NeuraLUT-Assemble: Hardware-aware Assembling of Sub-Neural Networks for Efficient LUT Inference

Marta Andronic
marta.andronic18@imperial.ac.uk
Imperial College London
London, UK

George A. Constantinides
g.constantinides@imperial.ac.uk
Imperial College London
London, UK

ABSTRACT

Efficient neural networks (NNs) leveraging lookup tables (LUTs) have demonstrated significant potential for emerging AI applications, particularly when deployed on field-programmable gate arrays (FPGAs) for edge computing. These architectures promise ultra-low latency and reduced resource utilization, broadening neural network adoption in fields such as particle physics. However, existing LUT-based designs suffer from accuracy degradation due to the large fan-in required by neurons being limited by the exponential scaling of LUT resources with input width. In practice, in prior work this tension has resulted in the reliance on extremely sparse models.

We present NeuraLUT-Assemble, a novel framework that addresses these limitations by combining mixed-precision techniques with the assembly of larger neurons from smaller units, thereby increasing connectivity while keeping the number of inputs of any given LUT manageable. Additionally, we introduce skip-connections across entire LUT structures to improve gradient flow. NeuraLUT-Assemble closes the accuracy gap between LUT-based methods and (fully-connected) MLP-based models, achieving competitive accuracy on tasks such as network intrusion detection, digit classification, and jet classification, demonstrating on average 6× reduction in inference latency and 22× reduction in LUT utilization compared to recent prior approaches.

CCS CONCEPTS

• **Hardware** → **Reconfigurable logic and FPGAs**; Hardware-software codesign; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Edge AI; FPGA; hardware-software codesign; LUT-based neural networks; soft logic; deep learning inference; hardware accelerator

ACM Reference Format:
Marta Andronic and George A. Constantinides. 2025. NeuraLUT-Assemble: Hardware-aware Assembling of Sub-Neural Networks for Efficient LUT Inference. In . ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Ultra-low latency neural network inference has become instrumental in advancing fields such as particle physics, network security, and autonomous vehicles. In particle physics, machine learning (ML) models are essential for handling the immense data volumes generated by detectors. For instance, in the latest upgrade to the CMS trigger system at CERN's Large Hadron Collider, the system processes information from six consecutive beam crossings, every 25 ns. The system must achieve nanosecond-level latency to be capable of accepting new data inputs continuously. Such real-time capabilities enable efficient event selection, by enabling the collection of meaningful events that could otherwise be lost. In network security, ML-driven intrusion detection systems swiftly identify threats and anomalies, safeguarding critical infrastructure by delivering near-instant insights. Autonomous vehicles also rely on ML to interpret sensor data and make split-second navigation and safety decisions, particularly in complex or high-speed scenarios. However, due to the resource-constrained environments and strict latency requirements in these fields, deployed deep neural networks (DNNs) often fall short of the state-of-the-art accuracy in ML.

FPGAs, with their highly customizable architecture, serve as a core platform for these applications, enabling optimized computation to meet stringent performance KPIs. Their reconfigurability supports rapid design iteration, making them ideal for applications that demand frequent model updates. Moreover, FPGAs are highly parallelizable, significantly reducing processing time.

Recent research in the field has focused on hardware-software co-design. Beyond designing efficient hardware, it is also important to adapt software for deployment efficiency. A growing area of research explores model architectures that map efficiently to hardware. For example, LUT-based approaches like NeuraLUT [3], PolyLUT [2], LogicNets [17], NullaNet [16], PolyLUT-Add [15], AmigoLUT [20]. Other recent works have adopted decision tree-based approaches like TreeLUT [14], or weightless neural networks [5].

Following [2], we refer to lookup tables of arbitrary size as Logical-LUTs (L-LUTs), highlighting their ability to exceed the capacity of the Physical-LUTs (P-LUTs) on the FPGA. When an L-LUT requires more inputs than a P-LUT can handle, logic synthesis tools map it as a circuit of multiple interconnected P-LUTs.

NeuraLUT, PolyLUT, LogicNets, and NullaNet encapsulate the entire computation of a neuron within a single L-LUT, creating a network of L-LUTs with no exposed datapaths. These prior works offer distinct trade-offs in model complexity and expressiveness.

However, for these approaches to be feasible, the lookup table size is constrained, which can limit the accuracy of these neural networks. PolyLUT-Add [15] takes a first step at trying to improve the connectivity of these networks by summing the results of multiple

L-LUTs across the network. However, this approach utilizes LUTs for the implementation of the sum which are also restricted by their fan-in. AmigoLUT [20] creates ensembles of smaller LUT-based NNs, including NeuraLUT, to tackle the scalability.

In this work, NeuraLUT-Assemble, we take a distinct approach to combat the challenge of the inherent limitation on the number of L-LUT inputs. We introduce a fully-parametrizable framework that assembles multiple NeuraLUT neurons as tree structures with larger fan-in, directly addressing the exponential scaling challenge. This customizable tool allows users to increase connectivity without fan-in restrictions by providing full control over the tree structure. The grouping of connections at the input of the tree structure is guided by the hardware-aware pruning strategy first introduced in the extended arxiv PolyLUT paper [4].

To enhance training stability, we propose a resource-efficient method that integrates skip-connections within the L-LUTs to ensure effective gradient flow in the individual L-LUTs or across the assembled tree structure.

In summary, the novel contributions are as follows:

- We introduce NeuraLUT-Assemble, an open-source[1] toolflow that leverages the FPGA architecture by embedding dense, full-precision sub-networks within tree-structures of synthesizable Boolean lookup tables.
- We develop a fully-parametrizable framework to increase connectivity by training larger fan-in tree structures of smaller L-LUT units, where connection grouping is determined post-initial training.
- We develop a resource-efficient approach that embeds skip-connections within L-LUTs, promoting smooth gradient flow at training throughout the entire assembled tree structure.
- We assess NeuraLUT-Assemble on three standard tasks used in the low-latency DNN research community: digit classification, jet substructure classification, and network intrusion detection. Our results show that compared to NeuraLUT, our method achieves the lowest area-delay product with up to 62× reduction on MNIST combined with a higher test accuracy, and up to 26× reduction on jet substructure for the same test accuracy.

2 BACKGROUND

2.1 DSP-based architectures

In the area of ultra-low latency, `hls4ml` [12] is a notable open-source framework created to enable inference on FPGAs, with a focus on low-latency applications. Duarte *et al.* [12] employ `hls4ml` to generate both fully-unrolled and rolled network architectures that target latency reduction. These designs, however, utilize high-precision networks, which result in considerable Digital Signal Processing (DSP) usage. Fahim *et al.* [8] further optimize `hls4ml` by incorporating techniques such as quantization-aware pruning.

2.2 XNOR-based architectures

Ngadiuba *et al.* [13] use the `hls4ml` framework to map binary and ternary neural networks onto FPGAs. Similarly, FINN [21] is an open-source framework that was initially tailored for deploying efficient binary neural networks (BNNs) on FPGAs. To improve hardware performance, FINN replaces traditional operations with hardware-friendly alternatives, such as popcount operators instead of additions, thresholding in place of the batch normalization and activation functions, and OR gates for max-pooling.

2.3 Decision tree-based architectures

Decision tree-based approaches, like TreeLUT [14] and POLYBiNN [1], are practical methods for efficient machine learning inference on hardware platforms like FPGAs. These methods use the structure of decision trees to break down problems into smaller, manageable decisions, which work well in hardware because of their parallel and low-latency nature.

2.4 Differentiable LUTs

2.4.1 LUTNet. LUTNet [18, 19], introduced by Wang *et al.*, replaces BNN XNOR operations with learned K-input Boolean functions mapped directly onto FPGA LUTs. This approach leverages the inherent flexibility of LUTs to implement complex Boolean functions, enhancing logic density and allowing for significant network pruning without accuracy degradation.

2.4.2 Differentiable Weightless Neural Networks. A different approach, Differentiable Weightless Neural Networks (DWNs) [5], utilizes an extended finite difference method to approximate gradients for training networks composed of interconnected LUTs. Additionally, in DWNs a distributive thermometer encoding scheme is employed for input representation to convert continuous input features into binary vectors. However, this thermometer encoding assigns distinct floating-point thresholds to each feature, leading to potentially large overhead in converting into thermometer-encoding.

2.5 LUT-based traditional NNs

What distinguishes this category is that, while designed for LUT-based netlist inference, the training process relies on traditional neural network models, which are later fully absorbed into LUT functions by complete enumeration. NullaNet [16] and LogicNets [17] were among the first to map entire neurons onto multi-input, multi-output Boolean functions. NullaNet minimizes these functions' footprint using Boolean logic minimization strategies and selectively determining output values for specific input combinations while treating the rest as don't-care conditions to conserve resources. In contrast, LogicNets trains neural networks that were *a priori* designed to be extremely sparse to overcome NullaNet's potential accuracy loss after their don't-care optimization step.

PolyLUT[2] LogicNets further by encoding the entire neuron's function within an L-LUT but uniquely expands each neuron's feature vector to include all monomials up to a user-defined degree D. This added complexity within each layer allows PolyLUT to reach target accuracies with fewer layers, enhancing efficiency.

NeuraLUT [3] explores an alternative universal function approximator that maintains training simplicity without requiring modifications to existing training frameworks: the multilayer perceptron (MLP) [7, 11]. By embedding MLPs within LUTs, NeuraLUT maximizes the neural network density within each individual L-LUT.

[1] https://github.com/MartaAndronic/NeuraLUT

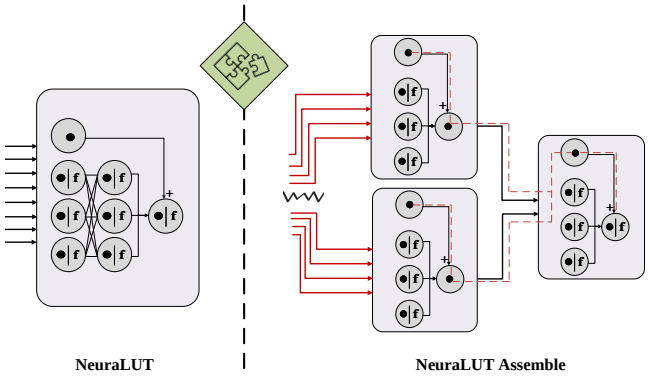

Figure 1: View of a NeuraLUT L-LUT on the left and a NeuraLUT-Assemble L-LUT tree on the right. • represents an affine transformation, whereas f is the activation function.

owever, this approach can unveil highly complex interaction between a limited number of features that is controlled by the L-LUT fan-in.

PolyLUT-Add [15] represents an initial effort to enhance network connectivity by aggregating the outputs of multiple L-LUTs across the network. While this approach effectively improves feature abstraction, it relies on LUTs to perform the summation. This creates a trade-off, as the fan-in limitations of these LUTs can constrain scalability.

AmigoLUT [20] creates ensembles of multiple small models of different LUT-based NNs, such as NeuraLUT, and computes the average of the outputs of all members. This method has proven to be effective at increasing the accuracy of very weak models.

2.6 Hardware-aware structured pruning

In LUT-based neural networks, a primary challenge lies in managing the exponential increase in LUT size with a growing number of inputs. Conventional approaches address this by imposing fixed random sparsity patterns *a priori*[2, 3, 17], but these methods are sensitive to initial seed selection, often resulting in performance inconsistencies.

PolyLUT [4] introduced a hardware-aware structured pruning strategy to overcome these limitations, promoting a tailored sparsity pattern. Rather than relying on predefined sparsity, PolyLUT defines a custom group regularizer designed to guide neuron connections according to hardware constraints.

The proposed method follows a sequential process: initially, the network undergoes dense training with a custom hardware-aware regularizer, establishing a foundation conducive to effective pruning. Following this dense training, a structured pruning stage is applied, and the resulting sparse network is subsequently retrained to restore any potential accuracy loss.

3 METHODOLOGY

Our work introduces a novel approach to designing LUT-based neural networks by leveraging a hardware-aware design to overcome the fan-in limitations of traditional LUT-based neural networks. We have designed a way to assemble tree structures of multiple L-LUTs and translate this structure onto the training framework to achieve higher connectivity. Figure 1 provides a small-scale example that illustrates our strategy. Instead of training a model with 8-input L-LUTs, we can train a fixed tree structure by combining 4-input L-LUTs with a 2-input L-LUT, thereby dramatically reducing the L-LUT cost. Since these two designs train fundamentally different functions, we train the tree structure from scratch rather than mapping a higher fan-in model onto it.

Training deep NNs often encounters difficulties due to the vanishing gradient problem [6] [9], where gradients can shrink substantially as they backpropagate through layers. While this issue is generally less prominent in prior NNs designed for ultra-low latency [2, 3, 17], which tend to have limited depth, it becomes more relevant in the NeuraLUT-Assemble framework. In NeuraLUT-Assemble, the tree-structures add additional depth to the neural network and it tends to be relatively deep compared to the number of inputs per tree, making training more difficult. To address this, we employ residual connections, which add outputs from certain layers to the activations from earlier layers, helping to preserve gradient flow [10].

A key advantage of our approach is that these residual connections traverse the entire assembled tree structure and are entirely hidden within the L-LUT synthesizable Boolean function. As a result, they do not cause an additional implementation cost or reduce regularity at inference. In NeuraLUT these connections were constrained to the L-LUT borders. As illustrated on the left side of Figure 1, a neuron without an activation function bypasses two fully connected layers, adding its output just before the activation function in the third layer. On the right side, in our strategy, all activation functions are removed except in the final tree layer, allowing a skip path with no activation function from the tree structure's input to its output, highlighted in dotted red.

3.1 Assembling strategies

Our fully customizable framework allows users to construct tree structures tailored to their needs, balancing the trade-off between the number of L-LUTs and their size. Figure 2 illustrates two possible configurations for a tree with 16 inputs. In the first configuration, each L-LUT has 4 inputs, and a number of entries in the LUT of $2^{4\beta}$, where β is the number of activation bits. In the second configuration, the number of L-LUTs increases threefold, but the size of each L-LUT decreases to the square root of its previous value, *i.e.* to $2^{2\beta}$, as the number of inputs per L-LUT is halved. NNs trained using NeuraLUT-Assemble resemble the structure shown in Figure 4.

This exponential reduction in size has significant implications for hardware implementation. This strategy enhances scalability, reducing the size of L-LUTs directly. However, the trade-off lies in the increased number of L-LUTs required to construct the tree, which can lead to higher interconnect complexity and potentially longer critical paths in the hardware. Therefore pipelining requires more attention here than in previous LUT-based NN work and we experiment with different strategies in Section 4.

There is also an inherent trade-off in training between the expressivity of the model and its hardware cost. Larger input L-LUTs have higher expressivity because they can capture more complex inter-dependencies among the input features. However, as the tree

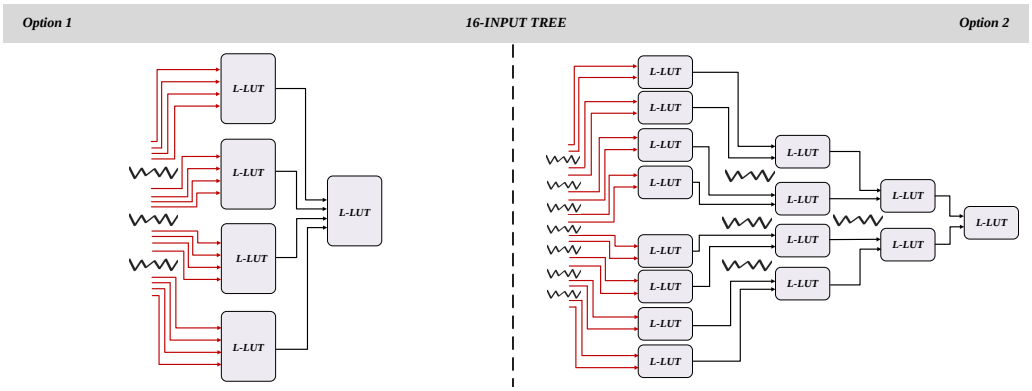

Figure 2: Example of two different NeuraLUT-Assemble configurations for a 16-input tree. The red connections are decided upon an initial stage of dense training, whereas the black connections always stay fixed.

Table 1: Overview of user-defined NN architecture parameters, emphasizing the flexibility of our toolflow. The last three parameters are specific to the NN inside the L-LUT structure.

Parameter	Notation	Description
Layer sizes	w_l	Number of L-LUTs units per layer.
Assemble layer	a_l	Boolean array to indicate assemble layers, which have fixed sparsity.
Fan-ins	F	The fan-ins are individually defined for the input, output, and inner-tree layers.
Bit-widths	β	The bit-widths are individually defined for the input, output, and inner-tree layers.
Depth of L-LUT NN	L	Depth parameter for the NN hidden inside the L-LUTs.
Width of L-LUT NN	N	Width parameter for the NN hidden inside the L-LUTs.
Skip-connection step	S	Skip-connection step for the NN hidden inside the L-LUTs.

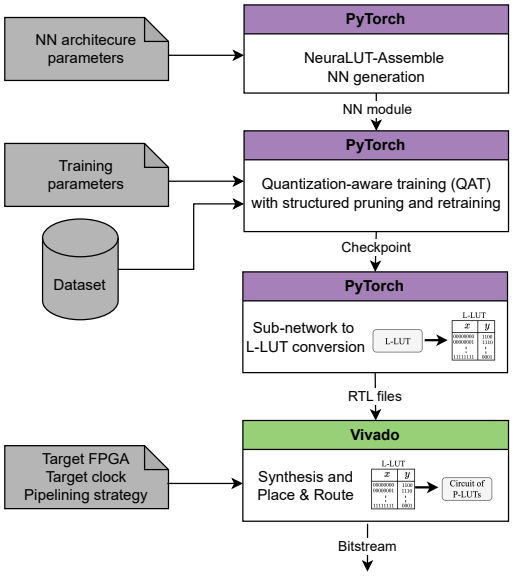

Figure 3: Visualization of NeuraLUT-Assemble's toolflow.

structure becomes deeper, the individual functions implemented by the L-LUTs at each layer become simpler. However, this simplicity makes the model easier and faster to train. To balance these competing factors, we allow the input connections to be decided upon an initial stage of training. This approach enables the framework to prioritize and preserve the most critical inter-dependencies among inputs. By doing so, the model retains a high degree of expressivity while mitigating the challenges of training and the hardware cost associated with larger L-LUTs. We investigate this tradeoff empirically in Section 4.

3.2 Toolflow

NeuraLUT-Assemble builds upon the NeuraLUT toolflow [3], enabling seamless DNN training, conversion into L-LUTs, RTL file generation, and hardware compilation and verification. The training implementation has been modified to accommodate the unique hidden NNs and the tree-based structure of NeuraLUT-Assemble. A high-level overview of the toolflow stages is presented in Fig. 3.

3.2.1 Quantization-aware training (QAT). The training code is written in PyTorch. Initially, the user needs to specify the learning parameters (*e.g.*, learning rate) and the topology parameters, as detailed in Table 1. The hyperparameters L, N, and S define the structure of the sub-networks within the L-LUTs as detailed in Table 1. The rest of the hyperparameters, such as w_l, a_l, F, and β dictate the tree-level topology.

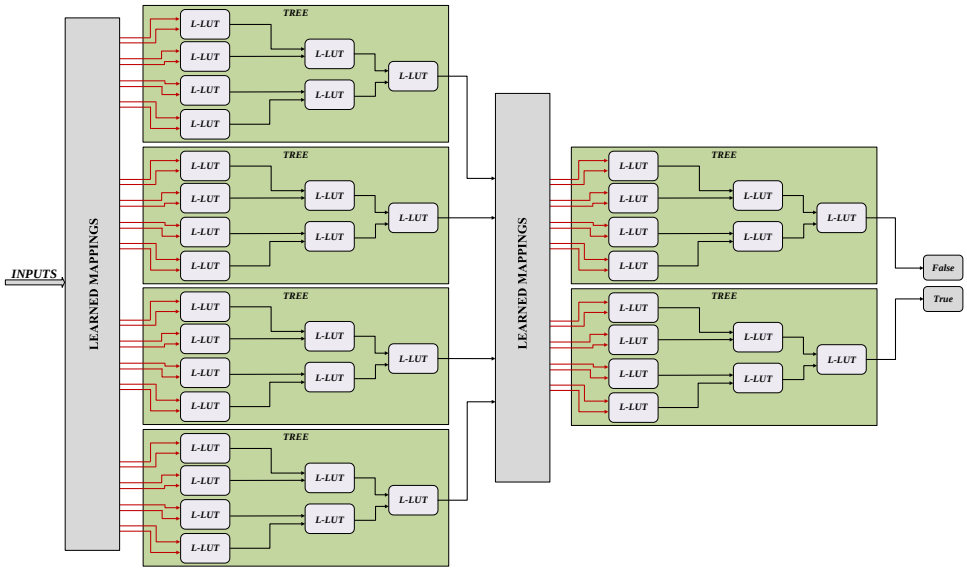

Figure 4: High-level view of toy 6-layer NeuraLUT-Assemble network.

Table 2: Reference floating-point (FP), fully-connected (FC) test accuracy, our test accuracy and architecture parameters for different models used for evaluation in Table 3. Despite the sparsity and low precision, NeuraLUT-Assemble achieves accuracy comparable to that of a dense floating-point model.

Dataset	Accuracy (%)		Parameters						
	FP FC	Ours	w_l	a_l	F	β	L	N	S
MNIST(+aug/-aug)	98.7%	98.6%	[2160,360,2160,360,60,10]	[0,1,0,1,1,1]	[6,6,6,6,6,6]	[1,1,1,1,1,6]	2	16	2
	98.4%	97.9%	[2160,360,2160,360,60,10]	[0,1,0,1,1,1]	[6,6,6,6,6,6]	[1,1,1,1,1,6]	2	16	2
JSC CERNBox	76.0%	75.0%	[320,160,80,40,20,10,5]	[0,1,1,1,1,1,1]	[1,2,2,2,2,2,2]	[8,4,4,4,4,4,8]	2	64	2
JSC OpenML	77.0%	76.0%	[320,160,80,40,20,10,5]	[0,1,1,1,1,1,1]	[1,2,2,2,2,2,2]	[6,3,3,3,3,3,8]	2	64	2
NID	92.5%	93.0%	[60,20,9,3,1]	[0,1,0,1,1]	[6,3,3,3,3]	[1,2,2,2,2,2]	2	16	2

3.2.2 Sub-network to L-LUT conversion. After training, each sub-network within the tree is transformed into an L-LUT. This conversion is automatically performed in PyTorch by generating all possible input combinations based on the specified bit-widths and evaluating the corresponding output values through inference. The number of entries in each L-LUT is $2^{\beta F}$, similar to LogicNets, but with differences in the specific lookup table content derived from the sub-network functions.

3.2.3 RTL file generation. Using the PyTorch framework, the trained network is automatically converted into Verilog RTL, where each L-LUT is implemented as a read-only memory (ROM) block.

3.2.4 Synthesis and Place & Route. To synthesize and compile the generated Verilog RTL files, we use Vivado 2020.1, targeting the `xcvu9p-flgb2104-2-i` FPGA. This enables direct comparison with [3, 4, 15, 17].

4 EVALUATION

4.1 Comparison with prior work

To ensure a fair comparison with prior works, we selected parameters and architectures that either match or exceed the test accuracy of other ultra-low-latency approaches. We evaluated metrics such as area, latency, maximum frequency, and the area-delay product. Table 2 summarizes the parameters used, along with a reference for the test accuracy achieved using the same-sized network with floating-point precision and fully-connected layers. Notably, none of our models deviate by more than 1 percentage point in test accuracy compared to these references.

Table 3 provides a comprehensive comparison, showing that NeuraLUT-Assemble outperforms all other prior works in terms of the area-delay product.

For MNIST, we evaluated two models: one without data augmentation to align with prior works, and another incorporating data augmentation. Compared to NeuraLUT, our method achieves a 62× reduction in the area-delay product while improving test accuracy

Table 3: Evaluation of NeuraLUT against other ultra-low latency neural networks, with results from cited conference papers. Our results are taken after running `Out-of-Context` synthesis and place & route, matching with prior work.

Dataset	Model	Accuracy (%)	LUT	FF	DSP	BRAM	F_{max} (MHz)	Latency (ns)	Area×Delay (LUT×ns)
MNIST	**NeuraLUT-Assemble^{+aug}**	**98.6%**	**5037**	**713**	**0**	**0**	**849**	**2.2**	$\mathbf{1.11 \times 10^4}$
	NeuraLUT-Assemble	**97.9%**	**5070**	**725**	**0**	**0**	**863**	**2.1**	$\mathbf{1.06 \times 10^4}$
	DWN [5]	97.8%	2092	1757	0	0	873	9.2	1.92×10^4
	TreeLUT [14]	97%	4478	597	0	0	791	2.5	1.12×10^4
	PolyLUT-Add [15]	96%	14810	2609	0	0	625	10	1.48×10^5
	AmigoLUT-NeuraLUT [20]	95.5%	16081	13292	0	0	925	7.6	1.22×10^5
	NeuraLUT [3]	96%	54798	3757	0	0	431	12	6.58×10^5
	PolyLUT [4]	97.5%	75131	4668	0	0	353	17	1.38×10^6
	FINN [21]	96%	91131	—	0	5	200	310	2.82×10^7
	`hls4ml` (Ngadiuba et al.) [13]	95%	260092	165513	0	345	200	190	4.94×10^7
JSC CERNBox	**NeuraLUT-Assemble**	**75.0%**	**8539**	**1332**	**0**	**0**	**352**	**5.7**	$\mathbf{4.87 \times 10^4}$
	AmigoLUT-NeuraLUT [20]	74.4%	42742	4717	0	0	520	9.6	4.10×10^5
	PolyLUT-Add [15]	75%	36484	1209	0	0	315	16	5.84×10^5
	NeuraLUT [3]	75%	92357	4885	0	0	368	14	1.29×10^6
	PolyLUT [4]	75.1%	246071	12384	0	0	203	25	6.15×10^6
	LogicNets [17]	72%	37931	810	0	0	427	13	4.93×10^5
JSC OpenML	**NeuraLUT-Assemble**	**76.0%**	**1780**	**540**	**0**	**0**	**941**	**2.1**	$\mathbf{3.92 \times 10^3}$
	TreeLUT [14]	76%	2234	347	0	0	735	2.7	6.03×10^3
	DWN [5]	76.3%	6302	4128	0	0	695	14.4	9.07×10^4
	`hls4ml` (Fahim et al.) [8]	76.2%	63251	4394	38	0	200	45	2.85×10^6
NID	**NeuraLUT-Assemble**	**93.0%**	**91**	**24**	**0**	**0**	**1471**	**1.4**	$\mathbf{1.27 \times 10^2}$
	TreeLUT [14]	93%	345	33	0	0	681	1.5	5.17×10^2
	PolyLUT-Add [15]	92%	1649	830	0	0	620	8	1.32×10^4
	PolyLUT [4]	92.2%	3165	774	0	0	580	9	2.85×10^4
	LogicNets [17]	91%	15949	1274	0	0	471	13	2.07×10^5

by 2 percentage points, underscoring the efficiency of our approach. Notably, while NeuraLUT relied on 12-input LUTs to reach this level of accuracy, NeuraLUT-Assemble achieves even higher accuracy with only 6-input LUTs, thus the exponential area reduction. Furthermore, compared to LUT-based approaches aimed at improving connectivity, such as PolyLUT-Add [15] and AmigoLUT-NeuraLUT [20], we observe 14× and 11.5× reductions in the area-delay product, respectively, while also achieving approximately 2 percentage points higher test accuracy. When compared to the decision-based TreeLUT approach [14], NeuraLUT-Assemble offers comparable hardware performance and delivers 1 percentage point higher accuracy. Similarly, against weightless neural networks, NeuraLUT-Assemble achieves a 1.8× improvement in the area-delay product for comparable accuracy.

For the JSC dataset from CERNBox, we focused on comparisons with other LUT-based approaches, as they are the only prior works utilizing this data source. Here, NeuraLUT-Assemble demonstrates a 26× reduction in the area-delay product while maintaining the same test accuracy as NeuraLUT. On the OpenML datasource, NeuraLUT-Assemble achieves at least a 1.5× reduction in the area-delay product compared to prior works.

On the network intrusion dataset (NID), compared to TreeLUT, NeuraLUT-Assemble reduces the area-delay product by 4× for the same accuracy. Compared to PolyLUT-Add, NeuraLUT-Assemble not only reduces the area-delay product by 104×, but also increases test accuracy by 2 percentage points.

5 CONCLUSION AND FUTURE WORK

NeuraLUT-Assemble advances LUT-based neural networks by addressing fan-in limitations and resource constraints through a flexible, hardware-aware framework. Our results demonstrate significant reductions in the area-delay product while maintaining competitive accuracy across benchmarks, outperforming prior approaches in efficiency and scalability.

REFERENCES

[1] Ahmed M. Abdelsalam, Ahmed Elsheikh, Jean-Pierre David, and J. M. Pierre Langlois. 2018. POLYBiNN: A Scalable and Efficient Combinatorial Inference Engine for Neural Networks on FPGA. In *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 19–24. https://doi.org/10.1109/DASIP.2018.8596871

[2] Marta Andronic and George A. Constantinides. 2023. PolyLUT: Learning Piecewise Polynomials for Ultra-Low Latency FPGA LUT-based Inference. In *2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE, Yokohama, Japan, 60–68. doi: 10.1109/ICFPT59805.2023.00012.

[3] Marta Andronic and George A. Constantinides. 2024. NeuraLUT: Hiding Neural Network Density in Boolean Synthesizable Functions. In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 140–148. doi: 10.1109/FPL64840.2024.00028.

[4] Marta Andronic, Jiawen Li, and George A. Constantinides. 2025. PolyLUT: Ultra-low Latency Polynomial Inference with Hardware-Aware Structured Pruning. arXiv:2501.08043 [cs.LG] https://arxiv.org/abs/2501.08043

[5] Alan Tendler Leibel Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy Kurian John, Priscila Machado Vieira Lima, and Felipe M.G. França. 2024. Differentiable Weightless Neural Networks. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 2277–2295.

[6] Y. Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (March 1994), 157–66. doi: 10.1109/72.279181.

[7] George V. Cybenko. 1989. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems* 2 (Dec. 1989), 303–314.

[8] F. Fahim *et al.* 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. In *TinyML Research Symposium'21*. San Jose, CA, USA. https://doi.org/10.48550/arXiv.2103.05579

[9] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*, Yee Whye Teh and Mike Titterington (Eds.). PMLR, Sardinia, Italy, 249–256.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA, 770–778. doi 10.1109/CVPR.2016.90.

[11] Kurt Hornik. 1991. Approximation Capabilities of Muitilayer Feedforward Networks. *NEURAL NETWORKS* 4, 2 (1991), 251–257.

[12] J. Duarte *et al.* 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation* 13, 7 (July 2018), P07027. doi: 10.1088/1748-0221/13/07/P07027.

[13] J. Ngadiuba *et al.* 2020. Compressing deep neural networks on FPGAs to binary and ternary precision with hls4ml. *Machine Learning: Science and Technology* 2, 1 (Dec. 2020), 015001. doi: 10.1088/2632-2153/aba042.

[14] Alireza Khataei and Kia Bazargan. 2025. TreeLUT: An Efficient Alternative to Deep Neural Networks for Inference Acceleration Using Gradient Boosted Decision Trees. arXiv:2501.01511 [cs.LG] https://arxiv.org/abs/2501.01511

[15] Binglei Lou, Richard Rademacher, David Boland, and Philip HW Leong. 2024. PolyLUT-Add: FPGA-based LUT Inference with Wide Inputs. In *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 149–155. doi: 10.1109/FPL64840.2024.00029.

[16] Mahdi Nazemi, Ghasem Pasandi, and Massoud Pedram. 2019. Energy-Efficient, Low-Latency Realization of Neural Networks through Boolean Logic Minimization. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. Association for Computing Machinery, Tokyo, Japan, 274–279. doi: 10.1145/3287624.3287722.

[17] Yaman Umuroglu, Yash Akhauri, Nicholas J. Fraser, and Michaela Blott. 2020. LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE Computer Society, Gothenburg, Sweden, 291–297. doi: 10.1109/FPL50879.2020.00055.

[18] Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. 2019. LUTNet: Rethinking Inference in FPGA Soft Logic. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE Computer Society, San Diego, CA, USA, 26–34. doi: 10.1109/FCCM.2019.00014.

[19] Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. 2020. LUTNet: Learning FPGA Configurations for Highly Efficient Neural Network Inference. *IEEE Transactions on Computers* 69, 12 (Dec. 2020), 1795–1808. doi: 10.1109/TC.2020.2978817.

[20] Olivia Weng, Marta Andronic, Danial Zuberi, Jiaqing Chen, Caleb Geniesse, George A. Constantinides, Nhan Tran, Nicholas J. Fraser, Javier Mauricio Duarte, and Ryan. Kastner. 2025. Greater than the Sum of its LUTs: Scaling Up LUT-based Neural Networks with AmigoLUT. https://kastner.ucsd.edu/wp-content/uploads/2025/01/admin/fpga25-amigoLUT.pdf Accessed: 2025-01-15.

[21] Y. Umuroglu *et al.* 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*. Association for Computing Machinery, Monterey, CA, USA, 65–74. doi: 10.1145/3020078.3021744.