model improvements are applied. Results presented for MNIST, KWS, and ToyADMOS datasets. Entries up to and including BTHOWeN represent the improvements made in recent prior work.

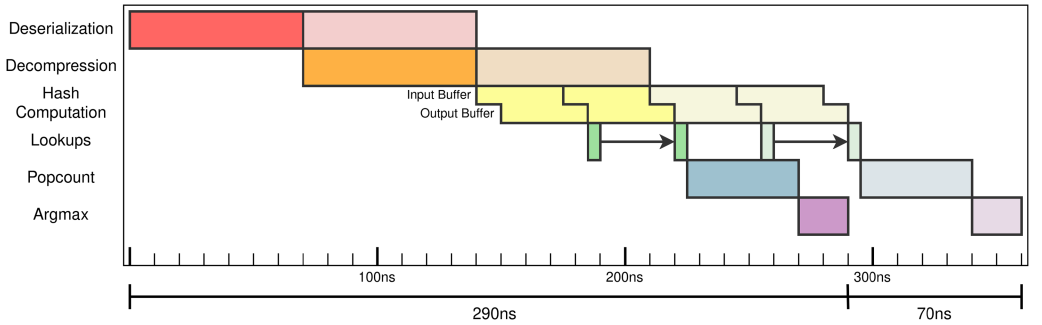

Fig. 12. Breakdown of the time spent in each stage for inference with ULEEN, showing two inputs being processed in a pipelined fashion.

About 55% of the improvement from BTHOWeN to ULEEN is attributable to the multi-pass learning rule, with the remainder coming from the addition of submodel ensembles. Pruning reduces model sizes by an average of 27% with at most a 0.6% increase in test error. In fact, for KWS, accuracy actually *improved* due to the elimination of noise from low-utility RAM nodes. These results, combined with the low overhead of the optimizations we propose, demonstrate the superiority of ULEEN over prior WNNs for edge inference.

Figure 12 shows the occupancy of the ULEEN accelerator's pipeline stages during inference, using the ULN-M model as an illustrative example. The pipelined design of our accelerator enables substantial overlapping of computation, with 290 ns of latency but one sample finished every 70 ns in the steady state. The hash computation stage is itself internally pipelined, as its functional units have a two-cycle latency. The latencies of the decompression and hash computation stages can be tuned by instantiating different numbers of functional units. While by default we instantiate only enough functional units for these two stages to match the throughput of the deserialization stage, increasing the counts of these functional units could decrease the latency of ULN-M to a theoretical minimum of 165 ns (a 43% reduction). However, this would significantly increase the area and routing complexity of the design with no benefit to throughput.

6 RELATED WORK

ULEEN targets applications with extremely low energy budgets. In this section, we discuss other works that explore related use cases.

Quantized DNNs: Quantization involves transforming higher-precision floating-point numbers to lower-precision floating-point, fixed-point, or integer [29] values. Quantizing networks leads to reduced memory access costs and increases compute efficiency. DNN accelerators commonly provide hardware support for lower precisions, such as 8-bit integer in Google's TPU [30] or 12-bit floating-point in Microsoft's Brainwave [22].

Ternary Weight Networks (TWNs) [38, 41, 68] constrain weights to +1, 0, and −1. The accuracy of TWNs has been shown in some cases to be only slightly worse than full-precision DNNs and superior to BNNs. Some ASIC and FPGA implementations of ternary neural networks for MNIST are presented in Reference [5], but these have lower accuracy, throughput, and energy efficiency than our ULEEN models.

Binary NNs: Many methods to binarize weights or activation functions in DNNs have been proposed [16, 17, 23, 27, 52]. In most modern BNNs, neurons take the XNOR of their input with a learned weight vector, perform a popcount on the result, and compare this value against a fixed or learned threshold to determine their output. Several works have explored ASIC and FPGA accelerators for BNNs (e.g., FINN [59], FP-BNN [39], and YodaNN [6]). As evidenced in our comparison with FINN, ULEEN WNNs can achieve better latency and energy than BNNs. ReBNet [24] is another FPGA-based BNN accelerator for MNIST. Our throughput and accuracy are higher than ReBNet, but no energy numbers are provided in the article.

Compressed DNNs: To efficiently run DNNs on devices with limited hardware resources, model compression is commonly used [20, 33, 42, 64]. Pruning [26, 28, 47] reduces the total number of parameters but may require retraining to avoid accuracy degradation. Compression techniques such as weight and bit sparsity have also been explored for CNNs [1, 2, 45, 49, 65, 67]. We use two forms of WNN compression in ULEEN: Bloom filters to reduce the size of the RAM nodes and pruning to reduce the number of RAM nodes.

Symbolist Approaches: Symbolist paradigms such as decision trees and Bayesian inference generally work well for small-scale problems. However, hardware implementations of these models typically require sophisticated, heavy domain-dependent preprocessing, or require large (e.g., half a million LUTs [15]) or very customized (e.g., clockless probabilistic p-bits [21]) circuit implementations.

TinyML: Microcontroller-based approaches such as TinyML leverage inexpensive off-the-shelf hardware for machine learning. However, the strengths of these approaches lie in their low cost of entry and their simplicity, not in their efficiency. For example, a TinyML MNIST implementation [56] on an Arduino Nano, using downscaled 8×8 images, had more than 130,000× lower throughput, more than 32,000× higher latency, and was less accurate than our FPGA implementation of ULN-M.

Prior WNNs: We have discussed several earlier WNN architectures throughout this work. Table 4 summarizes their attributes in comparison to ULEEN.

System Effects: We have limited the scope of our analysis in this work to the performance of the ULEEN and FINN models and accelerators themselves, ignoring broader-scope system effects. At the same time, recent work has demonstrated that system-level latency and energy overheads—the so-called "AI Tax"—can be substantial in edge inference contexts [9, 53]. Since the unary and binary encoding formats needed as input to ULEEN and FINN are unlikely to be supported by commercial off-the-shelf sensors, data preprocessing needs to be performed either on the FPGA (at an overhead to area) or by the general-purpose CPU in an SoC (which may struggle to provide sufficient throughput). In either case, this will also introduce a significant latency overhead. In a

Table 4. Qualitative Comparison of ULEEN with Prior WNNs

Model	Bloom filters	Thermometer encoding	Submodel ensembles	Bleaching	Multi-pass training	Pruning
WiSARD [4]	✗	✗	✗	✗	✗	✗
Bloom WiSARD [55]	✓	✗	✗	✗	✗	✗
WiSARD Encodings [31]	✗	✓	✗	✓	✗	✗
Regression WiSARD [44]	✗	✓	✓	✓	✗	✗
BTHOWeN [57]	✓	✓	✗	✓	✗	✗
ULEEN (this work)	✓	✓	✓	✗*	✓	✓

*The multi-pass learning rule introduced in ULEEN supplants the bleaching algorithm.

more fully customized environment such as a smart sensor, it should be possible to transform raw sensor data directly into the input format needed by ULEEN, reducing preprocessing latency. Due to the very large design space inherent in these considerations, we leave this analysis to future work.

7 CONCLUSION

Inference on the extreme edge demands new approaches to machine learning. While existing techniques use quantized DNNs or BNNs, we propose ULEEN, an approach based on WNNs. We augment a state-of-the-art WNN architecture with submodel ensembles, RAM node pruning, and a novel multi-pass learning rule to improve model accuracies and reduce parameter sizes. We present FPGA-based implementations of an inference accelerator for ULEEN. Compared against iso-accuracy models using the FINN platform for FPGA-based BNNs, we improve steady-state energy efficiency by 3.8–9.1× and area-delay product by 1.7–7.8× across the MNIST, KWS, and ToyADMOS/car datasets.

The most direct opportunity for future work that we see is the development of convolutional WNNs. Convolution is critical to achieving high accuracies on larger image datasets such as CIFAR-10 and VWW. Although support for convolution increases the complexity of models and accelerators, we believe that WNNs have the potential to excel in this domain, as they can in principle learn nonlinear convolutional filters.

Overall, ULEEN significantly advances the state-of-the-art for WNN accuracy, demonstrating that WNNs can outperform even highly optimized BNNs for some applications. WNNs are a superior approach for low-latency, high-throughput inference in ultra-low-energy environments.

ACKNOWLEDGMENTS

The authors thank the reviewers for their constructive feedback and suggestions.

REFERENCES

[1] Jorge Albericio, Alberto Delmás, Patrick Judd, Sayeh Sharify, Gerard O'Leary, Roman Genov, and Andreas Moshovos. 2017. Bit-pragmatic deep neural network computing. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. 382–394.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA'16)*. 1–13. https://doi.org/10.1109/ISCA.2016.11

[3] Igor Aleksander, Massimo De Gregorio, Felipe França, Priscila Lima, and Helen Morton. 2009. A brief introduction to Weightless Neural Systems. In *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN'09)*. 299–305.

[4] I. Aleksander, W. V. Thomas, and P. A. Bowden. 1984. WISARD·a radical step forward in image recognition. *Sensor Rev.* 4, 3 (1984), 120–124. https://doi.org/10.1108/eb007637

[5] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. 2017. Ternary neural networks for resource-efficient AI applications. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'17)*. 2547–2554. https://doi.org/10.1109/IJCNN.2017.7966166

[6] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2018. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 37, 1 (2018), 48–60. https://doi.org/10.1109/TCAD.2017.2682138

[7] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. MLPerf tiny benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS'21)*.

[8] Michael Bayer. 2021. Mako Templates for Python. Retrieved from https://www.makotemplates.org/

[9] Michael Buch, Zahra Azad, Ajay Joshi, and Vijay Janapa Reddi. 2021. AI tax in mobile SoCs: End-to-end performance analysis of machine learning in smartphones. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'21)*. 96–106. https://doi.org/10.1109/ISPASS51385.2021.00027

[10] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-All: Train One Network and Specialize it for Efficient Deployment. Retrieved from https://arxiv.org/abs/1908.09791 https://doi.org/10.48550/ARXIV.1908.09791

[11] Douglas O. Cardoso, Danilo Carvalho, Daniel S. F. Alves, Diego F. P. de Souza, Hugo C. C. Carneiro, Carlos E. Pedreira, Priscila M. V. Lima, and Felipe M. G. França. 2016. Financial credit analysis via a clustering weightless neural classifier. *Neurocomputing* 183 (2016), 70–78. arXiv:10.1016/j.neucom.2015.06.105

[12] Hugo Carneiro, Carlos Pedreira, Felipe França, and Priscila Lima. 2018. The exact VC dimension of the WiSARD n-tuple classifier. *Neural Comput.* (Nov. 2018), 1–32. https://doi.org/10.1162/neco_a_01149

[13] J. Lawrence Carter and Mark N. Wegman. 1979. Universal classes of hash functions. *J. Comput. Syst. Sci.* 18, 2 (1979), 143–154. https://doi.org/10.1016/0022-0000(79)90044-8

[14] Danilo Carvalho, Hugo Carneiro, Felipe França, and Priscila Lima. 2013. B-bleaching: Agile overtraining avoidance in the wisard weightless neural classifier. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'13)*.

[15] Rituparna Choudhury, S. R. Ahamed, and Prithwijit Guha. 2020. Efficient hardware implementation of decision tree training accelerator. In *Proceedings of the IEEE International Symposium on Smart Electronic Systems (iSES'20) (Formerly iNiS)*. 212–215. https://doi.org/10.1109/iSES50453.2020.00055

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, 3123–3131.

[17] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or −1. Retrieved from https://arXiv:1602.02830

[18] Erika Covi, Elisa Donati, Xiangpeng Liang, David Kappel, Hadi Heidari, Melika Payvand, and Wei Wang. 2021. Adaptive extreme edge computing for wearable devices. *Front. Neurosci.* 15 (2021). https://doi.org/10.3389/fnins.2021.611300

[19] Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *Multiple Classifier Systems*. Springer, Berlin, 1–15.

[20] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. 2017. CirCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. 395–408.

[21] Rafatul Faria, Jan Kaiser, Kerem Y. Camsari, and Supriyo Datta. 2021. Hardware design for autonomous Bayesian networks. *Front. Comput. Neurosci.* 15 (2021). https://doi.org/10.3389/fncom.2021.584797

[22] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[23] Joshua Fromm, Meghan Cowan, Matthai Philipose, Luis Ceze, and Shwetak Patel. 2020. Riptide: Fast end-to-end binarized neural networks. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 379–389. Retrieved from https://proceedings.mlsys.org/paper/2020/file/2a79ea27c279e471f4d180b08d62b00a-Paper.pdf

[24] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual binarized neural network. In *Proceedings of the IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'18)*. 57–64. https://doi.org/10.1109/FCCM.2018.00018

[25] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. Retrieved from https://arxiv:2103.13630

[26] Song Han, Huizi Mao, and William Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*.

[27] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf

[28] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1 MB model size. Retrieved from https://arxiv.org/abs/1602.07360

[29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. Retrieved from https://arxiv.org/abs/1712.05877 https://doi.org/10.48550/ARXIV.1712.05877

[30] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[31] Andressa Kappaun, Karine Camargo, Fabio Rangel, Fabrício Firmino, Priscila Machado Vieira Lima, and Jonice Oliveira. 2016. Evaluating binary encoding techniques for WiSARD. In *Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS'16)*. 103–108. https://doi.org/10.1109/BRACIS.2016.029

[32] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

[33] Jong Hwan Ko, Duckhwan Kim, Taesik Na, Jaeha Kung, and Saibal Mukhopadhyay. 2017. Adaptive weight compression for memory-efficient neural networks. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'17)*. 199–204. https://doi.org/10.23919/DATE.2017.7926982

[34] Yuma Koizumi, Shoichiro Saito, Hisashi Uematsu, Noboru Harada, and Keisuke Imoto. 2019. ToyADMOS: A dataset of miniature-machine operating sounds for anomalous sound detection. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA'19)*. 313–317. https://doi.org/10.1109/WASPAA.2019.8937164

[35] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report 0. University of Toronto, Toronto, Ontario.

[36] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. Retrieved from http://yann.lecun.com/exdb/mnist/ http://yann.lecun.com/exdb/mnist/

[37] Walter D. Leon-Salas, Thomas Fischer, Xiaozhe Fan, Golsa Moayeri, and Shaocheng Luo. 2016. A 64×64 image energy harvesting configurable image sensor. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'16)*. 1914–1917. https://doi.org/10.1109/ISCAS.2016.7538947

[38] Fengfu Li and Bin Liu. 2016. Ternary weight networks. Retrieved from http://arxiv.org/abs/1605.04711

[39] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086. https://doi.org/10.1016/j.neucom.2017.09.046

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV'14)*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 740–755.

[41] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural networks with few multiplications. Retrieved from https://arxiv.org/abs/1510.03009

[42] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'18)*. ACM, New York, NY, 389–400. https://doi.org/10.1145/3210240.3210337

[43] Teresa Ludermir, Andre de Carvalho, Antônio Braga, and M. C. P. Souto. 1999. Weightless neural models: A review of current and past works. *Neural Comput. Surveys* 2 (Jan. 1999), 41–61.

[44] Leopoldo A. D. Lusquino Filho, Luiz F. R. Oliveira, Aluizio Lima Filho, Gabriel P. Guarisa, Lucca M. Felix, Priscila M. V. Lima, and Felipe M. G. França. 2020. Extending the weightless WiSARD classifier for regression. *Neurocomputing* 416 (2020), 280–291. https://doi.org/10.1016/j.neucom.2019.12.134

[45] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. 2018. Diffy: A Déjà vu-free differential deep neural network accelerator. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 134–147. https://doi.org/10.1109/MICRO.2018.00020

[46] Igor D. S. Miranda, Aman Arora, Zachary Susskind, Luis A. Q. Villon, Rafael F. Katopodis, Diego L. C. Dutra, Leandro S. De Araújo, Priscila M. V. Lima, Felipe M. G. França, Lizy K. John, and Mauricio Breternitz. 2022. LogicWiSARD: Memoryless synthesis of weightless neural networks. In *Proceedings of the IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP'22)*. 19–26. https://doi.org/10.1109/ASAP54787.2022.00014

[47] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. *PatDNN: Achieving Real-time DNN Execution on Mobile Devices with Pattern-based Weight Pruning*. ACM, New York, NY, 907–922. https://doi.org/10.1145/3373376.3378534

[48] Alessandro Pappalardo. 2021. Xilinx/brevitas. Retrieved from https://zenodo.org/records/8364211

[49] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA'17)*. 27–40. https://doi.org/10.1145/3079856.3080254

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, 8024–8035. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[51] Jorge Portilla, Gabriel Mujica, Jin-Shyan Lee, and Teresa Riesgo. 2019. The extreme edge at the bottom of the internet of things: A review. *IEEE Sensors J.* 19, 9 (2019), 3179–3190. https://doi.org/10.1109/JSEN.2019.2891911

[52] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-net: ImageNet classification using binary convolutional neural networks. Retrieved from http://arxiv.org/abs/1603.05279

[53] Daniel Richins, Dharmisha Doshi, Matthew Blackmore, Aswathy Thulaseedharan Nair, Neha Pathapati, Ankit Patel, Brainard Daguman, Daniel Dobrijalowski, Ramesh Illikkal, Kevin Long, David Zimmerman, and Vijay Janapa Reddi. 2021. AI tax: The hidden cost of AI data center applications. *ACM Trans. Comput. Syst.* 37, 1–4, Article 3 (Mar. 2021), 32 pages. https://doi.org/10.1145/3440689

[54] Richard Rohwer and Michal Morciniec. 1998. The theoretical and experimental status of the n-tuple classifier. *Neural Netw.* 11, 1 (Jan. 1998), 1–14.

[55] Leandro Santiago, Leticia Verona, Fabio Rangel, Fabricio Firmino, Daniel S. Menasché, Wouter Caarls, Mauricio Breternitz Jr, Sandip Kundu, Priscila M. V. Lima, and Felipe M. G. França. 2020. Weightless neural networks as memory segmented bloom filters. *Neurocomputing* 416 (2020), 292–304.

[56] Simone. 2020. TinyML or Arduino and STM32: Convolutional Neural Network (CNN) Example. Retrieved from https://eloquentarduino.github.io/2020/11/tinyml-on-arduino-and-stm32-cnn-convolutional-neural-network-example/

[57] Zachary Susskind, Aman Arora, Igor D. S. Miranda, Luis A. Q. Villon, Rafael F. Katopodis, Leandro S. De Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França, Mauricio Breternitz, and Lizy K. John. 2022. Weightless neural networks for efficient edge inference. In *Proceedings of the 31st International Conference on Parallel Architectures and Compilation Techniques (PACT'22)*. https://doi.org/10.1145/3559009.3569680

[58] Zachary Susskind, Alan T. L. Bacellar, Aman Arora, Luis A. Q. Villon, Renan Mendanha, Leandro S. De Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França, Igor D. S. Miranda, Mauricio Breternitz, and Lizy K. John. 2022. Pruning weightless neural networks. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'22)*. 37–42. https://doi.org/10.14428/esann/2022.ES2022-55

[59] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium on Field-programmable Gate Arrays (FPGA'17)*. ACM, New York, NY, 65–74. https://doi.org/10.1145/3020078.3021744

[60] Pete Warden. 2018. Speech Commands: A Dataset for Limited-vocabulary Speech Recognition, Retrieved from https://arxiv.org/abs/1804.03209 https://doi.org/10.48550/ARXIV.1804.03209

[61] Iuri Wickert and Felipe França. 2002. Validating an unsupervised weightless perceptron. In *Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02)*. 537–541. https://doi.org/10.1109/ICONIP.2002.1198114

[62] Pedro Xavier, Massimo De Gregorio, Felipe M. G. França, and Priscila M. V. Lima. 2020. Detection of elementary particles with the WiSARD n-tuple classifier. In *Proceedings of the 28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'20)*. 643–648. Retrieved from https://www.esann.org/sites/default/files/proceedings/2020/ES2020-170.pdf

[63] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin. 2019. Understanding straight-through estimator in training activation quantized neural nets. Retrieved from http://arxiv.org/abs/1903.05662

[64] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. 2017. Scalpel: Customizing DNN pruning to the underlying hardware parallelism. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA'17)*. 548–560. https://doi.org/10.1145/3079856.3080215

[65] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783723

[66] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 191–207.

[67] Xuda Zhou, Zidong Du, Qi Guo, Shaoli Liu, Chengsi Liu, Chao Wang, Xuehai Zhou, Ling Li, Tianshi Chen, and Yunji Chen. 2018. Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 15–28. https://doi.org/10.1109/MICRO.2018.00011

[68] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained ternary quantization. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*. OpenReview.net. Retrieved from https://openreview.net/forum?id=S1_pAu9xl

Received 14 April 2023; revised 22 August 2023; accepted 8 October 2023