Differentiable Weightless Neural Networks

Alan T. L. Bacellar [*1] **Zachary Susskind** [*2] **Mauricio Breternitz Jr.** [3] **Eugene John** [4] **Lizy K. John** [2]
Priscila M. V. Lima [1] **Felipe M. G. França** [5]

Abstract

We introduce the Differentiable Weightless Neural Network (DWN), a model based on interconnected lookup tables. Training of DWNs is enabled by a novel Extended Finite Difference technique for approximate differentiation of binary values. We propose Learnable Mapping, Learnable Reduction, and Spectral Regularization to further improve the accuracy and efficiency of these models. We evaluate DWNs in three edge computing contexts: (1) an FPGA-based hardware accelerator, where they demonstrate superior latency, throughput, energy efficiency, and model area compared to state-of-the-art solutions, (2) a low-power microcontroller, where they achieve preferable accuracy to XGBoost while subject to stringent memory constraints, and (3) ultra-low-cost chips, where they consistently outperform small models in both accuracy and projected hardware area. DWNs also compare favorably against leading approaches for tabular datasets, with higher average rank. Overall, our work positions DWNs as a pioneering solution for edge-compatible high-throughput neural networks. https://github.com/alanbacellar/DWN

1. Introduction

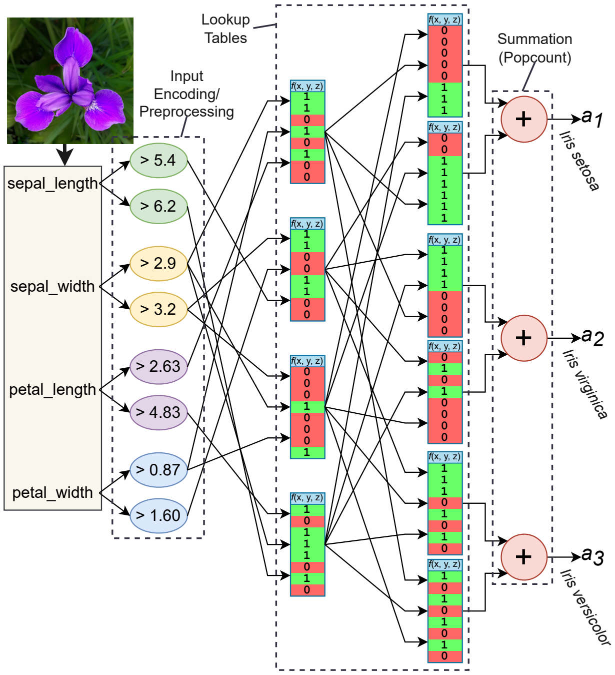

Figure 1. A very simple DWN for the Iris (Fisher, 1988) dataset, shown at inference time. DWNs perform computation using multiple layers of directly chained lookup tables (LUT-3s, in this example). Inputs are binarized using a unary "thermometer" encoding, formed into tuples, and concatenated to address the first layer of LUTs. Binary LUT outputs are used to form addresses for subsequent layers. Outputs from the final layer of LUTs are summed to derive activations for each output class. No arithmetic operations are performed between layers of LUTs.

Despite the rapid advancement of deep learning, optimizing computational efficiency, especially during inference, remains a critical challenge. Efforts to mitigate computational demands have led to innovations in model pruning (Dong et al., 2017a;b; Lin et al., 2018), quantization (Banner et al., 2018; Chmiel et al., 2021; Faghri et al., 2020), and sparse neural networks (Sung et al., 2021; Sun et al., 2021; Ma & Niu, 2018). However, these approaches do not fundamentally address the inherent cost of multiplication in neural networks. Consequently, multiplication-free architectures such as binary neural networks (BNNs) (Hubara et al., 2016), AddNets (Chen et al., 2021), and DeepShift (Elhoushi et al., 2021) have also been proposed, demonstrating impressive computational efficiency (Samragh et al., 2021; Qin et al., 2022; He & Xia, 2018).

Within the domain of multiplication-free models, weightless neural networks (WNNs) stand out as a distinct category. Diverging from the norm, WNNs forgo traditional weighted connections, opting instead for lookup tables (LUTs) with binary values to drive neural activity (Aleksander et al.,

[*] Equal contribution [1] Federal University of Rio de Janeiro, Brazil [2] The University of Texas at Austin, USA [3] ISCTE - Instituto Universitario de Lisboa, Lisbon, Portugal [4] The University of Texas at San Antonio, USA [5] Instituto de Telecomunicações, Porto, Portugal. Correspondence to: Alan T. L. Bacellar <alanbacellar@poli.ufrj.br>, Zachary Susskind <zsusskind@utexas.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

1984; 2009), where number of inputs to each LUT, n, is a hyperparameter. This enables WNNs to represent highly nonlinear behaviors with minimal arithmetic. However, a notable limitation of WNNs architectures is their restriction to single-layer models. This constraint is primarily due to the discrete structure of LUTs, which has historically made training complex multi-layer WNNs infeasible.

Despite this limitation, recent advancements have illuminated the potential of WNNs to achieve high efficiency in neural computation. The ULEEN model (Susskind et al., 2023) exemplifies this, showcasing a WNN's capabilities on an FPGA platform. Remarkably, ULEEN has outperformed BNNs in accuracy, energy efficiency, latency, memory consumption, and circuit area. This is particularly noteworthy considering that ULEEN operates with a single-layer structure, in contrast to the deep architecture of BNNs. This success not only underscores the inherent efficiency of WNNs but also implies the immense potential they could unlock if developed beyond their current single-layer constraints.

The recent Differentiable Logic Gate Networks (DiffLogicNet) (Petersen et al., 2022) proposes a gradient descent-based method for training *multi-layer* logic gate networks. The hypothesis space of DiffLogicNet's two-input binary nodes is exactly the same as a WNN with two-input LUTs (LUT-2s), showing that training multi-layer WNNs is theoretically possible. However, the cost of this method scales double-exponentially ($\mathcal{O}(2^{2^n})$) with the number of inputs to each LUT, requiring an astonishing *18.4 quintillion* parameters to represent a single LUT-6. The inability to train models with larger LUTs is detrimental for two main reasons: (1) the VC dimension of WNNs grows exponentially with the number of inputs to each LUT (Carneiro et al., 2019), making a WNN with fewer but larger LUTs a more capable learner than one with many small LUTs; (2) the ability to train LUTs with varying sizes facilitates hardware-software co-design, leading to more efficient models. For instance, a LUT-6-based WNN could be significantly more efficient on modern AMD/Xilinx FPGAs, which employ LUT-6s in their configurable logic blocks (CLBs), aligning WNN implementation more closely with FPGA architecture.

Furthermore, both WNNs and DiffLogicNet currently have three other vital limitations. First, they rely on pseudo-random connections between LUTs, which leaves the optimal arrangement of the neural network to chance. Second, they use population counts (popcounts) to determine activation values for each class, which incurs a large area overhead in hardware, with the popcount circuit in some cases being as large as the network itself. Finally, the binary nature of LUTs means that traditional DNN regularization techniques are ineffective; thus, there is a pressing need to develop specialized regularization techniques.

Recognizing the critical need for innovation in this area, our work introduces the **Differentiable Weightless Neural Network (DWN)** (Figure 1), tackling all of these limitations. This is achieved with a suite of innovative techniques:

- **Extended Finite Difference:** This technique enables efficient backpropagation through LUTs by approximate differentiation, allowing the development of multi-layer WNN architectures with bigger LUTs.
- **Learnable Mapping:** This novel layer allows DWNs to learn the connections between LUTs during training, moving beyond fixed or random setups in WNNs and DiffLogicNet, enhancing adaptability and efficiency without additional overhead during inference.
- **Learnable Reduction:** Tailored for tiny models, this approach replaces popcount with decreasing pyramidal LUT layers, leading to smaller circuit sizes.
- **Spectral Normalization:** A normalization technique specifically developed for LUTs in WNNs to improve model stability and avoid overfitting.

Our results demonstrate the DWN's versatility and efficacy in various scenarios:

1. **FPGA deployment**: DWNs outperform DiffLogicNet, fully-connected BNNs, and prior WNNs in latency, throughput, energy efficiency, and model area across all tested datasets. By aligning model LUTs with the native FPGA LUT size, DWNs achieve a geometric average 2522× improvement in energy-delay product versus the FINN BNN platform and a 63× improvement versus ULEEN, the current state-of-the-art for efficient WNNs.
2. **Constrained edge devices**: On a low-end microcontroller (the Elegoo Nano), our throughput-optimized implementations of DWNs achieve on average 1.2% higher accuracy than XGBoost with a 15% speedup. Our accuracy-optimized implementations achieve 5.4% improvement, at the cost of execution speed.
3. **Ultra-low-cost chips:** The DWN reduces circuit area by up to 42.8× compared to leading Tiny Classifier models (Iordanou et al., 2023), and up to 310× compared to DiffLogicNet.
4. **Tabular data**: DWN surpasses state-of-the-art models such as XGBoost and TabNets, achieving an average rank of 2.5 compared to 3.4 and 3.6 respectively.

2. Background & Related Work

2.1. Weightless Neural Networks

Weightless neural networks (WNNs) eschew traditional weighted connections in favor of a multiplication-free approach, using binary-valued lookup tables (LUTs), or "RAM nodes", to dictate neuronal activity. The connections between the input and these LUTs are randomly initialized and remain static. The absence of multiply-accumulate (MAC) operations facilitates the deployment of high-throughput models, which is particularly advantageous in edge computing environments. A notable recent work in this domain is ULEEN(Susskind et al., 2023), which enhanced WNNs by integrating gradient-descent training and utilizing straight-through estimators (Bengio et al., 2013) akin to those employed in BNNs, and outperformed the Xilinx FINN (Umuroglu et al., 2017) platform for BNN inference in terms of latency, memory usage, and energy efficiency in an FPGA implementation. A significant limitation of most WNN architectures is their confinement to single-layer models. While some prior works experimented with multi-layer weightless models (Al Alawi & Stonham, 1992; Filho et al., 1991), they relied on labyrinthine backward search strategies which were impractical for all but very simple datasets, and did not use gradient-based methods for optimization.

2.2. Thermometer Encoding

The method of encoding real-valued inputs into binary form is a critical aspect of WNNs, as the relationship between bit flips in the encoded input and corresponding changes in actual values is essential for effective learning (Kappaun et al., 2016). To address this, Thermometer Encoding was introduced (Carneiro et al., 2015), which uses a set of ordered thresholds to create a unary code (see Appendix B).

2.3. DiffLogicNet

DiffLogicNet (Petersen et al., 2022) proposed an approach to learning multi-layer networks exclusively composed of binary logic. In this model, an input binary vector is processed through multiple layers of binary logic nodes. These nodes are randomly connected, ultimately leading to a final summation determining the output class score. For training these networks via gradient descent, DiffLogicNet proposes a method where binary values are relaxed into probabilities. This is achieved by considering all possible binary logic functions (as detailed in Appendix A, Table 6), assigning a weight to each, and then applying a softmax function to create a probability distribution over these logic functions. See Appendix C for more details.

2.4. Other LUT-Based Neural Networks

Recently, other LUT-based neural networks such as LogicNets (Umuroglu et al., 2020a), PolyLUT (Andronic et al., 2023), and NeuraLUT (Andronic et al., 2024) have been proposed to improve DNN efficiency, rediscovering (Ferreira & França, 1997; Burattini et al., 2003). LogicNets suggest training sparse DNNs with binary activations and converting their neurons into LUTs by considering all possible input combinations. This aims to achieve efficient inference but fails to fully utilize the computational capacity of LUTs. An n-input LUT has a known VC-dimension of 2^n (Carneiro et al., 2019), while a DNN neuron with n inputs has a VC-dimension of $n+1$. Consequently, they effectively train a LUT with a reduced VC-dimension of $n+1$, leading to larger and less efficient models.

PolyLUTs tries to address this limitation by utilizing feature mappings in the sparse neuron inputs to learn more complex patterns. The most recent NeuraLUTs fit multiple neurons and layers with skip connections that receive the same input into a LUT, rather than a single neuron. However, both approaches still fall short of fully exploiting LUT computational capabilities, as we will demonstrate in the experiments section.

In contrast, our approach fully leverages the computational capabilities of LUTs by proposing a method to update and perform backpropagation with LUTs during the training phase, rather than merely using LUTs as a speedup mechanism for DNN neurons or layers.

3. Methodology

3.1. Extended Finite Difference

DiffLogicNet introduced a technique for learning binary logic with gradient descent and backpropagation that is readily applicable to WNNs employing two-input RAM nodes, as LUT-2s inherently represent binary logic. However, this technique is impractical for even slightly larger RAM nodes due to its $\mathcal{O}(2^{2^n})$ space and time complexities for a single LUT-n. Crucially, our approach reduces this to $\mathcal{O}(2^n)$, cutting the weights and computations needed to represent a LUT-6 from 18,446,744,073,709,551,616 to 64.

Finite Difference (FD) is a powerful tool for approximating derivatives, especially for functions with binary vector inputs. This approach is centered on evaluating the impact of minor input alterations on the input, specifically flipping a single bit in the binary case. For a given function $f : \{0,1\}^n \rightarrow \mathbb{R}^m$, the FD Δf is computed as $\Delta f(x)_j = f(^1_j x) - f(^0_j x)$ where x is the binary input vector, and $^1_j x$ and $^0_j x$ represents the vector x with its j-th bit set to 1 and 0, respectively. This formula shows how f's output changes when flipping the j-th bit in x, capturing the output's sensitivity to specific input bits.

The derivatives of a lookup table's addressing function can be approximated using FD. Consider $A : \mathbb{R}^{2^n} \times \{0,1\}^n \rightarrow \mathbb{R}$ as the addressing function that retrieves values from a lookup table $U \in \mathbb{R}^{2^n}$ using address $a \in \{0,1\}^n$. Define $\delta : \{0,1\}^n \rightarrow \{1,\ldots,2^n\}$ as the function converting a binary string to its integer representation $+1$. The partial derivatives of A can be approximated by finite differences:

$$\frac{\partial A}{\partial U_i}(U,a) = \begin{cases} 1, & \text{if } i = \delta(a) \\ 0, & \text{otherwise} \end{cases},$$

$$\frac{\partial A}{\partial a_j}(U,a) = A(U, {}^1_j a) - A(U, {}^0_j a)$$

where ${}^1_j a$ and ${}^0_j a$ signifies the address a with its j-th bit set to 1 and 0, respectively.

Using FD is our first proposed approach to ”Differentiable” WNNs (DWN). However, while FD approximates the partial derivatives of a lookup table’s addressing function, it only considers addresses within a Hamming distance of 1 from the targeted position. This limitation may hinder learning by ignoring optimal addressing positions beyond this proximity. For example, in a LUT-6 scenario, FD considers only 7 out of 64 positions, potentially neglecting more relevant ones.

To address this limitation, we introduce an **Extended Finite Difference (EFD)** method for more comprehensive derivative approximation. This technique considers variations in the addressed position relative to all possible positions, not just those one-bit apart:

$$\frac{\partial A}{\partial a_j}(U, a) = \sum_{k \in \{0,1\}^n} \frac{(-1)^{(1-k_j)} A(U, k)}{H(k, a, j) + 1}$$

where $H : \{0,1\}^n \times \{0,1\}^n \times \mathbb{N} \to \mathbb{N}$ calculates the Hamming distance between k and a, excluding the j-th bit. This formula integrates contributions from all lookup table positions, weighted by their relative distance (in terms of Hamming distance) to the address in use, with an added term for numerical stability. EFD provides a more holistic view, potentially capturing address shifts to more distant positions that conventional FD might miss.

3.2. Learnable Mapping

WNNs and DiffLogicNet both rely on pseudo-random mappings to route inputs, to LUTs in the former and between binary logic nodes in the latter. The specific choice of mapping can have a substantial impact on model accuracy, but is largely dependent on chance. In response, we introduce a new method that learns these connections through gradient descent-based optimization, without additional computational overhead during inference. This involves a weight matrix $W \in \mathbb{R}^{P \times Q}$ during training, where P is the input bit length or output bit count from the previous layer, and Q is the number of input connections in the next layer. Input selection for LUTs during the forward pass is based on the maximum weights in W, determined by $I(W, x)_i = x_{\text{argmax}(W[i,:])}$.

The backward pass involves calculating partial derivatives with respect to W and input x. For W, we use the product of the transformed input vector $(2x - 1)$ and the backpropagated gradient matrix G, where the transformation maps binary inputs to -1 and 1. The derivative is $\frac{\partial I}{\partial W} = ((2x-1)^\top \cdot G)$. For input x, the derivative is obtained by multiplying G with the transposed softmax of W over the first dimension, as $\frac{\partial}{\partial x} = G \cdot \text{softmax}_{\text{dim}=0}(W)^\top$. These gradients allow the learnable mapping (Figure 2) to iteratively refine the LUTs connections, optimizing DWN performance. During inference, the argmax of W remains constant since it is independent of the input. Consequently, the weight matrix W is discarded, and the LUTs’ connections become fixed, meaning there is no overhead from this technique at inference time.

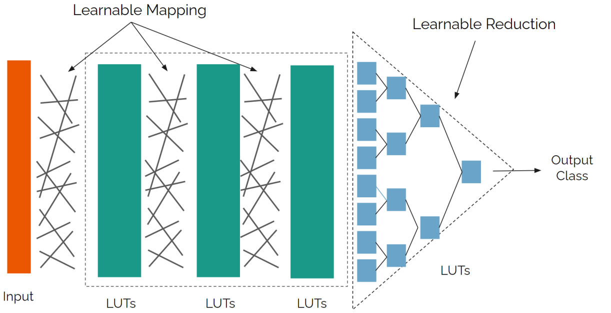

Figure 2. Learnable Mapping & Learnable Reduction in DWNs.

3.3. Learnable Reduction

When deploying tiny DWNs targetting ultra-low-cost chips, the popcount following the final LUT layer can constitute a large fraction of circuit area. This size disparity hinders efforts to reduce the overall circuit size. To address this, we propose a novel approach that deviates from the conventional method of determining a WNN’s output class; i.e., the argmax of popcounts from the last LUT layer’s feature vector. Instead, our method involves learning the reduction from the feature vector to the output class using layers of LUTs configured in a decreasing pyramidal architecture as in Figure 2. This technique enables the model to discover more efficient methods for determining the output class. It moves away from the reliance on a fixed structure of popcounts and argmax computations, resulting in smaller and more efficient circuit designs suitable for deployment.

3.4. Spectral Regularization

The inherent nonlinearity of WNNs is a double-edged sword: it contributes to their remarkable efficiency but also makes them very vulnerable to overfitting. Even very small DWNs may sometimes perfectly memorize their training data. Unfortunately, conventional DNN regularization techniques can not be applied directly to DWNs. For instance, since only the sign of a table entry is relevant for address computation, using L1 or L2 regularization to push entries towards 0 during training results in instability.

To address this issue, we propose a novel WNN-specific technique: *spectral regularization*. For an n-input pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$, we define the L2 spectral norm of f as:

$$\frac{1}{2^n} \left\| \left\{ \sum_{x \in \{0,1\}^n} f(x) \Big(\prod_{i \in S} (2x_i - 1) \Big) \;\Big|\; S \in [n] \right\} \right\|_2$$

Note that this is simply the L2 norm of the Fourier coefficients of f (O'Donnell, 2021). Additionally, since all terms except $f(x)$ are constant, we can precompute a coefficient matrix $\mathcal{C} \in \mathbb{R}^{2^n \times 2^n}$ to simplify evaluating the spectral norm at runtime. In particular, for a layer of u LUTs with n inputs each and data matrix $L \in [-1, 1]^{u \times 2^n}$, we express the spectral norm as:

$$\text{specnorm}(L) = \|L\mathcal{C}\|_2,\ \mathcal{C}_{ij} := \frac{1}{2^n} \prod_{a \in \{b \mid i_b = 1\}} (2j_a - 1)$$

The effect of spectral regularization is to increase the resiliency of the model to perturbations of single inputs. For instance, if an entry in a RAM node is never accessed during training, but all locations at a Hamming distance of 1 away hold the same value, then the unaccessed location should most likely share this value.

4. Experimental Evaluation

To demonstrate the effectiveness and versatility of DWNs, we evaluate them in several scenarios. First, we assess their performance on a custom hardware accelerator, implemented using a field-programmable gate array (FPGA), to demonstrate DWNs' extreme speed and energy efficiency in high-throughput edge computing applications. Next, we implement DWNs on an inexpensive off-the-shelf microcontroller, demonstrating that they can operate effectively on very limited hardware, and emphasizing their practicality in cost-sensitive embedded devices. We also consider the incorporation of DWNs into logic circuits, assessing their potential utility in ultra-low-cost chips.

Beyond hardware-focused evaluations, we also compare the accuracy of DWNs against state-of-the-art models for tabular data, with an emphasis on maximizing accuracy rather than minimizing model parameter size. Overall, while DWNs are chiefly engineered for edge inference applications, we aim to demonstrate their effectiveness in multiple contexts.

Binary Encoding: All datasets in the experimental evaluation are binarized using the Distributive Thermometer (Bacellar et al., 2022) for both DWN and DiffLogicNet. The sole exception is the DiffLogicNet model for the MNIST dataset, for which we use a threshold of 0, following the strategy outlined in their paper.

4.1. DWNs on FPGAs

FPGAs enable the rapid prototyping of hardware accelerators without the lead times associated with fabricating a custom IC. We deploy DWN models on the Xilinx Zynq Z-7045, an entry-level FPGA that was also used for the BNN-based FINN (Umuroglu et al., 2017) and WNN-based ULEEN (Susskind et al., 2023). We adopt the input data compression scheme used in ULEEN, which allows for more efficient loading of thermometer-encoded values. This is important due to the limited (112 bits per cycle) interface bandwidth of this FPGA. As in prior work, all designs are implemented at a clock speed of 200 MHz.

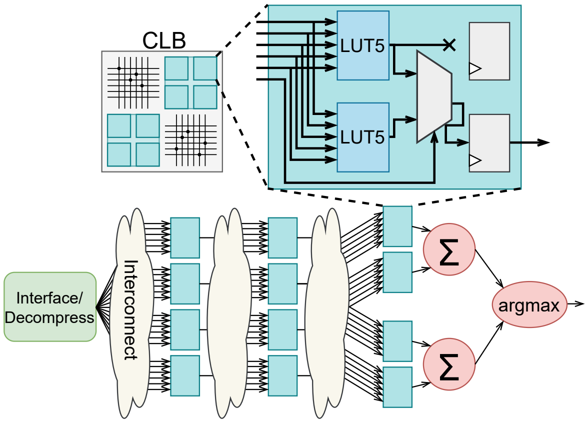

Figure 3. Implementation of a DWN on an FPGA. Each hardware LUT-6 (subdivided into two LUT-5s and a 2:1 MUX) can implement a six-input RAM node. Registers buffer LUT outputs.

Figure 3 gives a high-level overview of our accelerator design. The FPGA is largely composed of configurable logic blocks (CLBs), which are in turn composed of six-input lookup tables (LUT-6s), flip-flops, and miscellaneous interconnect and muxing logic (Xilinx, 2016).[1] The Z-7045 provides a total of 218,600 LUT-6s, each of which can represent a six-input RAM node. Hence, DWNs with n=6 make efficient use of readily available FPGA resources.

Table 1 compares the FPGA implementations of our DWN models against prior work. We include DWNs with both two and six-input RAM nodes. The original DiffLogicNet paper (Petersen et al., 2022) does not propose an FPGA implementation, but we observe that their model is structurally identical at inference time to DWNs with n=2 inputs per LUT, with all substantive differences restricted to the training process. Therefore, we can directly implement their models using our DWN RTL flow.

All datasets were chosen due to their use in prior work except for FashionMNIST (Xiao et al., 2017), which is identical in size to MNIST but intentionally more difficult. We directly reuse the MNIST model topologies, and thus hardware results, for FINN and ULEEN on this dataset.

Excluding CIFAR-10, our DWN models are smaller, faster, and more energy-efficient than prior work, with comparable or better accuracy. In particular, latency, throughput, energy per sample, and hardware area (in terms of FPGA LUTs) are improved by geometric averages of (20.7, 12.3, 121.6, 11.7)× respectively versus FINN, and

[1] As shown in Figure 3, a LUT-6 can also function as two LUT-5s, but this requires both LUTs to have identical inputs, and is not something we explore in this paper.

Table 1. Implementation results for DWNs and prior efficient inference models on a Z-7045 FPGA with input buffers and decompression implemented, with data being passed to the board using its I/O pins, which have a bandwidth of 112 bits per cycle, and the clock is limited to 200 MHz for a fair comparison to prior work. For results in out-of-context mode and with the clock is set to 250 MHz, see Appendix D. *Model could not be synthesized; hardware values are approximate. †ULEEN used augmented data for MNIST; we present MNIST results with and without augmentation.

Dataset	**Model**	**Test Accuracy%**	**Parameter Size (KiB)**	**Latency (ns)**	**Throughput (Samples/s)**	**Energy (nJ/Sample)**	**LUTs (1000s)**
MNIST	FINN	98.40	355.3	2440	1.56M	5445	83.0
	ULEEN†	98.46	262.0	940	4.05M	823	123.1
	DiffLogicNet *(xs)*	96.87	11.7	90	33.3M	17.2	9.6
	DiffLogicNet *(sm)**	97.62	23.4	*95*	*33.3M*	—	*19.1*
	DWN *(n=2; lg)*	98.27	**5.9**	135	25.0M	42.3	10.3
	DWN *(n=6; sm)*	97.80	11.7	**60**	**50.0M**	**2.5**	**2.1**
	DWN *(n=6; lg)*	98.31	23.4	125	25.0M	19.0	4.6
	DWN *(n=6; lg; +aug)*†	**98.77**	23.4	135	22.2M	21.6	4.6
FashionMNIST	FINN	84.36	40.8	2440	1.56M	5445	83.0
	ULEEN	87.86	262.0	940	4.05M	823	123.1
	DiffLogicNet	87.44	11.7	270	9.52M	119.8	11.4
	DWN *(n=2)*	**89.12**	**7.8**	255	**10.0M**	145.4	13.6
	DWN *(n=6)*	89.01	31.3	**250**	**10.0M**	**90.9**	**7.6**
KWS	FINN	70.60	324	7780	0.67M	5716	42.8
	ULEEN	70.34	101.0	390	10.0M	642	141.1
	DiffLogicNet*	64.18	23.4	*265*	*10.0M*	—	*20.3*
	DWN *(n=2)*	70.92	**2.9**	**235**	**10.5M**	79.2	6.8
	DWN *(n=6)*	**71.52**	12.5	**235**	**10.5M**	**42.3**	**4.8**
ToyADMOS/car	FINN	86.10	36.1	3520	1.57M	547	14.1
	ULEEN	86.33	16.6	340	11.1M	143	29.4
	DiffLogicNet	86.66	3.1	165	16.7M	23.2	3.9
	DWN *(n=2; sm)*	86.68	**0.9**	**115**	**25.0M**	7.6	2.2
	DWN *(n=6; sm)*	86.93	3.1	120	22.2M	**5.8**	**1.3**
	DWN *(n=6; lg)*	**89.03**	28.1	165	16.7M	45.4	6.2
CIFAR-10	FINN	**80.10**	183.1	283000	21.9K	150685	46.3
	ULEEN	54.21	1379	—	—	—	—
	DiffLogicNet*	57.29	250.0	*11510*	*87.5K*	—	*283.3*
	DWN *(n=2)**	57.51	**23.4**	*2200*	*468K*	—	*45.7*
	DWN *(n=6)*	57.42	62.5	**2190**	**468K**	**3972**	**16.7**

(3.3, 2.3, 19.0, 22.7)× respectively versus ULEEN, the prior state-of-the-art for efficient WNNs. Unlike the other architectures in Table 1, FINN supports convolution. This gives it vastly superior accuracy on the CIFAR-10 dataset, albeit at a hefty penalty to speed and energy efficiency.

Several models in Table 1 could not be implemented on our target FPGA (indicated by '$*$'). The primary cause of this was routing congestion: since it would be infeasibly expensive for FPGAs to implement a full crossbar interconnect, they instead have a finite number of wires to which they assign signals during synthesis. The irregular connectivity between layers in DiffLogicNet and DWNs with n=2 proved impossible to map to the FPGA's programmable interconnect in these cases. However, note that all DWNs with n=6 were successfully routed and implemented.

An interesting takeaway from these results is that the parameter sizes of DWN models are not necessarily good predictors of their hardware efficiency. For instance, the large MNIST model with n=2 has ≈1/4 the parameter size of the n=6 model, yet more than twice the area and energy consumption. Since our target FPGA uses LUT-6s natively, models with n=6 are inherently more efficient to implement. Although the synthesis tool can perform logic optimizations that map multiple DWN LUT-2s to a single FPGA LUT-6, this is not enough to offset the ≈4× larger number of RAM nodes needed to achieve the same accuracy with n=2.

Table 2. FPGA implementation results comparing DWNs with other LUT-based neural networks. Results for LogicNets, PolyLUT, and NeuraLUT are sourced from their respective papers, while new LogicNets results (*) are from their official GitHub (Umuroglu et al., 2020b). "Time/Sample" denotes the inference time per sample (calculated as 1/Throughput). Latency refers to the total end-to-end time of the model execution.

Dataset	Model	Accuracy	LUT	FF	DSP	BRAM	Fmax (MHz)	Time/Sample	Latency	Area×Lat. (LUT×ns)
MNIST	PolyLUT	96%	70673	4681	0	0	378	2.6ns	16.0ns	1.1e+06
	NeuraLUT	96%	54798	3757	0	0	431	2.3ns	12.0ns	6.6e+05
	DWN *(n=6; sm)*	97.1%	**692**	**422**	0	0	827	1.2ns	**2.4ns**	**1.6e+03**
	DWN *(n=6; md)*	97.8%	2055	1675	0	0	**873**	**1.1ns**	4.6ns	9.4e+03
	DWN *(n=6; md)*	97.9%	1413	1143	0	0	827	1.2ns	3.6ns	5.1e+03
	DWN *(n=6; lg)*	**98.3%**	4082	3385	0	0	827	1.2ns	6.0ns	2.4e+04
JSC	LogicNets	71.8%	37931	810	0	0	427	2.3ns	13.0ns	4.9e+05
	LogicNets *(sm)**	69.8%	244	270	0	0	1353	0.7ns	5.0ns	1.2e+03
	LogicNets *(lg)**	73.1%	36415	2790	0	0	390	2.6ns	6.0ns	2.2e+05
	PolyLUT	72%	12436	773	0	0	646	1.5ns	5.0ns	6.2e+04
	NeuraLUT	72%	4684	341	0	0	727	1.4ns	3.0ns	1.4e+04
	DWN *(n=6; sm)*	71.1%	**20**	**22**	0	0	**3030**	**0.3ns**	**0.6ns**	**1.3e+01**
	DWN *(n=6; sm)*	**74.0%**	110	72	0	0	1094	0.9ns	1.5ns	2.0e+02
JSC	PolyLUT	75%	236541	2775	0	0	235	4.3ns	21.0ns	5.0e+06
	NeuraLUT	75%	92357	4885	0	0	368	2.7ns	14.0ns	1.3e+06
	DWN *(n=6; md)*	**75.6%**	**720**	**457**	0	0	**827**	**1.2ns**	**3.6ns**	**2.6e+03**
JSC	hls4ml	76.2%	63251	4394	38	0	200	-	45.0ns	2.8e+06
	DWN *(n=6; lg)*	**76.3%**	**4972**	**3305**	0	0	**827**	**1.2ns**	**7.3ns**	**3.6e+04**

Comparison to Other LUT-Based NNs: We also compare DWNs against LogicNets, PolyLUT, and NeuraLUT, which convert models to LUTs for inference but do not use them during training. We follow their experimental methodology by targeting the xcvu9p-flgb2104-2-i FPGA, running synthesis with Vivado's Flow_PerfOptimized_high strategy in out-of-context mode, and using the highest possible clock frequency. We use the MNIST and Jet Substructure (JSC) datasets, as in the NeuraLUT paper, and compare our models against published results. This comparison is shown in Table 2.

DWNs achieves superior accuracy with significantly reduced LUT usage and area-delay product (ADP) compared to all other methods. Notably, on JSC ($\geq$75%), our *md* model achieves slightly higher accuracy then NeuraLUT with 43.1× fewer LUTs, 1.57× reduced latency, and 67.8× reduced ADP.

4.2. DWNs on Microcontrollers

While FPGAs can be extraordinarily fast and efficient, they are also expensive, specialized devices. We also consider the opposite extreme: low-cost commodity microcontrollers. The Elegoo Nano is a clone of the open-source Arduino Nano, built on the 8-bit ATmega328P, which at the time of writing retails for $1.52 in volume. The ATmega provides 2 KB of SRAM and 30 KB of flash memory and operates at a maximum frequency of 20 MHz. We can not expect performance comparable to an FPGA on such a limited platform. Our goal is instead to explore the speeds and accuracies of DWNs which can fit into this device's memory.

We use two strategies for implementing DWNs on the Nano. Our first approach uses aggressive bit packing to minimize memory usage, allowing us to fit more complex models on the device. For instance, the 64 entries of a LUT-6 can be packed in 8 bytes of memory, and the six indices for its inputs can be stored in 7.5 bytes by using 10-bit addresses (for more details on our bit-packing strategy, see Appendix E). However, this approach needs to perform bit manipulation to unpack data, which is fairly slow. Therefore, we also explore an implementation without bit-packing, which greatly increases inference speed but reduces the maximum size (and therefore accuracy) of feasible models.

XGBoost (Chen & Guestrin, 2016) is a widely-used tree boosting system notable for its ability to achieve high accuracies with tiny parameter sizes. This makes it a natural choice for deployment on microcontrollers. We use the MicroML (Salerno, 2022) library for XGBoost inference on the Nano and compare it against DWNs. To fit entire samples into SRAM, we quantize inputs to 8 bits. We did not observe a significant impact on accuracy from this transformation.

Table 3 compares DWNs against XGBoost on the Nano. We present results for the datasets from Table 1, excluding CIFAR-10, which was too large to fit even after quantization. We also include three additional *tabular* datasets, a category on which XGBoost excels. All XGBoost models have a maximum tree depth of 3, with forest size maximized to fill the board's memory. We found that this gave better results than the default max depth of 6, which required extremely narrow forests in order to fit. The parameter sizes of these XGBoost models are generally quite small, but their complex control flow means that their source code footprints are large, even after compiler optimizations (with `-Os`).

Table 3. Model accuracies and throughputs (in inferences per second) for DWNs and XGBoost on the Elegoo Nano, a low-end microcontroller. All models are as large as possible within the constraints of the device's memory. We consider an accuracy-optimized DWN implementation which uses bit-packing, and a throughput-optimized implementation which does not.

Dataset	DWN				XGBoost	
	Acc-Optim		Thrpt-Optim			
	Acc.	Thrpt	Acc.	Thrpt	Acc.	Thrpt
MNIST	97.9%	16.5/s	94.5%	108/s	90.2%	81/s
F-MNIST	88.2%	16.4/s	84.1%	95/s	83.2%	81/s
KWS	69.6%	16.4/s	53.6%	109/s	51.0%	103/s
ToyADMOS	88.7%	17.7/s	86.1%	112/s	85.9%	94/s
phoneme	89.5%	17.8/s	87.5%	298/s	86.5%	265/s
skin-seg	99.8%	17.5/s	99.7%	298/s	99.4%	268/s
higgs	72.4%	17.1/s	71.2%	254/s	71.8%	245/s

Our bit-packed DWN implementation is consistently more accurate than XGBoost, by an average of 5.4%, and particularly excels on non-tabular multi-class datasets such as MNIST and KWS. However, it is also 8.3× slower on average. Our unpacked implementation is 15% faster than XGBoost and still 1.2% more accurate on average, but is less accurate on one dataset (higgs). Overall, DWNs are good models for low-end microcontrollers when accuracy is the most important consideration, but may not always be the best option when high throughput is also needed.

4.3. DWNs for Ultra-Low-Cost Chips

To assess the viability of DWNs for ultra-low-cost chip implementations, we analyze their performance in terms of accuracy and NAND2 equivalent circuit area, comparing them with Tiny Classifiers (Iordanou et al., 2024), a SOTA work for ultra-low-cost small models, and DiffLogicNet. The datasets for this analysis are those shared between the Tiny Classifiers (see Appendix H) and AutoGluon (to be used in the next subsection) studies, providing a consistent basis for comparison. We also adhere to their data-splitting methods, using 80% of the data for the training set and 20% for the testing set.

Our DWN, utilizing the Learnable Reduction technique with LUT-2s, is designed to inherently learn two input binary logic, which directly correlates to logic gate formation in a logic circuit. The NAND2 equivalent size of our model is calculated by converting each LUT-2 into its NAND2 equivalent (e.g., a LUT-2 representing an OR operation equates to 3 NAND gates). For DiffLogicNet, we adopt a similar approach, translating the converged binary logic nodes into their NAND2 equivalents, plus the additional NAND2 equivalent size required for each class output popcount (Appendix F), as per their model architecture. Notably, our DWN model, due to Learnable Reduction, does not incur this additional computational cost. For Tiny Classifiers we utilize the results reported in their paper.

Our results, presented in Table 4, highlight DWN's exceptional balance of efficiency and accuracy across a range of datasets. Notably, DWN consistently outperforms Tiny Classifiers and DiffLogicNet in accuracy, while also showcasing a remarkable reduction in model size. In the 'skin-seg' dataset, DWN achieves 98.9% accuracy with a model size of only 88 NAND, compared to 93% with 300 NAND for Tiny Classifiers and 98.2% with 610 NAND for DiffLogicNet, demonstrating reductions of approximately 3.4× and 6.9×, respectively. Similarly, in the 'jasmine' dataset, DWN reaches 80.6% accuracy with just 51 NAND gates, while Tiny Classifiers and DiffLogicNet achieve 72% with 300 NAND and 76.7% with 1816 NAND, respectively, indicating reductions of 5.9× and 35.6×.

These findings demonstrate DWN's potential in ASIC and Ultra-Low-Cost Chip implementations, offering a blend of high accuracy and compact circuit design.

Table 4. Comparison of the accuracy and NAND2 equivalent circuit size for DWN, Tiny Classifiers, and DiffLogicNet across various datasets. This comparison highlights DWN's increased accuracy and significantly smaller circuit sizes, underscoring its effectiveness for ultra-low-cost chip implementations.

Dataset	DWN		DiffLogicNet		Tiny Class.	
	Acc.	NAND	Acc.	NAND	Acc.	NAND
phoneme	**85.7%**	**168**	83.2%	836	79%	300
skin-seg	**98.9%**	**88**	98.2%	610	93%	300
higgs	**67.8%**	**94**	67.5%	658	62%	300
australian	**88.5%**	**7**	87.7%	379	85%	300
nomao	**93.5%**	**87**	93.5%	4955	80%	300
segment	**99.4%**	**71**	99.4%	610	95%	300
miniboone	**90.1%**	**60**	90.1%	619	82%	300
christine	**70.6%**	**53**	68.3%	16432	59%	300
jasmine	**80.6%**	**51**	76.7%	1816	72%	300
sylvine	**92.2%**	**44**	85.7%	501	89%	300
blood	**78.4%**	**49**	78.4%	365	63%	300

Table 5. A comprehensive evaluation of the accuracy of our proposed DWN against prominent state-of-the-art models in handling tabular data. Key metrics include the average ranking (Avg Rank), indicating each model's relative rank across datasets, and the average L1 norm (Avg Dist) from the top accuracy per dataset, assessing how closely each model approaches the best performance, with lower values indicating superior performance for both metrics.

Dataset	DWN	DiffLogicNet	AutoGluon XGBoost	AutoGluon CatBoost	AutoGluon LightGBM	AutoGluon TabNN	AutoGluon NNFastAITab	Google TabNet
phoneme	**0.895**	0.891	0.886	0.868	0.873	0.884	**0.895**	0.844
skin-seg	**1.000**	0.999	**1.000**	**1.000**	0.999	0.999	**1.000**	0.999
higgs	0.727	0.711	0.728	0.730	**0.743**	0.731	0.727	0.726
australian	**0.901**	0.862	0.870	0.862	0.870	0.870	0.855	0.529
nomao	0.966	0.966	**0.973**	0.963	0.964	0.972	**0.973**	0.959
segment	**0.998**	**0.998**	0.989	**0.998**	0.996	0.996	0.994	0.857
miniboone	0.946	0.944	0.948	**0.952**	0.860	0.948	0.947	0.717
christine	0.736	0.710	0.750	0.728	0.734	0.754	**0.756**	0.547
jasmine	**0.816**	**0.816**	0.812	**0.816**	0.781	0.806	0.809	0.759
sylvine	0.952	0.945	0.944	0.921	0.941	0.951	**0.953**	0.921
blood	0.780	0.760	0.773	**0.787**	**0.787**	0.753	0.753	0.740
Avg Rank	**2.5**	4.5	3.4	3.6	4.5	3.6	3.5	7.5
Avg L1	**0.005**	0.016	0.009	0.014	0.021	0.010	0.010	0.107

4.4. DWNs on Tabular Data

In this subsection, we explore benchmarking DWNs against a range of prominent state-of-the-art models in the field of tabular data processing. This includes a thorough evaluation alongside the AutoGluon suite (Erickson et al., 2020) — encompassing models like XGBoost, CatBoost, LightGBM, TabNN, and NNFastAITab — Google's TabNet, and DiffLogicNet. These benchmarks are crucial in demonstrating the efficacy and competitiveness of DWN in handling structured data, a key requirement for numerous real-world applications. Additionally, they show that DWN's efficient inference does not come at the cost of accuracy, highlighting DWN's remarkable ability to learn from tabular data.

To ensure a fair comparison, all models in the AutoGluon suite were trained under 'best quality' configurations, which involve extended training times and fine-tuning for optimal accuracy. DWN model sizes were restricted to match those of the other models, ensuring that any performance gains were not simply due to a larger model size. The datasets and train-test splits are the same as in the previous subsection.

For training our DWN, we adopted an AutoML-like approach: setting the number of layers to 3 for small datasets and 5 for larger ones. The number of LUTs per layer was adjusted to align with the final model sizes of XGBoost, thereby maintaining a comparable model size. See Appendix J for more model configuration details.

Results are detailed in Table 5. Key metrics in our analysis are the Average Rank and Average L1 norm. Average Rank is calculated by ranking models on each dataset according to accuracy and then averaging these ranks across all datasets. This metric provides a comparative view of each model's performance relative to others. The Average L1 norm, on the other hand, measures the average L1 distance of each model's accuracy from the highest-achieving model on each dataset. This offers insight into how closely each model approaches the best possible accuracy.

As shown in Table 5, DWN achieves an impressive average rank of 2.5 and an average L1 norm of 0.005, indicating its leading performance in accuracy among the compared models. Notably, it surpasses renowned models such as XGBoost, CatBoost, and TabNN, which have respective average rankings of 3.4, 3.6, and 3.6, and average L1 norms of 0.009, 0.014, and 0.010.

5. Conclusion

In this paper, we have introduced the Differentiable Weightless Neural Network (DWN), which features the novel Extended Finite Difference technique along with three significant enhancements: Learnable Mapping, Learnable Reduction, and Spectral Regularization. Our results underscore the versatility and efficiency of our approach, demonstrating up to 135× reduction in energy costs in FPGA implementations compared to BNNs and DiffLogicNet, up to 9% higher accuracy in deployments on constrained devices, and culminating in up to 42.8× reduction in circuit area for ultra-low-cost chip implementations. Moreover, DWNs have achieved an impressive average ranking of 2.5 in processing tabular datasets, outperforming state-of-the-art models such as XGBoost and TabNets, which have average rankings of 3.4 and 3.6 respectively.

These significant contributions pave the way for a plethora of future work. In the context of FPGAs, extending our approach to include CNN and Transformer architectures could lead to significant advancements in deep learning. This is especially relevant considering the resource-intensive nature of current models used in computer vision and language processing. Furthermore, immediate future work on FPGAs could involve applying our models to domains where high-throughput or low latency are critical, such as anomaly detection, data streaming, and control systems. In the realm of ultra-low-cost chip implementations, DWNs hold significant promise for smart applications such as heart monitoring and seizure detection. These chips necessitate extremely compact architectures and operation on very low power, making them ideal for our approach. Furthermore, in the field of tabular data, the application of Differentiable Weightless Neural Networks (DWNs) could be transformative. Areas such as predictive analytics, financial modeling, and healthcare data analysis could greatly benefit from the efficiency and accuracy of DWNs, particularly in managing large and complex datasets. Overall, DWNs show great potential for a range of edge and tabular applications.

Acknowledgements

This research was supported in part by Semiconductor Research Corporation (SRC) Task 3148.001, National Science Foundation (NSF) Grant #2326894, NVIDIA Applied Research Accelerator Program Grant, and by FCT/MCTES through national funds and, when applicable, co-funded by EU funds under the project UIDB 50008/2020, and by Next Generation EU through PRR Project Route 25 (C645463824-00000063). Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies.

Impact Statement

This paper presents work to advance the field of Machine Learning, particularly on low-power, low-latency, and low-cost devices. There are many potential societal consequences of our work, including making tiny intelligent models with high accuracies for edge applications such as healthcare monitoring. Works that seek to lower the adoption cost and improve the energy efficiency of machine learning, such as this one, additionally have the inherent potential to improve global equality of access to technology, particularly as it impacts communities with limited financial resources and unreliable access to electricity.

References

Image Segmentation. UCI Machine Learning Repository, 1990. DOI: https://doi.org/10.24432/C5GP4N.

Al Alawi, R. and Stonham, T. A training strategy and functionality analysis of digital multi-layer neural networks. *Journal of Intelligent Systems*, 2, 12 1992. doi: 10.1515/JISYS.1992.2.1-4.53.

Aleksander, I., Thomas, W., and Bowden, P. WISARD·a radical step forward in image recognition. *Sensor Review*, 4(3):120–124, 1984. ISSN 0260-2288. doi: 10.1108/eb007637. URL https://www.emerald.com/insight/content/doi/10.1108/eb007637/full/html.

Aleksander, I., De Gregorio, M., França, F., Lima, P., and Morton, H. A brief introduction to weightless neural systems. In *17th European Symposium on Artificial Neural Networks (ESANN)*, pp. 299–305, 04 2009.

Alinat, P. Periodic progress report 4. Technical report, ROARS Project ESPRIT II- Number 5516, 1993.

Andronic et al., M. Polylut: learning piecewise polynomials for ultra-low latency fpga lut-based inference. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pp. 60–68. IEEE, 2023.

Andronic et al., M. Neuralut: Hiding neural network density in boolean synthesizable functions. *arXiv preprint arXiv:2403.00849*, 2024.

Bacellar, A., Susskind, Z., Villon, L., Miranda, I., Santiago, L., Dutra, D., Jr, M., JOHN, L., Lima, P., and França, F. Distributive thermometer: A new unary encoding for weightless neural networks. pp. 31–36, 01 2022. doi: 10.14428/esann/2022.ES2022-94.

Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., Montino, P., Kanter, D., Ahmed, S., Pau, D., et al. Mlperf tiny benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

Banner, R., Hubara, I., Hoffer, E., and Soudry, D. Scalable methods for 8-bit training of neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Bengio, Y., Léonard, N., and Courville, A. C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

Bhatt, R. and Dhall, A. Skin Segmentation. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5T30C.

Burattini, E., De Gregorio, M., Ferreira, V. M. G., and França, F. M. G. Nsp: a neuro–symbolic processor. In Mira, J. and Álvarez, J. R. (eds.), *Artificial Neural Nets Problem Solving Methods*, pp. 9–16, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-44869-3.

Candillier, L. and Lemaire, V. Nomao. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C53G79.

Carneiro, H., França, F., and Lima, P. Multilingual part-of-speech tagging with weightless neural networks. *Neural Networks*, 66, 03 2015. doi: 10.1016/j.neunet.2015.02.012.

Carneiro, H. C. C., Pedreira, C. E., França, F. M. G., and Lima, P. M. V. The exact VC dimension of the WiSARD n-tuple classifier. *Neural Computation*, 31:176–207, 2019. URL https://api.semanticscholar.org/CorpusID:53715711.

Chen, H., Wang, Y., Xu, C., Shi, B., Xu, C., Tian, Q., and Xu, C. Addernet: Do we really need multiplications in deep learning?, 2021.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

Chmiel, B., Ben-Uri, L., Shkolnik, M., Hoffer, E., Banner, R., and Soudry, D. Neural gradients are near-lognormal: improved quantized and sparse training. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=EoFNy62JGd.

CodaLab. ChaLearn automatic machine learning challenge (AutoML), 2016. URL https://competitions.codalab.org/competitions/2321.

Deng, L. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Dong, X., Chen, S., and Pan, S. J. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *CoRR*, abs/1705.07565, 2017a. URL http://arxiv.org/abs/1705.07565.

Dong, X., Chen, S., and Pan, S. J. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *CoRR*, abs/1705.07565, 2017b. URL http://arxiv.org/abs/1705.07565.

Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B., Ngadiuba, J., Pierini, M., Rivera, R., Tran, N., et al. Fast inference of deep neural networks in fpgas for particle physics. *Journal of instrumentation*, 13(07): P07027, 2018.

Elhoushi, M., Chen, Z., Shafiq, F., Tian, Y. H., and Li, J. Y. Deepshift: Towards multiplication-less neural networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 2359–2368, 2021. doi: 10.1109/CVPRW53098.2021.00268.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Faghri, F., Tabrizian, I., Markov, I., Alistarh, D., Roy, D. M., and Ramezani-Kebrya, A. Adaptive gradient quantization for data-parallel sgd. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3174–3185. Curran Associates, Inc., 2020.

Ferreira, V. M. G. and França, F. M. G. Weightless implementations of weighted neural networks. In *Anais do IV Simpósio Brasileiro de Redes Neurais*, 1997.

Filho, E., Fairhurst, M., and Bisset, D. Adaptive pattern recognition using goal seeking neurons. *Pattern Recognition Letters*, 12(3):131–138, 1991. ISSN 0167-8655. doi: https://doi.org/10.1016/0167-8655(91)90040-S. URL https://www.sciencedirect.com/science/article/pii/016786559190040S.

Fisher, R. A. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

Google, S. Skywater open source pdk, 2020. URL https://github.com/google/skywater-pdk.

He, H. and Xia, R. Joint binary neural network for multi-label learning with applications to emotion classification. In *Natural Language Processing and Chinese Computing: 7th CCF International Conference, NLPCC 2018, Hohhot, China, August 26–30, 2018, Proceedings, Part I 7*, pp. 250–259. Springer, 2018.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Iordanou, K., Atkinson, T., Ozer, E., Kufel, J., Biggs, J., Brown, G., and Lujan, M. Tiny classifier circuits: Evolving accelerators for tabular data, 2023.

Iordanou, K., Atkinson, T., Ozer, E., Kufel, J., Aligada, G., Biggs, J., Brown, G., and Luján, M. Low-cost and efficient prediction hardware for tabular data using tiny classifier circuits. *Nature Electronics*, April 2024.

Kappaun, A., Camargo, K., Rangel, F., Faria, F., Lima, P., and Oliveira, J. Evaluating binary encoding techniques for WiSARD. In *BRACIS*, pp. 103–108, 10 2016. doi: 10.1109/BRACIS.2016.029.

Koizumi, Y., Saito, S., Uematsu, H., Harada, N., and Imoto, K. Toyadmos: A dataset of miniature-machine operating sounds for anomalous sound detection. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 313–317, 2019. doi: 10.1109/WASPAA.2019.8937164.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

Lin, S., Ji, R., Li, Y., Wu, Y., Huang, F., and Zhang, B. Accelerating convolutional networks via global & dynamic filter pruning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pp. 2425–2432. AAAI Press, 2018. ISBN 9780999241127.

Ma, R. and Niu, L. A survey of sparse-learning methods for deep neural networks. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pp. 647–650, 2018. doi: 10.1109/WI.2018.00-20.

O'Donnell, R. Analysis of boolean functions, 2021.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Deep differentiable logic gate networks. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 2006–2018. Curran Associates, Inc., 2022.

Qin, H., Ma, X., Ding, Y., Li, X., Zhang, Y., Tian, Y., Ma, Z., Luo, J., and Liu, X. Bifsmn: Binary neural network for keyword spotting. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 4346–4352, 2022.

Quinlan, R. Statlog (Australian Credit Approval). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C59012.

Roe, B. MiniBooNE particle identification. UCI Machine Learning Repository, 2010. DOI: https://doi.org/10.24432/C5QC87.

Salerno, S. micromlgen 1.1.28, 2022. URL https://pypi.org/project/micromlgen/.

Samragh, M., Hussain, S., Zhang, X., Huang, K., and Koushanfar, F. On the application of binary neural networks in oblivious inference. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4625–4634, 2021. doi: 10.1109/CVPRW53098.2021.00521.

Sun, W., Zhou, A., Stuijk, S., Wijnhoven, R., Nelson, A. O., Li, h., and Corporaal, H. Dominosearch: Find layer-wise fine-grained n:m sparse schemes from dense neural networks. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 20721–20732. Curran Associates, Inc., 2021.

Sung, Y.-L., Nair, V., and Raffel, C. A. Training neural networks with fixed sparse masks. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 24193–24205. Curran Associates, Inc., 2021.

Susskind, Z., Arora, A., Miranda, I. D. S., Bacellar, A. T. L., Villon, L. A. Q., Katopodis, R. F., de Araújo, L. S., Dutra, D. L. C., Lima, P. M. V., França, F. M. G., Breternitz Jr., M., and John, L. K. Uleen: A novel architecture for ultra-low-energy edge neural networks. *ACM Trans. Archit. Code Optim.*, 20(4), dec 2023. ISSN 1544-3566. doi: 10.1145/3629522. URL https://doi.org/10.1145/3629522.

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., and Vissers, K. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pp. 65–74, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450343541. doi: 10.1145/3020078.3021744. URL https://doi.org/10.1145/3020078.3021744.

Umuroglu, Y., Akhauri, Y., Fraser, N. J., and Blott, M. Logicnets: Co-designed neural networks and circuits for extreme-throughput applications. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 291–297. IEEE, 2020a.

Umuroglu, Y., Akhauri, Y., Fraser, N. J., and Blott, M. Xillinx logicnets github updated results. 2020b. URL https://github.com/Xilinx/logicnets/tree/master/examples/jet_substructure.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition, 2018. URL https://arxiv.org/abs/1804.03209.

Whiteson, D. HIGGS. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5V312.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Xilinx. 7 series FPGAs configurable logic block user guide, 2016. URL https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB.

Yeh, I.-C. Blood Transfusion Service Center. UCI Machine Learning Repository, 2008. DOI: https://doi.org/10.24432/C5GS39.

A. Table of Boolean Functions

Table 6. There are a total of 2^{2^n} logic functions for n logic variables. This table shows the 16 different functions for 2 variables. For 3 variables, there will be 256 logic operators. This list shows the functions in logic function and real-valued forms. The four columns on the right show the 4 values for each function for the input variable values of 00, 01, 10 and 11.

ID	Operator	Real-valued	00	01	10	11
1	False	0	0	0	0	0
2	$A \wedge B$	$A \cdot B$	0	0	0	1
3	$\neg(A \Rightarrow B)$	$A - AB$	0	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
14	$A \Rightarrow B$	$1 - A + AB$	1	1	0	1
15	$\neg(A \wedge B)$	$1 - AB$	1	1	1	0
16	True	1	1	1	1	1

B. Formal Definition of Thermometer Encoding

For a real-valued input $q \in \mathbb{R}$, let z be the number of bits in the encoding, and $\vec{t} \in \mathbb{R}^z$ be the ordered Thermometer thresholds ($t_{i+1} > t_i$). The Thermometer Encoding $T \in \{0,1\}^z$ of q is given by:

$$T(q) = (1_{q>t_1}, 1_{q>t_2}, \ldots, 1_{q>t_z})$$

Here, $1_{q>t_i}$ is an indicator function, returning 1 if $q > t_i$ and 0 otherwise.

C. DiffLogicNet's Binary Logic Node

Formally, for two input bit probabilities $A, B \in [0,1]$, the output of each binary logic node during training is computed as:

$$\sum_{i=1}^{16} \frac{e^{w_i}}{\sum_j e^{w_j}} \cdot f_i(A, B)$$

Here, f_i denotes the i-th real-valued binary operator among the set of 16 distinct operators (a comprehensive list of which can be found in Appendix A, Table 6). During inference, the network is discretized, with each binary logic node adopting the binary operator that has the highest associated weight.

D. Out-of-Context Results

In our primary Table 1, we reported FPGA results where input buffers and decompression were implemented, assuming data is streamed to the board using I/O pins (112 bits per cycle on the Zynq Z-7045 FPGA) and limited the clock speed to 200 MHz. These design choices were made to enable a fair comparison to prior work, particularly FINN, which adheres to similar constraints.

However, we noticed that other papers report results assuming data is already available on the board without accounting for streaming or decompression overheads (i.e., operating in out-of-context mode). To provide a comprehensive comparison, we present these out-of-context results in the Table 7, following the methodology of other papers by setting the clock speed to the board's maximum of 250 MHz.

In this configuration, our model achieves an effective throughput of 250 million images per second. This is because our design is fully pipelined, taking only 4 ns per image during inference as the model is compact enough to process one image per clock cycle.

E. Bit-Packed Microcontroller Implementation Details

Figure 4 shows the memory layout for our bit-packed DWN inference model on the Elegoo Nano microcontroller. Flash memory is used to store program code, RAM node mapping information, and the contents of the RAM nodes themselves.

Table 7. Implementation results for DWNs on a Z-7045 FPGA compiled in out-of-context mode with the clock set to an optimal 250 MHz (4 ns per cycle), the maximum frequency of the board.

Dataset	Model	Test Accuracy%	Parameter Size (KiB)	Time/Sample	Throughput (Samples/s)	LUTs (1000s)
MNIST	DWN *(n=2; lg)*	98.27	**5.9**	4ns	250 M	6.5
	DWN *(n=6; sm)*	97.80	11.7	4ns	250 M	**2.1**
	DWN *(n=6; lg)*	98.31	23.4	4ns	250 M	4.1
	DWN *(n=6; lg; +aug)*	**98.77**	23.4	4ns	250 M	4.1
FashionMNIST	DWN *(n=2)*	**89.12**	**7.8**	4ns	250 M	8.1
	DWN *(n=6)*	89.01	31.3	4ns	250 M	**6.2**
KWS	DWN *(n=2)*	70.92	**2.9**	4ns	250 M	3.3
	DWN *(n=6)*	**71.52**	12.5	4ns	250 M	**3.3**
ToyADMOS/car	DWN *(n=2; sm)*	86.68	**0.9**	4ns	250 M	1.0
	DWN *(n=6; sm)*	86.93	3.1	4ns	250 M	**0.8**
	DWN *(n=6; lg)*	**89.03**	28.1	4ns	250 M	5.5
CIFAR-10	DWN *(n=2)*	57.51	**23.4**	4ns	250 M	45.7
	DWN *(n=6)*	57.42	62.5	4ns	250 M	**16.7**

Mapping arrays specify the indices of each input to each RAM node in a layer. These indices can be 8, 10, 12, or 16 bits, depending on the number of unique inputs to the layer. In the case of an irregular bit width (10 or 12 bits), the map is further subdivided into `map_lo`, which stores the low 8 bits, and `map_hi`, which stores the remaining high bits. RAM node entries are packed, with 8 entries per byte. For simplicity, this figure only shows mapping and data arrays for a single layer.

SRAM is divided into two large scratchpads, which layers alternate between. Layer i of a model reads its input activations from scratchpad $(i-1) \bmod 2$ and writes its outputs to scratchpad $i \bmod 2$.

Bit-packing incurs a heavy runtime overhead. For instance, a single 6-input RAM node on a layer with 12-bit input mappings must (1) read the low byte of an input index from `map_lo`, (2) read the high byte from `map_hi`, (3) shift and mask the data read from `map_hi` to isolate the relevant 4 bits, (4) shift and OR these bits with the byte read from `map_lo`, (5) isolate and read the indicated bit from the input scratchpad, (6) repeat steps 1-5 five additional times, (7) construct an address from the six bits thus retrieved, (8) isolate and read the addressed bit from the data array, and lastly (9) set the result bit in the output scratchpad.

Bit-packing reduces the data footprint of models in flash memory by $\sim$4.5$\times$ and in SRAM by 8$\times$, which significantly increases the complexity of the models we can implement. Our unpacked microcontroller inference model is 9.5$\times$ faster than this packed implementation on average, but also 4.2% less accurate.

All models on the Nano (for both DWNs and XGBoost) communicate with a host PC over a 1 Mbps serial connection, which they use to receive input samples and send back predictions. This was chosen to minimize data movement overhead while maintaining reliability; while a 2 Mbps connection is theoretically possible, we found that it was unstable in practice.

F. Estimating NAND2 Equivalents for Popcount Circuits

To obtain NAND2 equivalent areas of DiffLogicNet models, we need a way to estimate the area of an N-input popcount. Let's first consider the number of half adders and full adders that are required. As shown in Figure 5, we can construct a popcount circuit by first passing trios of 1-bit inputs into full adders, which results in $\lceil \frac{N}{3} \rceil$ partial sums which are each 2 bits wide. We then pass these signals into a reduction tree of adders. The ith layer of this tree requires approximately $\frac{n}{3}2^{-i}$ multi-bit adders which are each i+1 bits wide. These multi-bit adders are in turn composed of one half adder and i full adders. Therefore, including $\frac{N}{3}$ full adders from the first layer, this popcount circuit requires approximately

$$\frac{N}{3} + \sum_{i=1}^{\log_2(N/3)} i\left(\frac{N}{3}2^{-i}\right) = N - \log_2\left(\frac{N}{3}\right) - 2$$

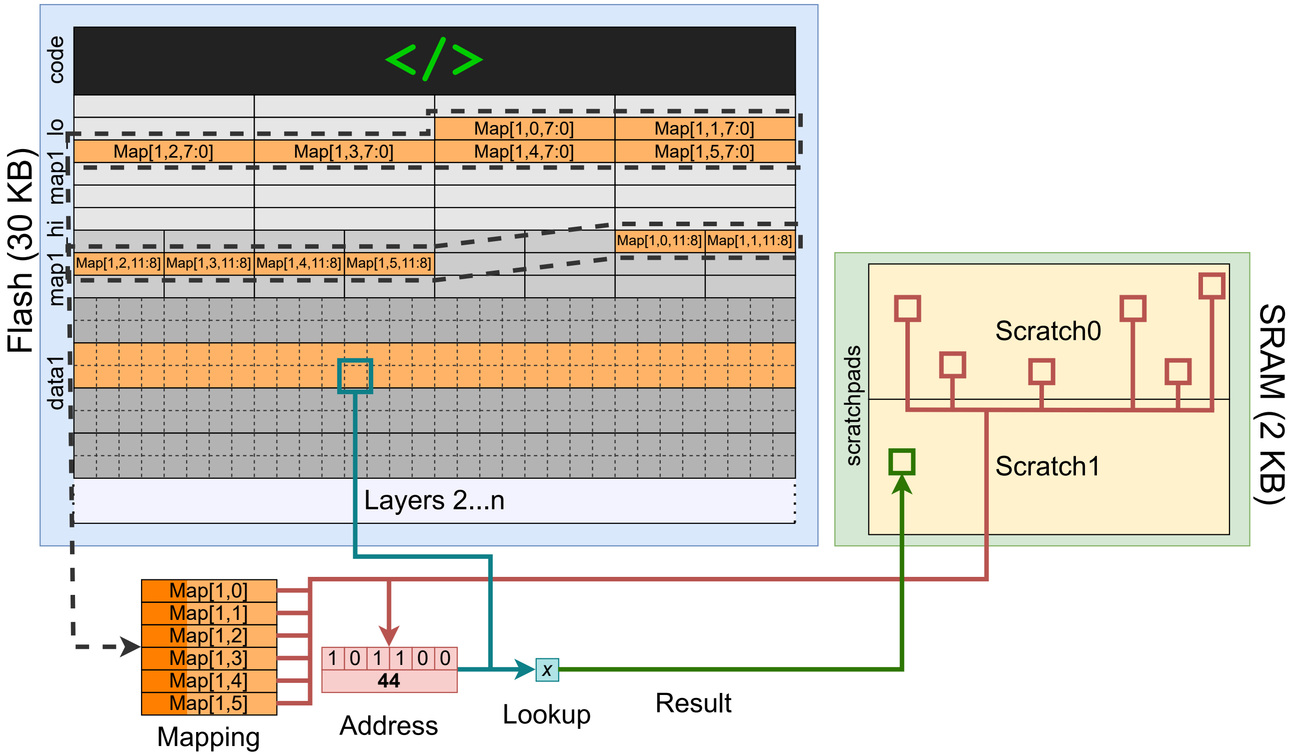

Figure 4. An overview of the data layout of a DWN model implemented on the Elegoo Nano. This microcontroller has very limited resources, which necessitates careful memory management.

full adders, and

$$\sum_{i=1}^{\log_2(N/3)} \left(\frac{N}{3}2^{-i}\right) = \frac{N}{3} - 1$$

half adders.

A naive implementation of this circuit requires 5 NAND2 gates per half adder and 9 gates per full adder. However, we can obtain a lower and more accurate estimate by looking at the transistor-level optimizations performed in standard cell libraries. For instance, the open-source SkyWater SKY130 PDK (Google, 2020) implements the smallest NAND2 gate using 4 transistors, a half-adder using 14 transistors, and a full-adder using 28 transistors. Therefore, we approximate a half adder as 3.5 NAND2 gates and a full adder as 7 NAND2 gates.

Our final NAND2 equivalent estimate for an N-input popcount is:

$$7\left(N - \log_2\left(\frac{N}{3}\right) - 2\right) + 3.5\left(\frac{N}{3} - 1\right) \approx 8.167N - 7\log_2(N) - 6.405$$

G. Expanded Details on Tabular Data Comparisons for Tiny Models

Table 8 provides additional details for our comparison between DWNs and tiny tabular models. For each DiffLogicNet model, we separately break out the NAND2 equivalent areas for the logic and the popcount circuit. For the Tiny Classifier models, we include additional, smaller configurations with 50, 100, or 200 NAND gates from their paper.

H. List of Datasets Used

See Table 9.

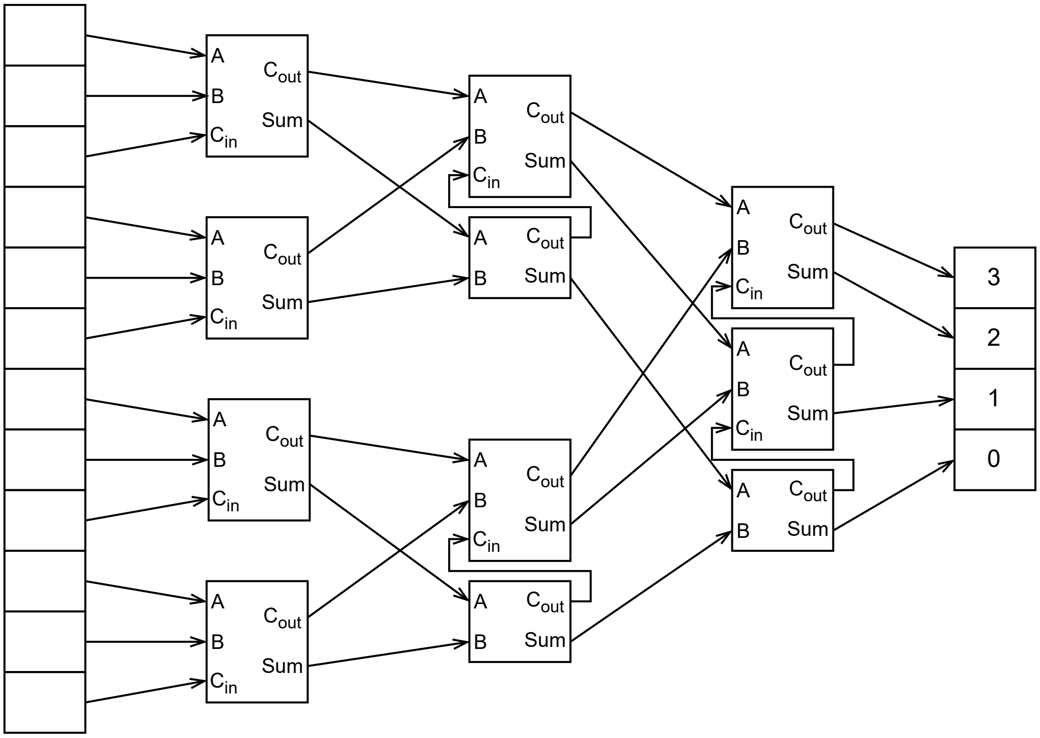

Figure 5. A 12:4 popcount tree composed of 8 full adders and 3 half adders.

Table 8. Expanded Table 4 detailing DiffLogicNets' Logic NAND count and Total NAND count, alongside additional model sizes for Tiny Classifiers.

Dataset	**DWN**		**DiffLogicNet**			**Tiny Classifier (Iordanou et al., 2023)**			
	Acc.	**NAND**	**Acc.**	**Logic NAND**	**Total NAND**	**50 NAND Acc.**	**100 NAND Acc.**	**200 NAND Acc.**	**300 NAND Acc.**
phoneme	0.857	168	0.832	394	836	0.700	0.700	0.780	0.790
skin-seg	0.989	88	0.982	327	610	0.900	0.890	0.950	0.930
higgs	0.678	94	0.675	232	658	0.600	0.520	0.620	0.620
australian	0.885	7	0.877	175	379	0.800	0.830	0.850	0.850
nomao	0.935	87	0.935	2612	4955	0.800	0.750	0.780	0.800
segment	0.994	71	0.994	248	610	0.500	0.790	0.790	0.950
miniboone	0.901	60	0.901	249	619	0.730	0.750	0.780	0.820
christine	0.706	53	0.683	3147	16432	0.550	0.540	0.580	0.590
jasmine	0.806	51	0.767	689	1816	0.500	0.710	0.710	0.720
sylvine	0.922	44	0.857	202	501	0.890	0.890	0.890	0.890
blood	0.784	49	0.784	82	365	0.600	0.600	0.620	0.630

Table 9. List of all datasets used in this paper.

Dataset Name	Source	Notes
Iris	(Fisher, 1988)	Extremely simple dataset; used in Figure 1 for illustrative purposes.
MNIST	(Deng, 2012)	
FashionMNIST	(Xiao et al., 2017)	Designed to be the same size as MNIST but more difficult.
KWS	(Warden, 2018)	Subset of dataset; part of MLPerf Tiny (Banbury et al., 2021).
ToyADMOS/car	(Koizumi et al., 2019)	Subset of dataset; part of MLPerf Tiny (Banbury et al., 2021).
CIFAR10	(Krizhevsky & Hinton, 2009)	Part of MLPerf Tiny (Banbury et al., 2021).
JSC	(Duarte et al., 2018)	
phoneme	(Alinat, 1993)	
skin-seg	(Bhatt & Dhall, 2012)	
higgs	(Whiteson, 2014)	
australian	(Quinlan)	
nomao	(Candillier & Lemaire, 2012)	
segment	(mis, 1990)	Binarized version of original dataset
miniboone	(Roe, 2010)	
christine	(CodaLab, 2016)	Synthetic dataset, created for competition.
jasmine	(CodaLab, 2016)	Synthetic dataset, created for competition.
sylvine	(CodaLab, 2016)	Synthetic dataset, created for competition.
blood	(Yeh, 2008)	

Table 10. Ablation Studies comparing Finite Difference (FD) (a minimal DWN) to Extended Finite Difference (EFD) and Learnable Mapping (LM). *These results are from models with only one layer, so input derivatives are not needed if LM is not employed. Therefore, FD and EFD behave the same when LM is not used.

Dataset	FD (Minimal DWN)	+ EFD	+ LM	+ EFD + LM
MNIST	96.15%	96.59%	98.30%	98.31%
FashionMNIST	85.74%	86.88%	87.94%	89.01%
KWS	52.33%*	52.33%*	70.24%	71.52%
ToyADMOS/car	87.73%	88.02%	88.52%	89.03%
CIFAR-10	48.37%*	48.37%*	55.36%	57.42%

I. Ablation Studies

See Tables 10, 11, 12 and 13.

J. Model Configurations

See Tables 14, 15, 16 and 17.

z indicates the number of thermometer bits per feature utilized in the binary encoding, **tau** the softmax temperature utilized after the popcount during training, and **BS** the batch size.

Table 11. Gains in accuracy observed by each method in Table 10. *These results are from models with only one layer, so input derivatives are not needed if LM is not employed. Therefore, FD and EFD behave the same when LM is not used.

Dataset	**+ EFD**	**+ LM**	**+ EFD + LM**
MNIST	0.44%	2.15%	2.16%
FashionMNIST	1.14%	2.20%	3.27%
KWS	0.00%*	17.91%	19.19%
ToyADMOS/car	0.29%	0.79%	1.30%
CIFAR-10	0.00%*	6.99%	9.05%

Table 12. Ablation Studies comparing a DWN with and without Spectral Normalization (SN)

Dataset	**DWN**	**+ SN**	**Change**
MNIST	97.82%	97.88%	+0.06%
FashionMNIST	87.10%	88.16%	+1.06%
KWS	67.17%	69.60%	+2.43%
ToyADMOS/car	88.04%	88.68%	+0.64%
phoneme	89.55%	89.50%	-0.05%
skin-seg	99.65%	99.83%	+0.18%
higgs	68.51%	72.42%	+3.91%

Table 13. Ablation Studies comparing the NAND2 gate count of DWN for Ultra-Low-Cost Chips with and without Learnable Reduction (LR). The results for DWN without LR are divided into three columns: the NAND count used by the LUTs (Logic), the NAND count used by the Popcount, and the total (Logic + Popcount). The DWN + LR column shows directly the total model size as no Popcount is needed when employing LR.

Dataset		**DWN**		**DWN + LR**	**Circuit**
	Logic	**Popcount**	**Total**		**Reduction**
phoneme	146	111	257	168	1.53x
skin-seg	189	157	346	88	3.93x
higgs	90	141	231	94	2.46x
australian	12	24	36	7	5.14x
nomao	84	126	210	87	2.41x
segment	49	111	160	71	2.25x
miniboone	55	66	121	60	2.02x
christine	45	52	97	53	1.83x
jasmine	48	66	114	51	2.24x
sylvine	63	96	159	44	3.61x
blood	60	96	156	49	3.18x

Table 14. DWN model configurations for Table 1 and Table 2. All models were trained for a total of 100 Epochs.

Dataset	Model	z	Layers	tau	BS	Learning Rate
MNIST	DWN (n=2; lg)	3	2x 6000	1/0.071	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6; sm)	1	1000, 500	1/0.245	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6; lg)	3	2000, 1000	1/0.173	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6; lg + aug)	3	2000, 1000	1/0.173	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
FashionMNIST	DWN (n=2)	7	2x 8000	1/0.061	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6)	7	2x 2000	1/0.122	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
KWS	DWN (n=2)	8	1x 3000	1/0.1	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6)	8	1x 1600	1/0.1	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
ToyADMOS/car	DWN (n=2; sm)	2	2x 900	1/0.1	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6; sm)	2	1x 400	1/0.1	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6; lg)	3	2x 1800	1/0.058	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
CIFAR-10	DWN (n=2)	10	2x 24000	1/0.03	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n=6)	10	8000	1/0.03	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
JSC	DWN (n-6; sm)	200	1x 50	1/0.3	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n-6; md)	200	1x 1000	1/0.3	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)
	DWN (n-6; lg)	200	1x 3000	1/0.03	100	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)

Table 15. DWN model configurations for Table 3.

Model	Dataset	z	Layers	tau	BS	Learning Rate	Epochs
Bit-Packed	MNIST	3	1000, 500	1/0.077	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	FashionMNIST	3	1000, 500	1/0.077	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	KWS	3	1000, 500	1/0.077	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	ToyADMOS/car	3	1000, 500	1/0.077	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	phoneme	128	1000, 500	1/0.077	256	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	skin-seg	128	1000, 500	1/0.077	256	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	higgs	128	1000, 500	1/0.077	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
Unpacked	MNIST	32	220, 110	1/0.165	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	FashionMNIST	32	220, 110	1/0.165	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	KWS	32	220, 110	1/0.165	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	ToyADMOS	32	220, 110	1/0.165	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	phoneme	255	80, 40	1/0.274	256	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	skin-seg	255	80, 40	1/0.274	256	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100
	higgs	255	90, 90	1/0.183	128	1e-2(30), 1e-3(30), 1e-4(30), 1e-5(10)	100

Table 16. DWN model configurations for Table 4.

Dataset	z	Layers	tau	BS	Learning Rate	Epochs
phoneme	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
skin-seg	200	64, 32, 16, 8, 4, 2, 1	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
higgs	200	64, 32, 16, 8, 4, 2, 1	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
australian	200	4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
nomao	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
segment	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
miniboone	200	64, 32, 16, 8, 4, 2, 1	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
christine	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
jasmine	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
sylvine	200	64, 32, 16, 8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
blood	200	8, 4, 2, 1	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200

Table 17. DWN model configurations for Table 5.

Dataset	z	Layers	tau	BS	Learning Rate	Epochs
phoneme	200	3x 21000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
skin-seg	200	5x 10000	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
higgs	200	5x 22000	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
australian	200	3x 12000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
nomao	200	3x 24000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
segment	200	3x 7500	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
miniboone	200	5x 22000	1/0.001	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
christine	20	3x 26000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
jasmine	200	3x 20000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
sylvine	200	3x 17000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200
blood	200	3x 13000	1/0.03	32	1e-2 (80), 1e-3 (80), 1e-4 (40)	200