Enhancing LUT-based Deep Neural Networks Inference through Architecture and Connectivity Optimization

Binglei Lou, Ruilin Wu, Philip Leong

School of Electrical and Computer Engineering

The University of Sydney, 2006, Australia

Email: {binglei.lou,ruilin.wu,philip.leong}@sydney.edu.au

Abstract—Deploying deep neural networks (DNNs) on resource-constrained edge devices such as FPGAs requires a careful balance among latency, power, and hardware resource usage, while maintaining high accuracy. Existing Lookup Table (LUT)-based DNNs—such as LogicNets, PolyLUT, and NeuraLUT—face two critical challenges: the exponential growth of LUT size and inefficient random sparse connectivity.

This paper presents SparseLUT, a comprehensive framework that addresses these challenges through two orthogonal optimizations. First, we propose an architectural enhancement that aggregates multiple PolyLUT sub-neurons via an adder, significantly reducing LUT consumption by $2.0\times$–$13.9\times$ and lowering inference latency by $1.2\times$–$1.6\times$, all while maintaining comparable accuracy. Building upon this foundation, we further introduce a non-greedy training algorithm that optimizes neuron connectivity by selectively pruning less significant inputs and strategically regrowing more effective ones. This training optimization, which incurs no additional area and latency overhead, delivers consistent accuracy improvements across benchmarks—achieving up to a 2.13% gain on MNIST and 0.94% on Jet Substructure Classification compared to existing LUT-DNN approaches.

Index Terms—Dynamic Sparsity, FPGA, Neural Network, Lookup Table

I. INTRODUCTION

DEEP neural networks (DNNs) have significantly improved our ability to solve pattern recognition problems across diverse data formats, including images, video, speech, audio and text [1]. Field-Programmable Gate Arrays (FPGAs) provide a unique implementation platform for deploying DNNs as they enable better integration with other system components such as networks and video decoders, allow signal processing to be combined with DNNs, and can operate with lower energy and latency than graphics processing units, particularly in real-time inference tasks.
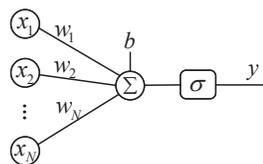

Fig. 1: Neuron computation with fan-in N. LUT-DNNs typically select a random subset (size $F \ll N$) of the inputs. SparseLUT is a training scheme to select the inputs while maximizing accuracy.
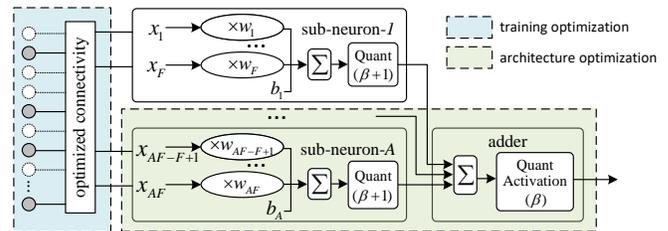

Fig. 2: Optimization strategies of SparseLUT

In an artificial neural network, the output of a neuron is given by

$$y = \sigma \left(\sum\nolimits_{k=1}^{N} w_k x_k + b \right) \qquad (1)$$

where N is the number of inputs, x_i are the inputs, w_i are the weights, b is a bias term and $\sigma(\cdot)$ is the activation function. Lookup Table (LUT)-based DNNs (LUT-DNN) exploit FPGA-native elements to achieve exceptional area efficiency and latency. This is done by combining the multiplication, sum, and activation operations of Figure 1 (Equation 1 in a single LUT). Recent approaches, including LogicNets [2], PolyLUT [3], and NeuraLUT [4], introduce *a priori* weight sparsity by constraining each neuron's fan-in, F. Put another way, they constrain the weight vector $W = \{w_1, w_2, \ldots, w_N\} \in \mathbb{R}^N$ to be determined *a priori*, sparse and have a maximum of F non-zero values ($0 < F \ll N$).

The accuracy of existing LUT-based DNNs is fundamentally constrained by limited scalability and suboptimal random sparsity. To keep the truth table size tractable, both the input bit-width β and the fan-in F must remain small, significantly limiting the network's representational capacity. In addition, the prevalent use of randomly assigned sparse connections often results in inefficient input selection, further impeding accuracy.

Although prior architectural enhancements have attempted to improve accuracy by boosting the expressive power of individual neurons, such as replacing linear operations with polynomial functions [3] or sub-network structures [4], these approaches still retain the fundamental limitation that the LUT size grows exponentially with βF, and thus fail to address the scalability bottleneck. Meanwhile, existing sparsity optimization techniques, including post-training pruning [5–8]
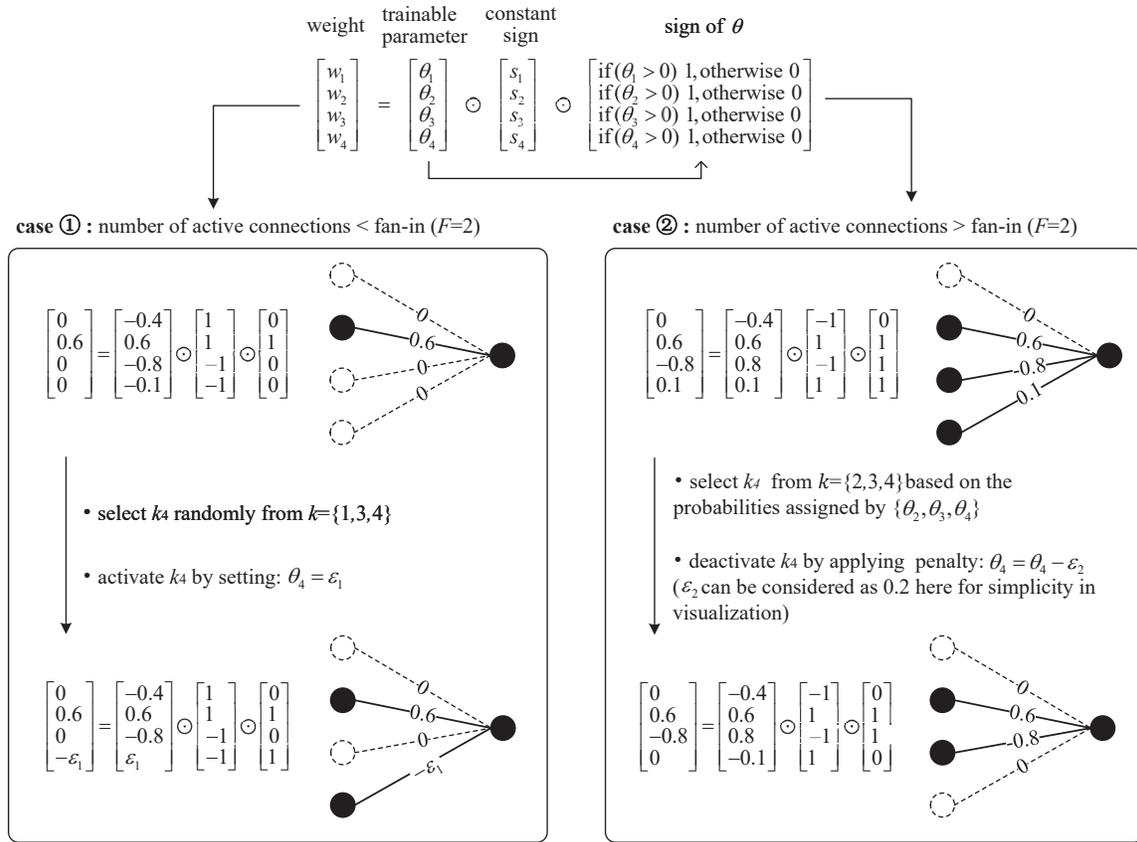
Fig. 3: Example iteration showing SparseLUT adjusting neuron connections to reach a target fan-in of 2, introducing new ones to increase the fan-in from 1 to 2 and eliminating connections to reduce the fan-in from 3 to 2. Note that the ϵ_2 parameter is typically set to a much smaller value, requiring multiple iterations to deactivate a connection.

and training-time sparsity methods [9–12], are designed for conventional DNNs. They typically operate at the model or layer level and fail to address the unique neuron-level fan-in constraints specific to LUT-based networks. These limitations underscore a critical gap and an opportunity for innovation.

This paper introduces SparseLUT, an architecture that addresses the unique challenges outlined through architectural and training algorithmic level optimization. Figure 2 shows the separate optimization respectively.

First, at the architecture level, we combine A copies of PolyLUT sub-neurons via an A-input adder to increase neuron fan-in. As shown in the right part of Figure 2, our approach restructures the neuron computation: weight multiplication, accumulation, batch normalization (BN), and quantized activation to two stages. 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. In this way, we can make better use of the FPGA fabric, where the lookup table cost of $\mathcal{O}(2^{\beta FA})$ is reduced to $\mathcal{O}(A \times 2^{\beta F} + 2^{A(\beta+1)})$.

Second, we proposed a non-greedy sparsification strategy through careful selection of the non-zero w_k values in Equation 1. Specifically, Figure 3, to be elaborated in Section III, illustrates our approach to training wherein active connections

are gradually modified to ultimately achieve a target fan-in of $F = 2$. As training progresses, new connections are added to neurons with fewer than F active connections (Case ①), while neurons with active connections exceeding F (Case ②) are deactivated. Over the training process, the target fan-in is achieved for all neurons.

The main contributions of this work are as follows:

1) At the computer architecture level, we propose an efficient reconstruction and extension of the Poly-LUT framework [3], which integrates A PolyLUT sub-neurons combined through an A-input adder, enhancing both scalability and accuracy.

2) At the tool level, we introduce the first non-greedy training algorithm for LUT-based neural networks that optimizes neuron input connectivity during training. This method imposes no additional LUT or routing resource overhead, as it solely modifies the selection strategy of the F inputs depicted in Figure 1.

3) These two optimizations are orthogonal. In our evaluation, we first apply the proposed architectural optimization to the original PolyLUT. At a comparable accuracy level, our method reduces LUT consumption by $4.8\times$ and $13.9\times$ on the MNIST and Jet Substructure Classification tasks, respectively, while also decreasing latency by $1.2\times$ to $1.6\times$. Furthermore, our proposed connectiv-
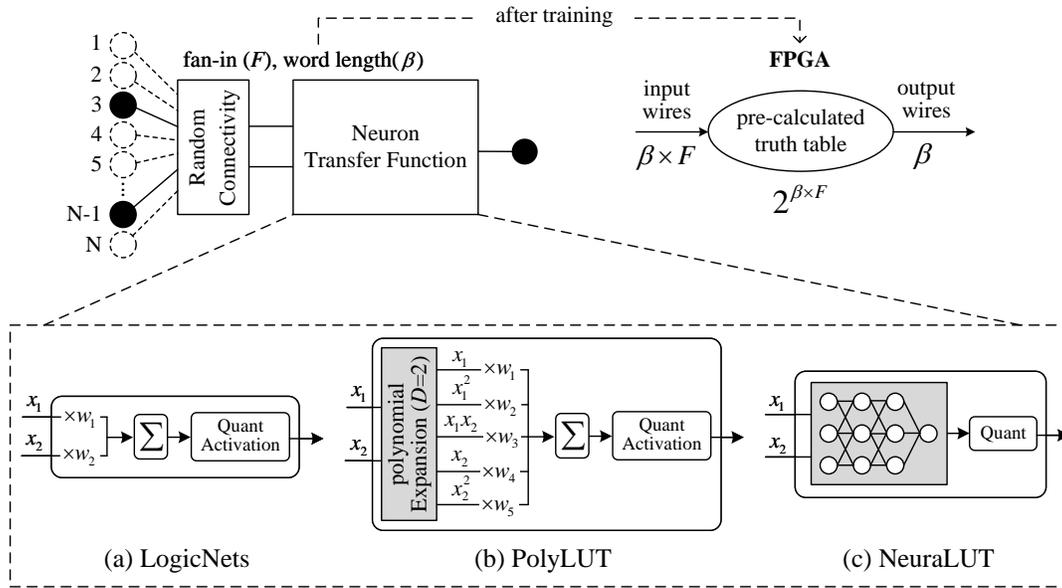
Fig. 4: Illustration of the generalized LUT-DNN architectures with sparse connectivity.

ity optimization consistently enhances accuracy across baseline LUT-DNN models, including LogicNets [2], PolyLUT [3], and NeuraLUT [4], without incurring any hardware or latency overhead. It achieves up to a 2.13% accuracy improvement on MNIST and a 0.94% gain on Jet Substructure Classification compared to random sparsity.

This work forms an extension of an earlier conference paper [13]; the key differences are the tool-level non-greedy training algorithm (Section III-B). The remainder of this paper is organized as follows: Section II reviews related work, including LUT-based DNNs and deep sparsification techniques. Section III describes the proposed method in detail, including the architecture-level optimization and non-greedy sparsification strategies. Section IV presents experimental results, showing accuracy and resource utilization improvements over existing methods. Finally, Section V concludes the paper and outlines future research directions.

II. BACKGROUND

Machine learning inference on FPGAs can be deployed through a variety of approaches. The most straightforward method is DSP-based inference, as exemplified by hls4ml [14], which enables developers to train models in Python using popular frameworks such as TensorFlow or PyTorch and automatically generate synthesizable HLS code. This approach leverages the full range of FPGA resources, including DSP blocks. Another line of work is XNOR-based inference, such as Binary Neural Networks (BNNs) [15], which employ 1-bit quantization to replace multipliers with efficient XNOR operations. Pure LUT-based methods can be further categorized into tree-based, differentiable LUT-based, and traditional DNN LUT-based approaches. For instance, TreeLUT [16] implements Gradient Boosted Decision Trees (GBDTs) using

only LUTs without relying on BRAMs or DSPs, while Differentiable Weightless Neural Networks (DWNs) [17] construct neural networks entirely from weightless nodes mapped to LUTs. This work primarily targets traditional DNN-based LUT designs, while comparisons with other categories are provided in the evaluation section.

A. LUT-based traditional DNN methods

LogicNets [2] is the first LUT-based DNN architecture that quantizes the inputs and outputs of each neuron and encapsulates 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 [3], proposed by Andronic *et al.*, further enhanced accuracy and reduced the number of required layers by introducing piecewise polynomial functions. Later, in the direction of continuing to improve the representation ability of each LUT, NeuraLUT [4] employs the network-in-network technique [18] and, therefore, maps entire sub-nets to a single LUT, enabling a deeper NN and better accuracy.

Figure 4 illustrates the main idea behind these LUT-DNN approaches. As shown in the top part, a maximum of F inputs are randomly selected from N nodes of the current layer to connect to each neuron of the next layer (only a single neuron of the next layer is demonstrated here). Additionally, the bit width of each neuron's inputs and outputs is quantized as β, and the rest of the parameters inside the neuron transfer function are maintained in full precision. Therefore, after training, the transfer function mapping an input vector $[x_1, x_2, \ldots, x_F]$ to the output node can be implemented using

TABLE I: Summary of dynamic sparsification approaches.

Method	Training	Drop (Prune) ↘	Regrowth ↗	*a priori* fixed fan-in constrain
GMP [9]	dense-to-sparse	magnitudes	✗	✗
DeepR [10]	sparse-to-sparse	stochastic	random	✗
RigL [11]	sparse-to-sparse	magnitudes	gradient	✗
SRigL [12]	sparse-to-sparse	magnitudes	gradient	✗
DWN [17]	sparse-to-sparse	magnitudes	✗	✓*
SparseLUT (ours)	flexible	stochastic+magnitudes	random	✓

* DWN is a special case here, designed specifically for weightless neural networks, and is not applicable to generalized weighted DNNs.

βF inputs in hardware, and hence its implementation requires $\mathcal{O}(2^{\beta F})$ LUTs in a pre-calculated truth table.

B. Sparsification

In practice, for most dense DNNs, a significant fraction of weights are redundant and can be set to zero with minimal effect on accuracy [8]. Research on sparsifying neural networks, known as network pruning, dates back to the late 1980s and mid-1990s. These techniques begin with a pre-trained, densely connected model and perform weight removal, where unimportant weights are identified and set to zero. Various approaches have been used as the removal criteria. For example, early work in 1989, Optimal Brain Damage [5], removed non-salient weights as determined via the diagonals of the Hessian matrix. In 1995, Thimm *et al.* [6] demonstrated that removing weights based on their magnitude was a simple yet effective technique. Gale *et al.* [7] further examined alternative pruning criteria on large-scale learning of transformers trained on WMT2014 English-to-German and ResNet-50 trained on ImageNet. They concluded that complex techniques such as L_0 regularization [19] and Variational Dropout [20] achieve similar accuracy-sparsity trade-offs compared to magnitude-based pruning. As for the LUT-DNN scope, PolyLUT [21] and NeuraLUT-Assemble [22] introduce a post-training pruning strategy based on weight ranking to replace the random connectivity. Specifically, after training, for each neuron, the weight branches are ranked according to their fan-in, and the top branches are selected in descending order of importance.

Gale *et al.* [7] also observed that unstructured sparse architectures learned through pruning alone cannot achieve as good performance as a model trained with joint sparsification and optimization. Thus, more recent methods commence from scratch rather than a pre-trained model, *i.e.*, they (1) start from the original initial training conditions and (2) update connectivity by doing weight removal and weight re-growth (optional) simultaneously. This can be further divided into two subcategories with representatives listed in Table I: (1) The training starts with a dense model and gradually removes connections during training to achieve the desired sparsity [9]. (2) The training begins with a sparse model to minimize training memory overhead, and during training, connections are dropped and regrown, guided by criteria such as weight magnitudes, gradients, or stochastically [10–12].

Gradual Magnitude Pruning (GMP) [9], removes connectivity from a dense initial solution during training until the

desired sparsity is reached. It begins with a dense network and removes weights with the smallest magnitudes. Its sparsity decreases monotonically until the target sparsity is reached.

In contrast, there are approaches that employ dynamic rewiring of neural networks [10–12]. This strategy is inspired by the human brain, where the majority of brain volume is occupied by white matter—the network of connections between neurons. In the brain, synaptic connectivity is highly dynamic, with the continuous formation and reorganization of synapses, particularly during learning [23–26].

Deep Rewiring (DeepR) [10] introduces a stochastic framework for dynamic rewiring. At initialization, DeepR employs a network with a limited number of random connections. During training, at each iteration, connections are removed if their weight updates lead to a sign change. A new connection is then randomly activated to preserve the target sparsity level. Our work extends this random walk mechanism, and further details are provided in subsequent sections. Rigged Lottery (RigL) [11] removes weights with the smallest magnitudes and regrows connections based on gradient magnitude during training. At each iteration, the number of dropped connections equals the number of regrown connections, maintaining a consistent sparsity level. RigL is designed to handle standard unstructured sparsity constraints at the layer level. Structured RigL (SRigL) [12] extends RigL by incorporating constant fan-in structured sparsity, which facilitates a compact weight matrix representation. Specifically, after training, the non-zero weights in each matrix exhibit a rectangular pattern. SRigL achieves its target fan-in target on average, rather than this being enforced on a per-neuron basis.

DWN's element-wise weight-free mapping is not directly applicable to our LUT-based weighted inference pipeline DWN [17] is a weightless neural network model, where its learnable mapping mechanism selects a fixed number of connections based on the largest weights during training. Specifically, its computation does not involve weighted summation, accumulation, nonlinear activation, or standard backpropagation. Instead, it provides a differentiable mechanism to decide the routing of inputs directly into LUTs, enabling training without conventional weights. Therefore, although DWN provides an insightful online learnable mapping, its weight-free, element-wise formulation is fundamentally different from the weighted computations underlying traditional LUT-based DNN inference, particularly for networks with customized architectures and training pipelines (e.g., the poly-
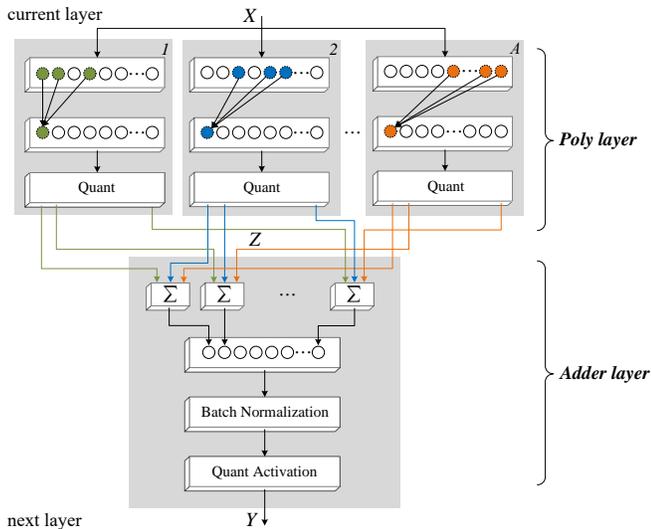
Fig. 5: A single-layer block diagram of the optimized architecture.

nomial formulation in PolyLUT and the network-in-network structure in NeuraLUT). As a result, its direct integration into a conventional weighted-DNN training workflow has not yet been demonstrated in the existing literature.

III. METHODOLOGY

A. DNN Architecture Optimization

Figure 5 outlines our proposed DNN architecture. Compared with Figure 4, 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. The connectivity between the Poly-layer and the Adder-layer is fixed to perform the addition operation for each neuron. 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) outputs 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 2.

B. Dynamic Sparsification

1) Model Representation: Unlike conventional DNNs, which focus solely on updating the network weights and biases, SparseLUT also incorporates weight removal and regrowth under a set of predefined constraints. A key factor enabling this is its weight representation. We follow the approach of DeepR [10] where each connection k in the network is represented by two components:

- A **trainable connection parameter** $\theta_k \in \mathbb{R}$, which controls both the magnitude of the connection and its status (active or inactive).

- A **predefined, non-trainable constant sign** $s_k \in \{-1, 1\}$, initialized randomly, which encodes the fixed polarity of the connection.

Referring to Figure 3, we use $W = \{w_1, w_2, \ldots, w_N\} \in \mathbb{R}^N$ to denote the weight vector of a neuron in Equations 1.

Algorithm 1 outlines the procedure for determining the effective weight vector for each neuron in SparseLUT. The algorithm begins with an initial weight vector $W_0 \in \mathbb{R}^N$, where the magnitude of each element is used as the initial value for θ_k. This ensures dense connectivity under the initial conditions, with all connections corresponding to θ_k being non-negative. The initial sparsity of the network is determined by the element-wise product of $|W_0|$ and a binary sparsity mask vector *is_con*, where only the element F_i is set to 1, and all other elements are set to 0. To finalize the effective weight vector W, an indicator function $1(\cdot)$ is applied to the product $\theta_k \odot s_k$, controlling the active status of each connection.

Algorithm 1: Weight Mapping Procedure

Input: Weight dimensions for W, initial fan-in F_i.

1 $W_0 \leftarrow$ Randomly initialize weights W_0 from a standard normal distribution with the same dimensions as W;

2 $is\_con \leftarrow$ Generate a binary sparsity mask is_con such that each output neuron is connected to F_i randomly chosen input neurons;

3 $\theta \leftarrow |W_0| \odot is\_con$ (Compute the initial θ values as the element-wise product);

4 $sign \in \{-1, +1\}$ Randomly initialize weight signs with uniform probability;

5 $W \leftarrow \theta \cdot sign \cdot 1(\theta > 0)$ (Compute the final weights, where $1(\cdot)$ is the indicator function that sets negative values of θ to zero);

6 **return** *Final weight vector W*

---

2) Sparse Training: The SparseLUT training procedure, as outlined in Algorithm 2, is a non-greedy algorithm. Each iteration updates the connection parameters and adjusts the network's sparsity to enforce the target fan-in F_o sparsity during training.

At each backpropagation training step [27], only the parameters θ_k of active connections ($\theta_k > 0$) are updated according to the rule in Line 6. Given that the learning rate is η, the term $\eta \frac{\partial E(\theta)}{\partial \theta_k}$ represents the derivative of the error function, while $\eta \alpha$ is regularization term (α is any regularization function). The final term, ηv_k, originally introduced in DeepR [10], implements a random walk scheme in the parameter space. Here, $v_k \sim N(0, G^2)$ denotes a noise matrix with the same dimensions as θ_k, a mean of 0, and a standard deviation of G. Following this update, any connection for which θ_k becomes negative is deactivated.

For each neuron, the difference between the existing number of active connections and the target fan-in is computed as R. A negative R indicates that the number of active connections is smaller than the target fan-in, prompting the activation of $|R|$ inactive connections k' selected uniformly at random

---

Algorithm 2: SparseLUT Training Procedure

Input: Target fan-in F_o, sparse-to-sparse training starting point T, learning rate η, regularization coefficient α, and noise $v_k \sim N(0, G^2)$.

Output: Feature mask $\mathcal{M}$

1 **Initialization:**

2 $W \leftarrow$ defined in Algorothm 1;

3 **Training:**

4 **for** *each training step t* **do**

5   **for** *all active connections k ($\theta_k > 0$)* **do**

6   Update $\theta_k \leftarrow \theta_k - \eta \frac{\partial E(\theta)}{\partial \theta_k} - \eta\alpha + \eta v_k$;

7   **if** $\theta_k < 0$ **then** set connection k non-active ;

8     $R \leftarrow$ number of active connections - F_o;

9   **if** $R < 0$ **then**

10   Select $|R|$ non-active connections k' with uniform probability and activate them;

11   Set $\theta_{k'} \leftarrow \epsilon_1$;

12   **else**

13   **if** $t < T$ **then**

14   Rank the θ values of all active connections;

15   Select $|R|$ active connections k' with the lowest rank probabilities and apply penalties;

16   Update $\theta_{k'} \leftarrow \theta_{k'} - \epsilon_2$;

17   **else**

18   Rank the θ values of all active connections;

19   Select $|R|$ active connections k' with the lowest rank probabilities and deactivate them;

20   Set $\theta_{k'} \leftarrow 0$;

21 **return** Feature mask $\mathcal{M}$, where $\mathcal{M}_k = 1$ if $\theta_k > 0$, otherwise $\mathcal{M}_k = 0$;

(Line 10). These connections are initialized with a small value ϵ_1 to allow them to rejoin the backpropagation process without dominating the updates immediately after reactivation.

Conversely, a positive R means that $|R|$ active connections need to be deactivated to meet the fan-in constraint. Different from approaches in Table I, SparseLUT employs a two-phase training strategy for this deactivation process:

1) progressive sparsification phase: During the early stages of training ($t < T$), a small penalty term ϵ_2 is subtracted from $\theta_{k'}$ in each iteration (Line 16). This gradual penalty reduces the values of smaller parameters to a greater extent, under the assumption that connections with smaller weights contribute less to the neuron's output. The speed of connectivity convergence is significantly influenced by the ϵ_2. As $\theta_{k'}$ becomes negative, these connections are eliminated. Specifically, a large value of ϵ_2 can cause nodes with relatively larger weights to change state immediately. This stage relaxes the strict requirement that the number of dropped and regrown connections must always match, as used in conventional

regrowth approaches such as DeepR [10], RigL [11], and SRigL [12]. We will later show that this greater flexibility during training leads to consistently improved accuracy.

2) fine-tuning phase: In the later stage of training ($t \geq T$), the model strictly enforces the fan-in constraint. Connections are directly deactivated when $|R| > 0$ (Line 20). This phase focuses on fine-tuning the remaining connections through periodically rewiring eliminated connections and selecting the same number of alternatives to maintain the fan-in constraint for each neuron.

Upon completing the training process, the final generated output of the algorithm is formed as the feature mask $\mathcal{M}$, where each element $\mathcal{M}_k$ indicates the status of connection k (Line 21).

The computational cost of SparseLUT, as shown in Algorithm 2, primarily arises from the connectivity search, specifically the Prune and Regrowth steps. When $R < 0$ (regrowth), selection is performed from all inactive candidates ($N - F \approx N$), while for $R > 0$ (pruning), ranking is conducted on active candidates. Empirically, the number of active candidates quickly stabilizes near the target fan-in value F_o during extended training periods. As a result, the ranking overhead mainly depends on F_o. Therefore, the training time is primarily influenced by the model size, the number of epochs, and F_o.

C. ToolFlow
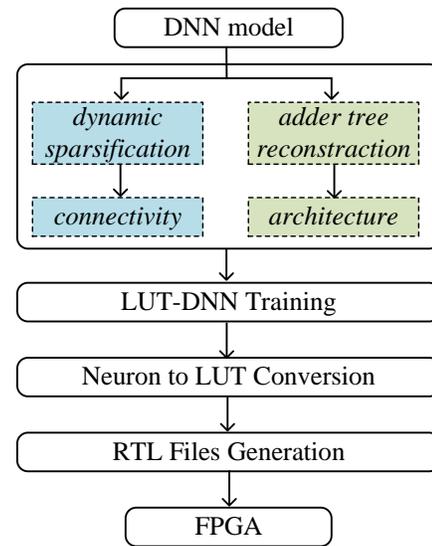
Fig. 6: Workflow of SparseLUT.

Figure 6 illustrates the overall workflow of the proposed SparseLUT framework, which is readily applicable to existing LUT-based DNN designs [2–4]. The process begins with the same model configuration as standard LUT-DNNs, thereby maintaining full compatibility with their training pipelines. On the left side of the flow, SparseLUT generates a *connectivity feature mask* $\mathcal{M}$, which conforms to the structural requirements of existing LUT-DNNs and directly replaces

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3654908

7

the randomly generated connectivity shown in Figure 4. In addition, the network architecture is modified to incorporate the fan-in factor A as a configurable parameter. This enables the reconstruction of the network topology using an adder tree structure.

Following this, LUT-DNN training is conducted offline using PyTorch [28]. Integration with the Brevitas library [29] enables quantization-aware training (QAT), ensuring that the trained model is suitable for hardware deployment. Post-training, the model's weights are transformed into lookup tables. This transformation begins by analyzing the quantized activations to determine each neuron's input range. For sub-neuron layers, all possible input combinations are generated based on the parameters β and F; for adder layers, combinations are generated using β and A. These input vectors are then propagated through their respective layers to obtain output values, forming input–output pairs that constitute the contents of the lookup tables.

These lookup tables are subsequently used to generate Register Transfer Level (RTL) code in Verilog, capturing the Boolean functions implemented by each neuron. Finally, the design is synthesized onto hardware using the AMD/Xilinx Vivado toolchain [30].

IV. Results

A. Datasets

We evaluated the proposed SparseLUT design using two widely adopted datasets for ultra-low latency inference. These datasets align with the evaluation benchmarks of existing techniques, including LogicNets [2], PolyLUT [3], and NeuraLUT [4]. In addition, to evaluate the generalizability of the optimized connectivity on more complex tasks, we also include an ablation study on the classification accuracy with the CIFAR-10 dataset. Since all the above LUT-DNN methods didn't conduct any CIFAR-10-based experiment and our focus is not on exploring the optimal hyperparameter search space here, the setups used for MNIST are directly applied to the CIFAR-10 task.

1) *Handwritten Digit Recognition*: In timing-critical sectors such as healthcare, medical imaging, and real-time object tracking, low latency is crucial. 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, so the MNIST [31] is utilized to compare our work with other LUT-based DNNs. MNIST is a dataset for handwritten digit recognition tasks with 28×28 pixels as input 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 [32–35] employed neural networks on an FPGA for this task to provide real-time inference capabilities. We also use the JSC dataset formulated from Ref. [33] to

evaluate our work, with the dataset having 16 substructure properties as input and 5 types of jets as outputs. FPGA-based classification in this task must be pipelined to manage a data rate of 40 MHz while ensuring the response latency remains under a microsecond.

3) *Object Recognition*: Similar to handwritten digit recognition, object recognition in real-world scenarios also demands low latency to enable fast and reliable decision-making. To evaluate our method, the CIFAR-10 [36] dataset is employed, which provides a more complex benchmark compared to MNIST. CIFAR-10 consists of 32×32 color images from 10 diverse object categories, making the classification task more challenging.

We emphasize that the architectural optimization and the connectivity training optimization proposed in SparseLUT are orthogonal and can be applied independently. To facilitate an ablation study, we first evaluate the model that incorporates only the architectural optimization. For consistency with the terminology used in our published work [13], we refer to this architecture-only variant as PolyLUT-Add.

B. Architectural Optimization Evaluation

Using AdamW as the optimizer [28], we trained the smaller models/datasets (JSC-M 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, JSC-M Lite) and MNIST, respectively. We inherit these configurations from PolyLUT [3] to ensure consistency in evaluation.

The hardware evaluation is compiled using Vivado 2020.1 on the `xcvu9p-flgb2104-2-i` FPGA part; the `Flow_PerfOptimized_high` settings were configured to perform synthesis in the `Out-of-Context` (`OOC`) mode, and Default values were used as the Place & Route settings.

1) PolyLUT-Add vs. original PolyLUT: As shown in Table II, for $A = 2$, PolyLUT-Add achieves accuracy improvements of 2.7%, 0.6%, and 2.3% over PolyLUT on the MNIST and Jet Substructure classification benchmarks, respectively, with only a 2–3× increase in LUT size. In contrast, achieving similar accuracy gains in PolyLUT by increasing the fan-in F results in a 256–1024× increase in LUT size. These results demonstrate that PolyLUT-Add offers a more efficient trade-off between accuracy and hardware cost. Furthermore, it's noteworthy that the RTL Generation time cost also correlates with the number of lookup table entries; it follows that a direct increase in fan-in would incur exponentially higher RTL Generation time costs. (The RTL Generation time was measured on a desktop with Intel(R) Core(TM) i7-10700F @2.9GHz and 64GB memory.)

2) PolyLUT-Add vs. Deeper and Wider PolyLUT: The accuracy comparison between PolyLUT-Add and the other two variants of PolyLUT is further tested (the baseline PolyLUT setup is shown in Table III):

1) **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 III. For example, for JSC-M Lite, if $\mathbb{D} = 2$, the

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 entries↓	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

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

TABLE III: Setups of PolyLUT baseline models for Figure 7.

Model	Neurons per layer	β	F	D
MNIST	256, 100, 100, 100, 100, 10	2	6	1,2
JSC-XL[1]	128, 64, 64, 64, 5	5	3	1,2
JSC-M Lite	64, 32, 5	3	4	1,2

PolyLUT with $D = 1$ is equivenlent to LogicNets.
Remarks: [1] $\beta_i = 7$, $F_i = 2$;

hidden layer is doubled, meaning the neurons per layer become (64,64,32,32,5).

2) **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 is applied to models in Table III. Once again, for JSC-M Lite, if $\mathbb{W} = 2$, the neurons per layer become (128,64,5).

Figure 7 shows the accuracy with the above configurations. PolyLUT-Add achieves the highest accuracy against all baselines on all datasets for both the linear ($D = 1$) and non-linear ($D = 2$) cases.

3) Comparisons with Comparable Accuracy: Finally, we studied area and latency with comparable accuracy. Table IV shows the comparisons between PolyLUT-Add and PolyLUT. $A = 2$ and a lower F are used for all PolyLUT-Add models (which are denoted as "HDR-Add2", "JSC-XL-Add2", "JSC-M Lite-Add2", see Table V for detailed setup). Notably, PolyLUT applied $D = 4$ for HDR and JSC-XL models and $D = 6$ for the JSC-M Lite model, while PolyLUT-Add used a smaller D.

The PolyLUT-Add achieved a LUT reduction of 4.8×, 6.5×, and 13.9× for the MNIST, JCS-XL, and JSC-M Lite benchmarks, respectively. As for latency, his approach

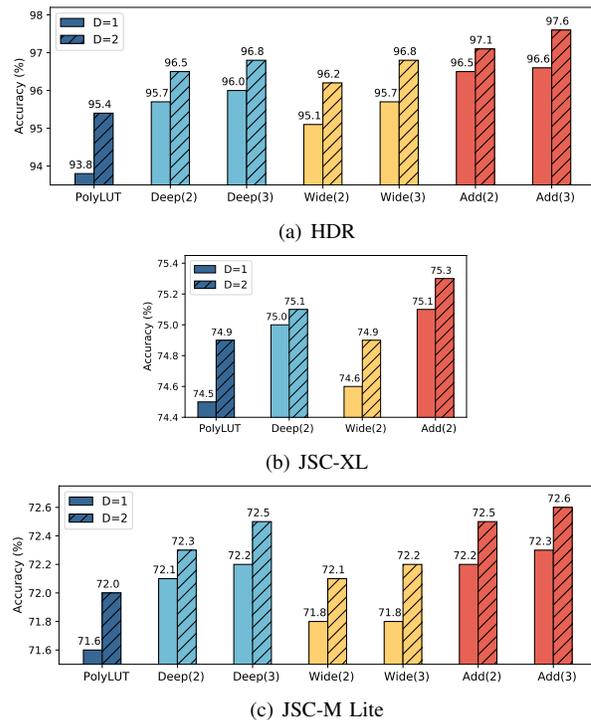

(a) HDR

(b) JSC-XL

(c) JSC-M Lite

Fig. 7: 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.

achieved a 1.6×, 1.3×, and 1.2× decrease for the three benchmarks, respectively. These significant reductions are attributed to a lower polynomial degree D and lower F.

TABLE IV: Comparison results between PolyLUT-Add and PolyLUT. The frequency and the area are collected from the Vivado post Place & Route reports.

Dataset	Model	Accuracy↑	LUT	FF	DSP	BRAM	F_max(MHz)↑	Latency(ns)↓
MNIST	**PolyLUT-Add (HDR-Add2, D=3)**	**96%**	**14810**	**2609**	**0**	**0**	**625**	**10**
	PolyLUT (HDR, D=4) [3]	96%	70673	4681	0	0	378	16
JSC (high accuracy)	**PolyLUT-Add (JSC-XL-Add2, D=3)**	**75%**	**36484**	**1209**	**0**	**0**	**315**	**16**
	PolyLUT (JSC-XL, D=4) [3]	75%	236541	2775	0	0	235	21
JSC (low accuracy)	**PolyLUT-Add (JSC-M Lite-Add2, D=3)**	**72%**	**895**	**189**	**0**	**0**	**750**	**4**
	PolyLUT (JSC-M Lite, D=6) [3]	72%	12436	773	0	0	646	5

∗: Paper reports "LUT+FF"

C. Sparsification Evaluation

To evaluate SparseLUT as a general approach for improving the accuracy of LUT-based DNNs, we tested its performance across several state-of-the-art methods, including LogicNets [2], PolyLUT [3], NeuraLUT [4], and our PolyLUT-Add variant enhanced with architectural optimization. Similar to the tree-structured fan-in expansion technique used in PolyLUT-Add, NeuraLUT-Assemble [22] is a fan-in optimized variant of NeuraLUT, which is also evaluated in this work. We compare the effectiveness of random connectivity against the optimized connectivity provided by SparseLUT. In addition, NeuraLUT-Assemble also reports results using static sparsification, a post-training pruning strategy based on weight ranking; this static sparsification performance is also included in our evaluation.

When searching for the optimized sparse connectivity, all modes follow the same training setup to ensure consistency. The AdamW optimizer [28] in PyTorch is used, and each model is trained for 300 epochs. The threshold T is set to 240 epochs, corresponding to 80% of the total training duration. The hyperparameters $\epsilon_1 = 10^{-12}$ and $10^{-5} \le \epsilon_2 \le 10^{-4}$ are employed. The F_i is set to N to set the training to start from a dense model. The primary objective is to obtain the optimized sparsity pattern $\mathcal{M}$. Full-precision training is used for faster convergence and better precision in identifying critical connections. The results presented in this section are based on these configurations, though further optimization or additional training techniques may enhance performance for specific applications.

The experiment comprises two steps: (1) derive the optimized connectivity feature mask $\mathcal{M}$. (2) Replace the original random connectivity in the baseline models with $\mathcal{M}$ and retrain the baselines.

To ensure a fair comparison, the configurations and training hyperparameters in Step 2 are consistent with the baseline setups as reported in the original papers and summarized in Table V. All results in this section are obtained by retraining the models.

In Step 1, we evaluate three sparse modes with a fully connected baseline, as shown in Table VI:

- **Random Sparsity**: Reflects the approach currently employed by existing techniques [2–4, 13].
- **DeepR**∗: The original DeepR [10] cannot be directly applied to our problem. We have developed a revised version DeepR∗, that supports fixed fan-in, which we use as a comparison baseline.

- **SparseLUT**: Reflects the proposed method.

We then conducted a case study on the MNIST dataset, focusing on two analyses: (1) visualizing the connectivity learned by SparseLUT using heatmaps to evaluate its quality and (2) comparing test accuracy between LUT-DNN models utilizing learned versus random connectivity.

1) Connectivity Distribution Visualization: We lack ground truth to evaluate connectivity quality before testing its performance in LUT-DNNs. However, for the MNIST dataset, where handwritten digits typically appear centered [1], it is reasonable to expect the most important connections to concentrate in the central region.

We adopted a fully connected model and three sparse models (random, DeepR∗, SparseLUT) with layer neurons $\{256, 100, 100, 100, 10\}$, consistent with the HDR-5L model from Table V. After training, we averaged the absolute values of the first-layer weight matrix $|W| \in \mathbb{R}^{784 \times 256}$ along the second axis and reshaped it into $W_{ave} \in \mathbb{R}^{28 \times 28}$, aligning with the input image dimensions.

Figure 8 is a visualization of W_{ave} as heatmaps. The fully connected model (last sub-figure) shows higher weights concentrated in the central region, aligning with our assumptions.

The first three sub-figures in Figure 8 show heatmaps for sparse models, where only $F = 6$ elements in each row of $|W|$ are non-zero. The random sparsity model shows a uniform distribution, reflecting unstructured connectivity. DeepR∗ exhibits a trend of concentrated central connectivity, suggesting meaningful adaptation to the task. SparseLUT's heatmap more closely resembles the fully connected case, indicating that SparseLUT effectively learns an optimized connectivity pattern, prioritizing the central region in alignment with the dataset's structure.

2) Comparison with baseline works: Figure 9 further compares the test accuracy of applying different connectivity patterns to five models—PolyLUT ($D = 1$), PolyLUT ($D = 2$), PolyLUT-Add ($D = 1$), PolyLUT-Add ($D = 2$), and NeuraLUT—under three sparsity configurations: random sparsity (boxplots with five seeds), DeepR∗, and SparseLUT. For DeepR∗ and SparseLUT, the optimized accuracy points are plotted separately for each model.

The results show that both DeepR∗ and SparseLUT outperform random sparsity, but the advantages of SparseLUT are more pronounced. DeepR∗ achieves limited improvements over random sparsity in certain cases, such as PolyLUT

[1]The handwritten digits dataset visualization:

TABLE V: Setups of baseline models.

Dataset	Method	Model	Neurons per layer	β	F	D
MNIST	PolyLUT	HDR	256, 100, 100, 100, 100, 10	2	6	1,2
	PolyLUT-Add	HDR-Add2	256, 100, 100, 100, 100, 10	2	4	1,2
	NeuraLUT	HDR-5L	256, 100, 100, 100, 10	2	6	—
	NeuraLUT-Assemble	HDR-5L-0.1	2160, 360, 2160, 360, 60, 10	1	6	—
JSC (high accuracy)	PolyLUT	JSC-XL[1]	128, 64, 64, 64, 5	5	3	1,2
	PolyLUT-Add	JSC-XL-Add2[2]	128, 64, 64, 64, 5	5	2	1,2
	NeuraLUT	JSC-5L[1]	128, 128, 128, 64, 5	4	3	—
	NeuraLUT-Assemble	jsc-cernbox[3]	320, 160, 80, 40, 20, 10	4	2	—
JSC (low accuracy)	PolyLUT	JSC-M Lite	64, 32, 5	3	4	1,2
	PolyLUT-Add	JSC-M Lite-Add2	64, 32, 5	3	2	1,2
	NeuraLUT	JSC-2L	32, 5	4	3	—
CIFAR-10	PolyLUT	HDR	256, 100, 100, 100, 100, 10	2	6	1,2
	PolyLUT-Add	HDR-Add2	256, 100, 100, 100, 100, 10	2	4	1,2
	NeuraLUT	HDR-5L	256, 100, 100, 100, 10	2	6	—
	NeuraLUT-Assemble	HDR-5L-0.1	2160, 360, 2160, 360, 60, 10	1	6	—

PolyLUT with $D = 1$ is equivenlent to LogicNets.
Remarks: [1] $\beta_i = 7$, $F_i = 2$; [2] $\beta_i = 7$, $F_i = 1$; [3] $\beta_i = 8$, $F_i = 1$;
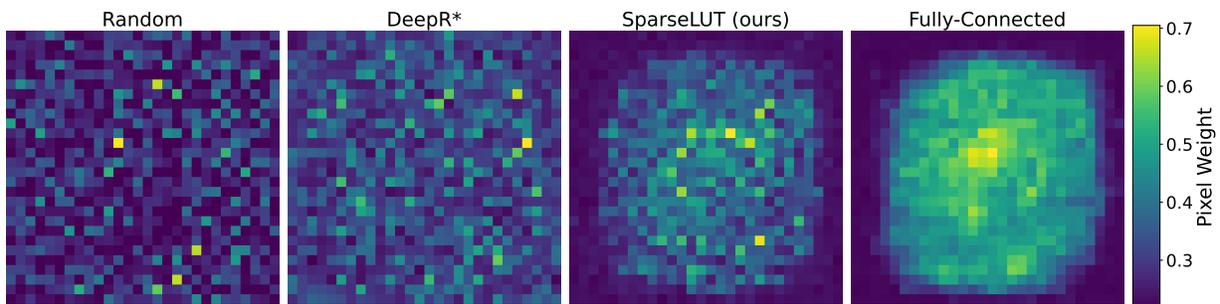

Fig. 8: Heatmaps of the average weight matrix for the first layer in three sparse modes: Random Sparsity, DeepR*, and SparseLUT, with a Fully Connected mode as a baseline.

TABLE VI: Sparsity Modes

Mode	Fixed fan-in constrain
Fully Connected	—
Random Sparsity	✓
DeepR* [10]	✓
SparseLUT (ours)	✓

Fig. 9: Test accuracy for five models with random sparsity (5 seeds), DeepR*, and SparseLUT.

($D = 1$) and PolyLUT-Add ($D = 2$), where its accuracy points overlap with the box regions of random sparsity, despite being slightly above the mean line. In contrast, SparseLUT consistently delivers significant gains, exceeding the random sparsity range by 1.4%–2.1%. This performance gap highlights the effectiveness of SparseLUT's non-greedy training approach, which provides a larger parameter search space in the early stages, allowing the model to explore a broader range of sparsity patterns.

3) Test Accuracy Analysis: In this subsection, we present the performance of SparseLUT across multiple baseline models. For the MNIST and JSC datasets, accuracy values under random sparsity are obtained from referenced papers and open-source implementations. As shown in Table VII, Sparse-LUT consistently outperforms random sparsity configurations across all baseline models, as well as the static-sparsification-

enhanced accuracy of the NeuraLUT-Assemble model (with random accuracy reported on the left and optimized accuracy on the right).

To further assess the generalizability of the proposed connectivity optimization on model accuracy, results on the CIFAR-10 dataset are reported in Table VII. The comparison demonstrates the superiority of SparseLUT over random sparsity across the evaluated models. The relatively low accuracy achieved on CIFAR-10 can be attributed to the simple

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3654908

11

TABLE VII: Performance comparison with accuracy reported from prior works. Accuracy^{+opt} indicates the accuracy with optimized connectivity by SparseLUT.

Dataset	Setup	Model	Accuracy	Accuracy^{+opt}	Dense
MNIST	HDR(D=1) [3]	PolyLUT	93.76%	**95.89%**	98.55%
	HDR(D=2) [3]	PolyLUT	95.42%	**97.13%**	
	HDR-Add2(D=1) [13]	PolyLUT-Add	95.09%	**96.52%**	
	HDR-Add2(D=2) [13]	PolyLUT-Add	95.87%	**97.26%**	
	HDR-5L [4]	NeuraLUT	95.20%	**96.96%**	98.61%
	HDR-5L-0.1 [22]	NeuraLUT-Assemble	95.52%, 97.90%	**98.10%**	98.87%
JSC (high accuracy)	JSC-XL(D=1) [3]	PolyLUT	74.48%	**74.65%**	75.46%
	JSC-XL(D=2) [3]	PolyLUT	74.94%	**75.01%**	
	JSC-XL-Add2(D=1) [13]	PolyLUT-Add	74.64%	**74.74%**	
	JSC-XL=Add2(D=2) [13]	PolyLUT-Add	74.98%	**75.04%**	
	JSC-5L [4]	NeuraLUT	74.93%	**74.98%**	75.31%
	jsc-cernbox [22]	NeuraLUT-Assemble	74.85%, 75.00%	**75.03%**	75.26%
JSC (low accuracy)	JSC-M Lite(D=1) [3]	PolyLUT	71.65%	**72.10%**	72.48%
	JSC-M Lite(D=2) [3]	PolyLUT	71.98%	**72.15%**	
	JSC-M Lite-Add2(D=1) [13]	PolyLUT-Add	71.53%	**71.96%**	
	JSC-M Lite-Add2(D=2) [13]	PolyLUT-Add	71.90%	**72.24%**	
	JSC-2L [4]	NeuraLUT	72.01%	**72.95%**	73.34%
CIFAR-10	HDR(D=1) [3]	PolyLUT	38.83%	**40.99%**	60.95%
	HDR(D=2) [3]	PolyLUT	42.66%	**46.10%**	
	HDR-Add2(D=1) [13]	PolyLUT-Add	40.81%	**42.74%**	
	HDR-Add2(D=2) [13]	PolyLUT-Add	44.27%	**46.82%**	
	HDR-5L [4]	NeuraLUT	42.98%	**46.38%**	61.38%
	HDR-5L-0.1 [22]	NeuraLUT-Assemble	44.84%	**46.27%**	62.51%

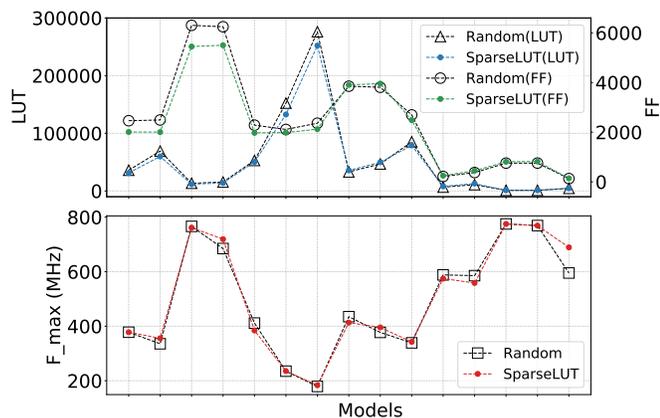LogicNets [2] cases are equivalent to all baselines with $D = 1$.

Fig. 10: Post place & route hardware comparison between models (in the same order from left to right (PolyLUT to NeuraLUT) as Table VII) with/without connectivity from SparseLUT.

architectural design of existing LUT-based DNNs. Developing LUT-based DNNs that achieve practical accuracy (e.g., exceeding 90%) on complex tasks such as CIFAR-10, or even larger benchmarks, remains an open challenge in the FPGA community.

To assess the ceiling for accuracy improvement achievable through optimized sparsity, we use the fully connected model of LogicNets (*i.e.*, polynomial degree $D = 1$) as a fair comparison point for PolyLUT ($D = 1$) and PolyLUT-Add ($D = 1$) configurations. For models with $D > 2$ or NeuraLUT's network-in-network architecture, fully connected implementations result in an exponential increase in parameters, rendering such implementations impractical. Consequently, the dense accuracy shown in the last column of Table VII for these configurations is considered a reasonable indicator rather than an upper bound.

The results highlight notable differences in accuracy deltas (δ = *Dense Accuracy - Random Sparsity Accuracy*) across datasets and models. For MNIST and CIFAR-10, SparseLUT achieves a substantial accuracy boost, attributed to the large δ exceeding 3%. In contrast, for most models in the JSC dataset, this δ is much smaller—approximately 0.8%—explaining the comparatively modest accuracy improvements achieved with optimized connectivity. A notable exception within the JSC dataset is the JSC-2L model on NeuraLUT (last row), which exhibits a δ of 1.33%. SparseLUT significantly boosts its accuracy from 72.01% to 72.95%, likely due to the higher word length ($\beta = 4$) of the JSC-2L model compared to the 3-bit word length ($\beta = 3$) in other JSC-M Lite models.

These findings reveal that SparseLUT's effectiveness is closely tied to the connectivity optimization potential of the underlying model, as reflected in the δ between dense and sparse configurations. This highlights the critical role of dataset characteristics and model architecture in determining the extent of accuracy improvements achievable with Sparse-LUT.

TABLE VIII: Comparison of generalized LUT-DNN against other ultra-low latency machine learning techniques.

Dataset	Model	Accuracy↑	Accuracy^{+opt}↑	LUT	FF	DSP	BRAM	F_max(MHz)↑	Latency(ns)↓
MNIST	PolyLUT (HDR, D=2) [3]	95.42%	97.13%	59790	2003	0	0	356	17
	PolyLUT-Add (HDR-Add2, D=2) [13]	95.87%	97.26%	13890	2768	0	0	635	8
	NeuraLUT (HDR-5L) [4]	95.20%	96.96%	50271	1973	0	0	384	13
	NeuraLUT-Assemble (HDR-5L-0.1) [22]	97.90%	**98.10%**	5076	725	0	0	863	**2.1**
	DWN [17]	97.8%	-	**2092**	1757	0	0	**873**	9.2
	DWN^{arxiv} [37]	**98.3%**	-	4082	3385	0	0	827	6
	TreeLUT [16]	96.6%	-	4478	**597**	0	0	791	2.5
	PolyLUT^{extension} [21]	97.5%	-	75131	4668	0	0	353	17
	FINN [38]	96%	-	91131	-	0	5	200	310
	`hls4ml` [14]	95%	-	260092	165513	0	345	200	190
JSC (high accuracy)	PolyLUT (JSC-XL, D=2) [3]	74.94%	75.01%	252077	2121	0	0	183	27
	PolyLUT-Add (JSC-XL-Add2, D=2) [13]	74.98%	**75.04%**	49766	1976	0	0	395	13
	NeuraLUT (JSC-5L) [4]	74.93%	74.98%	79012	2485	0	0	342	15
	NeuraLUT-Assemble (jsc-cernbox) [22]	75.00%	75.03%	8556	1335	0	0	352	5.7
	Duarte *et al.* [33]	75%	-	88797*	954	0	200	75
	TreeLUT [16]	75.6%	-	**2234**	**347**	0	0	735	**2.7**
	DWN^{arxiv} [37]	**76.3%**	-	4972	3305	0	0	**827**	7.3
	PolyLUT^{extension} [21]	75.1%	-	246071	12384	0	0	203	25
	Fahim *et al.* [35]	76.2%	-	63251	4394	38	0	200	45
JSC (low accuracy)	LogicNets [2]	71.65%	72.10%	7995	249	0	0	574	5
	PolyLUT (JSC-M Lite, D=2) [3]	71.98%	72.15%	12855	447	0	0	558	5
	PolyLUT-Add (JSC-M Lite-Add2, D=2) [13]	71.90%	72.24%	1197	410	0	0	769	4
	NeuraLUT (JSC-2L) [4]	72.01%	**72.95%**	4679	140	0	0	689	3
	DWN^{arxiv} [37]	**74.0%**	-	**110**	72	0	0	**1094**	**1.5**
	PolyLUT^{extension} [21]	72.5%	-	10169	631	0	0	598	5

1. ∗: Paper reports "LUT+FF"
2. DWN^{arxiv} is the arXiv version of DWN with updated results, PolyLUT^{extension} is the journal extension version of PolyLUT with updated results.

SparseLUT does not alter the number of LUT entries in the generated RTL design; Figure 10 illustrates the post-Place&Route results for area (LUTs and flip-flops (FFs)) and maximum frequency (F_max) from Vivado, constrained within the same clock cycle. Notably, SparseLUT achieves similar hardware consumption, and the F_max comparison shows no speed penalty. Therefore, we conclude that the accuracy improvements achieved by SparseLUT are realized without incurring additional hardware and latency overhead.

Table VIII compares the connectivity-optimized models discussed above with other low-latency machine learning methods. Since DWN achieves 57.42% accuracy on CIFAR-10, whereas all traditional LUT-based DNNs remain below 50% accuracy (see Table VII), evaluating their hardware cost at this stage is not meaningful. Consequently, exploring the potential of both approaches remains an open direction for future research within the FPGA community.

TABLE IX: Task Execution Time

Task	Time
Connectivity Searching	2.6h
LUT-DNN Training (PolyLUT)	3.9h
RTL Generation	1.4h
Synthesis & Place & Route	2.5h

4) Run Time Results: Finally, Table IX breaks down the execution times for the various tasks involved and shows that the runtime of connectivity searching does not significantly impact the overall system performance. To illustrate this, we measured the runtime for implementing a PolyLUT ($D = 2$) model on the MNIST task. Synthesis, Place, and Route were performed on a machine with an Intel Xeon E-2356G CPU and 94GB of RAM, while all other processes were executed on a system with an Intel Core i7-10700F CPU, 64GB of RAM, and an NVIDIA RTX 2060 Super GPU.

V. CONCLUSION

This paper introduced SparseLUT, a comprehensive framework that enhances the deployment of LUT-based deep neural networks (DNNs) on FPGAs through two orthogonal optimizations: architectural enhancement and training-based connectivity optimization.

At the architectural level, we proposed an efficient structure that combines multiple PolyLUT sub-neurons via an adder, significantly reducing hardware cost and latency without sacrificing accuracy. Experimental results show that this approach achieves $2.0\times$–$13.9\times$ LUT savings and $1.2\times$–$1.6\times$ latency reduction across diverse benchmarks such as MNIST and Jet Substructure Classification.

On top of this, we introduced a non-greedy dynamic sparsity training algorithm that optimizes the connectivity pattern of each neuron under fixed fan-in constraints, with no additional LUT or routing overhead. By intelligently pruning and regrowing connections, SparseLUT consistently improves accuracy over existing LUT-DNNs such as LogicNets [2], PolyLUT [3], and NeuraLUT [4], achieving up to 2.13% higher accuracy on MNIST and 0.94% on Jet Substructure Classification.

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[2] Yaman Umuroglu, Yash Akhauri, Nicholas James Fraser, and Michaela Blott. LogicNets: Co-designed neural networks and circuits for extreme-throughput applications. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 291–297. IEEE, 2020.

[3] Marta Andronic and George A Constantinides. PolyLUT: learning piecewise polynomials for ultra-low latency FPGA LUT-based inference. In *2023 International Conference on Field Programmable Technology (ICFPT)*, pages 60–68. IEEE, 2023.

[4] Marta Andronic and George A. Constantinides. Neura-LUT: Hiding neural network density in boolean synthesizable functions. *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 140–148, 2024.

[5] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[6] Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *Proceedings of the International Symposium on Artificial neural networks*, pages 20–25, 1995.

[7] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[8] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. *Efficient processing of deep neural networks*. Springer, 2020.

[9] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[10] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[11] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International conference on machine learning*, pages 2943–2952. PMLR, 2020.

[12] Mike Lasby, Anna Golubeva, Utku Evci, Mihai Nica, and Yani Ioannou. Dynamic Sparse Training with Structured Sparsity. *arXiv preprint arXiv:2305.02299*, 2023.

[13] Binglei Lou, Richard Rademacher, David Boland, and Philip H.W. Leong. PolyLUT-Add: FPGA-based LUT Inference with Wide Inputs. *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 149–155, 2024.

[14] Jennifer Ngadiuba, Vladimir Loncar, Maurizio Pierini, Sioni Summers, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Mia Liu, et al. Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml. *Machine Learning: Science and Technology*, 2(1):015001, 2020.

[15] Md Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2022.

[16] Alireza Khataei and Kia Bazargan. Treelut: An efficient alternative to deep neural networks for inference acceleration using gradient boosted decision trees. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '25, 2025.

[17] Alan T. L. Bacellar, Zachary Susskind, Mauricio Breternitz Jr., Eugene John, Lizy K. John, Priscila M. V. Lima, and Felipe M. G. França. Differentiable weightless neural networks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[18] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations (ICLR) 2014*, pages 1–10, Banff, AB, Canada, April 2014.

[19] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through L_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[20] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International conference on machine learning*, pages 2498–2507. PMLR, 2017.

[21] Marta Andronic, Jiawen Li, and George A. Constantinides. Polylut: Ultra-low latency polynomial inference with hardware-aware structured pruning. *IEEE Transactions on Computers*, 74(9):3181–3194, 2025.

[22] Marta Andronic and George A. Constantinides. Neuralut-assemble: Hardware-aware assembling of sub-neural networks for efficient lut inference. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 208–216. IEEE, 2025.

[23] Anthony JGD Holtmaat, Joshua T Trachtenberg, Linda Wilbrecht, Gordon M Shepherd, Xiaoqun Zhang, Graham W Knott, and Karel Svoboda. Transient and persistent dendritic spines in the neocortex in vivo. *Neuron*, 45(2):279–291, 2005.

[24] Dan D Stettler, Homare Yamahachi, Wu Li, Winfried Denk, and Charles D Gilbert. Axons and synaptic boutons are highly dynamic in adult visual cortex. *Neuron*, 49(6):877–887, 2006.

[25] Alessio Attardo, James E Fitzgerald, and Mark J Schnitzer. Impermanence of dendritic spines in live adult CA1 hippocampus. *Nature*, 523(7562):592–596, 2015.

[26] Anna R Chambers and Simon Rumpel. A stable brain from unstable components: Emerging concepts and implications for neural computation. *Neuroscience*, 357:172–184, 2017.

[27] Shiliang Sun, Zehui Cao, et al. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy,

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2026.3654908

14

Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[29] Alessandro Pappalardo. Xilinx/brevitas, 2023.

[30] Xilinx Inc. Vivado Design Suite User Guide (UG949), 2020.

[31] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[32] Jennifer Ngadiuba, Vladimir Loncar, Maurizio Pierini, Sioni Summers, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Mia Liu, et al. Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml. *Machine Learning: Science and Technology*, 2(1):015001, 2020.

[33] Javier Duarte, Song Han, Philip Harris, Sergo Jindariani, Edward Kreinar, Benjamin Kreis, Jennifer Ngadiuba, Maurizio Pierini, Ryan Rivera, Nhan Tran, et al. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*, 13(07):P07027, 2018.

[34] Claudionor N Coelho, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence*, 3(8):675–686, 2021.

[35] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*, 2021.

[36] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[37] Alan TL Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy K John, Priscila Lima, and Felipe MG França. Differentiable weightless neural networks. *arXiv preprint arXiv:2410.11112*, 2024.

[38] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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*, pages 65–74, 2017.

VI. Biography Section

Binglei Lou received the Ph.D. degree from the University of Sydney in 2024. He is currently a Research Associate in the Department of Electrical and Electronic Engineering at Imperial College London. His research interests focused on efficient AI and NPU design.

Ruilin Wu received the M.Eng. degree in Electrical Engineering from the University of Sydney in 2025 and is currently pursuing a Ph.D. at the University of Sydney. From 2023 to 2025, he conducted research on FPGA and AI Engine architectures for high-performance signal processing applications. His recent work has focused on efficient implementations of the hardware-aware neural network optimization, and low-power reconfigurable computing.

Philip Leong received the B.Sc., B.E. and Ph.D. degrees from the University of Sydney. In 1993 he was a consultant to ST Microelectronics in Milan, Italy working on advanced flash memory-based integrated circuit design. From 1997-2009 he was with the Chinese University of Hong Kong. He is currently Professor of Computer Systems in the School of Electrical and Computer Engineering at the University of Sydney, the Chief Technology Advisor to Cluster Technology and Chief Technology Officer at CruxML Pty Ltd.

He was the co-founder and program co-chair of the International Conference on Field Programmable Technology (FPT); program co-chair of the International Conference on Field Programmable Logic and Applications (FPL) and is the Senior Associate Editor for the ACM Transactions on Reconfigurable Technology and Systems. The author of more than 150 technical papers and 4 patents, Dr. Leong was the recipient of the 2005 FPT conference Best Paper as well as the 2007 and 2008 FPL conference Stamatis Vassiliadis Outstanding Paper awards.