PolyLUT-Add: FPGA-based LUT Inference with Wide Inputs

Binglei Lou, Richard Rademacher, David Boland and Philip H.W. Leong

School of Electrical and Computer Engineering

The University of Sydney, Australia 2006

Email: {binglei.lou,richard.rademacher,david.boland,philip.leong}@sydney.edu.au

Abstract—FPGAs have distinct advantages as a technology for deploying deep neural networks (DNNs) at the edge. Lookup Table (LUT) based networks, where neurons are directly modelled using LUTs, help maximize this promise of offering ultra-low latency and high area efficiency on FPGAs. Unfortunately, LUT resource usage scales exponentially with the number of inputs to the LUT, restricting PolyLUT to small LUT sizes. This work introduces PolyLUT-Add, a technique that enhances neuron connectivity by combining A PolyLUT sub-neurons via addition to improve accuracy. Moreover, we describe a novel architecture to improve its scalability. We evaluated our implementation over the MNIST, Jet Substructure classification and Network Intrusion Detection benchmark and found that for similar accuracy, PolyLUT-Add achieves a LUT reduction of $1.3 - 7.7\times$ with a $1.2 - 2.2\times$ decrease in latency.

I. INTRODUCTION

Deep neural networks (DNNs) have been shown to provide powerful feature extraction and regression capabilities and are widely employed across a spectrum of applications, including image classification for autonomous driving [1], data analysis in particle physics [2], and anomaly detection for cybersecurity [3, 4]. Field-Programmable Gate Arrays (FPGAs) provide a unique implementation platform for deploying DNNs, with significant advantages over other technologies, particularly in real-time inference tasks.

Lookup Table (LUT) based neurons on FPGAs offer high area efficiency and ultra-low latency. Examples of accelerators published using this approach include LUTNet [5], LogicNets [6], NullaNet [7] and PolyLUT [8]. Compared with Binary Neural Networks (BNNs [9]), which utilize 1-bit quantization to replace multipliers with simple XNOR gates, LUT-based neurons further optimize FPGA resource utilization using LUTs as direct inference operators.

Building upon the PolyLUT framework, this work introduces an enhancement called PolyLUT-Add, where we combine A copies of PolyLUT sub-neurons via an A-input adder to increase neuron fan-in. Figure 1 highlights how our approach builds from PolyLUT for a simple example where $A = 2$. The computation process of a PolyLUT neuron: weight multiplication, accumulation, batch normalization (BN), and quantized activation, is first shown in Figure 1(a). Our PolyLUT-Add approach, shown in Figure 1(b) restructures the neuron computation. The first stage is similar to PolyLUT without the batch normalization and is repeated for each sub-neuron. Instead, the batch normalization is performed after the results are accumulated, with the resulting activation quantized again if necessary.

While the same functionality of PolyLUT-Add could be achieved with PolyLUT with a single lookup table, PolyLUT-Add can make better use of the FPGA fabric. In this example, with the input word length, $\beta = 2$, PolyLUT-Add uses three distinct lookup tables, each of size 2^6; the single lookup table equivalent would be of size 2^{12}. In the general case, if we define the fan-in to be F of (β-bit words), for each output bit, PolyLUT requires a lookup table of $2^{\beta FA}$, while PolyLUT-Add only requires reducing the size to $A \times 2^{\beta F} + 2^{A(\beta+1)}$.

The contributions of this work can be summarized as:

- At the algorithmic level, we introduce PolyLUT-Add, an extension of the PolyLUT framework [8], which incorporates A PolyLUT sub-neurons, combined via an A-input adder to enable improved accuracy.
- At the computer architecture level, we propose an efficient FPGA implementation of PolyLUT-Add.
- To the best of our knowledge, for similar accuracy, PolyLUT-Add produces the best-reported FPGA latency and area results on the three benchmarks tested. To facilitate experimentation with our design, the work in this paper is reproducible. Source code and data to reproduce our results are available from Github[1].

We evaluated PolyLUT-Add across three datasets with four DNN models and demonstrated significant accuracy improvements. Specifically, for the same polynomial degree D and fan-in F setups, $A = 2$ achieves an accuracy improvement of up to 2.7%, albeit with a 2-3 fold increase in area. Latency and clock frequency are unchanged in most cases. However, we also see that when $A = 2$, we can choose a lower D and F and obtain accuracy levels comparable to those achieved by the original PolyLUT. This reduces LUT consumption by factors of 4.6, 7.7 and 1.3 for the MNIST, Jet Substructure classification, and Network Intrusion Detection benchmarks, respectively, with a 1.2 to 2.2 times decrease in latency.

The remainder of this paper is organized as follows. In Section II we review previous work on LUT-based neurons. In Section III, the design of PolyLUT-Add is described. Results are presented in Section IV and conclusions drawn in Section V.
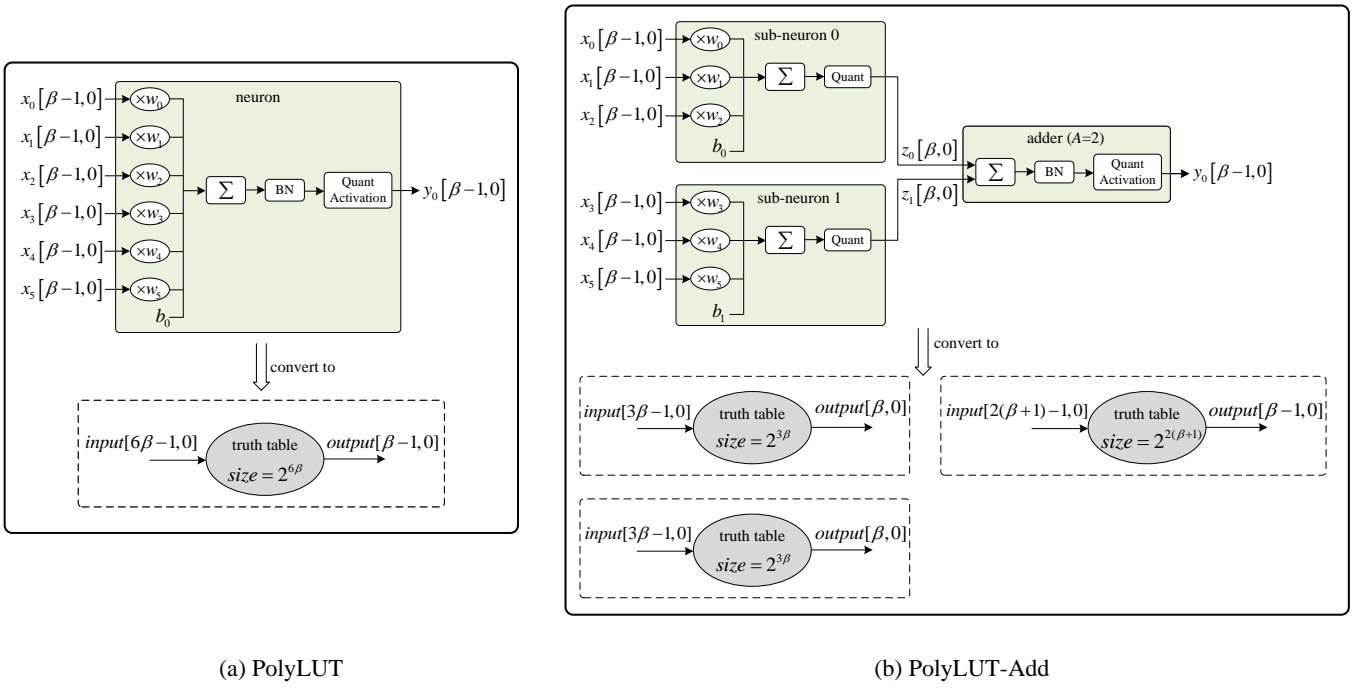
---

[1]PolyLUT-Add: https://github.com/bingleilou/PolyLUT-Add

(a) PolyLUT

(b) PolyLUT-Add

Fig. 1: Architecture of a (a) PolyLUT and (b) PolyLUT-Add neuron, Fan-in $F = 6$ (β-bit words) and $A = 2$ sub-neurons. For simplicity, the polynomial order of each PolyLUT neuron [8] is set to 1 in this example. For each output bit, PolyLUT requires a lookup table of $2^{6\beta}$, while PolyLUT-Add requires $(2^{3\beta} + 2^{3\beta} + 2^{2(\beta+1)})$.

II. BACKGROUND

Wang *et al.* introduced LUTNet, the first LUT-optimized FPGA inference scheme [5]. Its approach was to prune a residual BNN: ReBNet [10] by mapping some of the XNOR-population count (popcount) operations directly to $k-$input LUTs to take advantage of FPGA architectures. LogicNets [6] and NullaNet [7] adopted a different approach by quantizing the inputs and outputs of each neuron and encapsulating the neuron's transfer function (*i.e.*, densely connected linear and activation functions) in a lookup table. This method enumerated all possible combinations of a neuron's inputs and determined the corresponding outputs based on the neuron's weights and biases. By replacing popcount operations with Boolean expressions, significant computational savings were made. Building upon the foundations of LogicNets, Poly-LUT [8], proposed by Andronic *et al.*, further enhanced accuracy and reduced the number of required layers by introducing piecewise polynomial functions.

Figure 2 illustrates the main idea behind the LogicNets, NullaNet and PolyLUT approaches. Only sparse connections of maximum F inputs from the previous layer are supported, and these can be directly mapped to the output via LUTs, eliminating the F-bit popcount operations required to form the sum for a dot product operation. As an example in Figure 2(a), the current layer has N neurons, of which only F random nodes are used as inputs to each neuron in the next layer. In Figure 2(b), the transfer function mapping an input vector $[x_0, x_1, \ldots, x_{F-1}]$ to the output node y_0 can be implemented
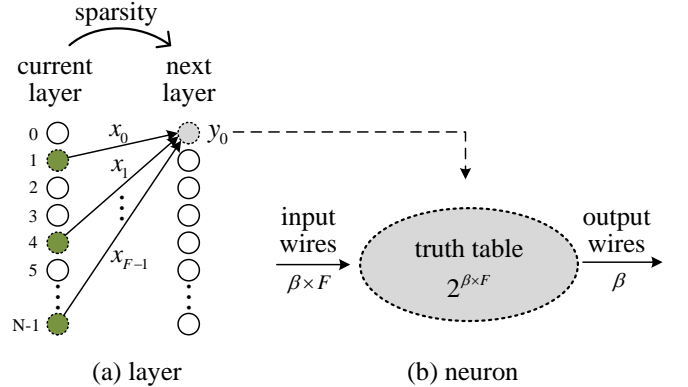

(a) layer

(b) neuron

Fig. 2: Illustration of the LUT-based DNN inference scheme used in LogicNets [6], NullaNet [7] and PolyLUT [8].

using βF inputs, and hence requires a lookup table of size $2^{\beta F}$. The constraint $F \ll N$ is manually applied to limit the size of the lookup table.

$$y_0 = \sigma \left(\sum_{i=0}^{M-1} w_i m_i(x) + b \right), \text{where } M = \binom{F+D}{D} \quad (1)$$

The computation inside the neuron y_0 can be described as Eq. (1), where σ is the quantized activation function, w and b denote weight and bias respectively, D is a polynomial degree. For LogicNets and NullaNet methods $D = 1$; Poly-LUT generalizes these methods by allowing larger polynomial

degree constructed from multiplicative combinations of the inputs up to degree D. For instance, if the input vector is two-dimensional and $D = 2$, the model construction proceeds as follows: $[x_0, x_1] \rightarrow \left[1, x_0, x_1, x_0^2, x_0 x_1, x_1^2\right]$. The value of M equals the number of monomials $m(x)$ of at most degree D in F variables.

While PolyLUT enriched the representational capability of previous solutions, this requires a lookup table of $O(2^{\beta F})$. For instance, in the context of the MNIST handwritten digit recognition task [11], which involves classifying 28×28 pixel images into 10 categories, PolyLUT's architecture employs layers with (784, 256, 100, 100, 100, 10) neurons, using parameters $\beta = 2$ and $F = 6$. This means that only 6 neurons are randomly selected from each layer to form extremely sparse connectivity to neurons in the next layer. To address scalability by avoiding very large table sizes, the size of each neuron's lookup table was capped at 2^{12}. The exponential LUT requirement of this approach precludes the selection of larger β and fan-in values, which can in turn limit accuracy.

III. DESIGN

A. DNN architecture

Figure 3 outlines our proposed DNN architecture. Compared with Figure 2, the fan-in F to sub-neurons remains the same, but the total fan-in to the neuron is increased by a factor of A at the output. This is achieved by summing A independent and parallel randomly connected Poly-layers. To elucidate the enhancement mechanism, we introduce the formulation detailed in Eq. (2).

$$\sum_{i=0}^{AF-1} w_i x_i + b = \sum_{a=0}^{A-1} \left(\sum_{i=0}^{F-1} w_{(aF+i)} x_{(aF+i)} + b_a \right) \quad (2)$$

During computation, the activation function, such as Rectified linear unit (ReLU) output bits, can be one bit less than the input bits because its output is non-negative. To avoid overflow in the Adder-layer, we increase its internal word length by one bit (to $\beta + 1$), as seen in Figure 1(b).

B. System Toolflow

Figure 4 shows the tool flow. Like PolyLUT, training is done offline using PyTorch [12], and then the resulting weights are used to create the LUTs that implement each neuron. These tables are then utilized to generate Register Transfer Level (RTL) files in Verilog, encapsulating the Boolean expressions derived from the neurons. The final stage involves synthesizing the LUT-based DNN design onto hardware, using the AMD/Xilinx Vivado tool [13].

The integration of Brevitas [14] with PyTorch facilitates quantization-aware training of DNNs. We modified the network implementation to accept A, the model's fan-in factor as a parameter. The model's weights are transformed into lookup tables following the training phase. This transformation begins by employing the quantized states inherent in the trained model to ascertain each neuron's input data range. For Poly-layers, we generate all possible input combinations based on β and F; In contrast, for the Adder-layer, all combinations are
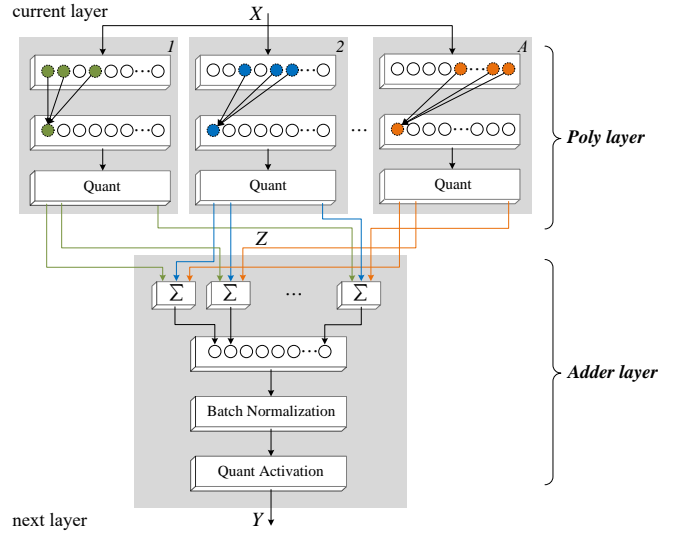

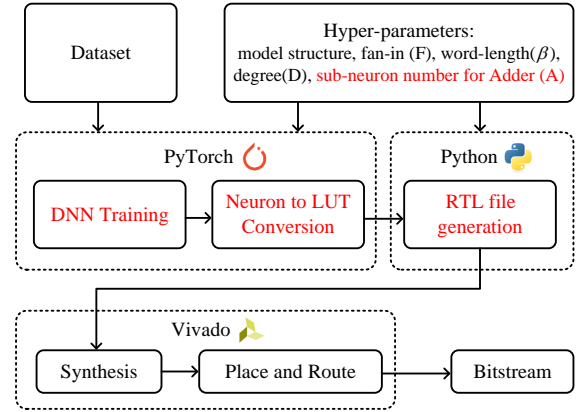Fig. 3: A single-layer block diagram of PolyLUT-Add.

Fig. 4: Tool flow for PolyLUT-Add. The original open-source PolyLUT toolflow [8] components are shown in black with modified elements in red.

generated based on β and A. These input combinations are subsequently fed into their respective layers—Poly-layer and Adder-layer—to generate the corresponding outputs. Finally, these input and output pairs form the individual values for the lookup table.

C. Pipelining

In our FPGA design, we treat each layer as an independent module and synthesize them separately, with the critical path in the layer with the largest delay determining the system's maximum clock frequency. Two implementation strategies were considered, as illustrated in Figure 5.

1) **Separate pipeline registers for each layer**. This strategy is best when the lookup table size (a proxy for critical path delay) of the Adder-layer ($2^{A(\beta+1)}$) and Poly-layer ($A2^{\beta F}$) are similar. Although the latency in clock cycles is doubled, this architecture maximizes clock frequency.
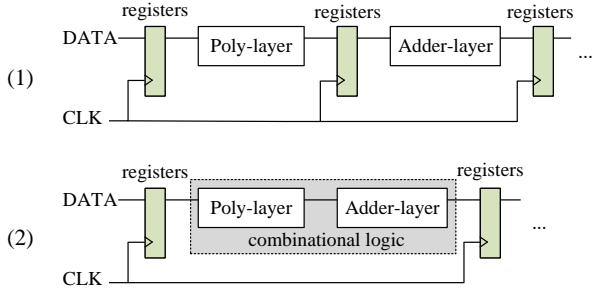
Fig. 5: Two synthesis strategies

2) **Single register for combined Poly-layer and Adder-layer**. When the Adder-layer is much smaller than Poly-layer, its processing time should not adversely impact the Poly-layer's performance, enabling a more efficient overall system design where latency is tightly controlled.

IV. RESULTS

A. Datasets

We evaluated the proposed PolyLUT-Add design on three commonly used datasets for ultra-low latency inference:

1) *Handwritten Digit Recognition*: In timing-critical sectors such as autonomous vehicles, medical imaging, and real-time object tracking, the demand for low-latency image classification is paramount. These applications underscore the necessity for swift and accurate decision-making, where even minimal delays can have significant repercussions. Unfortunately, there is no public dataset specialized for low-latency image classification tasks. The MNIST [11] is therefore utilized to benchmark our work on its image classification performance which is a dataset for handwritten digit recognition tasks with 28×28 pixels as input image and 10 classes as outputs.

2) *Jet Substructure Classification*: Real-time decision-making is often important for physics experiments such as the CERN Large Hadron Collider (LHC). Jet Substructure Classification (JSC) is one of its applications that requires high-throughput data processing. Prior works [2, 15–17] employed neural networks on FPGA for this task to provide real-time inference capabilities. We also use the JSC dataset formulated from Ref. [2] to evaluate our work, with the dataset having 16 substructure properties as input and 5 types of jets as outputs.

3) *Network Intrusion Detection*: In the field of cybersecurity, the swift detection and mitigation of network threats are important for the preservation of digital infrastructure integrity (*e.g.* fibre-optic throughput can reach 940 Mbps). Prior works have used FPGAs to accelerate DNNs, enabling real-time Network Intrusion Detection Systems (NIDS) with high accuracy and enabling privacy on edge devices [6, 8, 18]. The UNSW-NB15 dataset [19] was used as the benchmark for our evaluation process. It has 49 input features and the classification is binary (bad or normal).

Results for the JSC and NID datasets were reported in the LogicNets paper [6], and all three datasets were used for PolyLUT [8].

B. Experimental Setup

Table I lists our neural network configurations for three datasets. As a foundation for our experiments and to ensure consistency in evaluation, our setup closely follows the PolyLUT study [8]. Our newly introduced A is set to $A \in \{2, 3\}$ for models (HDR and JSC-M Lite) with small truth table ($2^{\beta F}$), and $A = 2$ is used for JSC-XL. Larger A ($A \geq 4$) can be supported using an adder tree, which is left as future work.

The polynomial degree $D = 1$ and $D = 2$ correspond to linear and quadratic representations respectively. We utilize $D \in \{1, 2\}$ to evaluate the performance of PolyLUT-Add. As will be seen later, to facilitate a comparison with existing literature and aim for enhanced accuracy, we also explore

TABLE I: Model setups used to evaluate different datasets.

Dataset	Model name	Neurons per layer	β	F	D	A
MNIST	HDR	256, 100, 100, 100, 100, 10	2	6	1, 2	2, 3
Jet Substructure	JSC-XL[1]	128, 64, 64, 64, 5	5	3	1, 2	2
Jet Substructure	JSC-M Lite	64, 32, 5	3	4	1, 2	2, 3
UNSW-NB15	NID Lite[2]	686, 147, 98, 49, 1	3	5	1	2

1: Remarks: $\beta_i = 7$, $F_i = 2$
2: Remarks: $\beta_i = 1$, $F_i = 7$
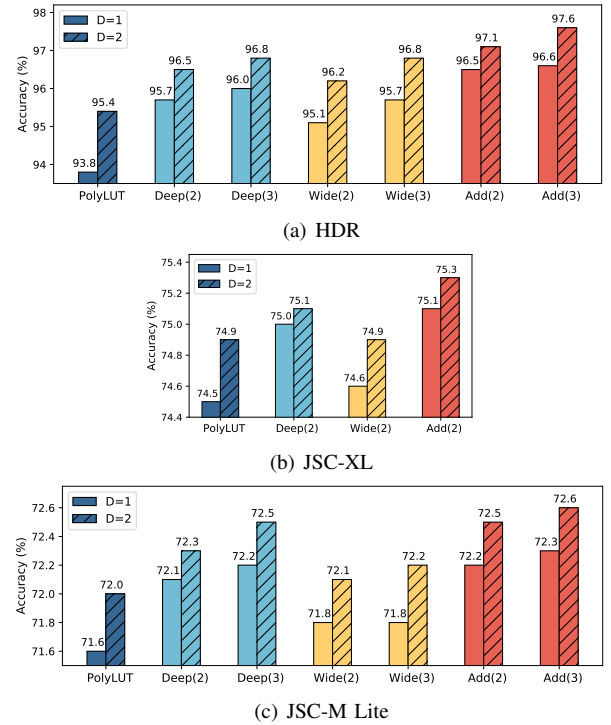

(a) HDR

(b) JSC-XL

(c) JSC-M Lite

Fig. 6: Accuracy results on different models. We use Deep($\mathbb{D}$), Wide($\mathbb{W}$) and Add(A) to denote "PolyLUT-Deeper", "PolyLUT-Wider" and "PolyLUT-Add" respectively.

$D = 3$ in Section IV-D. This contrasts with PolyLUT, for which higher degrees ($D \in \{4, 6\}$) achieve the best accuracy. We comment, the case $A = 1$ is identical to PolyLUT, and $A = 1, D = 1$ corresponds to LogicNets. We also note that the training convergence of the UNSW dataset is sensitive to the initial random seed, and hence, multiple trials were necessary before a result with good accuracy was achieved. As an exception, we therefore apply $A = 2$ and $D = 1$ to evaluate the NID Lite model.

Using AdamW as the optimizer [12], we trained the smaller models/datasets: (JSC-M Lite, NID Lite) for 1000 epochs, and used 500 epochs for (JSC-XL and HDR). The mini-batch size is set to 1024 and 128 for (JSC-XL and JSC-M Lite, NID-Lite) and MNIST, respectively. We inherit these configurations from PolyLUT [8] to ensure consistency in evaluation.

We used the AMD/Xilinx `xcvu9p-flgb2104-2-i` FPGA part for evaluation to facilitate comparisons with PolyLUT [8] and LogicNets [6]. The designs are compiled using Vivado 2020.1 with `Flow_PerfOptimized_high` settings, and are configured to perform synthesis in the `Out-of-Context` (`OOC`) mode. The RTL Generation time was measured on a desktop with Intel(R) Core(TM) i7-10700F @2.9GHz and 64GB memory.

C. PolyLUT-Add vs Deeper and Wider PolyLUT (small D)

We first present results comparing PolyLUT-Add with Poly-LUT in configurations with the same polynomial degree. Three configurations were tested:

1) **Original PolyLUT**: This serves as the baseline network.
2) **PolyLUT-Deeper**: This explores the impact of increasing network depth. We denote the depth factor as $\mathbb{D}$.

Then $\mathbb{D}\times$ the number of layers is applied to models in Table I. For example, for JSC-M Lite, if $\mathbb{D} = 2$, the hidden layer is doubled meaning the neurons per layer becomes (64,64,32,32,5).

3) **PolyLUT-Wider**: This examines the impact of a wider network model. We denote the width factor as $\mathbb{W}$. Then $\mathbb{W}\times$ the number of neurons per layer are applied to models in Table I. Once again, for JSC-M Lite, if $\mathbb{W} = 2$, the neurons per layer becomes (128,64,5).

Figure 6 shows the accuracy of the configurations with parameter settings detailed in Table I. PolyLUT-Add achieves the highest accuracy against all baselines on all datasets for both the linear ($D = 1$) and non-linear ($D = 2$) cases.

D. Optimizing for Accuracy

In terms of accuracy and hardware, Table II shows that for $A = 2$, PolyLUT-Add achieved accuracy improvements of 2.7%, 0.6% and 2.3% over PolyLUT on the MNIST, Jet Substructure classification and Network Intrusion Detection benchmarks respectively. However, this required a 2-3$\times$ increase in LUT size.

We also evaluate the performance of simply increasing PolyLUT's fan-in, F. This has a lookup table consumption of 256-1024$\times$ for similar accuracy, showing that PolyLUT-Add can improve model accuracy without an excessive impact on LUT size. Furthermore, it's noteworthy that the RTL Generation time cost also correlates with the lookup table size; it follows that a direct increase in fan-in would incur exponentially higher RTL Generation time costs.

In terms of latency, we apply single registers for combined Poly-layer and Adder-layer (pipeline strategy-(2) in Figure 5)

TABLE II: Comparison of accuracy and hardware results between PolyLUT and PolyLUT-Add ($\mathbb{D} = 1$, $\mathbb{W} = 1$)

Models	Degree D	Model	Fan-in $(F \times A)$	Acc(%)↑	lookup table Size↓	LUT (% of 1182240)	FF (% of 2364480)	F_max (MHz)	Latency (cycles)	RTL Gen. (hours)
HDR	1	PolyLUT	6	93.8	2^{12}	3.43	0.12	378	6	1.40
			10	96.1	$2^{12} \times 256$		—			—
		PolyLUT-Add	6×2	96.5	$2^{12} \times 2 + 2^6$	12.69	0.12	378	6	3.00
			6×3	**96.6**	$2^{12} \times 3 + 2^9$	20.67	0.12	378	6	4.40
	2	PolyLUT	6	95.4	2^{12}	6.62	0.12	378	6	1.40
			10	97.3	$2^{12} \times 256$		—			—
		PolyLUT-Add	6×2	97.1	$2^{12} \times 2 + 2^6$	19.78	0.07	378	6	3.00
			6×3	**97.6**	$2^{12} \times 3 + 2^9$	31.36	0.07	378	6	4.50
JSC-XL	1	PolyLUT	3	74.5	2^{15}	19.55	0.07	235	5	2.10
			5	74.9	$2^{15} \times 1024$		—			—
		PolyLUT-Add	3×2	**75.1**	$2^{15} \times 2 + 2^{12}$	50.10	0.07	235	5	5.17
	2	PolyLUT	3	74.9	2^{15}	37.40	0.07	235	5	2.30
			5	75.2	$2^{15} \times 1024$		—			—
		PolyLUT-Add	3×2	**75.3**	$2^{15} \times 2 + 2^{12}$	89.60	0.07	235	5	5.24
JSC-M Lite	1	PolyLUT	4	71.6	2^{12}	0.97	0.01	646	3	0.16
			7	72.1	$2^{12} \times 512$		—			—
		PolyLUT-Add	4×2	72.2	$2^{12} \times 2 + 2^8$	2.62	0.01	488	3	0.35
			4×3	**72.3**	$2^{12} \times 3 + 2^{12}$	4.33	0.01	363	3	0.63
	2	PolyLUT	4	72.0	2^{12}	1.51	0.01	568	3	0.16
			6	72.5	$2^{12} \times 512$		—			—
		PolyLUT-Add	4×2	72.5	$2^{12} \times 2 + 2^8$	4.29	0.01	440	3	0.34
			4×3	**72.6**	$2^{12} \times 3 + 2^{12}$	6.57	0.01	373	3	0.64
NID Lite	1	PolyLUT	5	89.3	2^{15}	6.86	0.15	529	5	4.09
			8	91.0	$2^{15} \times 512$		—			—
		PolyLUT-Add	5×2	**91.6**	$2^{15} \times 2 + 2^8$	21.41	0.15	529	5	8.76

$-$: Data for very high fan-in settings is omitted due to exceeding FPGA memory capacity limits.

TABLE III: Comparison results with prior works. PolyLUT-Add uses smaller F and D (see Table IV), whereas PolyLUT uses larger D, F for accuracy.

Dataset	Model	Accuracy↑	LUT	FF	DSP	BRAM	F_max(MHz)↑	Latency(ns)↓
MNIST	**PolyLUT-Add (HDR-Add2, D=3)**	**96%**	**15272**	**2880**	**0**	**0**	**833**	**7**
	PolyLUT (HDR, D=4) [8]	**96%**	70673	4681	0	0	378	16
	FINN [20]	**96%**	91131	-	0	5	200	310
	`hls4ml` [21]	95%	260092	165513	0	0	200	190
Jet Substructure	**PolyLUT-Add (JSC-XL-Add2, D=3)**	75%	**47639**	**1712**	**0**	**0**	**400**	**13**
	PolyLUT (JSC-XL, D=4) [8]	75%	236541	2775	0	0	235	21
	Duarte *et al.* [2]	75%	88797*		954	0	200	75
	Fahim *et al.* [17]	**76%**	63251	4394	38	0	200	45
Jet Substructure	**PolyLUT-Add (JSC-M Lite-Add2, D=3)**	**72%**	**1618**	**336**	**0**	**0**	**800**	**4**
	PolyLUT (JSC-M Lite, D=6) [8]	**72%**	12436	773	0	0	646	5
	LogicNets [6]	**72%**	37931	810	0	0	427	13
UNSW-NB15	**PolyLUT-Add (NID-Add2, D=1)**	**92%**	**2591**	1193	**0**	**0**	**620**	**8**
	PolyLUT (NID-Lite D=4) [8]	**92%**	3336	686	0	0	529	9
	LogicNets [6]	91%	15949	1274	0	5	471	13
	Murovic *et al.* [18]	**92%**	17990	**0**	**0**	**0**	55	18

∗: Paper reports "LUT+FF"

TABLE IV: Model setups for smaller F of PolyLUT-Add.

Dataset	Model name	Neurons per layer	β	F	D	A
MNIST	HDR-Add2	256, 100, 100, 100, 100, 10	2	4	3	2
Jet Substructure	JSC-XL-Add2[1]	128, 64, 64, 64, 5	5	2	3	2
Jet Substructure	JSC-M Lite-Add2	64, 32, 5	3	2	3	2
UNSW-NB15	NID-Add2[2]	100, 100, 50, 50, 1	2	3	1	2

1: Remarks: $\beta_i = 7$, $F_i = 1$
2: Remarks: $\beta_i = 1$, $F_i = 6$, $\beta_o = 2$, $F_o = 7$

TABLE V: Comparison of two pipeline strategies on PolyLUT-Add with JSC-M Lite as the case study

Degree D	Fan-in $F \times A$	pipeline strategy	F_max (MHz)↑	Latency Results clock cycles↓	Latency(ns)↓
1	4×2	(1)	**646**	6	9
		(2)	488	**3**	**6**
	4×3	(1)	**571**	6	11
		(2)	363	**3**	**8**
2	4×2	(1)	**568**	6	11
		(2)	440	**3**	**7**
	4×3	(1)	**568**	6	11
		(2)	373	**3**	**8**

to models in Table II. For HDR, JSC-XL and NID Lite, PolyLUT-Add achieve the same latency (with maximum frequency (F_max), which was constrained at 378 MHz, 235 MHz and 529 MHz respectively in Ref. [8]). However, on the JSC-M Lite model, F_max is decreased. Therefore, we use the JSC-M Lite model as a case study to analyze its maximum frequency and clock cycles for pipeline strategies (1) and -(2). The results are shown in Table V. As expected, the separate pipeline registers for each layer (strategy-(1)) do not affect overall system performance, whereas strategy-(2) results in the lowest overall latency with lower F_max. We suggest that the best choice will be dependent on specific system requirements.

We conducted additional experiments with PolyLUT-Add using the setup in Table IV. This utilizes lower F compared with the PolyLUT setup in Table I. $A = 2$ is used for all models (which are denoted as "HDR-Add2", "JSC-XL-Add2", "JSC-M Lite-Add2", "NID-Add2"). We also reduced the layer

sizes in the DNN model for the UNSW-NB15 dataset. These configurations were found to reduce area whilst maintaining comparable accuracies. Optimization of these parameters may further improve results for specific applications.

Table III shows the results and comparisons with prior works. Notably, PolyLUT applied $D = 4$ for HDR, JSC-XL and NID Lite models and $D = 6$ for the JSC-M Lite model, while PolyLUT-Add used smaller D. For comparable accuracy, the proposed PolyLUT-Add achieved a LUT reduction of $4.6\times$, $5.0\times$, $7.7\times$ and $1.3\times$ for the MNIST, JCS-XL, JSC-M Lite and UNSW-NB15 benchmarks respectively.

Finally, we studied latency with comparable accuracy. Pipeline strategy-(2) in Figure 5 was used to minimize the number of clock cycles. Compared with PolyLUT, this approach achieved a $2.2\times$, $1.7\times$, $1.2\times$ and $1.2\times$ decrease for the four benchmarks respectively. These significant reductions are attributed to lower polynomial degree D, and lower F.

V. CONCLUSION

We introduced PolyLUT-Add, a novel technique designed to enhance connectivity between neurons in LUT-based networks to deploy DNNs at the edge efficiently. By combining base PolyLUT models, our approach mitigates scalability issues associated with conventional implementations and significantly improves efficiency. Specifically, we demonstrated that by utilizing a configuration of $A = 2$, PolyLUT-Add with a lower polynomial degree D and fan-in F are sufficient to achieve comparable accuracy to PolyLUT. Over our benchmarks, PolyLUT-Add achieved reductions in LUT consumption by factors of 1.3-7.7 with a 1.2-2.2 times decrease in latency. The PolyLUT-Add architecture enhances LUT-based neural network performance in terms of area efficiency and latency.

Future work could be developing a targeted optimization technique for the individualized adjustment of F, A and β parameters within each layer or neuron, aiming to substantially boost network accuracy, reduce latency, and optimize area efficiency.

REFERENCES

[1] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 751–766, 2018.

[2] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran, *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[3] T. Murovic and A. Trost, "Massively parallel combinational binary neural networks for edge processing," *Elektrotehniski Vestnik*, vol. 86, no. 1/2, pp. 47–53, 2019.

[4] B. Lou, D. Boland, and P. Leong, "Fsead: A composable fpga-based streaming ensemble anomaly detection library," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 3, pp. 1–27, 2023.

[5] E. Wang, J. J. Davis, P. Y. Cheung, and G. A. Constantinides, "LUTNet: Rethinking inference in FPGA soft logic," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 26–34, IEEE, 2019.

[6] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "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, 2020.

[7] M. Nazemi, G. Pasandi, and M. Pedram, "Energy-efficient, low-latency realization of neural networks through boolean logic minimization," in *Proceedings of the 24th Asia and South Pacific design automation conference*, pp. 274–279, 2019.

[8] M. Andronic and G. A. Constantinides, "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.

[9] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, "Efficient acceleration of deep learning inference on resource-constrained edge devices: A review," *Proceedings of the IEEE*, vol. 111, no. 1, pp. 42–91, 2022.

[10] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "Rebnet: Residual binarized neural network," in *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*, pp. 57–64, IEEE, 2018.

[11] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[13] Xilinx Inc., "Vivado design suite user guide (ug949)," 2020.

[14] A. Pappalardo, "Xilinx/brevitas," 2023.

[15] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, *et al.*, "Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, p. 015001, 2020.

[16] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.

[17] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. Di Guglielmo, P. Harris, J. Krupa, *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.

[18] T. Murovič and A. Trost, "Genetically optimized massively parallel binary neural networks for intrusion detection systems," *Computer Communications*, vol. 179, pp. 1–10, 2021.

[19] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, pp. 1–6, IEEE, 2015.

[20] 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*, pp. 65–74, 2017.

[21] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, *et al.*, "Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, p. 015001, 2020.