Soon Filter: Advancing Tiny Neural Architectures for High Throughput Edge Inference

Alan T. L. Bacellar[1], Zachary Susskind[2], Maurício Breternitz Jr.[3],
Lizy K. John[2], Felipe M. G. França[4] and Priscila M. V. Lima[1]
1 - Universidade Federal do Rio de Janeiro, Brazil; 2 - The University of Texas at Austin, USA
3 - Instituto Universitário de Lisboa ISCTE-IUL ISTAR, Portugal; 4 - Instituto de Telecomunicações, Portugal

***Abstract*—As Deep Neural Networks become more complex and computationally demanding, efficient models for inference at the edge, particularly multiplication-free ones, have gained significant attention. The Ultra Low-Energy Edge Neural Network (ULEEN) is a notable architecture optimized for high throughput edge designs. ULEEN uniquely employs Bloom Filters with binary values to compute neuron activation, boasting better efficiency metrics than Binary Neural Networks (BNNs). This work uncovers a gradient back-propagation bottleneck within ULEEN's Bloom filters and introduces a simplified version of it as a solution: the "Soon Filter". Both theoretically and empirically, we demonstrate that our approach improves gradient back-propagation efficiency. Tests on MLPerf Tiny, MNIST and various UCI datasets reveal that our method surpasses ULEEN, BNN, and DeepShift. Notably, with MLPerf KWS (Key Word Spotting) dataset, we achieve 69.6% accuracy with only 101KiB, while ULEEN, BNN and DeepShift achieve only 67.4%, 55.9%, and 24.9% respectively. Remarkably, we also achieve 67.7% accuracy with only 50KiB, resulting in a 2x model size reduction compared to ULEEN while maintaining similar accuracy (+0.3%). This results underscores the promising potential of our solution for efficient inference at the edge in applications that rely on high throughput architectures.**

I. Introduction

In recent years, the field of artificial intelligence has witnessed a remarkable transformation due to the advent of Deep Neural Networks (DNNs). These powerful models have pushed the boundaries of what AI can achieve, making significant strides in areas like computer vision, speech recognition, and natural language processing [1, 2, 3, 4, 5]. However, this performance comes at a cost, with increasingly complex models demanding higher computational resources. As these models grow in size and complexity, the computational overhead for training and inference becomes substantial, posing a challenge for their deployment in resource-constrained environments.

Inference at the edge, particularly in the growing realm of the Internet of Things (IoT), demands ultra-efficient models. The rapid expansion of the IoT ecosystem has seen an explosion of interconnected devices, from smart thermostats to wearable health monitors. These devices often operate under stringent energy and latency constraints, making it imperative to deploy models that can deliver competitive accuracy without taxing the limited resources available [6].

Several optimization techniques, including including Pruning [7, 8], Weight Quantization [9, 10] and Sparse Neural Networks [11, 12], have been developed, indicating promise in boosting computational efficiency. Nevertheless, while they help in reducing memory consumption, they don't alleviate the inherent computational expenses tied to multiplication operations during the inference stage.

In response, recent research has shifted towards the development of multiplication-free architectures. Binary Neural Networks (BNNs) [13] stand out as a prominent example, in which both weights and activations are quantized to binary values. This paradigm enables the substitution of multiplication operations with XOR gates, substantially reducing memory and computational overheads [14]. Concurrently, Deep Shift Networks [15] have been introduced, leveraging shift operations in lieu of multiplications, offering a novel viewpoint on model computational efficiency [16]. As a result, multiplication-free models have been deployed in numerous applications [17, 18, 19, 20].

In recent advancements in multiplication-free designs, Susskind et al. (2023) introduced the Ultra Low Energy Edge Neural Network (ULEEN) aimed at applications that rely on high throughput architectures. By utilizing binary-valued bloom filters, along with the use of Straight Through Estimators (STE) and a continuous relaxation of these bloom filters for training, this approach has demonstrated notable improvements in latency, memory consumption, and energy efficiency compared to BNNs, setting new state-of-the-art results and paving the way for the implementation of highly energy-efficient and high throughput models at the edge.

In this work, we theoretically and empirically demonstrate a gradient back-propagation bottleneck present in ULEEN, caused by the use of continuous relaxation of Bloom filters, which hinders learning. Drawing insights and inspiration from ResNet [21, 22] — which emphasizes the benefits of tweaking network architecture to enhance gradient flow — we introduce our solution: the "Soon filter". Both theoretically and empirically, we demonstrate that that our proposed solution ensures more seamless gradient back-propagation to filter locations. Consequently, we set new state-of-the-art benchmarks for multiplication-free high throughput models.

II. Background

In this section, we provide an overview of the underlying mechanisms of Bloom Filters and ULEEN, which will form the foundation for our methodology.

This work is funded 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).

Bloom Filter The Bloom Filter [23] is a space-efficient data structure that probabilistically determines membership in a set. It's comprised of a bit array $F \in \{0,1\}^L$ of fixed size L, and K hash functions, denote h_k for each $k \in \{1, \ldots, K\}$. Each h_k function maps an element to one of the L positions in the array, indicating its possible presence or absence. Initially, every position in the array is set to 0. When we add an element e to the Bloom Filter, it is hashed, and the corresponding bits in the array change to 1. This is represented as:

$$\forall k \in \{1, \ldots, K\} : F_{h_k(e)} \leftarrow 1$$

To verify an element's membership, it's hashed using the same functions, and we check the relevant bits in the array:

$$\theta(e) = \bigwedge_{k=1}^{K} F_{h_k(e)}$$

Here, $\theta(e)$ is the Bloom Filter's output function. If any checked bit is 0, the element isn't in the set. If all are 1, the element might be in the set, but we can't be certain because of potential hash collisions. Hence, the Bloom Filter guarantees true negatives but may produce false positives.

Straight Through-Estimator The Straight Through-Estimator (STE) [24, 25] is a widely adopted technique for learning binary variables using gradient descent and is commonly employed in BNNs. The STE functions as the sign function during the forward pass and as the derivative of the hardtanh function during the backward pass. This approach allows gradients to pass through the function, which would otherwise be impossible since the derivative of the sign function is infinite at zero and is zero everywhere else. The STE can be expressed as:

$$STE(w) = \begin{cases} 1, & \text{if } w > 0 \\ 0, & \text{otherwise} \end{cases} \qquad \frac{\partial STE(w)}{\partial w} = \begin{cases} 1, & \text{if } |w| < 1 \\ 0, & \text{otherwise} \end{cases}$$

During training, the binary variable to be learned is treated as a real-valued parameter passing through the STE. During inference, as this variable becomes a constant, it is binarized and the STE is removed.

Weightless Neural Networks Weightless Neural Networks (WNNs) are a type of neural model that achieve a multiplication-free characteristic by completely eliminating the use of weights. Instead, they utilize lookup tables with binary values to determine neural activity, allowing for the deployment of high-throughput models at the edge. Consequently, WNNs have been used in many applications requiring real-time performance [26, 27, 28]. A drawback of using lookup tables (LUTs) is that the memory requirement grows exponentially with the number of inputs, making the deployment of larger models unfeasible. To address this, [29] proposed substituting LUTs with Bloom Filters, demonstrating that this allows for more efficient and smaller models with negligible changes in accuracy. Additionally, [30] showed that using H3 hash functions [31] made the filters extremely efficient for deployment at the edge. Recently, [32] further improved upon this work by incorporating gradient-descent training into WNNs using Straight Through Estimators, commonly employed in Binary Neural Networks (BNNs). They also developed a hardware implementation for WNNs named ULEEN, demonstrating its superiority over BNNs in terms of latency, memory usage, and energy efficiency.

ULEEN ULEEN serves as a classification model designed to distinguish C distinct classes from a binary input $x \in \{0,1\}^{nN}$. Each class is represented by a discriminator D_c where $c \in \{1, 2, \ldots C\}$. Each discriminator is composed of N Bloom Filters of length of L. Every Bloom Filter in the discriminator processes a unique subset of n bits from the input x, selected pseudo-randomly. Let $F_{c,i} \in \{0,1\}^L$ denote the bit array of the i-th Bloom Filter of the of the c-th discriminator. Let $\delta_{c,i,k} \in \{1, 2, \ldots L\}$ represent the hash value of the binary subset produced by the k-th hash function, of the i bloom filter, of the c-th discriminator. The output of discriminator D_c is expressed by:

$$s_c(x) = \sum_{i=1}^{N} \bigwedge_{k=1}^{K} F_{c,i,\delta_{c,i,k}}$$

where $s_c : \{0,1\}^{Nn} \rightarrow \{1, 2, \ldots, N\}$ indicates the response of the c-th discriminator. The discriminator yielding the highest response determines the model's output class. Refer to Figure [fig:uleen]1 for a graphical representation.

To learn the Bloom Filter's binary values via gradient descent, ULEEN replaces the Bloom Filter AND aggregation with a continuous relaxation—specifically, the min function—during training. The Straight Through-Estimator is employed to learn the filter array's binary values. Specifically, during training, the output of discriminator D_c is expressed as:

$$s_c(x) = \sum_{i=1}^{N} \min_{k=1}^{K} STE(F_{c,i,\delta_{c,i,k}})$$

III. SoonFilter

In this section, we elucidate the inspiration behind our proposed methodology. We theoretically identify a gradient bottleneck in ULEEN's Bloom Filter and introduce our solution: The Soon Filter.

Inspiration ResNets [33] have shown that altering model architecture to enhance gradient flow during back-propagation can lead to significant improvements in model performance. Let f_i be the i-th layer function in a DNN model. Given a model with P consecutive layers, the partial derivative of the final layer f_P in the with respect to the i-th layer f_i can be expressed as:

$$\frac{\partial f_P}{\partial f_i} = \prod_{p=i}^{P-1} \frac{\partial f_{p+1}}{\partial f_p}$$

The authors observed that gradients in earlier layers (those closer to the input) can vanish due to the multiplication of numerous values less than 1 (the partial derivative of one layer with respect to its predecessor) in the chain rule during back-propagation. To mitigate this, they introduced a skip connection between layers. In their new design, assuming a skip connection

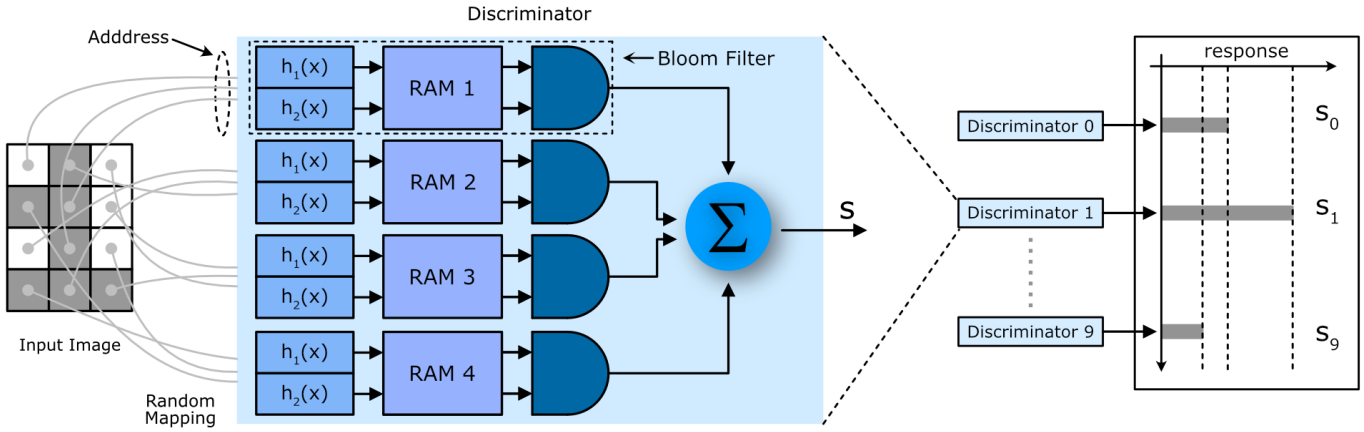


Fig. 1: ULEEN doing digit recognition. Each output class has a discriminator. The input image has digit 1 and the discriminator corresponding to digit 1 has the highest response here.

between every 2 layers, the partial derivative of f_P with respect to f_i became:

$$\frac{\partial f_P}{\partial f_i} = \sum_{u=0}^{P/2-1} \prod_{p=i+2u}^{P-1} \frac{\partial f_{p+1}}{\partial f_p}$$

This architectural alteration ensured that gradients effectively propagated to earlier layers, as elucidated in [22]. This simple yet impactful technique has been integrated into many contemporary DNN architectures [34, 35, 36].

Taking cues from this approach of tailoring model architectures to optimize gradient flow, we propose replacing the Bloom Filter in ULEEN with our novel Soon Filter.

Gradient Bottleneck Consider $A : \mathbb{R}^K \to \mathbb{R}$, an arbitrary continuous aggregation function. For a loss function $\mathcal{L}$, the partial derivative with respect to an arbitrary content of the ULEEN's Bloom Filter, denoted as $F_{c,i,j}$ (denoting the content at the j-th position of the i-th Bloom Filter of the c-th discriminator) can be written as:

$$\frac{\partial \mathcal{L}}{\partial F_{c,i,j}} = \frac{\partial \mathcal{L}}{\partial A} \frac{\partial A}{\partial STE} \frac{\partial STE}{\partial F_{c,i,j}}$$

We will demonstrate that the middle term of this expression can form a bottleneck, potentially hindering filter positions from updating.

For gradient descent training on ULEEN, a continuous relaxation of the Bloom Filter's AND aggregation function, specifically the min function, is employed. Let $\vec{x} \in \mathbb{R}^K$ be an input representing the output of the STE at the positions accessed by the K hash functions. For the min function, we have:

$$A(\vec{x}) = \min_{k=1}^{K} x_k \qquad \frac{\partial A}{\partial x_k}(\vec{x}) = \begin{cases} 1, & \text{if } x_k = A(\vec{x}) \\ 0, & \text{otherwise} \end{cases}$$

This function only permits gradients to flow to inputs identical to its own value, effectively blocking gradients to filter positions with values exceeding its own.

Another potential continuous relaxation for the AND aggregation function is the product operation: However, its derivative reveals it to be even more obstructive to gradient back-propagation, resulting in an even greater bottleneck than the min function:

$$A(\vec{x}) = \prod_{k=1}^{K} x_k \qquad \frac{\partial A}{\partial x_k}(\vec{x}) = \prod_{\substack{l=1 \\ l \neq k}}^{K} x_l$$

In this scenario, the gradient propagates to all filter positions only when the STE for every accessed position yields a 1. If a single STE of a filter position produces a 0, just that particular position is updated by the gradient. When two or more outputs register as zero, none of the filter positions receive a gradient update, effectively halting gradient back-propagation entirely.

Soon Filter To overcome this gradient bottleneck issue, we do not constrain ourselves to continuous relaxations of the AND aggregation function. Instead, we choose an aggregation function without a gradient bottleneck:

$$\frac{\partial A}{\partial x_k}(\vec{x}) = 1$$

This function corresponds to the sum operation:

$$A(\vec{x}) = \sum_{k=1}^{K} x_k$$

Using the sum operation as the aggregation function alters the filter's characteristics. In this modified filter, the number of false positives rises. While in the Bloom Filter, indexing both a 0 and a 1 position would produce a true negative, this new filter will output a 1. Due to this filter's tendency to output a result prematurely, we named it the "Soon Filter" — a simplified version of the Bloom Filter that has a higher rate of false positives.

In our approach, the model is trained and deployed exactly like ULEEN but replacing Bloom Filters with Soon Filters. The discriminator's response in our model is as follows:

$$s_c(x) = \sum_{i=1}^{N} \sum_{k=1}^{K} F_{c,i,\delta_{c,i,k}}$$

During training, the discriminator response is given by:

$$s_c(x) = \sum_{i=1}^{N} \sum_{k=1}^{K} STE(F_{c,i,\delta_{c,i,k}})$$

Refer to Figure [fig:discriminators]2 for a visual representation comparing a discriminator that employs a Bloom Filter with a discriminator that uses a Soon Filter. This comparison illustrates how the increased rate of false positives affects the discriminator's response.

Analysis of Filter Equivalence A noteworthy observation arises when examining the Bloom Filter and the Soon Filter within the context of the number of hash functions employed. Specifically, when only one hash function, both the Soon Filter and the Bloom Filter essentially operate as identical filters. This is due to the fact that in the presence of a single hash function, there's no necessity for an aggregation function. Thus the output of both filters is simply the accessed position by that hash function.

Delving deeper into the continuous relaxations of the Bloom Filter, an interesting parallelism can be discerned. For scenarios with one or two hash functions, both continuous relaxations exhibit identical derivatives. This means that, in terms of behavior, the two relaxations are indistinguishable under these conditions. This congruence is evident for a single hash function since it negates the need for an aggregation function. For two hash functions, we can construct a truth table of the partial derivatives to elucidate this:

Input		*Min*		*Product*	
x_1	x_2	$\frac{\partial A}{\partial x_1}$	$\frac{\partial A}{\partial x_2}$	$\frac{\partial A}{\partial x_1}$	$\frac{\partial A}{\partial x_2}$
0	0	0	0	0	0
0	1	1	0	1	0
1	0	0	1	0	1
1	1	1	1	1	1

When three or more hash functions are introduced, the different continuous relaxations unique characteristics become apparent, highlighting the distinctions in their operational behavior. Below is the truth table showcasing a 3-bit input into the continuous relaxation aggregations functions and their respective derivatives:

Input			*Min*			*Product*		
x_1	x_2	x_3	$\frac{\partial A}{\partial x_1}$	$\frac{\partial A}{\partial x_2}$	$\frac{\partial A}{\partial x_3}$	$\frac{\partial A}{\partial x_1}$	$\frac{\partial A}{\partial x_2}$	$\frac{\partial A}{\partial x_3}$
0	0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0	0
0	1	0	1	0	1	0	0	0
0	1	1	1	0	0	1	0	0
1	0	0	0	1	1	0	0	0
1	0	1	0	1	0	0	1	0
1	1	0	0	0	1	0	0	1
1	1	1	1	1	1	1	1	1

Hardware Considerations As depicted, the modifications introduced by our model to the ULEEN hardware are minimal. Specifically, we remove the AND gate from the Filter output and connect the filter outputs directly to the discriminator's pop-count. This ensures that ULEEN's remarkable hardware features and performance remain intact.

IV. Experiments

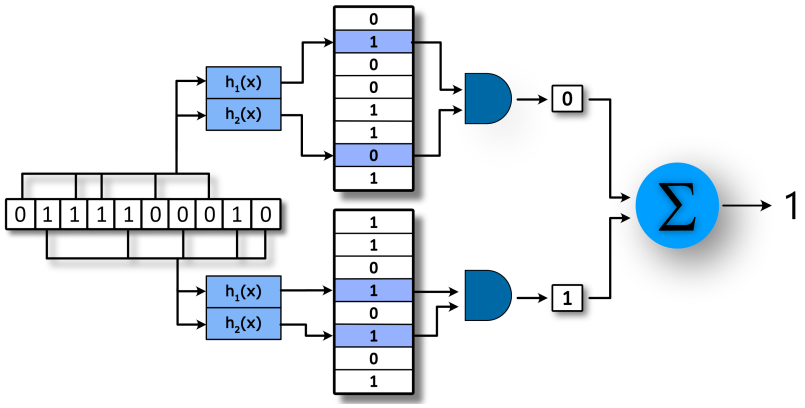
In this section, we present the experimental evaluation of our approach, comparing it with ULEEN, BNN, and DeepShift across the MLPerf Tiny benchmark datasets [37], various UCI datasets [38] and MNIST [39]. We designate our model as SULEEN to distinguish it from ULEEN, signifying the incorporation of our proposed Soon Filters within the ULEEN architecture. Moreover, we introduce ablation studies in which we vary the number of hash functions in the Soon Filter and contrast it with the continuous relaxations of the Bloom Filter to corroborate our theoretical filter equivalence conclusions.

A. Empirical Evaluation Across Diverse Datasets

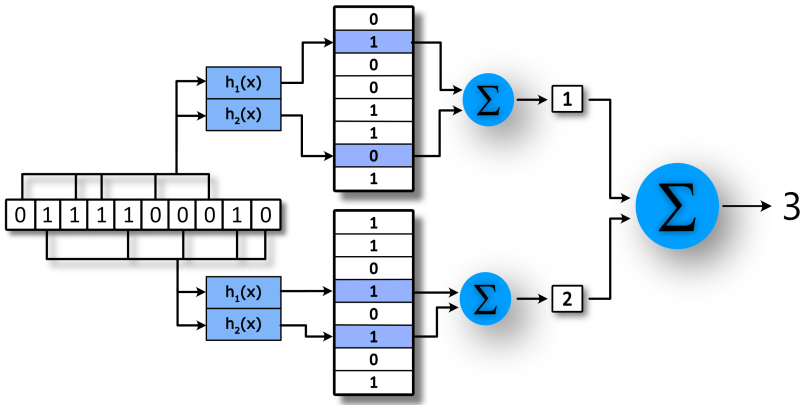
Model Implementations: To evaluate BNNs, we utilize FINN [14], a specialized tool for developing high-performance neural network architectures on Field-Programmable Gate Arrays (FPGAs). FINN is designed to enable the efficient implementation of BNNs, allowing for significant acceleration in inference compared to traditional architectures. This aligns with the most performant version of BNNs as outlined in the accompanying paper. For DeepShift, we employ the implementation detailed and made available in their paper [15], conducting a grid search on both the Q (quantized) and PS (parameterized shift) variations during hyperparameter tuning. For SULEEN and ULEEN, we developed the code in PyTorch [40], incorporating a custom CUDA kernel. The code is made publicly accessible at: *link omitted due to the double-blind review process*.

Hyperparameter Tuning: To optimize each model for every dataset, we employed grid search, utilizing 10% of the training data as a validation set. It is important to note that the test dataset is only used for the final evaluation and not for hyperparameter tuning. For SULEEN and ULEEN, the hyperparameters were varied as follows: $n \in \{2, 3, \ldots, 28\}$, $K \in \{1, 2, 3, 4\}$, and $L \in \{2^1, 2^2, \ldots, 2^n\}$. Common to all models, we explored dropout rates $p \in \{0.0, 0.1, \ldots, 0.8\}$. For BNN and DeepShift, the grid search included the number of hidden layers $P \in \{1, 2, \ldots, 6\}$, with the number of neurons per layer being automatically adjusted to meet the targeted model size. In the case of DeepShift, we additionally conducted a grid search on both the Q and PS variations to discern the optimal choice. Hyperparameter tuning spanned 30 epochs, employing a batch size of 32 and the Adam Optimizer with $\alpha = 0.9$, $\beta = 0.999$. We initiated with a learning rate of 1e-2, reducing it by a factor of 0.1 every 10 epochs.

Data Splits: Following the approach in [32], we split the datasets into 66% train and 33% test sets for datasets where no test data was available.



(a) ULEEN Discriminator



(b) SULEEN Discriminator

Fig. 2: Graphical Representation of ULEEN (a) and SULEEN (b) discriminators. The two images provide an example of a discriminator that receives a 10-bit input. Each discriminator in the example has two filters that are addressed by two hash functions. Each hash function takes a 5-bit word as input and produces a 3-bit address as output. The distinction between the Bloom Filter and Soon Filter is evident in the images: In (a), the upper Bloom Filter provides a true negative output of 0, while the lower indicates a potential positive, outputting a 1. In (b), the upper Soon Filter outputs a 1 (a false positive) and the lower outputs a 2, indicating a potential positive, totaling an output of 3.

Preprocessing: For Keyword Spotting (KWS), we employed Mel Frequency Cepstral Coefficients (MFCC) preprocessing, complemented by cepstral mean and variance normalization. For the UCI datasets and MNIST, we followed the methodology outlined in the ULEEN paper [32].The other datasets underwent no preprocessing. To train both SULEEN and ULEEN models, we employed the Distributive Thermometer Encoding [41] to binary encode the inputs of all datasets. In the MLPerf Tiny benchmark datasets, we assigned 8 bits for CIFAR-10, 12 bits for KWS, 6 bits for ToyADMOS, and 12 bits for Visual Wake Words (VWW). For the UCI datasets, we utilized a 24-bit encoding across all datasets. For MNIST, we employed a 5-bit encoding.

Training: All models was trained for 240 epochs with a batch size of 32. We used the Adam Optimizer with hyperparameters $\alpha = 0.9$ and $\beta = 0.999$. The learning rate was initialized at 1e-2 and decayed by a factor of 0.1 every 80 epochs. Each model was trained and tested 10 times, and we report the average results.

MLPerf Tiny: A standard benchmark suite in [37] for edge devices, MLPerf Tiny includes four datasets. Keyword Spotting (KWS) features 105,829 utterances for keyword recognition [42]. CIFAR-10 comprises 32x32 RGB images across 10 classes for image classification [43]. ToyADMOS/car, with audio recordings of toy cars, focuses on anomaly detection in damaged cars [44]. Visual Wake Words (VWW) uses 96x96 grayscale images from MSCOCO 2014 [45] to detect human presence.

MLPerf Tiny Results: We conducted experiments across three distinct model sizes (small, medium, and large) for each model and dataset. The results are captured in Table I. SULEEN consistently excels across all MLPerf Tiny datasets for every model size. Specifically, in the KWS dataset, our large model outperforms DeepShift by 44.7%, BNNs by 13.7%, and ULEEN by 2.2%. Moreover, our medium-sized model (50KiB) achieves comparable accuracy (+0.3%) to ULEEN large model (101KiB), showing an outstanding 2x reduction in memory footprint when compared at iso-accuracy. A similar trend is observed

Dataset	Model Size	SULEEN	ULEEN	BNN	DeepShift
MLPerf Tiny					
KWS(small)	23KiB	**58.2%**	57.2%	47.2%	18.6%
KWS (medium)	50KiB	**67.7%**	66.1%	53.3%	22.2%
KWS (large)	101KiB	**69.6%**	67.4%	55.9%	24.9%
CIFAR-10 (small)	24KiB	**49.7%**	45.3%	40.0%	40.3%
CIFAR-10 (medium)	250KiB	**55.6%**	53.5%	46.5%	53.0%
CIFAR-10 (large)	625KiB	**57.3%**	54.5%	48.0%	54.1%
ToyADMOS (small)	7KiB	**88.4%**	**88.4%**	84.8%	57.8%
ToyADMOS (medium)	15KiB	**89.3%**	**89.3%**	85.9%	57.8%
ToyADMOS (large)	30KiB	**90.5%**	**90.5%**	86.6%	57.9%
VWW (small)	12KiB	**57.4%**	**57.4%**	51.7%	52.9%
VWW (medium)	120KiB	**59.8%**	**59.8%**	52.1%	53.8%
VWW (large)	250KiB	**60.6%**	**60.6%**	52.3%	54.6%
UCI					
Ecoli	0.87KiB	**87.5%**	**87.5%**	68.9%	43.6%
Iris	0.28KiB	**98.3%**	98.0%	69.2%	33.3%
Letter	78.00KiB	**96.0%**	95.3%	4.79%	19.2%
SatImage	9.00KiB	**91.7%**	90.9%	30.8%	48.0%
Vehicle	2.25KiB	**78.3%**	77.1%	27.2%	28.3%
Vowel	3.44KiB	**94.0%**	91.7%	17.7%	8.4%
Wine	0.42KiB	**98.3%**	**98.3%**	14.0%	27.3%
MNIST	(**98**/262/355/408)KiB	**98.6%**	98.5%	98.4%	98.3%

TABLE I: Accuracy comparison of SULEEN, ULEEN, BNN, and DeepShift Across MLPerf Tiny at three different model sizes (small, medium, and large), various UCI datasets and MNIST. The highest accuracy for each dataset and, for MLPerf Tiny, each model size, is highlighted in bold.

in the CIFAR-10 dataset, where our medium model (250KiB) achieves 55.6%, compared to the ULEEN large model (625KiB) which achieves 54.5%, thereby demonstrating a substantial 2.5x reduction in model size. On ToyADMOS and VWW, SULEEN and ULEEN perform identically. This is due to both achieving optimal results in the hyper-parameter tuning when using a single hash function, causing them to operate essentially as identical models, a phenomenon we detailed theoretically in our methodology and that will be further substantiated in the next subsection.

UCI: Our proposed approach is evaluated using a selection of datasets from the UCI Machine Learning Repository [38]. This evaluation aims to verify its applicability for edge inference in applications that utilize structured data.

UCI Results: Table I encapsulates our findings. SULEEN consistently ranks first in accuracy across all datasets. When juxtaposed against ULEEN, SULEEN exhibits superior accuracy in all cases, save for the Ecoli and Wine datasets, where both models achieve parity. It's striking to note that the top-ranking models in terms of accuracy predominantly belong to WNNs (SULEEN and ULEEN). Both BNN and DeepShift fall short in matching the their performance, accentuating the distinct advantage of WNNs in edge inference applications that utilize structued data.

MNIST: In our study, we utilized the MNIST dataset, a classic and foundational benchmark in the field of edge inference. Our focus was on comparing model sizes while maintaining a similar accuracy level, close to 98.5%, a common practice in this domain. We utilize the results reported in [14, 32, 46] for ULEEN, BNN and DeepShift repectively.

MNIST Results: The results, as presented in Table I, demonstrate a significant advancement achieved by SULEEN. Notably, SULEEN attained an impressive accuracy of 98.6% with a model size of only 98KiB. This performance is particularly remarkable when compared to ULEEN, achieving a 2.67x reduction in model size, without compromising on accuracy. Furthermore, when compared to BNN and DeepShift, we acheive a model size reduction of 3.62x and 4.16x respectively. These results underscore SULEEN's significant contribution to the field of edge inference, offering a powerful yet compact solution that does not sacrifice accuracy for size efficiency.

B. Ablation Studies and Theoretical Validation

In this subsection, we evaluate our Soon Filter against the continuous Min-Relaxation and Prod-Relaxation of the Bloom Filter. We do this by varying the number of hash

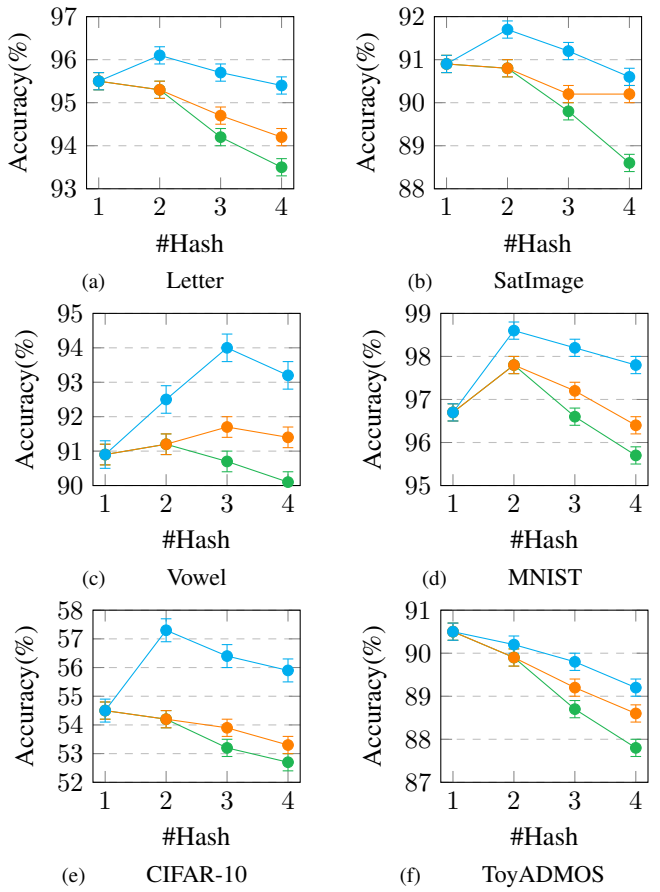


Fig. 3: Ablation study graphs comparing the Number of Hash Functions versus Accuracy(%) for the Soon Filter (in Blue), the Bloom Filter with Min function continuous relaxation (in Orange), and the Bloom Filter with Product operation continuous relaxation (in Green). Datasets include (a) Letter, (b) SatImage, (c) Vowel, (d) MNIST, (e) CIFAR-10 , and (f) ToyADMOS .

functions used, ranging from 1 to 4. Our goal is to validate our theoretical findings: that the three filter versions operate as the same model with one hash function and that both the Mean and Product relaxations function identically with two hash functions. Additionally, we aim to confirm that our model consistently outperforms the others regardless of the number of hash functions employed. We test this hypothesis on the Letter, SatImage, Vowel, MNIST, CIFAR-10, and ToyADMOS datasets.

The results are illustrated in Figure 3. As can be observed, with one hash function, all filters yield identical accuracy across all datasets. When using two hash functions, both Min and Product relaxations also produce identical results. Notably, for every dataset and number of hash functions, the Soon Filter consistently surpasses both the Min and Product relaxations of the Bloom Filter. These findings validate our theoretical assertions.

V. Conclusion

In this study, we introduced the Soon Filter, an innovative approach designed to enhance the performance of ULEEN, a multiplication-free model tailored for high-throughput edge inference. By theoretically demonstrating its efficiency and conducting rigorous experimentation on the MLPerf Tiny, UCI, and MNIST datasets, we have distinctly underscored the robustness and efficiency of our proposed methodology. Notably, our results have surpassed the performance benchmarks across all datasets, outperforming well-established models like ULEEN, BNN, and DeepShift.

Through ablation studies, we have empirically verified our theoretical assertions regarding filter equivalence, showing that the Soon Filter consistently outperforms its counterparts by maximizing the gradient updates of the filter RAM positions.

Given the minimal deviations between our model and ULEEN, as outlined in the methodology section, we can consider ULEEN's hardware results as a performance upper bound for our model. Additionally, our approach achieves, on average, a 2x reduction in model size compared to ULEEN. Based on this significant size reduction, we project that the hardware deployment of our model could be twice as efficient. This projection underscores the importance of hardware testing as a vital and immediate direction for future research.

Furthermore, numerous edge applications that rely on high-throughput architectures could greatly benefit from our approach. The integration of SULEEN into these applications has the potential to redefine their efficiency and robustness, thereby setting a new standard for edge inference.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *CoRR*, vol. abs/2112.10752, 2021. [Online]. Available: https://arxiv.org/abs/2112.10752

[3] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv*, vol. 1409, 09 2014.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *CoRR*, vol. abs/1705.07565, 2017. [Online]. Available: http://arxiv.org/abs/1705.07565

[8] ——, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," *CoRR*, vol. abs/1705.07565, 2017. [Online]. Available: http://arxiv.org/abs/1705.07565

[9] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/e82c4b19b8151ddc25d4d93baf7b908f-Paper.pdf

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

[11] Y.-L. Sung, V. Nair, and C. A. Raffel, "Training neural networks with fixed sparse masks," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 24 193–24 205. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/cb2653f548f8709598e8b5156738cc51-Paper.pdf

[12] W. Sun, A. Zhou, S. Stuijk, R. Wijnhoven, A. O. Nelson, h. Li, and H. Corporaal, "Dominosearch: Find layer-wise fine-grained n:m sparse schemes from dense neural networks," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 20 721–20 732. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/ad68473a64305626a27c32a5408552d7-Paper.pdf

[13] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "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., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf

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

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

[16] H. You, X. Chen, Y. Zhang, C. Li, S. Li, Z. Liu, Z. Wang, and Y. Lin, "Shiftaddnet: A hardware-inspired deep network," *CoRR*, vol. abs/2010.12785, 2020. [Online]. Available: https://arxiv.org/abs/2010.12785

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

[18] H. Udagawa, T. Okano, and T. Saito, "Permutation binary neural networks: Analysis of periodic orbits and its applications," pp. 748–764, 2023. [Online]. Available: /article/id/62948d012d80b70dfd2582b8

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

[20] H. He and R. Xia, "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*. Springer, 2018, pp. 250–259.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[22] ——, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, p. 422–426, Jul. 1970. [Online]. Available: https://doi.org/10.1145/362686.362692

[24] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *CoRR*, vol. abs/1308.3432, 2013. [Online]. Available: http://arxiv.org/abs/1308.3432

[25] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Skh4jRcKQ

[26] M. De Gregorio, "An intelligent active video surveillance system based on the integration of virtual neural sensors and bdi agents," *IEICE TRANSACTIONS on Information and Systems*, vol. 91, no. 7, pp. 1914–1921, 2008.

[27] P. Coraggio and M. De Gregorio, "WiSARD and NSP for robot global localization," in *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer, 2007, pp. 449–458.

[28] C. B. Do Prado, F. M. G. França, E. Costa, and L. Vasconcelos, "A new intelligent systems approach to 3D animation in television," in *Proceedings of the 6th ACM international conference on Image and video retrieval*. ACM, 2007, pp. 117–119.

[29] L. Santiago, L. Verona, F. Rangel, F. Firmino, D. S. Menasché, W. Caarls, M. Breternitz Jr, S. Kundu, P. M. Lima, and F. M. França, "Weightless neural networks as memory segmented bloom filters," *Neurocomputing*, vol. 416, pp. 292–304, 2020.

[30] Z. Susskind, A. Arora, I. D. Miranda, L. A. Villon, R. F. Katopodis, L. S. de Araújo, D. L. Dutra, P. M. Lima, F. M. França, M. Breternitz Jr *et al.*, "Weightless neural networks for efficient edge inference," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2022, pp. 279–290.

[31] J. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022000079900448

[32] Z. Susskind, A. Arora, I. D. Miranda, A. T. Bacellar, L. A. Villon, R. F. Katopodis, L. S. de Araujo, D. L. Dutra, P. Lima, F. M. Franca *et al.*, "Uleen: A novel architecture for ultra low-energy edge neural networks," *arXiv preprint arXiv:2304.10618*, 2023.

[33] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. C. Courville, and Y. Bengio, "Renet: A recurrent neural network based alternative to convolutional networks," *CoRR*, vol. abs/1505.00393, 2015. [Online]. Available: http://arxiv.org/abs/1505.00393

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[36] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

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

[38] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[39] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[41] A. Bacellar, Z. Susskind, L. Villon, I. Miranda, L. Santiago, D. Dutra, M. Jr, L. JOHN, P. Lima, and F. França, "Distributive thermometer: A new unary encoding for weightless neural networks," 01 2022, pp. 31–36.

[42] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[43] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Toronto, Ontario, Tech. Rep. 0, 2009.

[44] Y. Koizumi, S. Saito, H. Uematsu, N. Harada, and K. Imoto, "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)*, 2019, pp. 313–317.

[45] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[46] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "Deepshift: Towards multiplication-less neural networks," 2021.