Differentiable Weightless Neural Networks for Regression Applications

Andrew Jacob[1], Shashank Nag[1], Alan T.L. Bacellar[1], Zachary Susskind[1], Eugene John[2], Priscila M.V. Lima[3], Felipe M.G. Franca[3], Lizy K. John[1]

[1]UT Austin, [2] UT San Antonio, [3] UFRJ Brazil

Abstract

Low latency and energy-efficient edge intelligence is of great importance in the current times. Conventional neural networks struggle to meet these efficiency targets when deployed at the edge due to their large model sizes and computational demands. LUT-based neural networks offer an effective alternative, where computations in the model are replaced by look-up operations, resulting in lower energy consumption and latency. These models are also a natural fit for FPGAs, with prior work demonstrating significant improvements in energy-delay products over aggressively quantized machine learning models on a wide range of classification tasks.

Despite their success in applications involving classification problems, LUT-based models have been underutilized in regression tasks, where they would benefit several applications. In this work, we propose Regression DWN, a regressive variant of the Differentiable Weightless Neural Network (DWN). We evaluate RDWN on ten regression problems from sensing, tabular and particle physics applications, and compare it with quantized neural networks and LUT-based models extended to regression, all implemented with the same FPGA flow. Within a budget of 12,000 LUTs, RDWN is the most accurate method on three of the datasets and within about 3% of the most accurate design on five others, while running at 450 to 580 MHz with latencies of 14 to 22 ns.

Keywords

LUT-based Neural Networks, Weightless Neural Networks, Efficient ML, Regression, FPGA

1 Introduction

With the prominence of physical AI and internet of things, intelligence at the edge has assumed greater significance than ever before [33]. Cloud-based solutions often cannot meet the low-latency demands of applications deployed in these environments, necessitating the need for local, on-device, real-time intelligence.

While conventional deep neural networks offer remarkable model performance across applications, they entail large model sizes and are computationally intensive, ranging from thousands to millions of computations per inference [8]. Consequently, these models are not well suited to be deployed in edge environments, where tight resource and power budgets place stringent constraints on the models that can meet the latency targets for real time inference.

Look Up Table (LUT) based neural networks present an alternate paradigm of models, where typical multiply-accumulate (MAC) operations are replaced by look-up operations [4, 6, 14, 16, 29]. Look-up operations being comparatively inexpensive, enable these models to perform inference at low-latency and minimal energy consumption, making them well-suited for edge deployment. Weightless Neural Networks (WNNs) are a distinct class of LUT-based networks, which learn the LUT neurons natively, rather than converting trained models to LUTs [2, 6, 26, 27]. This enables WNNs to effectively utilize the expressive power of logical LUTs to develop smaller and shallower models, further increasing the efficiency of these models.

FPGAs are a natural choice for these LUT-based models, as the LUTs map directly to the underlying logic fabric on the device. Further, their reconfigurability enables this direct mapping of the model to the hardware LUTs, and the device can be reconfigured to update models even after deployment. The rapid enablement offered by FPGAs also makes them strong candidates for edge deployment.

LUT-based neural networks have been predominantly developed for high-throughput, sub-microsecond classification tasks. Early architectures like LogicNet [29] laid the groundwork by replacing quantized and pruned neurons with equivalent LUTs [29]. Works like NeuraLUT [4] and PolyLUT [3] significantly enhanced the expressiveness of LUT-native models by integrating polynomial learning and sub-network of neurons within each LUT. Furthermore, extensions like AmigoLUT [31] have pushed LUT-based inference toward scalable architectures by creating ensembles of LUT-based models. Differentiable Weightless Neural Networks (DWN) [6] proposed an extended finite-difference based gradient mechanism to natively train multi-layer LUT-based networks. Alongside these, alternative functional representations such as Kolmogorov-Arnold Networks (KAN) have also inspired new paradigms in localized non-linear activations [14].

In spite of these advances, we note that all these prior LUT-based networks have primarily targeted classification tasks, and their architectures have not focused on regression applications. Several regression-based tasks could benefit from an efficient LUT-based network. Across domains like high-energy physics and on-sensor intelligence, several applications demand predicting continuous-valued outputs in real time and in edge/embedded scenarios [7, 11, 24]. For instance: low latency, high throughput regression models are frequently employed in physics experiments such as those in particle colliders to estimate energy of tau particles, or pion calibration. Because tau particles decay extremely quickly, physicists cannot measure the tau directly; they must reconstruct it from its decay products (charged and neutral pions). In general firmware is built to achieve the low latency required of these energy estimators and FPGAs are often employed in these applications [15]. For these, and several other applications, in-situ, low latency regression is extremely important for such scientific experiments [1, 19].

Tech Report,
2026. ACM ISBN NA
https://doi.org/nan

In this paper, we extend Differentiable Weightless Neural Networks (DWNs) to the realm of regression tasks (RDWN). We introduce a specialized, fully differentiable output stage that effectively bridges the gap between the discrete, integer-based processing of the core DWN module and the continuous, real-valued output required for regression. Specifically, we introduce the ternary popcount learnable layer, the polynomial feature expansion module, and the full-precision linear head to learn and predict real-valued output responses, while the core DWN block transforms the input features into a single value representation. We evaluate RDWN and its FPGA implementation on a broad set of regression problems, and compare it with quantized neural networks and LUT-based models extended to regression. While RDWN does not outperform other approaches on all datasets, within a budget of 12,000 LUTs, RDWN is the most accurate method on three of the datasets and within about 3% of the most accurate design on five others. Overall, we make the following contributions in this paper: (i) We propose RDWN which extends DWN to regression tasks and validate its regression performance on various datasets. (ii) We create regression extensions for other LUT based networks like TreeLUT, KANELE, etc. and compare RDWN's performance against them. (iii) We incorporate hardware optimizations for RDWN and evaluate all works on the target FPGA, and report hardware utilization, latency, and other metrics.

2 Background and Motivation

2.1 LUT-based Neural Networks

Over the recent past, Look-Up Table (LUT)-based neural networks have emerged as a compelling paradigm for extreme ultra-low-latency inference on FPGAs, by directly mapping quantized neuron activation functions into boolean truth tables. LogicNets [29] pioneered this idea by training quantized and pruned neural network topologies, where each neuron can then be represented by an equivalent X:Y look-up table (LUT) that encodes the output response of that neuron. By effectively mapping these software LUTs to hardware LUTs on FPGAs, LogicNets offers low nanosecond latencies per inference. To further exploit the capability of LUTs, PolyLUTs [3] introduces multivariate polynomials as the building block that gets encapsulated within each LUT – thereby enabling equivalent classification accuracy with fewer layers, offering area and latency improvements. While these works mapped a single neuron to a LUT, NeuraLUT [4] relaxes this constraint, and maps a sub-network of neurons (sparse and quantized fully connected layers) within a single LUT. Enabled by this greater function expressivity within each LUT, NeuraLUT achieves significant latency reductions. Expanding on this approach, NeuraLUT-Assemble [5] assembles multiple NeuraLUT neurons as tree structures with a larger effective fan-in, thereby achieving significant reductions in area-delay products. AmigoLUT [31] improves the accuracy of weak LUT-based models by creating ensembles of LUT-based networks.

KANELÉ [14] proposes an alternate approach of using Kolmogorov-Arnold Networks (KANs), which utilizes a structure that can be efficiently discretized and mapped to LUTs. This enables it to achieve comparable accuracy with fewer LUTs than most prior LUT-based neural networks, across datasets. TreeLUT [16] introduces a method to implement Gradient Boosted Decision Trees (GBDTs) with LUTs on FPGAs, using an efficient quantization scheme, hardware architecture, and pipelining strategy. With this, TreeLUTs achieve improved hardware utilization, latency, and throughput against prior works, with competitive accuracy. Finally, HGQ (High Granularity Quantization) [25] introduces fine-grained, non-uniform bit-width quantization across individual layers and channels, striking an optimal balance between accuracy preservation and LUT utilization on the FPGA.

While most of these models convert quantized and sparse neural networks to LUTs post training, we now look at Weightless Neural Networks — an alternate class of models that train LUTs natively.

2.2 Weightless Neural Networks

Weightless Neural Networks (WNNs), first proposed by Igor Aleksander [2], represent a distinct paradigm in neural computation. Unlike conventional neural networks that rely on multiply-accumulate (MAC) operations involving weighted connections, WNNs utilize Random Access Memory (RAM) nodes or Lookup Tables (LUTs) to store and retrieve output values based on input patterns. As WNNs natively train LUTs, they are able to better exploit the expressive power of LUTs, often resulting in shallower and smaller models. This architecture makes them exceptionally fast, power-efficient, and well-suited for deployment on FPGAs.

A significant historical limitation of WNNs was the difficulty in training deep, multi-layer architectures using gradient-based optimization methods [18, 26, 27]. The discrete nature of LUT addressing and storage made backpropagation of errors challenging. Recent advances, such as the Differentiable Weightless Neural Network (DWN) [6], have overcome this limitation. By introducing techniques like the Extended Finite Difference (EFD), DWNs allow for the approximate differentiation of operations within the network, enabling end-to-end training of deep, multi-layer WNNs. These networks have demonstrated state-of-the-art performance on various classification tasks, particularly with tabular data, often surpassing traditional models in both accuracy and efficiency.

However, the existing DWN architecture is inherently designed for classification. The final layer of a DWN typically outputs binary or integer values from its LUTs, which are then summed or counted (a "popcount" operation) to generate logits for each class. This process is effective for selecting a discrete category but is not directly applicable to regression tasks, where the objective is to predict a continuous, real-valued output. The fundamental challenge lies in mapping the discrete, integer-based computational result of the core WNN structure to a continuous output space in a way that remains fully differentiable and trainable. Therefore, there is a need for architectural enhancements that extend the benefits of DWNs, namely, high speed, low power consumption, and hardware efficiency, to the domain of regression. Such a solution would enable the application of this powerful and efficient architecture to a broader class of problems, including time-series prediction, healthcare, and scientific computing, where continuous value prediction is essential.

3 The RDWN Design

We build upon the foundations of Differentiable Weightless Neural Networks and introduce an extension architecture for regression,

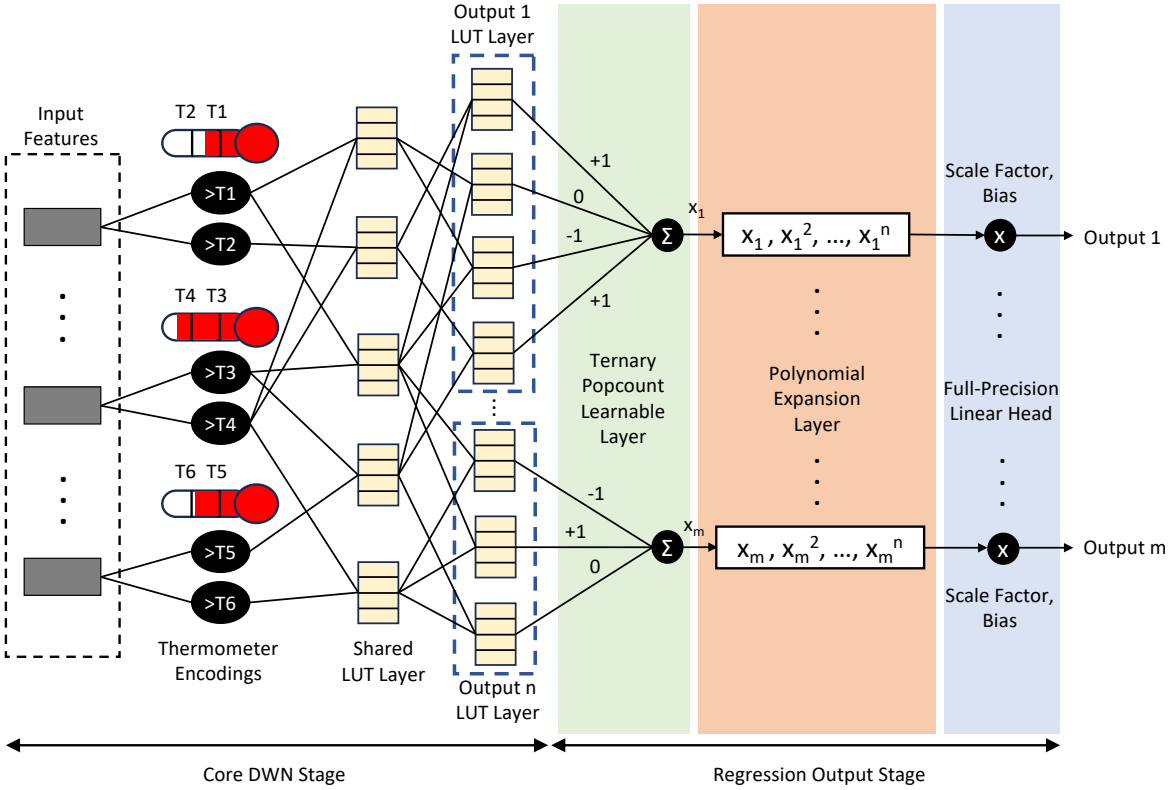


Figure 1: Regression Differentiable Weightless Neural Network (RDWN) Architecture.

the Regression Differentiable Weightless Neural Network (RDWN) (see Fig. 1). The design comprises a standard multi-layer DWN composed of learnable lookup layers. This core network processes an input vector and produces a set of binary or integer outputs from its final layer. The subsequent regression output stage that we introduce processes these outputs.

The overview of the regression output stage is as follows. Following the final DWN lookup layer, a "Ternary Popcount Learnable Layer" is introduced. This layer takes the outputs of the final DWN layer and computes a scalar value. Unlike a simple summation, this layer may employ learnable parameters to interpret the incoming patterns, assigning different weights or significance to different input states, specifically ternary weights like positive (+1), negative (-1), or zero contribution (0). The output of this layer is a single scalar value, herein referred to as 'x', which serves as a compressed, learned representation of the DWN's output state.

This scalar value 'x' is then fed into a polynomial feature expansion module. This module generates a vector of polynomial features by computing successive powers of 'x' (e.g., $[x, x^2, x^3, ..., x^n]$ for a degree-n polynomial). This expansion transforms the single scalar representation into a higher-dimensional feature space, allowing for the modeling of complex, non-linear relationships.

Finally, the vector of polynomial features is processed by a full-precision linear regression head. This head consists of a learnable weight vector and a learnable bias term. It computes the dot product of the polynomial feature vector and the weight vector, and adds the bias. The result is a single, continuous, real-valued number, which is the final regression output of the network.

The entire architecture, from the DWN lookup layers through the ternary popcount layer and the polynomial regression head, is constructed to be fully differentiable. This allows the network to be trained end-to-end using standard gradient-based optimization algorithms, such as stochastic gradient descent (SGD) or its variants (e.g., Adam). During training, the gradients flow back through the linear head, the polynomial expansion, the ternary popcount layer, and into the DWN's learnable lookup tables, allowing all components of the system to be jointly optimized to minimize a regression-specific loss function (e.g., Mean Squared Error).

This proposal successfully extends the highly efficient LUT-based paradigm of DWNs to regression problems, creating a new class of models that are both powerful and suitable for low-latency, low-power edge computing applications.

3.1 Core DWN Architecture

The initial stages of the network in Fig. 1 follow the established DWN structure. An input data vector is first processed. If the input is not binary, it is typically encoded into a binary representation using a method such as thermometer encoding [26]. This binary vector serves as the input to the first layer of the DWN.

The core of the DWN consists of one or more layers of interconnected Lookup Tables (LUTs). Each LUT is effectively a small memory that stores learnable values. A subset of the binary inputs

to a layer is used as an address to index into a specific LUT, retrieving the value stored at that address. In a DWN, these stored values are typically real-valued (e.g., within [-1, 1]) during training to facilitate differentiation, and are binarized for inference. The connections that determine which inputs address which LUTs can be fixed, random, or learnable. The output of one layer of LUTs is concatenated to form the input (address lines) for the next layer. This layered composition allows the network to learn hierarchical and complex features from the input data. The final layer of the core DWN produces a vector of binary values.

3.2 Regression Output Stage

The key contribution of this work is the structure that follows the final LUT layer of the core DWN. This output stage is designed to convert the discrete-style output of the DWN into a continuous value for regression. It comprises three main components: a Ternary Popcount Learnable Layer, a Polynomial Feature Expansion Module, and a Full-Precision Linear Head. The model also supports predicting multiple target values, by replicating the regression output stage for each of the targets. As shown in Fig. 1, the LUTs in the last layer can be separately grouped to be assigned to each output, or can all be shared for all the outputs.

3.2.1 Ternary Popcount Learnable Layer. The output from the final DWN layer, which can be seen as a bit-string or a vector of activations, is fed into the Ternary Popcount Learnable Layer. The term "popcount" traditionally refers to counting the number of set bits (1s) in a binary string. In this context, the operation is generalized and made learnable. The layer's operation can be considered a learnable weighted summation. For each output bit b_i from the final DWN layer, the ternary layer has a corresponding learnable weight w_{ij}. This weight can be positive (+1), negative (-1), or zero, effectively treating the input bit's contribution in a ternary fashion (excitatory, inhibitory, or neutral). The output of the layer, a scalar 'x', is computed as: $x_j = \Sigma_i w_{ij}.b_i$ The "ternary" nature arises from the effect of the weights, which can be trained to push certain inputs toward a positive contribution, others toward a negative one, and others to have no contribution. The purpose of this layer is to learn the most effective way to collapse the high-dimensional binary output of the DWN into a single, meaningful scalar representation relevant to the regression task.

3.2.2 Polynomial Feature Expansion Module. The scalar 'x' produced by the ternary popcount layer represents a learned, one-dimensional feature. To allow the model to capture more complex, non-linear relationships without adding more DWN layers, this scalar is expanded into a higher-dimensional space. The Polynomial Feature Expansion Module takes 'x' and generates a vector containing the powers of 'x' up to a predefined degree n: $P(x) = [x, x^2, x^3, ..., x^n]$. This is a non-learnable, deterministic transformation. The choice of the degree n is a hyperparameter of the model. This step provides the final linear layer with a richer set of features, enabling it to approximate a more complex function of 'x' than a simple linear relationship.

3.2.3 Full-Precision Linear Head. The final component is a standard linear regression layer. This layer has its own set of learnable, full-precision parameters: a scalar matrix W of dimension $n \times m$ and a bias B vector of dimension m. It takes the polynomial feature vector $P(x)$ as input and computes the final regression prediction y as the row wise dot product of $P(x)$ and W, plus the bias B: $y = W.P(x) + B = (\Sigma_i^n W_i x^i) + B$. This structure is effectively a polynomial regression model, but its input feature 'x' is not a raw input variable; instead, it is the dynamically computed, learned output of the underlying DWN and ternary popcount layer.

3.3 End-to-End Differentiability and Training:

A critical aspect of the design is that every component is differentiable with respect to its inputs and learnable parameters. The DWN core is differentiable via the Extended Finite Difference method. The ternary popcount layer is a weighted sum, which is made differentiable using Straight Through Estimators (STEs). The polynomial expansion is a deterministic function whose derivatives are well-defined. The final linear head is also differentiable. This complete differentiability allows the entire network to be trained end-to-end. A regression loss function, such as Mean Squared Error (MSE), L1 loss, or Huber loss, is computed between the network's prediction y and the ground truth value. Gradients of this loss are then calculated with respect to all learnable parameters in the network—the linear head weights W and bias B, the ternary popcount weights w_{ij}, and the contents of all the LUTs in the core DWN, and these parameters are updated via an optimization algorithm like Adam. This process allows the network to holistically learn the optimal LUT contents, the best way to interpret the LUT outputs into a scalar, and the ideal polynomial function to map that scalar to the final regression target.

3.4 Hardware Implementation

Each trained DWN LUT is implemented and mapped to physical LUTs on the FPGA fabric. Typically, DWN models trained with 6:1 offer a more effective and direct mapping to the physical LUTs on the FPGA (which are 6:1 LUTs).

The ternary popcount layer needs no additional storage for ternary weights, as its weights are folded into the connections of the last LUT layer. For inference, each ternary weight is quantized to +1, −1 or 0, so each final-layer output is wired to a positive count, a negative count or to nothing, and each output of the layer is the difference of its two counts. Synthesis then removes the final-layer LUTs whose weight is 0 for every output, together with any logic that feeds only them. The counts are summed in an adder tree that is registered after every second level rather than after every level, which saves pipeline registers and cycles, and the learned scale and offset of each output are applied by a single constant multiplication in the same pipeline stage as the final shift to the 16-bit prediction.

4 Methodology

Datasets and protocol. Table 1 lists the regression datasets used in our evaluation. For C-MAPSS and SARCOS we use their published training and test sets, while the other datasets follow the resampling rule of the OpenML-CTR23 benchmark suite [12]. The two datasets with less than 10,000 samples, air quality and wine quality, use one fold of a 10-fold split as the test set, and the larger datasets use a 33% holdout. For each of the datasets, 20% of the training set is reserved

Table 1: Regression datasets and the splits used by every model.

Dataset	Domain	Description	Target	Train/Val/Test	In/Out
Air quality [10]	Sensor	Gas sensor readings, temperature, humidity	NO_2 ($\mu g/m^3$)	5,554/1,389/772	8/1
Wine quality [9]	Tabular	Physicochemical properties of red and white wines	Quality score (0–10)	4,678/1,169/650	11/1
California housing [20]	Tabular	Census block statistics and geographic location	Median value ($100k)	11,063/2,766/6,811	8/1
C-MAPSS RUL [23]	Sensor	Turbofan engine sensor readings (FD001)	RUL (cycles)	16,505/4,126/100[†]	18/1
Diamonds [32]	Tabular	Diamond attributes and physical dimensions	Price (US$)	28,912/7,228/17,800	26/1
LHC MET (FWX) [7, 22]	Physics	Simulated collision and missing-energy features	True MET (GeV)	107,200/26,800/66,000	8/1
SARCOS [21, 30]	Sensor	Robot arm positions, velocities, accelerations	Joint torques	35,587/8,897/4,449	21/7
Superconductivity [13]	Tabular	Elemental properties of superconducting materials	Critical temp. (K)	11,397/2,849/7,017	81/1
CMS muon q/p_T [17]	Physics	Muon trigger-stub positions and bend angles	q/p_T	632,135/158,034/389,187	31/1
Smart pixels [28]	Physics	Simulated silicon-sensor charge profiles	$\cot\alpha$, $\cot\beta$	535,962/133,991/329,977	60/2

[†]C-MAPSS uses every cycle of its 100 training engines for training and validation, and only the last recorded cycle of each of its 100 test engines, the one with a published remaining useful life, for testing.

for validation, with individual samples drawn at random. All models and their specific configurations are trained on the same training set and evaluated on the same test set. We note that while most datasets used in our evaluation are applicable for edge use-cases, we also include popular datasets like California housing which are more generic. In our evaluation, accuracy is reported as test RMSE, with RMSE pooled over samples and outputs for multi-output datasets. We report the accuracy-versus-area fronts and the budget tables for each method, wherein the evaluated configuration with the lowest test RMSE among those within the area considered, is reported.

Model Comparisons. We compare RDWN models against HGQ [25], KANELÉ [14], NeuraLUT-Assemble [5], TreeLUT [16] and an int8 MLP. Each method is swept over its own size parameters, and all designs in each sweep is synthesized, placed and routed. HGQ was already evaluated on regression in its original work, so we use it without changes to its method. then the chTreeLUT and NeuraLUT-Assemble were originally evaluated only on classification, and KANELÉ on classification, anomaly detection and control but not on supervised regression, so **we extend them to regression** with the following changes. TreeLUT trains XGBoost regression trees, quantizes the leaf values to TreeLUT's leaf width using its per-tree offset, and replaces the argmax with the sum of the leaf outputs. For tasks with several outputs, it trains one forest per output. NeuraLUT-Assemble is trained with an MSE loss on standardized targets instead of cross-entropy, and the ReLU quantizer on its output neurons is replaced with a signed quantizer so that it can produce negative predictions. KANELÉ already computes the fixed-point sum of its last layer on chip, with the argmax applied off chip in the original work, so only its loss function and output stage change. The int8 MLP is quantized after training, with an input standardization stage before the network and the same output stage after it. Table 2 describes the major changes made in these architectures that we compare against. For each of the models, the training procedures, truth-table extraction and RTL generators are unchanged except for the changes described above. Each extended design ends in the same dequantization stage, a constant multiply-add per output that produces the int16 prediction. In the tables and figures, the three methods that we extend to supervised regression are marked with an asterisk, for example, TreeLUT*.

Table 2: Changes made to each method for regression. Every design, including HGQ's, ends in the same dequantization stage, a constant multiply-add per output that produces the int16 prediction.

Method	Change for regression
HGQ	None, as its original work already includes a regression task.
TreeLUT*	XGBoost regression objective, leaf outputs summed instead of an argmax, and one forest per output.
NeuraLUT-A*	MSE loss on standardized targets instead of cross-entropy, and a signed output quantizer in place of the ReLU quantizer.
KANELÉ*	MSE loss on standardized targets instead of cross-entropy.

Hardware flow. Each design is generated as RTL and simulated against a bit-exact software model of its integer datapath. The designs are synthesized, placed and routed out of context for a Xilinx xcvu9p part using Vivado 2022.1, with a 2.0 ns target clock period and DSP inference disabled. No design uses block RAM or DSP slices, so the LUT and FF counts taken after routing, cover the entire area of each design. $F_{\max}$ is taken from the post-route timing report, and latency is the pipeline depth in cycles divided by $F_{\max}$. All designs are fully pipelined and accept one sample per clock cycle.

Every design produces a 16-bit integer prediction.The comparisons in Section 5 count each design alone, reading its own input format, which is the learned fixed-point code of HGQ, KANELÉ, NeuraLUT-Assemble and the int8 MLP, the quantized input grid or the 32-bit float features for TreeLUT, and quantized fixed-point or 32-bit float features for RDWN, whichever of its builds is most accurate within the area considered. The reported RMSE is that of the bit-exact integer model on the full test set, so it includes the effects of input and output quantization. TreeLUT is built in two forms, one on its quantized input grid and the other with 32-bit floating-point split keys, for which XGBoost is trained on raw 32-bit floating-point features and each split threshold is stored as an order-preserving 32-bit key. Both forms are included in TreeLUT's model selection, and the design with the lowest test RMSE within each LUT budget is reported. Section 6 extends the comparison to end-to-end designs that receive 32-bit float features and include the conversion of these features into each method's input code.

5 Results

5.1 Accuracy versus area

Figure 2 shows the trade-off between accuracy and area up to 70,000 LUTs. Below about 300 LUTs, NeuraLUT-Assemble* or HGQ provides the most accurate design on every dataset, as their smallest designs are smaller than RDWN's, which start at about 300 to 500 LUTs and at 1,200 LUTs on SARCOS. From there, RDWN provides the most accurate design up to about 7,000 LUTs on air quality, 10,000 LUTs on California housing, 13,000 LUTs on wine quality and 21,000 LUTs on superconductivity, and TreeLUT* leads above these LUT counts except on air quality, where HGQ does. On wine quality, RDWN and TreeLUT* alternate below about 3,000 LUTs. On C-MAPSS, RDWN leads between about 600 and 2,400 LUTs, and it does not lead at any area on LHC MET, SARCOS, diamonds and CMS muon. KANELÉ* leads on SARCOS from about 4,000 to 60,000 LUTs, and on diamonds RDWN, KANELÉ* and TreeLUT* come within 0.1% of each other at 12,000 LUTs. HGQ and KANELÉ* alternate as the most accurate method on CMS muon, with HGQ leading beyond about 9,500 LUTs. On smart pixels, RDWN and TreeLUT* alternate as the most accurate method over the whole range from about 800 LUTs, with RDWN ahead over most of it up to about 43,000 LUTs, while HGQ, KANELÉ*, NeuraLUT-Assemble* and the int8 MLP remain several times less accurate.

The other models compared outperform the RDWN in the LHC MET dataset. LHC MET is largely linear, as four of its eight inputs are reconstructed estimates of the same missing transverse energy that it predicts. A ridge regression reaches a test RMSE of 26.2 on it, only about 5% above the most accurate design of any method, and every method comes within 3% of the lowest RMSE within 12,000 LUTs. HGQ represents a linear dependence directly with a few multiplications and reaches an RMSE of 25.0 with a design of about 2,300 LUTs, which no larger design of any method improves on. RDWN's strength lies in LUTs that each hold an arbitrary non-linear function of several inputs, whereas its output stage simply counts their single-bit outputs. On a target that is mostly a weighted sum of its inputs, that strength goes largely unused, and RDWN's designs remain about 3% less accurate than HGQ's at every area.

Superconductivity, a dataset where the RDWN performs well especially at smaller sizes, has 81 features derived from the elemental properties of each compound. RDWN provides the most accurate design on it from about 700 to 21,000 LUTs. Within 12,000 LUTs, the most accurate TreeLUT*, NeuraLUT-Assemble* and KANELÉ* designs have 5 to 10% higher RMSE than RDWN's, and HGQ's is 35% higher, its largest shortfall on any dataset apart from smart pixels. The wide input makes each first-layer neuron of HGQ costly, so that its most accurate design within 12,000 LUTs has only eight hidden neurons. TreeLUT* trails by 8 to 25% up to about 8,000 LUTs and closes the gap only at larger areas, leading beyond about 21,000 LUTs. Unlike on LHC MET, a ridge regression on superconductivity has about 80% higher RMSE than the most accurate designs, and this non-linear target may put the capacity of RDWN's LUTs to use in small designs.

Table 3 compares the methods at a budget of 12,000 LUTs, with each method's lowest RMSE within the budget normalized to the lowest RMSE of any method within the same budget on that dataset. This is the smallest of the comparison budgets at which every method has a design on all ten datasets, as the smallest int8 MLP on superconductivity occupies 11,558 LUTs. RDWN achieves the lowest geometric mean of normalized RMSE at 1.053, followed by TreeLUT* at 1.084, KANELÉ* at 1.224, HGQ at 1.328, NeuraLUT-Assemble* at 1.347 and the int8 MLP at 1.527. The most accurate method differs between datasets, with HGQ leading on four datasets, RDWN on three, TreeLUT* on two and KANELÉ* on SARCOS. RDWN is within 3.1% of the lowest RMSE on five of the datasets it does not lead, while its RMSE is 11% higher than HGQ's on CMS muon and 39% higher than KANELÉ*'s on SARCOS.

Smart pixels weighs heavily on the geometric means of HGQ, KANELÉ*, NeuraLUT-Assemble* and the int8 MLP, whose RMSE on it is five to twelve times the lowest. Without smart pixels, KANELÉ* has the lowest geometric mean at 12,000 LUTs, 1.045, followed by RDWN at 1.060, TreeLUT* at 1.079 and HGQ at 1.094, with NeuraLUT-Assemble* and the int8 MLP at 1.14 and 1.22.

Table 4 gives the same geometric mean at budgets from 2,000 to 48,000 LUTs. RDWN has the lowest geometric mean at every budget up to 40,000 LUTs, and its lead is largest at small budgets, 1.078 against 1.174 for TreeLUT* and KANELÉ* at 2,000 LUTs. TreeLUT*'s geometric mean falls from 1.174 at 2,000 LUTs to 1.040 at 48,000 LUTs, slightly below RDWN's 1.044, while those of the other four methods remain between 1.17 and 1.53 at every budget.

Table 5 reports the minimum observed LUT count required by each method to match or improve on the RMSE of RDWN's most accurate design within 12,000 LUTs. TreeLUT* needs 1.0 to 2.6 times as many LUTs as RDWN on seven datasets, and no evaluated TreeLUT* design reaches RDWN's accuracy on C-MAPSS and CMS muon. HGQ, KANELÉ*, NeuraLUT-Assemble* and the int8 MLP do not reach RDWN's accuracy with any evaluated design on five to eight of the ten datasets, up to the largest designs given in the last row of the table. The exceptions are LHC MET, on which every method matches RDWN's accuracy with only 3 to 25% of its LUTs, and CMS muon, on which HGQ and KANELÉ* need 18% and 16% of RDWN's LUTs. KANELÉ* also needs fewer LUTs than RDWN on SARCOS and diamonds, 53% and 61% as many.

SARCOS and smart pixels are the two datasets on which the lowest RMSE continues to fall substantially with area. Doubling

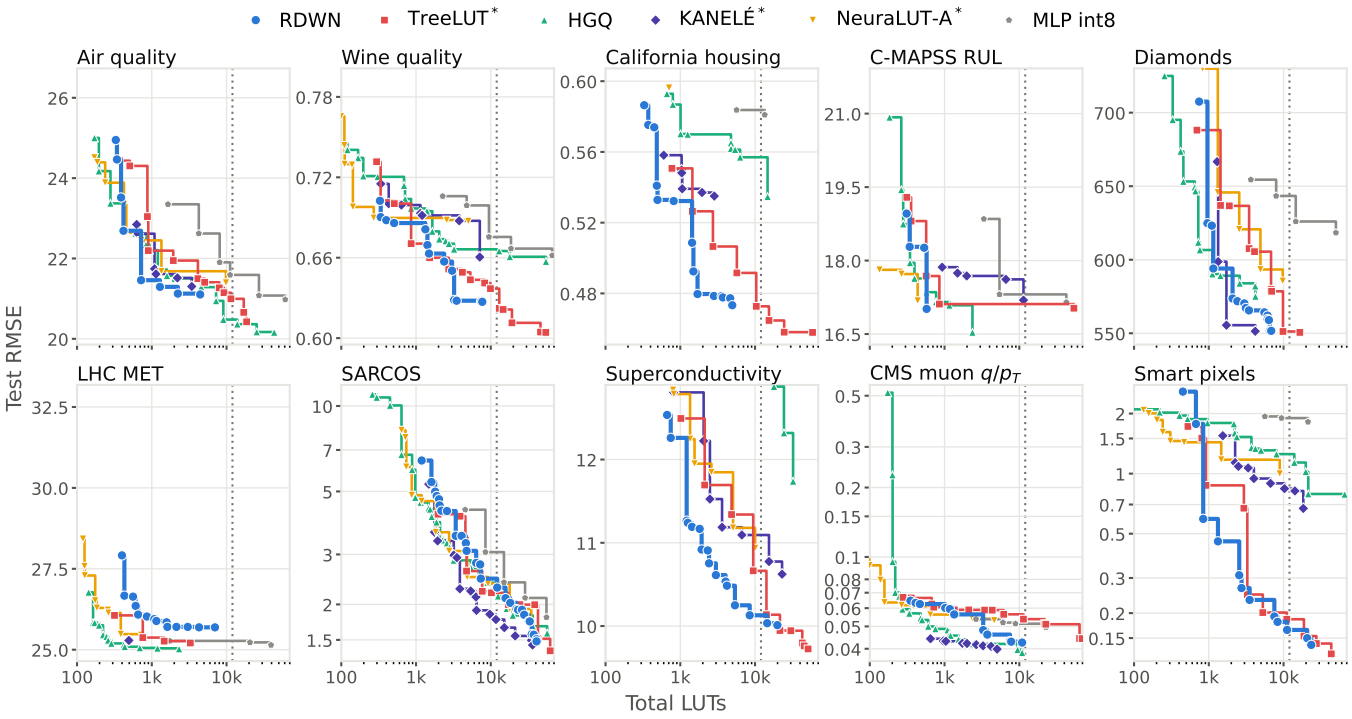


Figure 2: Test RMSE versus total LUTs, design alone, one panel per dataset, up to 70,000 LUTs. Each step curve represents a method's non-dominated designs. The dotted line marks 12,000 LUTs.

Table 3: RMSE of each method's most accurate design within 12,000 LUTs, design alone, divided by the lowest RMSE of any method within the same budget on that dataset. Bold: the lowest unrounded value in each row.

Dataset	RDWN	TreeLUT*	HGQ	KANELÉ*	NeuraLUT-A*	MLP int8
Air quality	1.031	1.025	**1.000**	1.040	1.045	1.054
Wine quality	**1.000**	1.016	1.063	1.053	1.096	1.077
California housing	1.001	**1.000**	1.178	1.132	1.261	1.235
C-MAPSS RUL	1.029	1.035	**1.000**	1.040	1.039	1.047
Diamonds	1.001	**1.000**	1.043	1.000	1.062	1.167
LHC MET (FWX)	1.027	1.007	**1.000**	1.011	1.017	1.010
SARCOS	1.393	1.242	1.274	**1.000**	1.337	1.728
Superconductivity	**1.000**	1.053	1.351	1.095	1.080	1.494
CMS muon q/p_T	1.107	1.399	**1.000**	1.037	1.382	1.336
Smart pixels	**1.000**	1.131	7.603	5.106	6.067	11.519
Geometric mean	**1.053**	1.084	1.328	1.224	1.347	1.527

the budget from 6,000 to 12,000 LUTs lowers the lowest RMSE of any method by 21% on SARCOS and by 18% on smart pixels, and raising it further to 48,000 LUTs lowers it by another 19% and 24%, respectively, whereas each of these reductions is at most 3.6% on the other eight datasets. These are also the two datasets with several outputs, and TreeLUT* trains one forest per output, so its 50-tree, depth-3 float32-key configuration occupies 9,521 LUTs on SARCOS and 2,947 LUTs on smart pixels, whereas the same configuration occupies only 897 to 2,170 LUTs on the eight single-output datasets.

5.2 Latency

Latency is the number of pipeline cycles from input to output, measured in simulation, divided by $F_{\max}$, which is derived from the worst negative slack after routing against the 2.0 ns clock constraint, and every design accepts a new sample every clock cycle. Figure 3 shows the trade-off between RMSE and latency for the designs within 12,000 LUTs. Over the ten datasets, the geometric mean latency of each method's most accurate design within this budget

Table 4: Geometric mean over the ten datasets of each method's RMSE relative to the most accurate design within the budget on each dataset, design alone. – implies no design within the budget on at least one dataset. Best per budget in bold.

Budget	RDWN	TreeLUT*	HGQ	KANELÉ*	NeuraLUT-A*	MLP int8
≤ 2,000 LUTs	**1.078**	1.174	1.232	1.174	1.203	–
≤ 5,000 LUTs	**1.052**	1.094	1.292	1.193	1.293	–
≤ 8,000 LUTs	**1.040**	1.085	1.322	1.207	1.342	–
≤ 12,000 LUTs	**1.053**	1.084	1.328	1.224	1.347	1.527
≤ 16,000 LUTs	**1.047**	1.086	1.315	1.222	1.361	1.501
≤ 24,000 LUTs	**1.047**	1.071	1.275	1.220	1.370	1.520
≤ 40,000 LUTs	**1.036**	1.070	1.263	1.223	1.365	1.511
≤ 48,000 LUTs	1.044	**1.040**	1.280	1.240	1.383	1.526

Table 5: Minimum observed LUT count of each method's evaluated designs that match or improve on the RMSE of RDWN's most accurate design within 12,000 LUTs, design alone, as a multiple of RDWN's LUTs. "Not reached" implies no evaluated design of the method reaches that RMSE. Bold entires imply fewer LUTs than RDWN.

	RDWN (<24k LUTs)		LUTs required to match RMSE				
Dataset	RMSE	LUTs	TreeLUT*	HGQ	KANELÉ*	NeuraLUT-A*	MLP int8
Air quality	21.10	4,438	2.6×	1.6×	not reached	not reached	6.1×
Wine quality	0.6271	7,692	1.7×	not reached	not reached	not reached	not reached
California housing	0.4733	4,958	2.1×	not reached	not reached	not reached	not reached
C-MAPSS RUL	17.01	576	not reached	4.1×	not reached	not reached	not reached
Diamonds	551.7	6,839	1.4×	not reached	**0.61**×	not reached	not reached
LHC MET (FWX)	25.69	6,995	**0.11**×	**0.03**×	**0.07**×	**0.06**×	**0.25**×
SARCOS	2.465	7,346	1.0×	1.2×	**0.53**×	1.3×	2.0×
Superconductivity	10.13	8,486	2.5×	not reached	not reached	not reached	not reached
CMS muon q/p_T	0.0426	10,905	not reached	**0.18**×	**0.16**×	not reached	not reached
Smart pixels	0.1648	11,140	1.7×	not reached	not reached	not reached	not reached

is 14.0 ns for TreeLUT*, 19.3 ns for RDWN, 20.3 ns for NeuraLUT-Assemble*, 23.1 ns for KANELÉ* and about 27 ns for HGQ and the int8 MLP.

TreeLUT* has the lowest latency of these designs on nine of the ten datasets, 13 to 17 ns, because every one of its designs within the budget takes only six cycles, as its trees are evaluated in parallel and their leaf values are summed by a shallow adder tree. This short pipeline outweighs its clock frequency of only 350 to 460 MHz. RDWN has the lowest latency on C-MAPSS, 13.9 ns, and 14 to 22 ns over the ten datasets. Its pipelines are longer, 7 to 11 cycles, but its designs run at 450 to 580 MHz, around the 500 MHz target. HGQ and the int8 MLP have pipelines as short as RDWN's, 7 to 10 cycles, but their multiplications and adder trees limit them to 240 to 360 MHz, which gives them the highest geometric mean latencies .

6 Discussion

Input Quantization Overheads. The comparison in Section 5 considers the hardware utilization for the core model design alone, with each model reading inits own input format. However, a fully end-to-end system for the evaluated datasets would receive its features as 32-bit floats, the format on which every model in this work is trained and evaluated. HGQ, KANELÉ*, NeuraLUT-Assemble* and the int8 MLP first map each feature to a learned fixed-point code by scaling, offsetting, rounding and clamping it, and TreeLUT* in its published form quantizes each feature uniformly over its training range by min-max scaling. Table 6 therefore reports each method's most accurate design within 24,000 LUTs end to end, from 32-bit float features to a 16-bit integer prediction, with a generated float-to-code converter included in every design that needs one. We prioritized latency and throughput in these converters, so each is fully pipelined and accepts one sample per clock cycle, like the design it feeds, rather than sharing its arithmetic between features over several cycles. Each converter adds five to seven pipeline stages and a number of LUTs that grows with the number of features, from about 1,800 to 3,300 LUTs for the datasets with 8 to 11 inputs to between 15,700 and 21,300 LUTs for the 81 inputs of superconductivity, and it accounts for a median of 30%, 41% and 61% of the LUTs of the HGQ, KANELÉ* and NeuraLUT-Assemble* designs in Table 6. RDWN without quantized inputs and TreeLUT*'s float32-key designs instead compare each float32 feature directly with constant thresholds and need no converter, but they pay for 32-bit comparators, which make RDWN's float32-input designs a median of 26% larger than the same models reading 16-bit inputs. The budget of 24,000 LUTs is the smallest of our comparison budgets at which every LUT-based method has a design on all ten datasets end to end. The converter cost is most visible at small budgets. Within 2,000 LUTs end to end, only RDWN and HGQ have a

Table 6: Each method's most accurate design within 24,000 LUTs, end to end (32 bit float inputs / 16 bit integer outputs). Conv.: LUTs of the float32 input converter, included in LUT. LUT×ns: total LUTs occupied × latency. Bold, within each dataset: the lowest RMSE, LUT, FF, cycle, latency and LUT×ns values and the highest $F_{\max}$, compared before rounding (ties all bold).

Dataset	Configuration	RMSE	LUT	Conv.	FF	$F_{\max}$ (MHz)	Cycles	Latency (ns)	LUT×ns (10^3)
Air quality	RDWN 3.16 KiB, f32 in	21.30	**3,659**	–	3,606	**592**	11	18.6	**68.0**
	TreeLUT* 100×d8, f32 keys	20.42	18,550	–	2,739	407	**6**	**14.8**	273.7
	HGQ 32×32, β 5e-7, ep 335	**20.36**	15,768	1,815	4,132	255	16	62.7	988.2
	KANELÉ* 16 p0.14	21.30	5,538	2,114	**1,434**	509	16	31.4	173.9
	NeuraLUT-A* 512 leaves 4-bit	21.40	11,664	1,834	4,112	515	20	38.9	453.3
Wine quality	RDWN 49.47 KiB, f32 in	0.6271	7,692	–	7,110	500	**11**	**22.0**	**169.3**
	TreeLUT* 300×d6	**0.6112**	21,765	2,574	3,326	414	13	31.4	683.0
	HGQ 32×32, β 1e-7	0.6607	20,558	2,576	6,307	257	17	66.3	1362.3
	KANELÉ* 64 p0.14	0.6605	9,282	2,154	3,100	469	17	36.2	336.3
	NeuraLUT-A* 256 leaves 4-bit	0.6873	**7,123**	2,118	**2,195**	**535**	20	37.4	266.3
California housing	RDWN 6.30 KiB, f32 in	0.4736	5,792	–	4,649	505	10	19.8	114.7
	TreeLUT* 300×d5, f32 keys	**0.4647**	15,369	–	3,907	450	**6**	**13.3**	205.0
	HGQ 32×32, β 9e-7, ep 488	0.5347	16,710	1,997	4,298	245	16	65.4	1092.4
	KANELÉ* 16 p0.3	0.5350	5,127	2,278	1,311	513	16	31.2	159.8
	NeuraLUT-A* 32 leaves 4-bit	0.5963	**2,846**	2,138	**409**	**524**	16	30.6	**87.0**
C-MAPSS RUL	RDWN 0.98 KiB, f32 in	17.01	**576**	–	703	504	**7**	**13.9**	**8.0**
	TreeLUT* 50×d3, lr 0.3	17.11	6,574	5,721	694	405	13	32.1	211.0
	HGQ 16, β 5e-6, ep 828	**16.53**	6,795	4,432	688	363	14	38.6	262.3
	KANELÉ* 32×32 p0.14	17.19	17,229	5,857	5,516	480	23	47.9	825.4
	NeuraLUT-A* 64 leaves 3-bit	17.18	5,502	5,065	**632**	**517**	18	34.8	191.7
Diamonds	RDWN 32 KiB, f32 in	551.7	7,062	–	9,843	522	**11**	**21.1**	**148.8**
	TreeLUT* 300×d6	**550.6**	19,527	2,991	2,843	446	13	29.1	568.9
	HGQ 64×64×32, β 9e-7, ep 486	575.1	**6,147**	1,944	**1,333**	348	16	46.0	282.8
	KANELÉ* 16 p0.14	551.4	7,801	3,621	2,131	449	17	37.9	295.5
	NeuraLUT-A* 512 leaves 4-bit	585.6	13,253	3,503	4,358	**539**	20	37.1	492.0
LHC MET (FWX)	RDWN 15.93 KiB, f32 in	25.69	6,995	–	6,421	**579**	**11**	**19.0**	132.9
	TreeLUT* 150×d4	25.21	7,194	3,301	1,754	460	13	28.3	203.3
	HGQ 16, β 6e-7, ep 119	**25.02**	4,701	2,383	590	288	13	45.1	211.9
	KANELÉ* 2 p0.14	25.29	3,722	3,228	**298**	502	16	31.9	118.7
	NeuraLUT-A* 32 leaves 4-bit	25.46	**3,472**	2,752	407	522	17	32.5	**113.0**
SARCOS	RDWN 10.89 KiB, f32 in	2.109	19,593	–	15,681	**515**	**11**	**21.3**	**418.1**
	TreeLUT* 150×d4	1.999	23,374	6,269	8,456	389	13	33.4	780.3
	HGQ 64, β 6e-7, ep 385	1.933	22,423	6,097	4,786	282	15	53.2	1192.7
	KANELÉ* 32 p0.3	**1.657**	21,536	6,374	6,694	399	18	45.2	972.6
	NeuraLUT-A* 64 leaves 4-bit	2.366	**15,714**	6,527	**4,079**	505	18	35.7	560.6
Superconductivity	RDWN 27.86 KiB, f32 in	**10.13**	**15,942**	–	7,664	**528**	9	17.0	**271.6**
	TreeLUT* 300×d4, f32 keys	10.45	20,636	–	5,224	416	**6**	**14.4**	297.5
	HGQ 16, β 6e-6, ep 896	13.83	23,585	15,858	2,319	318	15	47.2	1113.7
	KANELÉ* 16 p0.14, 6-bit in	11.19	19,362	15,741	3,330	492	17	34.6	669.2
	NeuraLUT-A* 128 leaves 4-bit	11.85	23,925	21,316	**1,790**	500	19	38.0	910.1
CMS muon q/p_T	RDWN 64.7 KiB, f32 in	0.0426	11,192	–	8,449	**525**	11	20.9	**234.3**
	TreeLUT* 300×d6, f32 keys	0.0511	22,638	–	3,889	415	**6**	**14.4**	327.1
	HGQ 32×32, β 1e-7	**0.0384**	14,318	3,402	3,548	250	15	59.9	857.6
	KANELÉ* 32×32 p0.14	0.0399	**7,521**	2,472	2,424	486	20	41.1	309.3
	NeuraLUT-A* 256 leaves 4-bit	0.0531	8,559	3,943	**2,355**	519	19	36.6	313.0
Smart pixels	RDWN 34.29 KiB, f32 in	**0.1386**	23,473	–	13,016	**505**	11	21.8	511.5
	TreeLUT* 300×d4, f32 keys	0.1535	18,734	–	5,394	428	**6**	**14.0**	**262.8**
	HGQ 64×64×32, β 2e-5, ep 166	1.253	**17,775**	9,715	2,477	283	17	60.1	1068.5
	KANELÉ* 16 p0.14, 6-bit in	0.9448	23,034	19,025	3,616	492	18	36.6	842.1
	NeuraLUT-A* 128 leaves 3-bit	1.174	19,247	17,780	**2,032**	504	19	37.7	726.3

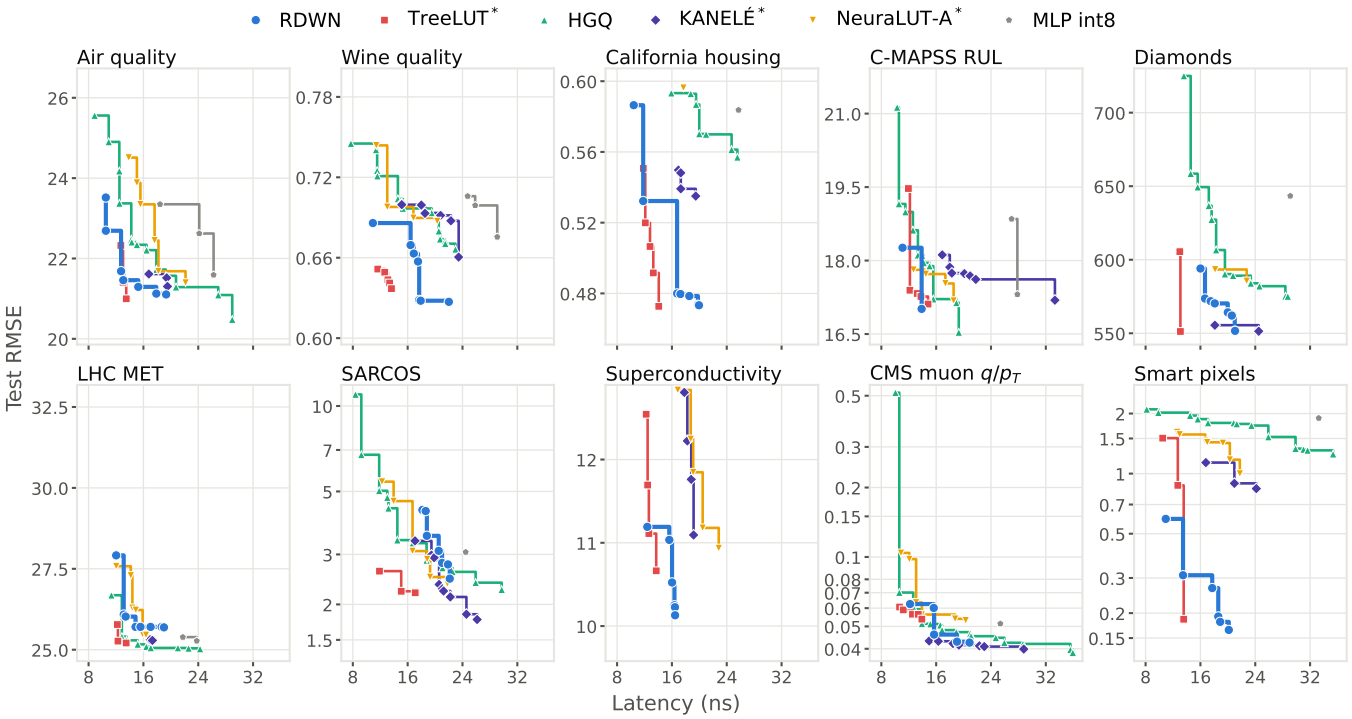


Figure 3: Test RMSE versus latency, design alone, over the designs within 12,000 LUTs. Each step curve represents a method's designs that are non-dominated in latency and RMSE. The RMSE axes are those of Figure 2.

design on every dataset, TreeLUT* has a design on nine, KANELÉ* and NeuraLUT-Assemble* on one each, and the int8 MLP on none. Within this budget RDWN is the most accurate method on seven of the ten datasets, with a geometric mean of normalized RMSE of 1.015 compared to 1.402 for HGQ.

Impact of Quantization on Accuracy. Unlike the uniformly spaced quantization levels used by the quantized-input baselines, RDWN places its thermometer thresholds at quantiles of the training data. This distinction has little effect on most datasets but is particularly important for smart pixels, whose features are concentrated near zero with long tails. It explains much of RDWN's accuracy advantage over HGQ, KANEL'E* and NeuraLUT-Assemble* on this dataset. Quantile-based or logarithmic input transformations could narrow this gap, but would increase the complexity and hardware cost of the baselines' float-to-code converters.

Hybrid Model Architectures. The results show that the models have different strengths. Within 12,000 LUTs, HGQ is the most accurate method on four datasets, RDWN on three, TreeLUT* on two and KANELÉ* on one, and on several datasets the most accurate model changes with area. A potential future work is therefore to consider ensembles of these models. Since RDWN, HGQ, KANELÉ* and NeuraLUT-Assemble* are all trained by gradient descent, their building blocks could be trained jointly in a single network. On CMS muon, for example, HGQ and KANELÉ* are the most accurate methods and RDWN's RMSE is about 11% higher within 12,000 LUTs, whereas on superconductivity RDWN leads and HGQ's RMSE is 35% higher, so a network that adds HGQ's weighted sums or KANELÉ*'s learned univariate functions to RDWN's LUTs could draw on both.

Polynomial Expansion. We note that for most datasets considered here, adding higher degrees of polynomial in the polynomial expansion layer did not necessarily result in improved model performance. However, this layer be valuable for certain other datasets having a higher correlation between higher order terms and the target response.

7 Conclusion

This work introduces RDWN, which extends Differentiable Weightless Neural Networks (DWNs) to regression applications, with a ternary popcount layer, polynomial feature expansion layer, and a learned regression head, which effectively get mapped onto an FPGA as layers of LUTs, comparators and adder trees. We compare RDWN with HGQ, KANELÉ*, NeuraLUT-Assemble*, TreeLUT*, and an int8 MLP, extending these LUT-based methods to regression wherever required. Evaluated on a Xilinx xcvu9p part, RDWN is competitive with the most accurate method on most datasets and has the lowest geometric mean of normalized RMSE over a wide range of LUT budgets. While no model is the most accurate on every dataset, however, we note that the methods have different strengths. RDWN provides the most accurate designs over most of the range from a few hundred LUTs up to between 7,000 and 21,000 LUTs on four of the sensor and tabular datasets, while HGQ and KANELÉ* are more accurate on the robotics dataset and on two of the particle physics datasets, one of which has a largely linear

target. RDWN's designs run at 450 to 580 MHz with latencies of 14 to 22 ns, and only TreeLUT* is consistently faster.

Overall, RDWN brings weightless neural networks to the realm of low-latency regression on FPGAs and is competitive with the strongest LUT-based methods. Placed side by side on the same regression tasks, FPGA and implementation flow, HGQ, TreeLUT, KANELÉ and NeuraLUT-Assemble also perform well on many of the datasets, which shows that LUT-based networks can deliver accurate regression under tight latency and area constraints.

References

[1] Georges Aad et al. 2021. Artificial Neural Networks on FPGAs for Real-Time Energy Reconstruction of the ATLAS LAr Calorimeters. *Comput. Softw. Big Sci.* 5, 1 (2021), 19. doi:10.1007/s41781-021-00066-y

[2] Igor Aleksander, WV Thomas, and PA Bowden. 1984. WISARD· a radical step forward in image recognition. *Sensor review* 4, 3 (1984), 120–124.

[3] 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. doi:10.1109/icfpt59805.2023.00012

[4] Marta Andronic and George A. Constantinides. 2024. NeuraLUT: Hiding Neural Network Density in Boolean Synthesizable Functions. arXiv:2403.00849 [cs.AR] https://arxiv.org/abs/2403.00849

[5] Marta Andronic and George A. Constantinides. 2025. 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)*. IEEE, 208–216. doi:10.1109/FCCM62733.2025.00077.

[6] 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. *2024 41st International Conference on Machine Learning (ICML)* (2024). https://icml.cc/virtual/2024/poster/34511

[7] Benjamin T. Carlson, Quincy Bayer, Tae Min Hong, and Stephen T. Roche. 2022. Nanosecond machine learning regression with deep boosted decision trees in FPGA for high energy physics. *Journal of Instrumentation* 17, 09 (2022), P09039. arXiv:2207.05602 doi:10.1088/1748-0221/17/09/P09039

[8] Jiasi Chen and Xukan Ran. 2019. Deep Learning With Edge Computing: A Review. *Proc. IEEE* 107, 8 (2019), 1655–1674. doi:10.1109/JPROC.2019.2921977

[9] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems* 47, 4 (2009), 547–553. doi:10.1016/j.dss.2009.05.016

[10] Saverio De Vito, Ettore Massera, Marco Piga, Luca Martinotto, and Girolamo Di Francia. 2008. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical* 129, 2 (2008), 750–757. doi:10.1016/j.snb.2007.09.060

[11] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P. Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, Dylan Rankin, Manuel Blanco Valentin, Josiah Hester, Yingyi Luo, John Mamish, Seda Orgrenci-Memik, Thea Aarrestad, Hamza Javed, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, Sioni Summers, Javier Duarte, Scott Hauck, Shih-Chieh Hsu, Jennifer Ngadiuba, Mia Liu, Duc Hoang, Edward Kreinar, and Zhenbin Wu. 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. In *Proceedings of TinyML Research Symposium (TinyML Research Symposium'21)*. ACM, 1–10. https://par.nsf.gov/servlets/purl/10300178

[12] Sebastian Felix Fischer, Liana Harutyunyan, Matthias Feurer, and Bernd Bischl. 2023. OpenML-CTR23 – A Curated Tabular Regression Benchmarking Suite. In *AutoML Conference 2023 (Workshop)*. https://openreview.net/forum?id=HebAOoMm94

[13] Kam Hamidieh. 2018. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science* 154 (2018), 346–354. doi:10.1016/j.commatsci.2018.07.052

[14] Duc Hoang, Aarush Gupta, and Philip C Harris. 2026. KANELÉ: Kolmogorov–Arnold Networks for Efficient LUT-based Evaluation. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 44–55. doi:10.1145/3748173.3779202

[15] Yutaro Iiyama, Gianluca Cerminara, Abhijay Gupta, Jan Kieseler, Vladimir Loncar, Maurizio Pierini, Shah Rukh Qasim, Marcel Rieger, Sioni Summers, Gerrit Van Onsem, Kinga Anna Wozniak, Jennifer Ngadiuba, Giuseppe Di Guglielmo, Javier Duarte, Philip Harris, Dylan Rankin, Sergo Jindariani, Mia Liu, Kevin Pedro, Nhan Tran, Edward Kreinar, and Zhenbin Wu. 2021. Distance-Weighted Graph Neural Networks on FPGAs for Real-Time Particle Reconstruction in High Energy Physics. *Frontiers in Big Data* 3 (Jan. 2021). doi:10.3389/fdata.2020.598927

[16] Alireza Khataei and Kia Bazargan. 2025. 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* (Monterey, CA, USA) *(FPGA '25)*. Association for Computing Machinery, New York, NY, USA, 14–24. doi:10.1145/3706628.3708877

[17] Emre Kurtoglu. [n. d.]. CMS_dataset. Kaggle. Dataset, version 1. https://www.kaggle.com/datasets/ekurtoglu/cms-dataset

[18] Igor D.S. Miranda, Aman Arora, Zachary Susskind, Luis A.Q. Villon, Rafael F. Katopodis, Diego L.C. Dutra, Leandro S. De Araújo, Priscila M.V. Lima, Felipe M.G. França, Lizy K. John, and Mauricio Breternitz. 2022. LogicWiSARD: Memoryless Synthesis of Weightless Neural Networks. In *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 19–26. doi:10.1109/ASAP54787.2022.00014

[19] Rustem Ospanov, Changqing Feng, Wenhao Dong, Wenhao Feng, Kan Zhang, and Shining Yang. 2022. Development of a resource-efficient FPGA-based neural network regression model for the ATLAS muon trigger upgrades. *The European Physical Journal C* 82, 6 (2022). doi:10.1140/epjc/s10052-022-10521-8

[20] R. Kelley Pace and Ronald Barry. 1997. Sparse spatial autoregressions. *Statistics & Probability Letters* 33, 3 (1997), 291–297. doi:10.1016/S0167-7152(96)00140-X

[21] Carl Edward Rasmussen and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA. doi:10.7551/mitpress/3206.001.0001

[22] Stephen Roche, Benjamin Carlson, and Tae Min Hong. 2022. fwXmachina example: Missing transverse energy regression. Mendeley Data. Version 1. doi:10.17632/d4c94r9254.1

[23] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. 2008. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*. IEEE, Denver, CO, USA, 1–9. doi:10.1109/PHM.2008.4711414

[24] Pavel Serhiayenka, Stephen T. Roche, Benjamin T. Carlson, and Tae Min Hong. 2025. Nanosecond hardware regression trees in FPGA at the LHC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1072 (2025), 170209. doi:10.1016/j.nima.2025.170209

[25] Chang Sun, Zhiqiang Que, Thea Aarrestad, Vladimir Loncar, Jennifer Ngadiuba, Wayne Luk, and Maria Spiropulu. 2026. HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs. In *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '26)*. Association for Computing Machinery, New York, NY, USA, 79–91. doi:10.1145/3748173.3779200

[26] Zachary Susskind, Aman Arora, Igor D.S. Miranda, Luis A.Q. Villon, Rafael F. Katopodis, Leandro S. De Araújo, Diego L.C. Dutra, Priscila M.V. Lima, Felipe M.G. França, Mauricio Breternitz, and Lizy K. John. 2022. Weightless Neural Networks for Efficient Edge Inference. In *31st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. doi:10.1145/3559009.3569680

[27] Zachary Susskind, Aman Arora, Igor D. S. Miranda, Alan T. L. Bacellar, Luis A. Q. Villon, Rafael F. Katopodis, Leandro S. de Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França, Mauricio Breternitz Jr., and Lizy K. John. 2023. ULEEN: A Novel Architecture for Ultra-low-energy Edge Neural Networks. *ACM Trans. Archit. Code Optim.* 20, 4, Article 61 (dec 2023), 24 pages. doi:10.1145/3629522

[28] Morris Swartz and Jennet Dickinson. 2024. Smart pixel dataset. Zenodo. Version 3. doi:10.5281/zenodo.10783560

[29] Yaman Umuroglu, Yash Akhauri, Nicholas J. Fraser, and Michaela Blott. 2020. LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications. *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)* (2020), 291–297.

[30] Sethu Vijayakumar and Stefan Schaal. 2000. Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*. Morgan Kaufmann, 1079–1086.

[31] 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. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, CA, USA) *(FPGA '25)*. Association for Computing Machinery, New York, NY, USA, 25–35. doi:10.1145/3706628.3708874

[32] Hadley Wickham. 2016. *ggplot2: Elegant Graphics for Data Analysis* (2 ed.). Springer International Publishing, Cham. doi:10.1007/978-3-319-24277-4

[33] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proc. IEEE* 107, 8 (2019), 1738–1762. doi:10.1109/JPROC.2019.2918951