Greater than the Sum of its LUTs: Scaling Up LUT-based Neural Networks with AmigoLUT

Olivia Weng
oweng@ucsd.edu
University of California San Diego
San Diego, CA, USA

Marta Andronic
Imperial College London
London, UK

Danial Zuberi
University of California San Diego
San Diego, CA, USA

Jiaqing Chen
Arizona State University
Tempe, AZ, USA

Caleb Geniesse
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

George A. Constantinides
Imperial College London
London, UK

Nhan Tran
Fermi National Accelerator Laboratory
Batavia, IL, USA

Nicholas J. Fraser
AMD Research and Advanced Development
Dublin, Ireland

Javier Mauricio Duarte
Ryan Kastner
University of California San Diego
San Diego, CA, USA

Abstract

Applications like high-energy physics and cybersecurity require extremely high throughput and low latency neural network (NN) inference. Lookup-table-based NNs address these constraints by implementing NNs as lookup tables (LUTs), achieving inference latency on the order of nanoseconds. Since LUTs are a fundamental FPGA building block, LUT-based NNs efficiently map to FPGAs. LogicNets (and its successors) form one class of LUT-based NNs that target FPGAs, mapping neurons directly to LUTs to meet low latency constraints with minimal resources. However, it is difficult to build larger, more performant LUT-based NNs like LogicNets because LUT usage increases exponentially with respect to neuron fan-in (i.e., number of synapses × synapse bitwidth). A large LUT-based NN quickly runs out of LUTs on an FPGA. Our work AmigoLUT addresses this issue by creating ensembles of smaller LUT-based NNs that scale linearly with respect to the number of models. AmigoLUT improves the scalability of LUT-based NNs, reaching higher throughput with up to an order of magnitude fewer LUTs than the largest LUT-based NNs.

CCS Concepts

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

Keywords

Edge AI; FPGA; hardware-software codesign; neural networks

ACM Reference Format:
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 (FPGA '25), February 27–March 1, 2025, Monterey, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3706628.3708874

FPGA '25, February 27–March 1, 2025, Monterey, CA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1396-5/25/02
https://doi.org/10.1145/3706628.3708874

1 Introduction

Machine learning has become increasingly prevalent in areas such as high-energy physics, cybersecurity, and autonomous vehicles [10, 12, 18, 29, 31]. Many of these domains require extremely high throughput and low latency. For example, at the CERN Large Hadron Collider (LHC), the Compact Muon Solenoid experiment [8] runs particle collision experiments that generate data rates of ~40 TB/s, which physicists need to compress in real time (≤25 ns) to lower the data bandwidth to a level that their system can handle [10, 12].

Recently, lookup-table-based neural networks (NNs) such as LogicNets [29] and NeuraLUT [4] have sought to meet this demand for ultra low-latency, high-throughput NNs. For instance, LogicNets achieve efficient computation by carefully mapping neurons to lookup tables (LUTs), a fundamental building block of field-programmable gate arrays (FPGAs). This makes LUT-based NNs efficient, yielding high throughput and low latency implementations while sacrificing some accuracy compared with dense NNs.

However, it is challenging to create more accurate LUT-based NNs because LUT usage increases exponentially ($\Omega(2^n)$) with respect to neuron fan-in (i.e., n = number of synapses × synapse bitwidth). As such, LUT-based NNs must be heavily quantized and very sparse to efficiently map to FPGA LUTs, which are 6 inputs or less. LUT-based NNs have a fundamental tradeoff between input size and LUT size because LUT size increases exponentially with input size. The standard method of improving NN accuracy [1, 7] by increasing NN depth, width, and neuron fan-in fails for LUT-based NNs because they quickly run out of the available LUTs on an FPGA due to exponential scaling. As a result, scaling up LogicNets by increasing NN size and density presents a poor tradeoff because we get minimal accuracy increase at the cost of exponential increases in LUT usage as well as decreases in throughput.

Previously, researchers have increased model accuracy through ensemble learning (*ensembling*) [11, 15, 24, 33, 35]. Ensembling increases model accuracy by having multiple, weaker NNs work together to make a decision. The idea behind ensembling is that the whole is greater than the sum of its parts; by having multiple NNs work together, we can achieve better performance than any one model individually. Prior work [6, 14] often forms ensembles of many small and weak models to promote diversity in the decision each model makes (i.e., they must disagree with each other to an extent [28]). Otherwise, if all the models always agree with each other, then there is no benefit to combining them together. Through this "diversity of thought," the ensemble can make a better decision as a group. Since LUT-based NNs are constrained by neuron fan-in, they must be extremely sparse and quantized to avoid an explosion in LUT usage, sacrificing some accuracy and making them relatively weak NNs. As a result, LUT-based NNs are good candidates for ensembling.

Our work AmigoLUT[1] creates ensembles of smaller LUT-based NNs to improve accuracy, scaling up performance *linearly* with respect to the number of models rather than *exponentially* with respect to neuron fan-in. Although AmigoLUT does not achieve higher accuracy or lower LUT usage than recent state-of-the-art work [5, 16], our results show that, for certain lower accuracy benchmarks, AmigoLUT increases throughput and reduces LUT usage by over an order of magnitude. Thus, the goal of our paper is to demonstrate the great potential of scaling up LUT-based NNs with ensembling.

How to apply ensembling to LUT-based NNs effectively presents many challenges: (1) which ensembling method should we use for LUT-based NNs? (2) how can we analyze their ensemble performance? (3) how can we implement these ensembles on FPGAs efficiently? Although many ensembling methods have been introduced over the years [33, 35], it is not immediately clear which one would work best on LUT-based NNs and why. Few ensembling studies have been completed on NNs that are both extremely sparse and quantized let alone LUT-based NNs [25, 32, 36]. Moreover, how to map LUT-based NN ensembles to FPGAs while minimizing the resource overhead needed to combine the ensemble members together is non-trivial.

In our paper, we tackle these challenges by evaluating three fundamental ensemble learning methods: averaging [35], bagging [6], and AdaBoost [14]. We study how LogicNets [29], PolyLUT [3] and NeuraLUT [4] ensemble together on two high-energy physics datasets [12] and the MNIST image classification dataset [34]. From our results, we find that averaging performs the best. We then analyze our ensemble learning results using diversity and disagreement metrics. We introduce a new diversity plot for visualizing ensemble diversity to understand why averaging is the best. Finally, we introduce an open-source tool AmigoLUT[2], a hardware-friendly ensemble learning framework for LogicNets, PolyLUTs, and NeuraLUTs that minimizes ensemble resource overhead when implementing ensembles on FPGAs.

Our primary contributions are: (1) evaluating various ensemble learning methods on LUT-based NNs such as LogicNets, PolyLUT, and NeuraLUT (2) analyzing ensembles using novel diversity plots and disagreement metrics to better understand their performance, and (3) introducing AmigoLUT, an ensemble learning framework that efficiently maps ensembles of LUT-based NNs to FPGAs.

[1]"Amigo" means "friend" in Spanish.
[2]https://github.com/KastnerRG/amigolut

2 Background & Related Work

This section reviews relevant background on ensembling and related work on LUT-based NNs.

2.1 Ensembling

Ensembling has been studied in the fields of statistics and machine learning for decades, and it has shown notable improvements in performance [6, 14, 15, 35]. While there are many ensembling methods to choose from, we start with three seminal ensembling methods: averaging [35], bagging [6], and AdaBoost [14]. The goal of each method is to promote diversity among the ensemble members so that together they can make a decision better than any one model could individually.

Averaging involves taking the average of the outputs of all ensemble members. Averaging introduces diversity by randomly initializing each ensemble member's weights differently, training them independently or dependently. In our work, we randomly initialize each model's weights differently and backpropagate the averaged outputs of the ensemble through each model, training the models dependently together as one.

Bagging, an abbreviation of bootstrap aggregating, involves randomly sampling the training set with replacement to create a unique training set per ensemble member, which is then each trained independently. These unique training sets introduce diversity into the ensemble because each ensemble member trains on different distributions of the data. The ensemble members' outputs can be combined in many ways, but in our work, we average the outputs.

AdaBoost is a subclass of *boosting* [35], which is an ensembling method that trains each ensemble member sequentially, where each model's performance informs how the subsequent model will be trained. AdaBoost, or adaptive boosting, achieves this by assigning weights to each training sample, starting with equal weights. AdaBoost uses these weights to randomly sample from the training set. As each model trains, the weights of the training samples are adjusted based on how the model performs on them. The samples with poor performance are assigned higher weight so that they are sampled more frequently when the next model trains. This way, each subsequent model targets training samples that the previous models performed poorly on, in an effort to "boost" each other's performance. To combine the ensemble's outputs, AdaBoost weights each model based on its performance, computing a weighted average of each model's outputs to produce the final output.

Bagging and AdaBoost were first studied on decision trees, which are relatively weak, increasing accuracy remarkably [6, 14]. Prior work [35] has attributed this increase in accuracy to the combination of weak models along with their unique methods of promoting diversity in training. Based on this observation, we hypothesize that sparse, quantized LUT-based NNs should ensemble together well because they are weaker than their dense, full-precision counterparts. Few works [13, 25] have evaluated ensembling NNs that are both sparse and quantized. We evaluate our hypothesis on averaging, bagging, and AdaBoost in Sec. 3.

2.2 LUT-based NNs

Many kinds of LUT-based NNs have been introduced in the literature.

LUTNet [30] uses LUTs as the fundamental inference operator to implement NNs on FPGAs. LUTNet seeks to improve binary NNs' reliance on XNOR operations by replacing them with more expressive, learnable K-LUT Boolean operations. While LUTNet improves NN resource efficiency on FPGAs, the number of parameters in LUTNet scales exponentially with respect to the number of LUT inputs, making it difficult to improve accuracy by increasing model size through parameter count.

Weightless NNs (WNNs) are another class of LUT-based NNs [2, 19, 26, 27]. WNNs consist of neurons, but they do not use weights to determine how they respond (i.e., they are weightless). WNN neurons are made up of logical LUTs that represent Boolean functions. WNNs suffer from exponential scaling with respect to the number of LUT inputs as well as difficulty in generalizing to unseen input data, as they primarily memorize patterns [26]. Recent work, such as differentiable WNNs (DWNs) [5] improve WNN generalization by making them differentiable so that they learn connections between LUTs rather than using fixed random setups.

Ensembling WNNs has been done before [13, 25]. ULEEN [25] ensembles WNNs by training many and selecting a few of the best performing models to train together, summing their outputs in the end to make decisions. Filho et al. [13] evaluate bagging and boosting on their WNNs; however, they show minimal accuracy gains, quickly hitting diminishing returns as ensemble size increases.

LogicNets [29] and NullaNet [20, 21] fully encapsulate a neuron's operation within a logical LUT. Given a trained quantized NN, LogicNets feed all possible quantized inputs to a neuron and captures the quantized outputs it computes, while NullaNet does so in a lossy manner. These inputs and outputs are then enumerated in a logical LUT and then implemented as physical LUTs by a logic synthesis tool.

PolyLUT [3] and NeuraLUT [4] are successors of LogicNets. Each method improves the accuracy of LogicNets by attempting to capture more complex functions within the logical LUT. PolyLUT learns a NN that is a piece-wise polynomial function whereas NeuraLUT encapsulates an arbitrary NN within a logical LUT. PolyLUT-Add [16] sums smaller PolyLUT logical LUTs together to build larger, more complex LUTs. However, similar to LogicNets, all three methods still scale exponentially with respect to LUT inputs. We evaluate AmigoLUT on LogicNets, PolyLUT, and NeuraLUT, creating ensembles with them as base models. Although each base model still suffers from exponential scaling with respect to LUT inputs, AmigoLUT alleviates this by relying on weaker models that have smaller LUT fan-in. As a result, AmigoLUT scales performance *linearly* with respect to the number of models.

3 Ensembling LUT-based NNs

The first step in ensembling LUT-based NNs is determining which ensembling method increases accuracy the most. We study three fundamental ensembling methods: (1) averaging, (2), bagging, and (3) AdaBoost. We seek to answer: (1) which ensembling method performs the best, and (2) why does one ensembling method outperform another?

3.1 Ensembling results

To determine which of our three ensembling methods is the best, we evaluate LogicNets models on three datasets: MNIST [34], high-granularity endcap calorimeter (HGCal) compression [12], and jet substructure classification (JSC) [12].

MNIST is a well-known dataset tasked with classifying handwritten digits. The HGCal and JSC datasets are latency-critical tasks that come from the LHC's Compact Muon Solenoid experiment [8]. At the LHC, physicists run particle collision experiments that generate data rates of ~40 TB/s. To reduce data rates, the physicists plan to deploy tens of thousands NN encoders in the LHC to compress particle collision data from the HGCal [9] sensor instrument into a smaller format. This encoded data is then passed to a decoder off-sensor for further processing in the trigger system. We evaluate HGCal model performance using Earth mover's distance (EMD), a distance measure between two probability distributions [23]. In our case, the EMD measures the distance between the encoder's input energy readings and the decoder's outputs, respectively. Lower EMD is better, and an EMD of 0 indicates the autoencoder is lossless. The trigger system, to which the decoder outputs data, performs many tasks, one of which is JSC. JSC involves classifying particle collision data into five types of jets, which informs physicists about potentially new physics. Of note, the HGCal encoder hardware must accept new input data at 40 MHz and complete inference in 25 ns. For each dataset, we ensemble NNs of different sizes.

From our experiments, we find that averaging works the best. As seen in Fig. 1, we see that for the small (JSC-S), medium (JSC-M), and large (JSC-L) models, averaging increases accuracy the most. For instance, averaging JSC-S increases accuracy by 4.1%, whereas bagging and AdaBoost increase accuracy by at most 1.8% and 0.4%, respectively. Similarly for MNIST, the extra small model (MNIST_XS) increases in accuracy by 4.4% at ensemble size 16, whereas bagging and AdaBoost increase accuracy by 3.6% and 3.1%, respectively (Fig. 2). For HGCal, averaging the encoders together lowers EMD significantly, where lower is better. Our goal is to hit ~1.1 EMD, which is a dense, `float32` NN's performance. As seen in Fig. 3, 32 small models (HGCal_S) decreases EMD by 28%, whereas bagging and AdaBoost decrease EMD by 5% and 9%, respectively. For the large HGCal model (HGCal_L), averaging 16 models decreases EMD the most, but the trend is nonlinear. This is likely due to the difficulty involved with encoding and decoding HGCal data, which has a larger margin of error than classifying data into a few discrete classes. In particular, when performing a NN architecture random search of ~500 LogicNets to find a low-EMD NN, we found that any NN larger than HGCal_L led to worse EMD. Only by ensembling these models together can we significantly lower EMD to get close to our goal of 1.1 EMD.

We also observe that smaller models benefit from ensembling the most, increasing in performance significantly. In Fig. 2, we see that averaging MNIST_XS improves accuracy by 4.9%, whereas the large model (MNIST_L) only increases accuracy by 0.2%. Notably, bagging performs the best, but only outperforms averaging by

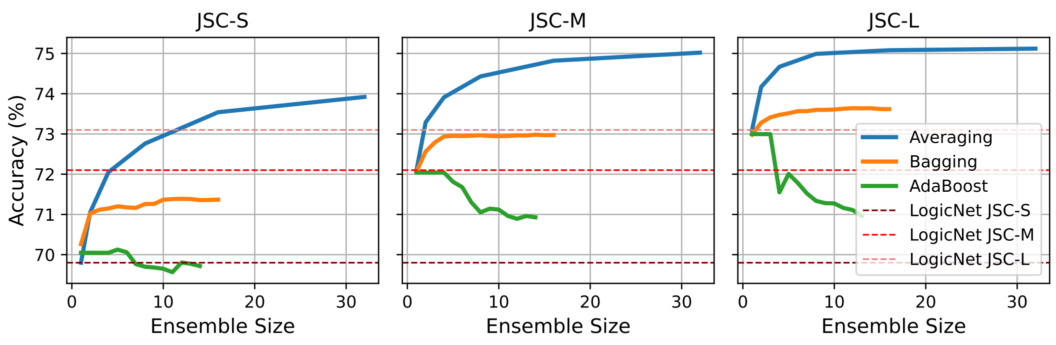

Figure 1: JSC Ensembling Comparison. Averaging is the best ensembling method for the JSC task, whereas bagging quickly hits diminishing returns and AdaBoost decreases performance overall.

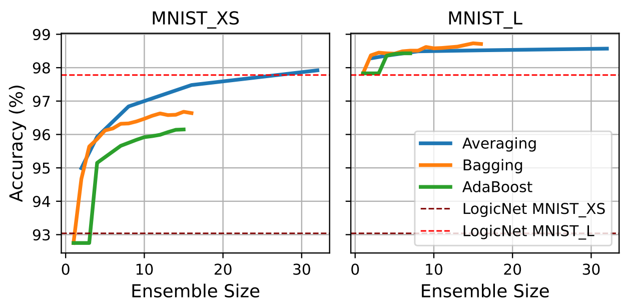

Figure 2: MNIST Ensembling Comparison. The extra small model MNIST_XS ensembles better than MNIST_L because as ensemble size increases it increases in accuracy more.

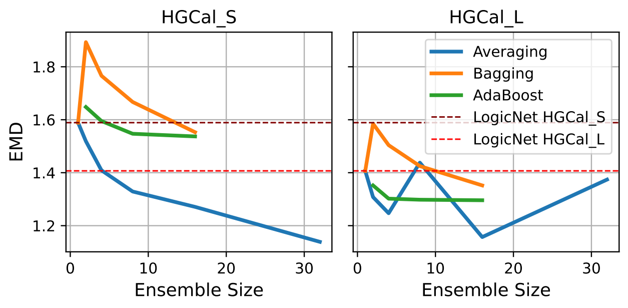

Figure 3: HGCal Ensembling Comparison. The small model HGCal_S responds to averaging the best, decreasing EMD (lower is better) by 28% whereas HGCal_L responds nonlinearly, decreasing EMD by only 17%.

0.14%, with all ensembling methods quickly hitting diminishing returns after two models. Similarly, in Fig. 3, ensembling HGCal_S improves EMD by 28% whereas the large model (HGCal_L) only improves by 17%.

3.2 Analyzing ensemble performance

We find averaging to perform the best, but we also want to understand why it does well whereas bagging and AdaBoost underperform. For instance, for JSC, AdaBoost decreases accuracy for all three models. As previously stated in Sec. 1 and Sec. 2, ensembling performs well because it promotes diversity, especially among weaker models, so that they can make a better decision collectively [35]. But, as we have seen, sometimes this is not the case. To that end, we analyze ensemble performance via a diversity analysis.

Diversity analyses have been performed in the machine learning community to better understand ensembling performance, primarily on classification tasks [17, 28]. Previously, researchers have quantified diversity by measuring disagreement among ensemble members as a proxy for diversity. In particular, recent work [28] argues that balancing disagreement with model error is important for good ensemble performance. We conduct a similar diversity analysis on our classification tasks, namely JSC and MNIST, to better understand their ensemble performance. Our diversity analysis can be used to analyze ensemble performance for not only other LUT-based NNs but any NN.

We introduce a new way of visualizing the diversity of ensemble member prediction, as seen in Fig. 4, Fig. 5, and Fig. 6, by showing ensemble member voting across individual samples (using $n = 50$). Our diversity visualizations reveal the extent to which the members of an ensemble agree with each other. We show diversity plots for the different ensembling methods on the MNIST and JSC tasks in Fig. 4 and Fig. 5, respectively. For the MNIST task, we show the diversity plots for MNIST_XS and ensemble size 16. For the JSC task, we show the diversity plots for JSC_S and ensemble size 32. Note, the x-axis corresponds to the different classes, and the y-axis corresponds to an input sample we pass through the ensemble members individually. Each line represents a sample, and a peak in class i means an individual ensemble member predicted class i. Higher peaks mean that more ensemble members voted for that class. The samples are ordered by target class, such that the first k samples correspond to the first target ($y = 0$), the second k samples correspond to the second target ($y = 1$), and so on. The lines are grouped by color based on the class the data sample belongs to.

Thus, for an ensemble with no diversity and zero error, we should only see peaks along the diagonal. For the MNIST task where we have ten classes, we use $k = 5$ samples per class. For the JSC task where we have five classes, we use $k = 10$ samples per class. This gives a total of 50 samples per plot. While we observe much lower diversity for the MNIST task compared to the JSC task, we find

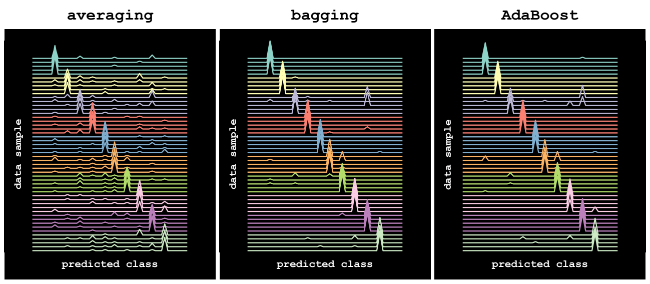

Figure 4: Diversity plots for MNIST_XS. Averaging exhibits the most diversity, as seen in the small peaks off the diagonal (which are the incorrect class predictions), meaning there is some disagreement among the ensemble members. Diversity has been shown to improve ensembling, confirming averaging's superior performance.

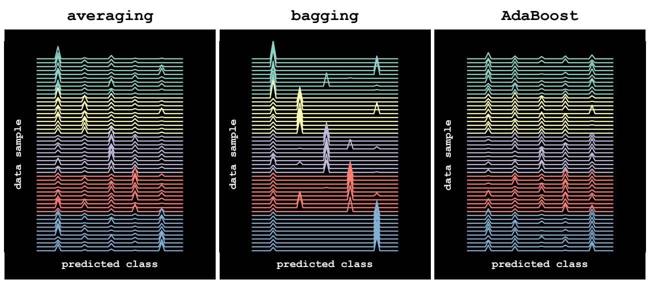

Figure 5: Diversity plots for JSC_S. While AdaBoost is very diverse, the high peak density off the diagonal shows that many ensemble members classify incorrectly. Averaging best balances diversity with correctness.

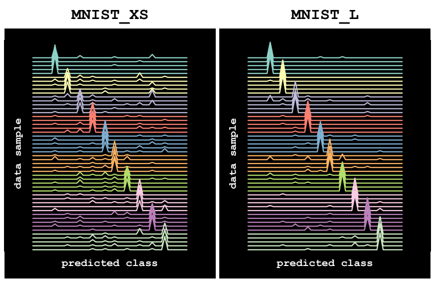

Figure 6: Diversity plots for averaging MNIST_XS and MNIST_L. MNIST_XS exhibits more diverse predictions than MNIST_L, suggesting small, weaker models ensemble better.

that the averaging method is consistently more diverse compared to the other ensembling methods, as seen in the small peaks off the diagonal. Another interesting observation is that for the JSC task, while AdaBoost appears to be more diverse than bagging, there appears to be more density off the diagonal, suggesting more incorrect predictions. Moreover, in Fig. 6, we observe more diversity in averaging MNIST_XS than in MNIST_L. This corresponds to the large increases in accuracy for MNIST_XS compared with the diminishing returns of MNIST_L we observe in Fig. 2.

	Disagreement (↑)	Error (↓)	DER (↑)
Averaging	**0.460**	0.302	**1.522**
Bagging	0.096	**0.078**	1.232
AdaBoost	0.099	0.082	1.208

Table 1: Diversity metrics for the MNIST task. Bagging and AdaBoost both have low disagreement and error. Their ensemble members agree with each other a lot, implying less benefit from ensembling as compared with Averaging.

	Disagreement (↑)	Error (↓)	DER (↑)
Averaging	0.704	0.620	**1.137**
Bagging	0.316	**0.374**	0.844
AdaBoost	**0.752**	0.693	1.085

Table 2: Diversity metrics for the JSC task. Averaging balances disagreement and error the best, achieving high DER.

To further verify our qualitative observations, we quantify model disagreement and compute the disagreement error ratio (DER) [28]. DER is defined as

$$\text{DER} = \frac{\mathbb{E}_{m_i,m_j\sim\rho}[\text{Disagreement}(m_i, m_j)]}{\mathbb{E}_{m_i\sim\rho}[\text{Error}(m_i)]} \tag{1}$$

provided that $\mathbb{E}_{m_i\sim\rho}[\text{Error}(m_i)] \neq 0$, and where ρ is the distribution of members m_i in the ensemble, $\text{Disagreement}(m_i, m_j)$ is the prediction disagreement between the two ensemble members m_i and m_j, and $\text{Error}(m_i)$ is the prediction error for ensemble member m_i. For a given data distribution $\mathcal{D}$, the disagreement between any two ensemble members is defined as

$$\text{Disagreement}(m_i, m_j) = \mathbb{E}_{x\sim\mathcal{D}}[\mathbf{1}(m_i(x) \neq m_j(x))] \tag{2}$$

where $\mathbf{1}(m_i(x) \neq m_j(x))$ is 1 if $m_i(x) \neq m_j(x)$ else it is 0. The prediction error for a single ensemble member is defined as

$$\text{Error}(m_i) = \mathbb{E}_{x,y\sim\mathcal{D}}[\mathbf{1}(m_i(x) \neq y)] \tag{3}$$

where $\mathbf{1}(m_i(x) \neq y)$ is 1 if $m_i(x) \neq y$ else it is 0. Importantly, DER provides a measure of diversity that balances model disagreement with prediction error, and a higher DER suggests that ensembling should improve performance [28].

We report these metrics (based on $n = 680$ samples) for the MNIST and JSC tasks in Tab. 1 and Tab. 2, respectively. For the MNIST task, we report the metrics for MNIST_XS and ensemble size 16. For the JSC task, we report the metrics for JSC_S and ensemble size 32. Notably, averaging had the highest DER across both tasks. While bagging has the lowest prediction error, it also had the lowest disagreement, implying low diversity. For the JSC task, while AdaBoost had a slightly higher level of disagreement compared to averaging, it also made more errors. Together these metrics reveal differences in the diversity of ensemble members that are related to model performance. We find that averaging balances disagreement and error the best, leading to superior ensembling performance.

4 Quantization Architecture Tradeoffs

As seen in the previous section, averaging works the best among the ensembling methods we study. Next, we want to implement

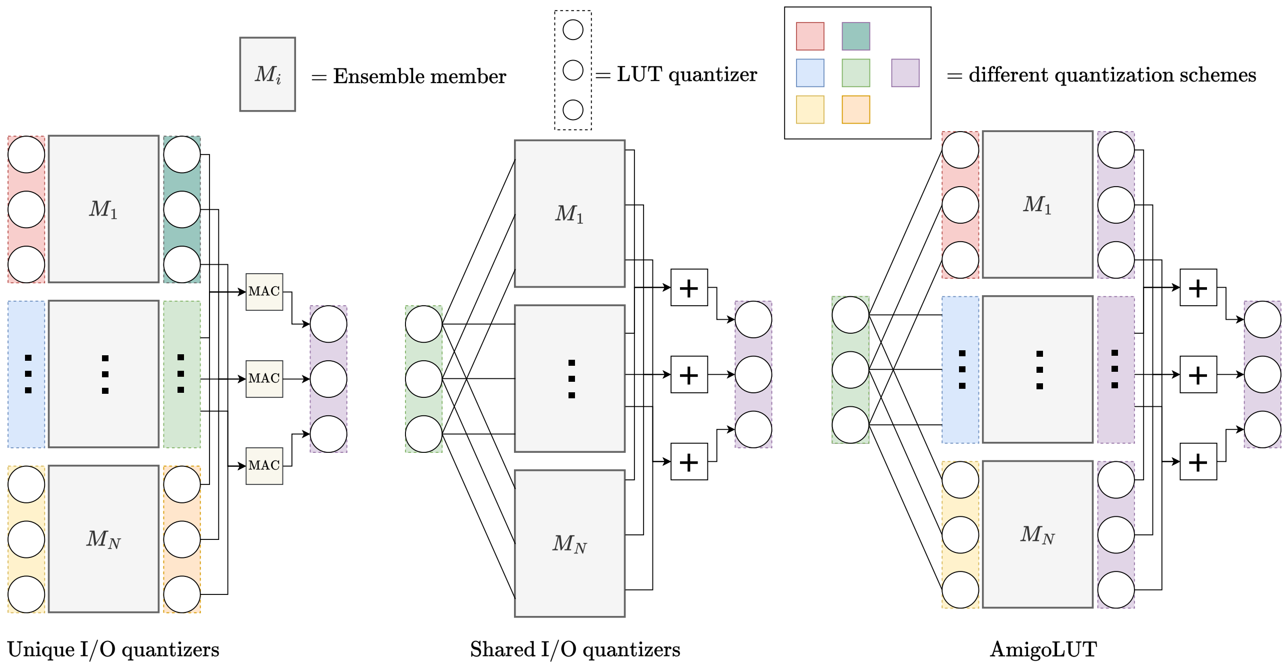

Figure 7: Mapping ensembles to FPGAs requires careful consideration of how to handle the different quantization schemes exhibited by each ensemble member M_i's inputs and outputs (a.k.a. I/O). With unique I/O quantizers (left), we must implement both input and output quantizers per M_i. This becomes expensive when quantizing from say JSC's 16-bit input data to the 2-bit precision each M_i expects. To combine outputs of different quantization schemes, we also need to perform a multiply-accumulate (MAC) to scale the outputs based on their learned scaling factor so that they can be combined. Ideally, we would share I/O quantizers (middle), however this decreases ensemble performance, especially for difficult tasks. AmigoLUT (right) balances accuracy and resources by first quantizing the input data to low precision (i.e., 6 bits) with a single shared quantizer (green) before sending the data to each M_i to be uniquely quantized. Each M_i's input neuron can be quantized with only two LUT6 to the lower 2-bit precision it expects. At the output, AmigoLUT implements an extra NN layer that quantizes each M_i's output to the same quantization scheme so that they can be summed together efficiently.

this ensemble on an FPGA; however, this presents many challenges in minimizing resource overhead while preserving ensemble performance.

If we were to directly map the averaging ensembles in the last section to an FPGA, we would need to map an ensemble that looks like the design on the left in Fig. 7. As previously stated in Sec. 1 and Sec. 2, LogicNets are heavily quantized to minimize LUT utilization. LogicNets are quantized using Brevitas [22], which quantizes values to a given precision, say 6-bit integers. However, to improve quantized NN performance, Brevitas learns a floating point scaling factor. The Brevitas quantizer works by mapping activations to a limited set of discrete integer levels based on the specified bitwidth (e.g., 2 bits). However, the output of the quantizer remains in floating-point format because these integer levels are scaled by a learned floating-point scale factor and shifted by a learned zero-point (bias). This becomes an issue when we want to combine ensemble outputs efficiently. When we naively train an ensemble of LogicNets, Brevitas will learn different floating point scaling factors for each ensemble member's input and output quantizers—even though the inputs and outputs are quantized to the same bitwidth. As a result, in hardware, we would have to implement a unique input quantizer per model in the ensemble. This is resource-intensive. For instance, the HGCal data is quantized to 8 bits and each LogicNet model expects say 4-bit inputs, so we would have to implement N $8 \rightarrow 4$ bit input quantizers, which costs $N\times$ as many resources compared with mapping a single LogicNet to an FPGA. Moreover, since each model's output also exhibits a different quantization scheme, we would need to scale each model's output by the floating point scaling factor Brevitas learns. This requires multiplication using either DSPs or LUTs, if we can manage to also quantize the floating point scaling factors to integers. As seen in Fig. 7 (left), combining all ensemble outputs requires implementing multiply-accumulate (MAC) operations, costing extra resources.

Ideally, we would want to implement the ensemble such that all models share a single input quantizer and quantize the outputs using the same scaling factor, as seen in Fig. 7 (middle). By quantizing the outputs using the same scaling factor, we would only need to sum the ensemble member's outputs to get the final result. This would drastically reduce the resource overhead needed to quantize the inputs and combine the outputs. However, quantizing all of the

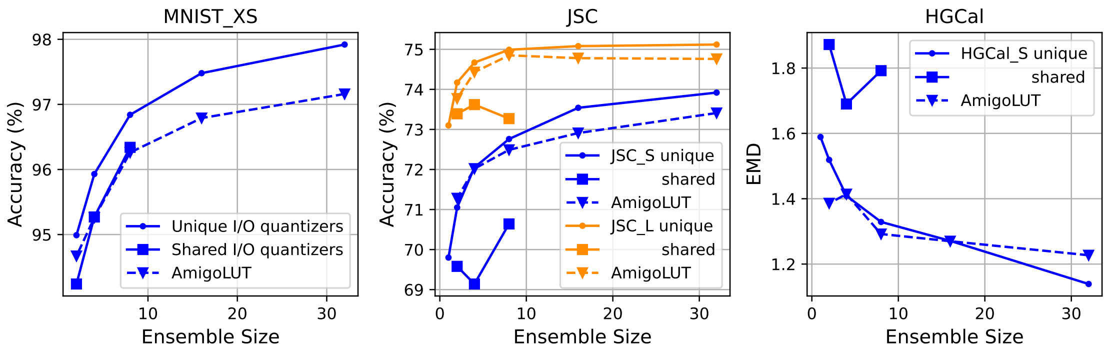

Figure 8: Unique quantizers vs. naively shared quantizers vs. AmigoLUT case study on LogicNets. The three graphs show the accuracy or EMD (y-axis) vs. the ensemble size (x-axis) for three different tasks (MNIST, JSC, and HGCal). Each graph compares the different input and output quantization methods as described in Fig. 7. The JSC application shows two base model—small (S) and large (L). Lower EMD results (HGCal) indicate better model performance.

inputs and outputs in a similar manner leads to poor performance, even undercutting the benefits of ensembling. In Fig. 8, we compare the performance of using a unique input/output (I/O) quantizer per ensemble member versus an I/O quantizer shared between all ensemble members. While for MNIST sharing I/O quantizers does not reduce accuracy by much (<1%), performance reduces drastically for JSC and HGCal tasks. This is likely because MNIST is an easier task than JSC and HGCal. When ensembling eight JSC_S models, using unique quantizers improves accuracy by 2.1%, while using shared I/O quantizers only improves accuracy by 0.8%, reducing the benefit of ensembling. Similarly, for HGCal, sharing I/O quantizers reduces HGCal_S's EMD by 35% to 1.792, which is worse than its single model EMD of 1.589, resulting in no benefit from ensembling. As a result, we need a way to maintain ensemble performance when mapping ensembles to FPGAs without significantly increasing resource overhead.

5 AmigoLUT

To solve the challenges presented in the previous section, we introduce AmigoLUT, a method for creating ensembles of LUT-based NNs that map to FPGAs efficiently. The idea behind AmigoLUT is to balance the performance gains from unique I/O quantizers with the resource efficiency of shared I/O quantizers. AmigoLUT quantizes the input data to low precision (i.e., 6 bits) with a single shared quantizer before sending the data to each ensemble member. Then, each ensemble member's input neuron can be quantized uniquely with a single LUT6 per bit to achieve the lower precision it expects. At the output, AmigoLUT implements an extra NN layer that quantizes each ensemble member's outputs to the same precision so that they can be summed together efficiently on an FPGA. Fig. 7 (right) describes our design.

5.1 Training results

Fig. 8 shows the accuracy improvements that AmigoLUT achieves with ensembling. We consider three tasks: MNIST, JSC, and HGCal; each graph corresponds to one task. We only consider LogicNets here and use averaging as it gives the best results. As before, we see that the accuracy generally increases as we add additional ensemble members until it typically plateaus. Note that a higher accuracy percentage is better in the MNIST and JSC graphs, but in HGCal, a lower EMD is better.

The graphs show three different methods of quantization: unique I/O quantizers, shared I/O quantizers, and the hybrid quantization scheme that we use in AmigoLUT (see Fig. 7). The general trend is that the unique I/O quantizer gives the best results, which is unsurprising given that it is unconstrained; each input and output can be uniquely quantized, providing the most flexibility.

In general, AmigoLUT's quantization scheme sits between the unique I/O quantization scheme and the shared I/O quantization scheme. In MNIST, AmigoLUT and shared quantization are very similar. Both provided less than a 1% accuracy drop from the unique I/O quantization. The JSC results show two models: small (S) and large (L). The larger models perform better than the small models across all ensemble sizes. Ensembling the small model achieves the accuracy of the baseline large model (ensemble size 1). AmigoLUT ensembling has very similar accuracy compared to unique I/O quantization. The shared I/O quantization method performs poorly for JSC. In general, AmigoLUT reduces ensemble performance by <1% compared with unique I/O quantizers for MNIST and JSC. HGCal results show a similar trend. The EMD is lower (better) for the unique and AmigoLUT quantization schemes, with the shared scheme being noticeably worse. The AmigoLUT quantization has the best results of all three quantization schemes for some ensemble sizes. For example, when ensembling two models together, AmigoLUT improves EMD by 9% compared with unique I/O quantizers. In summary, AmigoLUT significantly improves ensemble performance compared to the naive approach of shared I/O quantizers.

Similar to Sec. 3.2, we visualize ensemble member prediction diversity by showing ensemble member voting across individual samples (using $n = 50$). We compare the diversity of averaging models with unique I/O quantizers to models ensembled with AmigoLUT on the MNIST and JSC tasks in Fig. 10 and Fig. 11, respectively. We observe similar diversity patterns, suggesting that the training with AmigoLUT preserves ensemble diversity in addition to performance.

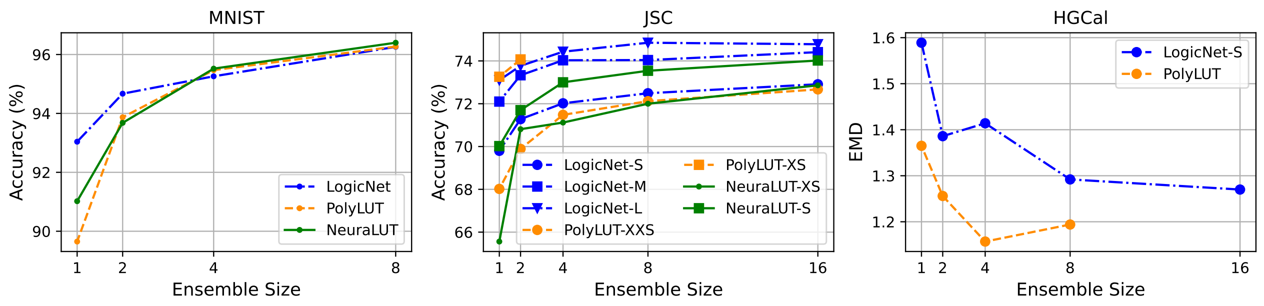

Figure 9: AmigoLUT can ensemble different LUT-based NNs (LogicNet, PolyLUT, and NeuraLUT) with different sizes (XXS, XS, S, M, L) to increase their accuracy. The accuracy improvements depend on the difficulty of learning the underlying task (MNIST, JSC, HGCal). In general, ensembling has greater relative improvement with smaller models compared to larger models.

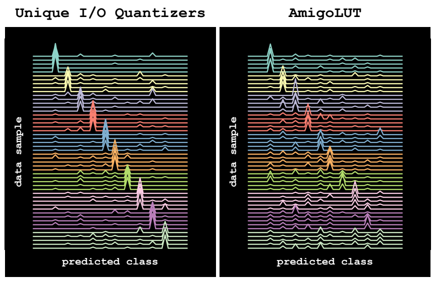

Figure 10: Diversity plots for MNIST with unique I/O quantizers versus AmigoLUT.

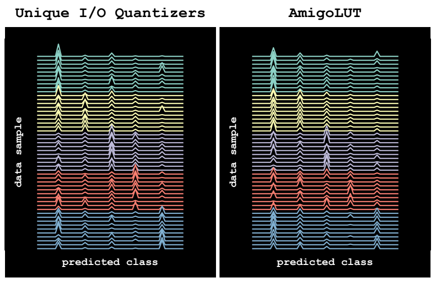

Figure 11: Diversity plots for JSC with unique I/O quantizers versus AmigoLUT.

5.2 Hardware results

We study the effect of ensembling different types of models in terms of their neuron structure (LogicNet, PolyLUT, NeuraLUT) and size (number of neurons/LUTs). Fig. 9 presents accuracy results across three tasks (MNIST, JSC, and HGCal) with different NN model architectures.

Tab. 3 shows the base model architecture parameters considered throughout the experiments. These models were selected by performing a model architecture search to find high-quality models that provided tradeoffs between model complexity/size and accuracy. The base models are replicated for each ensemble member; the structure of the models of an ensemble are identical though their weights will differ.

Dataset	Model	Neurons per layer	Input bit width	Input fan-in	Layer bit width	Layer fan-in
JSC	AmigoLUT-LogicNet-S	64, 32, 32, 32, 5	2	3	2, 2, 2, 2, 2	3, 3, 3, 3, 3
	AmigoLUT-NeuraLUT-XS (width = 16)	64, 5	2	3	2, 2	3, 3
	AmigoLUT-NeuraLUT-S (width = 16)	64, 5	4	3	2, 2	3, 3
	AmigoLUT-PolyLUT-XXS (degree = 3)	64, 5	2	3	2, 2	3, 3
	AmigoLUT-PolyLUT-XS (degree = 3)	64, 5	2	3	2, 2	3, 3
MNIST	AmigoLUT-LogicNet-XS	1024, 1024, 128, 10	1	8	1, 1, 1, 4	8, 8, 8, 8
	AmigoLUT-NeuraLUT (width = 16)	600, 300, 300, 10	2	4	2, 2, 2, 2	4, 4, 4, 4
	AmigoLUT-PolyLUT (degree = 3)	600, 300, 300, 10	2	4	2, 2, 2, 2	4, 4, 4, 4
HGCal	AmigoLUT-LogicNet-S	128, 256, 128, 128, 16	4	2	2, 2, 2, 6, 6	6, 5, 4, 2, 6
	AmigoLUT-PolyLUT (degree = 2)	512, 512, 64, 16	6	2	3, 3, 4, 5	4, 5, 3, 3

Table 3: Base model architecture used for AmigoLUT ensembling, (i.e., the parameters of each ensemble member). "Width" and "degree" are NeuraLUT and PolyLUT parameters, respectively.

The general trend is that ensembling increases the accuracy regardless of the underlying NN model. However, the rate of improvement from ensembling depends upon the task being learned by the NN and the base ensemble model size (S, M, L, etc.). Ensembling generally sees its greatest relative gains with smaller ensemble members. For example, MNIST sees a several percentage point increase when going from one ensemble member (baseline) to two ensembles in Fig. 9, which is true across all of the different types of models (LogicNet, PolyLUT, NeuraLUT). Increasing the ensemble size from two to four members increases the accuracy further but not as dramatically. The returns of increasing the ensemble size generally continue to diminish with more ensemble members.

The JSC task considers more than one model for LogicNet, PolyLUT, and NeuraLUT; denoted by the appended relative sizes (XXS, XS, S, M, L) from smallest to largest w.r.t. the number of LUTs. Consider LogicNet models with ensemble size 1. LogicNet-S has the lowest accuracy, LogicNet-M is approximately 2% more accurate, and LogicNet-L is the most accurate. As we ensemble each of these models, they all get more accurate at relatively the same rate, with the rate of change in accuracy being higher at lower ensemble members and relatively unchanged at ensemble sizes 8 and 16. The PolyLUT-XXS and PolyLUT-XS models follow a similar trend.

The HGCal task is likely the most challenging of the three tasks. First, as an autoencoder, the output result is much more continuous than the rather small, discrete number of output results for MNIST

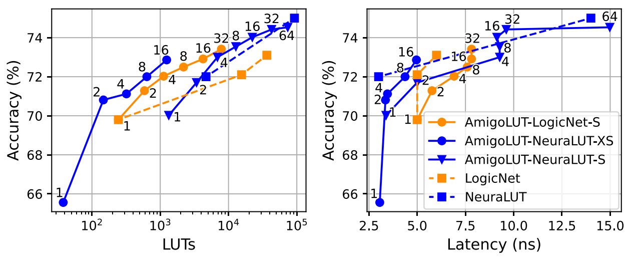

Figure 12: Jet Substructure Pareto Front. The results compare accuracy, LUTs, and latency. The solid lines are the AmigoLUT models with labels indicating the number of ensemble members. The dotted lines are the standalone versions of the LUT-based NNs we ensemble. AmigoLUT with NeuraLUT as its ensembling model gives small, accurate, and low latency results. The latency only increases slightly as ensemble size increases until we hit large ensembles (i.e., 64 models).

and JSC, 10 and 5 classes, respectively. Regardless, we see that ensembling HGCal encoders yields better EMD. The EMD reduces as the ensemble members increase, meaning the error is lower; an EMD = 0 is the most accurate result.

Our next series of experiments focuses on comparing the best models across LUT-based NNs and model sizes. We compare our AmigoLUT results for ensembling against the standalone models that we ensemble, showing how AmigoLUT improves the scalability of LogicNets and NeuraLUT. We also compare with the published related work. We synthesize and place-and-route our ensembles with Vivado 2020.2, targeting part `xcvu9p-flgb2104-2-i`, a Xilinx Virtex UltraScale+ FPGA. We compile the projects using Vivado's `Flow_PerfOptimized_high` setting and execute synthesis in `Out-of-Context` mode.

We note that prior work assumes that the inputs are quantized and this quantization is done offline, (i.e., there is no quantization hardware on the inputs). This is a reasonable assumption, but the reality is that this quantization must be performed somewhere. To make a fair comparison, we assume that the AmigoLUT inputs are similarly quantized to 6 bits offline; however, we do add the hardware required to quantize those 6 bits into say unique 2 bit inputs for each ensemble member (see Fig. 7).

Tab. 4 shows a selection of ensembling models developed in this work and results presented in previous works across the three tasks. The table provides precise numbers for all the relevant metrics (accuracy, LUTs, FFs, latency, FMax, and area delay product) typically discussed in related work. Note that the high FMax numbers are a direct result from synthesizing in `Out-of-Context` mode. Most FPGA devices do not provide a 1 GHz clock, so in a real system we would be limited by the global clock network, which is around ~800 MHz for the VU9P device we target. In Tab. 4, we see that DWN and PolyLUT-Add provide superior performance for some benchmarks compared with AmigoLUT; however, we demonstrate AmigoLUT's potential for improving these works' state-of-the-art performance in the following figures and discussion. In particular, we show how AmigoLUT significantly reduces LUT utilization for the models we ensemble, specifically LogicNet and NeuraLUT.

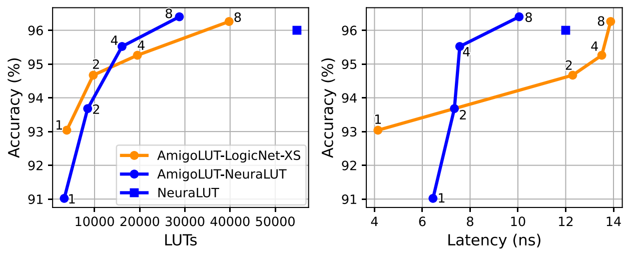

Figure 13: MNIST Pareto Front comparing accuracy, LUTs, and latency. Solid lines are different ensemble sizes (noted by the labels) generated from the same base model. The remaining point is a standalone NeuraLUT.

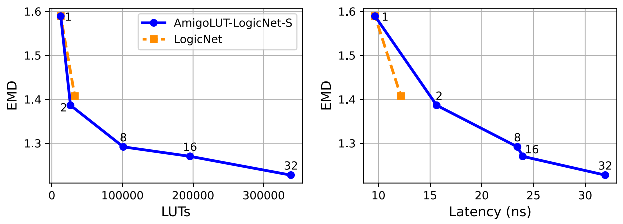

Figure 14: HGCal Pareto Front comparing accuracy, LUTs, and latency. The labels on the solid lines indicate the number of ensemble members. We only achieved better EMD through ensembling. We did not find a larger LogicNet model that provided better EMD with a reasonable number of LUTs.

Fig. 12 shows different models' accuracy versus latency and LUTs. The models include our AmigoLUT ensembling results (solid lines) and their standalone versions (dotted lines). The labels indicate the number of ensemble members. The different models in the standalone LUT-based NNs architectures are generated by changing the model architecture, (e.g., the number of layers, the number of neurons per layer, the layer bit width, and the layer fan-in). Our results show that selecting the best model architecture parameters is challenging; it requires extensive tuning and trial and error.

Fig. 12 shows that adding more ensemble members increases the accuracy, resulting in a linear increase in LUTs. We also see over an order of magnitude drop in LUT utilization when comparing a standalone LogicNet versus AmigoLUT-LogicNet at 72% accuracy. Except for large ensembles (i.e., ensemble size 64), the latency stays relatively the same as ensemble size increases. Note that the latency for larger non-ensembled models increases substantially; thus, there is a clear benefit for increasing model size/accuracy using AmigoLUT compared to the more traditional NN architecture search strategy done in previous work (modifying the layers, neurons per layer, etc.). This is seen in the drastic increase in latency for the higher accuracy dotted line results.

The AmigoLUT architectures scale well regardless of the baseline architecture model characteristics, though tuning the baseline architecture does affect the overall quality. A better baseline model will generally ensemble better than a lower quality baseline model.

Fig. 13 shows results for MNIST, which are relatively similar to JSC. Ensembling has a linear relationship between ensemble

Dataset	Model	Accuracy /EMD	LUT	FF	DSP	BRAM	Latency (ns)	FMax (MHz)	Area × Delay (LUT × ns)
MNIST	**AmigoLUT-LogicNet-XS** (2 models)	94.7	9 711	9 047	0	0	12.3	569	119 445
	AmigoLUT-NeuraLUT (4 models)	95.5	16 081	13 292	0	0	**7.6**	**925**	122 216
	PolyLUT [3]	96	70 673	4 681	0	0	16	378	1 130 768
	NeuraLUT [4]	96	54 798	3 757	0	0	12	431	657 576
	PolyLUT-Add [16]	96	14 810	2 609	0	0	10	625	148 100
	DWN [5]	**97.8**	**2 092**	**1 757**	0	0	9.2	873	**19 246**
JSC	**AmigoLUT-NeuraLUT-XS** (4 models)	71.1	320	482	0	0	3.5	**1 445**	1 120
	AmigoLUT-NeuraLUT-XS (16 models)	72.9	1 243	1 240	0	0	5.0	1 008	6 215
	AmigoLUT-NeuraLUT-S (32 models)	74.4	42 742	4 717	0	0	9.6	520	410 323
	LogicNet-L	73.1	36 415	2 790	0	0	6	390	218 490
	PolyLUT	72	12 436	773	0	0	5	646	62 180
	PolyLUT	75	236 541	2 775	0	0	21	235	4 967 361
	NeuraLUT	72	4 684	341	0	0	**3**	727	14 052
	NeuraLUT	75	92 357	4 885	0	0	14	368	1 292 998
	PolyLUT-Add	75	36 484	1 209	0	0	16	315	583 744
	PolyLUT-Add	72	895	1 649	0	0	4	750	3 580
	DWN	73.7	**134**	**106**	0	0	3.7	1 361	**496**
	DWN	**76.3**	6 302	4 128	0	0	14.4	695	90 749
HGCal	**AmigoLUT-LogicNet-S** (2 models)	1.386	**26 400**	4 049	0	0	15.6	**512**	411 840
	AmigoLUT-LogicNet-S (16 models)	**1.270**	195 724	23 515	0	0	24.0	334	4 697 376
	LogicNet-L	1.407	32 529	**2 340**	0	0	**12.2**	323	**396 854**

Table 4: Comparing model resource utilization and performance metrics across the three datasets/tasks with prior work.

members and LUTs. The latency increases as more ensemble members are added, but generally, ensembling provides a good tradeoff between accuracy, LUTs, and latency.

The HGCal dataset was not considered in previous LUT-based NN publications. We feel this is a good benchmark for LUT-based NNs because it has become a canonical task in the FastML community. It has very high constraints on latency and throughput and thus is a task well-matched for LUT-based NNs. Furthermore, it has a more continuous output space as it is trying to compress data.

We generated the LogicNets results by performing a NN architecture search that modified the number of layers, the number of neurons per layer, the input size, etc. We performed a design space exploration to find architectures that would train well and provide good tradeoffs between accuracy and size. We could not find a larger LogicNets architecture that provided better accuracy with a reasonable number of LUTs. We considered nearly 500 models in this search exploration, which highlights the challenges of scaling up LUT-based NN architecture. As our results show in Fig. 14, we used ensembling to generate larger models with better EMD.

6 Conclusion

We address the challenges of implementing large, high-performance NNs for tasks that require high throughput and low latency. AmigoLUT creates ensembles of smaller LUT-based NNs that map efficiently to FPGAs. AmigoLUT provides a straightforward methodology to increase model accuracy while maintaining a linear scaling in LUTs, minimally increasing latency, and creating a high-frequency design. We evaluate three ensemble methods (averaging, bagging, and AdaBoost) and show that averaging outperforms the other methods. We analyze the ensemble's performance using novel diversity plots and disagreement metrics, which help explain why averaging is effective, and provide tools to evaluate the general quality of ensembling LUT-based NNs and NNs in general. We describe architecture tradeoffs for efficiently implementing the ensembles, developing a quantization architecture that allows for effective ensemble scaling to increase accuracy while minimizing resource usage. We perform a comprehensive study on three tasks, showing that AmigoLUT can effectively ensemble existing LUT-based NN models, demonstrating the potential that ensembling has to scale up state-of-the-art and future LUT-based NNs.

Acknowledgments

The authors thank the anonymous referees and our shepherd Kia Bazargan for their valuable comments and helpful suggestions. Thank you to Abarajithan Gnaneswaran, Zhenghua Ma, and Andres Meza for helpful consultation. Thanks to Ryan Theisen for helpful discussions about how to visualize ensemble diversity. CG and JC were supported by the U.S. Department of Energy (DOE), Office of Science, Advanced Scientific Computing Research (ASCR) program under Contract Number DE-AC02-05CH11231 to Lawrence Berkeley National Laboratory and Award Number DE-SC0023328 to Arizona State University ("Visualizing High-dimensional Functions in Scientific Machine Learning"). JD is supported by DOE ASCR under the "Real-time Data Reduction Codesign at the Extreme Edge for Science" Project (DE-FOA-0002501) and the National Science Foundation (NSF) under Cooperative Agreement OAC-2117997 (A3D3). This material is based upon work supported by the NSF Graduate Research Fellowship Program under Grant No. DGE-2038238. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[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, 60–68.

[4] 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.

[5] Alan Tendler Leibel Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy Kurian John, Priscila Machado Vieira Lima, and Felipe MG França. 2024. Differentiable Weightless Neural Networks. In *Forty-first International Conference on Machine Learning*.

[6] Leo Breiman. 1996. Bagging predictors. *Machine learning* 24 (1996), 123–140.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[8] S Chatrchyan, G Hmayakyan, V Khachatryan, AM Sirunyan, W Adam, T Bauer, T Bergauer, H Bergauer, M Dragicevic, J Eroe, et al. 2008. The CMS experiment at the CERN LHC. *Journal of instrumentation* 3 (2008).

[9] CMS collaboration et al. 2017. The phase-2 upgrade of the CMS endcap calorimeter. *CMS Technical Design Report CERN-LHCC-2017-023. CMS-TDR-019, CERN* (2017).

[10] Giuseppe Di Guglielmo, Farah Fahim, Christian Herwig, Manuel Blanco Valentin, Javier Duarte, Cristian Gingu, Philip Harris, James Hirschauer, Martin Kwok, Vladimir Loncar, et al. 2021. A reconfigurable neural network ASIC for detector front-end data compression at the HL-LHC. *IEEE Transactions on Nuclear Science* 68, 8 (2021), 2179–2186.

[11] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. 2020. A survey on ensemble learning. *Frontiers of Computer Science* 14 (2020), 241–258.

[12] Javier Duarte, Nhan Tran, Ben Hawks, Christian Herwig, Jules Muhizi, Shvetank Prakash, and Vijay Janapa Reddi. 2022. FastML Science Benchmarks: Accelerating Real-Time Scientific Edge Machine Learning. *arXiv preprint arXiv:2207.07958* (2022).

[13] Leopoldo Lusquino Filho, Felipe MG França, and Priscila MV Lima. 2023. WiSARD-based Ensemble Learning. (2023).

[14] Yoav Freund, Robert E Schapire, et al. 1996. Experiments with a new boosting algorithm. In *icml*, Vol. 96. Citeseer, 148–156.

[15] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. 2009. Multi-class adaboost. *Statistics and its Interface* 2, 3 (2009), 349–360.

[16] Binglei Lou, Richard Rademacher, David Boland, and Philip HW Leong. 2024. PolyLUT-Add: FPGA-based LUT Inference with Wide Inputs. *arXiv preprint arXiv:2406.04910* (2024).

[17] Haiquan Lu, Xiaotian Liu, Yefan Zhou, Qunli Li, Kurt Keutzer, Michael W. Mahoney, Yujun Yan, Huanrui Yang, and Yaoqing Yang. 2024. Sharpness-diversity tradeoff: improving flat ensembles with SharpBalance. arXiv:2407.12996 [stat.ML] https://arxiv.org/abs/2407.12996

[18] Yifang Ma, Zhenyu Wang, Hong Yang, and Lin Yang. 2020. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA Journal of Automatica Sinica* 7, 2 (2020), 315–329.

[19] Igor DS Miranda, Aman Arora, Zachary Susskind, Luis AQ Villon, Rafael F Katopodis, Diego LC Dutra, Leandro S De Araújo, Priscila MV Lima, Felipe MG França, Lizy K John, et al. 2022. Logicwisard: Memoryless synthesis of weightless neural networks. In *2022 IEEE 33rd International conference on application-specific systems, architectures and processors (ASAP)*. IEEE, 19–26.

[20] Mahdi Nazemi, Arash Fayyazi, Amirhossein Esmaili, Atharva Khare, Soheil Nazar Shahsavani, and Massoud Pedram. 2021. NullaNet Tiny: Ultra-low-latency DNN inference through fixed-function combinational logic. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 266–267.

[21] Mahdi Nazemi, Ghasem Pasandi, and Massoud Pedram. 2018. Nullanet: Training deep neural networks for reduced-memory-access inference. *arXiv preprint arXiv:1807.08716* (2018).

[22] Alessandro Pappalardo. 2023. *Xilinx/brevitas*. https://doi.org/10.5281/zenodo.3333552

[23] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The Earth Mover's Distance as a Metric for Image Retrieval. *Int. J. Comput. Vis.* 40 (2000), 99. https://doi.org/10.1023/A:1026543900054

[24] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 8, 4 (2018), e1249.

[25] Zachary Susskind, Aman Arora, Igor DS Miranda, Alan TL Bacellar, Luis AQ Villon, Rafael F Katopodis, Leandro S de Araújo, Diego LC Dutra, Priscila MV Lima, Felipe MG França, et al. 2023. ULEEN: A Novel Architecture for Ultra-low-energy Edge Neural Networks. *ACM Transactions on Architecture and Code Optimization* 20, 4 (2023), 1–24.

[26] Zachary Susskind, Aman Arora, Igor DS Miranda, Luis AQ Villon, Rafael F Katopodis, Leandro S De Araújo, Diego LC Dutra, Priscila MV Lima, Felipe MG França, Mauricio Breternitz Jr, et al. 2022. Weightless neural networks for efficient edge inference. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 279–290.

[27] Zachary Susskind, Alan TL Bacellar, Aman Arora, Luis AQ Villon, Renan Mendanha, Leandro Santiago de Araújo, Diego Leonel Cadette Dutra, Priscila MV Lima, Felipe MG França, Igor DS Miranda, et al. 2022. Pruning weightless neural networks. *ESANN 2022 proceedings* (2022).

[28] Ryan Theisen, Hyunsuk Kim, Yaoqing Yang, Liam Hodgkinson, and Michael W Mahoney. 2024. When are ensembles really effective? *Advances in Neural Information Processing Systems* 36 (2024).

[29] Yaman Umuroglu, Yash Akhauri, Nicholas James 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, 291–297.

[30] Erwei Wang, James J Davis, Peter YK 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, 26–34.

[31] Olivia Weng, Alexander Redding, Nhan Tran, Javier Mauricio Duarte, and Ryan Kastner. 2024. Architectural implications of neural network inference for high data-rate, low-latency scientific applications. *arXiv preprint arXiv:2403.08980* (2024).

[32] Tim Whitaker and Darrell Whitley. 2022. Prune and tune ensembles: low-cost ensemble learning with sparse independent subnetworks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8638–8646.

[33] Yongquan Yang, Haijun Lv, and Ning Chen. 2023. A survey on ensemble learning under the era of deep learning. *Artificial Intelligence Review* 56, 6 (2023), 5545–5589.

[34] Corinna Cortes Yann LeCun and Chris Burges. [n. d.]. MNIST handwritten digit database. https://yann.lecun.com/exdb/mnist/

[35] Zhi-Hua Zhou. 2012. *Ensemble methods: foundations and algorithms*. CRC press.

[36] Shilin Zhu, Xin Dong, and Hao Su. 2019. Binary ensemble neural network: More bits per network or more networks per bit?. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 4923–4932.