NeuSym-HLS: Learning-Driven Symbolic Distillation in High-Level Synthesis of Hardware Accelerators

Chung-Mou Pan, Salma Elmalaki, Yasser Shoukry, Sitao Huang
Department of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, CA 92697
{chungmop, selmalak, yshoukry, sitaoh}@uci.edu

Abstract

Domain-specific hardware accelerators for deep neural network (DNN) inference have been widely adopted. Traditional DNN compression techniques such as pruning and quantization help but can fall short when aggressive hardware efficiency is required. We present *NeuSym-HLS*, a partial symbolic distillation and high-level hardware synthesis flow to compress and accelerate DNN inference for edge computing. NeuSym-HLS replaces a portion of the layers of a trained DNN model with compact analytic expressions obtained via symbolic regression, and generates efficient hardware accelerators. The resulting hardware accelerator of the hybrid DNN-symbolic model provides well balanced performance in algorithmic accuracy, hardware resource, and inference latency. Our evaluation on vision tasks showed that NeuSym-HLS reduces hardware resource usage, reduces latency, while maintaining model inference accuracy.

1 Introduction

Deep neural networks (DNNs) have been widely adopted in a wide range of application domains. Despite their significant accuracy performance, DNN models require intensive computation, and they often lack explainability, which posts risks for certain applications and systems like real-time systems and cyber-physical systems. To improve DNN's computational performance, researchers have proposed various solutions at both the model architecture level and the computing system level, including model compression techniques and domain-specific acceleration with application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs). Model compression and hardware acceleration techniques often need to be applied together with specialized hardware architecture and DNN retraining to gain computing efficiency and maintain model accuracy.

Compared to the computational challenges discussed above, the explainability of DNN-based machine learning models is an even harder problem and is more serious in certain domains that require rigorous robustness and verification standard. In recent years, researchers have proposed techniques aiming to create interpretable versions of DNN-based models. Symbolic regression (SR) has become one of the popular techniques and has been shown to be able to replace some DNN models without much accuracy degradation.

To address the computation and explainability challenges in DNN-based models, we propose ***NeuSym-HLS***, a DNN model compression and hardware accelerator generation framework that performs co-optimization of DNN model compression and high-level synthesis (HLS) based hardware architecture design space exploration. Our proposed NeuSym-HLS flow features three key components, including (i) *partial symbolic distillation* and quantization pass that compresses given DNN models by replacing computational heavy layers with symbolic expressions; (ii) hardware-aware design space exploration of *symbolic operation selection*; and (iii) *symbolic-architecture co-design* engine which finetune the

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

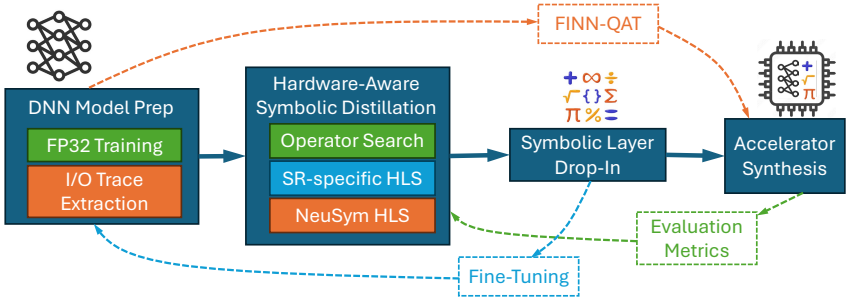

Figure 1: NeuSym-HLS Overview. FP32 training produces an activation trace that feeds hardware-aware symbolic regression (SR). The resulting analytic layer is *dropped-in* and the hybrid design synthesized. Optional branches (blue dashed boxes/paths) support QAT and post-SR fine-tuning.

symbolic distilled model and generates efficient hardware accelerator design via high-level synthesis (HLS). NeuSym-HLS significantly reduces hardware resource usage and inference latency of DNN models accelerators, while delivering near-original inference accuracy.

2 Background and Related Works

Deep learning (DL) inference on hardware accelerators hinges on two orthogonal themes: *(i) hardware-aware design flows* that translate high-level specifications (e.g., DNN models and C/C++ implementations) into efficient RTL designs, and *(ii) model-compression techniques* that shrink model size without sacrificing accuracy. We review the state-of-the-art techniques in both areas before introducing our NeuSym-HLS flow.

High-Level Synthesis of Hardware Accelerators. Modern tools such as AMD Vitis HLS let developers express kernels in C/C++ and refine them with HLS pragmas (e. g. `dataflow`, `pipeline`) to guide HLS compiler to optimize hardware designs and meet tight latency and resource utilization targets on FPGAs or ASICs [1]. Dozens of works show that careful loop unrolling, tiling, and memory partitioning can deliver real-time CNN inference even on low-power resource-constrained devices [2, 3].

Symbolic Regression and Hardware Acceleration. Tsoi et al. [4] introduce FPGA-resident symbolic regression (SR) accelerators that entirely replace dataset-level inference with compact analytic expressions, achieving sub-μs latency on physics workloads. Later, SymbolNet [5] bridges this idea with DNNs by by combining neural symbolic regression with adaptive dynamic pruning to achieve efficient model compression, enabling low-latency inference even for high-dimensional datasets on FPGA hardware.

Challenges in Scaling Symbolic Regression. Differentiable Genetic Programming (DGP) [6] tackles the high-dimensional SR problem by relaxing tree structures into continuous representations. Even so, the authors note that discovering compact formulas for thousands of features remains expensive, confirming that "all-layer" SR on deep networks is impractical.

3 NeuSym-HLS Framework

3.1 NeuSym-HLS Framework Overview

Figure 1 illustrates the end-to-end **NeuSym-HLS** flow, starting from a floating-point PyTorch model and ending with post-synthesis metrics. Solid arrows represent the default path; dashed arrows represent optional branches such as quantization-aware training (QAT) and end-to-end fine-tuning of the neural-symbolic hybrid network.

Hardware-Aware Symbolic Regression (SR). For each of the N output channels fit an analytic function

$$\ell_j \;=\; f_j(\mathbf{h}) \quad \text{with} \quad f_j \in \mathrm{SR}\big(\mathcal{O},\, C_{\max}\big)\,,$$

where $\mathcal{O}$ is the operator set (e.g., $\{+, -, *, \sin, \exp\}$) and $C_{\max}$ a complexity budget.

Table 1: Example snapshot of symbolic regression Hall of Fame produced by PySR.

Complexity	Loss	Equation
1	1926.54	0.7614
3	0.4929	$x_1 \cdot x_1$
5	0.4835	$(x_1 \cdot x_1) \cdot 0.9978$
6	1.10×10^{-11}	$\sin(x_0) + x_1^2$
13	9.88×10^{-12}	$x_1^2 + \sin(x_0 + \cos((x_1 - x_1) - \frac{\pi}{2}))$

Table 2: Evaluating Generated Accelerators on the MNIST Task

Model ID	Acc (%)	Latency	LUT (%)	FF (%)	DSP (%)
BL–MLP	96.4	106k	21k (41%)	44k (41%)	976 (443%)
SR–MLP	94.4	110k	9,499 (17%)	6,833 (6%)	96 (43%)
Q–MLP	96.3	158	1,347 (3%)	1,955 (2%)	0 (0%)
Q–SR–MLP	95.8	**75**	**982 (2%)**	**1,645 (2%)**	0 (0%)
BL–LeNet	98.4	130k	602k (1,132%)	435k (408%)	2,769 (1,258%)
SR–LeNet	**99.0**	130k	395k (743%)	256k (241%)	2,194 (997%)
Q–LeNet	97.7	8,810	7,173 (13%)	10k (10%)	0 (0%)
Q–SR–LeNet	96.2	35k	3,146 (6%)	3,576 (3%)	0 (0%)
SR Paper[4]	85.3	13[1]	7,592 (14%)	6,424 (6%)	160 (73%)

The equation with the lowest validation loss under the complexity threshold (e.g., Entry #13) is selected for deployment. This expression is translated into synthesizable HLS C code for hardware integration, as shown in Listing 1.

```
float fc3_symbolic(float x0, float x1) {
  // (x1 * x1) + sin(x0 + cos((x1 - x1) - 1.5707971f))
  #pragma HLS INLINE
  float term1 = x1 * x1;
  float term2 = sinf(x0 + cosf((x1 - x1) - 1.5707971f));
  return term1 + term2;
}
```

Listing 1: Auto-generated HLS C implementation of Entry #13

3.2 Hardware-Aware Symbolic Regression in NeuSym-HLS

NeuSym-HLS builds on **PySR** [7], an open-source symbolic regression library for discovering human-interpretable equations through a high-performance Genetic Programming approach with its unique evolve–simplify–optimize loop algorithm, but enhances its fitness metric to account for hardware-aware trade-offs, including *inference latency and FPGA resource utilization*. This section details the two key design dimensions that must be fixed *before* large-scale symbolic regression: **Search-space definition**: selecting unary/binary operator sets that offer the best balance between interpretability and synthesis overhead (Section A). **Distillation placement** — deciding which DNN layers to be replaced by symbolic expressions to optimize synthesis metrics while preserving accuracy (Section B).

4 Evaluation Results and Analysis

Our evaluation setup is explained in Appendix C.

TABLE 2 and TABLE 3 summarized all the evaluation results on MNIST task and SVHN task, respectively. According to the evaluation results shown in these two tables, NeuSym-HLS generated compressed models and accelerators (names start with “SR-” or “Q-”) achieve more performance and resource efficient hardware acceleration compared to baseline implementations, while maintaining high accuracy. When compared against state-of-the-art works, NeuSym-HLS achieves much higher accuracy while using much less resource, and having similar levels of latency.

Table 3: Evaluating Generated Accelerators on the SVHN Task

Model ID	Acc (%)	Latency	LUT (%)	FF (%)	DSP (%)
BL–MLP	94.1	12.6m	4,665 (8%)	3,182 (2%)	15 (6%)
SR-1L-SCE	97.7	18.4m	5,598 (10%)	3,316 (3%)	16 (7%)
SR-1L-SRL	97.6	18.4m	2,969 (5%)	1,693 (1%)	7 (3%)
SR-1L-POL	**98.2**	18.4m	2,894 (5%)	1,630 (1%)	7 (3%)
SR-2L-SCE	90.3	12.6m	7,803 (14%)	4,662 (4%)	23 (10%)
SR-2L-SRL	91.8	12.6m	4,089 (7%)	2,555 (2%)	7 (3%)
SR-2L-POL	90.7	12.6m	3,627 (6%)	2353 (2%)	5 (2%)
Q–MLP	93.9	8,737	48,357 (90%)	32,367 (30%)	0 (0%)
Q–SR-1L-SCE	91.7	**65,550**	45,682 (86%)	**14,287 (13%)**	0 (0%)
Q–SR-1L-SRL	93.9	65,550	46,289 (87%)	14,368 (13%)	0 (0%)
Q–SR-1L-POL	90.9	65,550	**45,661 (86%)**	14,291 (13%)	0 (0%)
SymbolNet [5]	94	520[1]	27,407 (52%)	16,286 (15%)	77 (35%)

Operator–Set Search: Identifying Key Unary Functions. We explore the design space of operator selection for symbolic regression (SR) to identify the most efficient combinations of operators. As discussed in Section A, we defined non-trig and trig-enriched sets of operators. Fig. 3 and Fig. 4 present the losses (the lower the better) of varies combinations of SR operators. According to the figures, trig-enriched unary operator sets tend to perform better than the non-trig operator sets. From Fig. 4, We can also observe that better performing non-trig operator sets consistently include "ReLu" operator.

Impact of Operator Selection on Hardware Accelerators. We selected various representative combinations of SR operators, and use them to approximate the *entire* DNN model, and evaluate the performance of generated hardware accelerators. Fig. 2 shows the evaluation results. When running symbolic regression on the whole DNN model, the hardware usage decreases significantly, however, the accuracy also drops significantly to 72%. This supports our motivation of doing **partial** symbolic distillation instead of full model symbolic approximation. Besides, different combinations of operators lead to different results of accuracy, hardware resource usage, and latency, therefore, the operator selection search in NeuSym-HLS is essential to generate efficient accelerator.

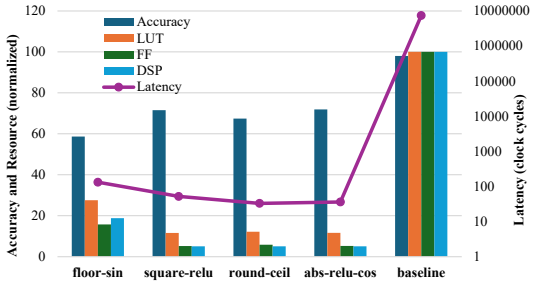

Figure 2: Comparing Symbolic Regression Operator Combinations on the Area and the Performance of Hardware Accelerators for LeNet (MNIST Task).

5 Conclusion

NeuSym-HLS bridges the gap between the efficiency of aggressive quantization and the interpretability of full-network symbolic regression. By selectively distilling the final layer(s) of a neural network into compact symbolic expressions, leveraging quantization-aware training and other standard compression techniques, our results demonstrate that symbolic distillation is especially advantageous for low-dimensional and high-latency output layers. Meanwhile, quantization remains an effective tool for compressing the remaining network, underscoring the complementary strengths of both approaches for resource-efficient and interpretable DNN deployment.

[1]Reported latency from [4] and [5] reflects only HLS kernel latency using hls4ml, and does not include end-to-end design latency; actual deployment latency may be higher.

References

[1] Xilinx Inc. Vitis high-level synthesis user guide, 2022. https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_2/ug1399-vitis-hls.pdf.

[2] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170, 2015.

[3] Stylianos I Venieris and Christos-Savvas Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47, 2016.

[4] Ho Fung Tsoi, Adrian Alan Pol, Vladimir Loncar, Ekaterina Govorkova, Miles Cranmer, Sridhara Dasu, Peter Elmer, Philip Harris, Isobel Ojalvo, and Maurizio Pierini. Symbolic regression on fpgas for fast machine learning inference. In *EPJ Web of Conferences*, volume 295, page 09036, 2024.

[5] Ho Fung Tsoi, Vladimir Loncar, Sridhara Dasu, and Philip Harris. SymbolNet: Neural Symbolic Regression with Adaptive Dynamic Pruning for Compression. *Machine Learning: Science and Technology*, 6(1):015021, 2025.

[6] Peng Zeng, Xiaotian Song, Andrew Lensen, Yuwei Ou, Yanan Sun, Mengjie Zhang, and Jiancheng Lv. Differentiable genetic programming for high-dimensional symbolic regression, 2023.

[7] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl. *arXiv preprint arXiv:2305.01582*, 2023.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[9] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 65–74. ACM, 2017.

[10] Marco Virgolin and Solon P. Pissis. Symbolic regression is np-hard. *arXiv preprint arXiv:2207.01018*, 2022.

Appendices

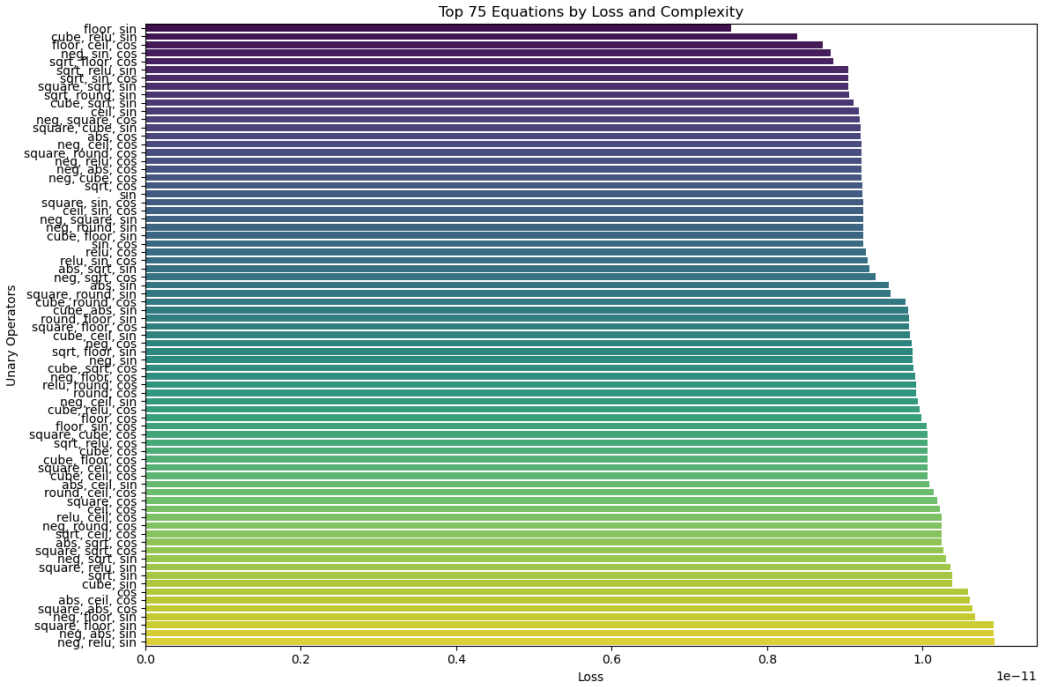

Figure 3: Top 75 trig-enriched operator combinations, sorted by loss.

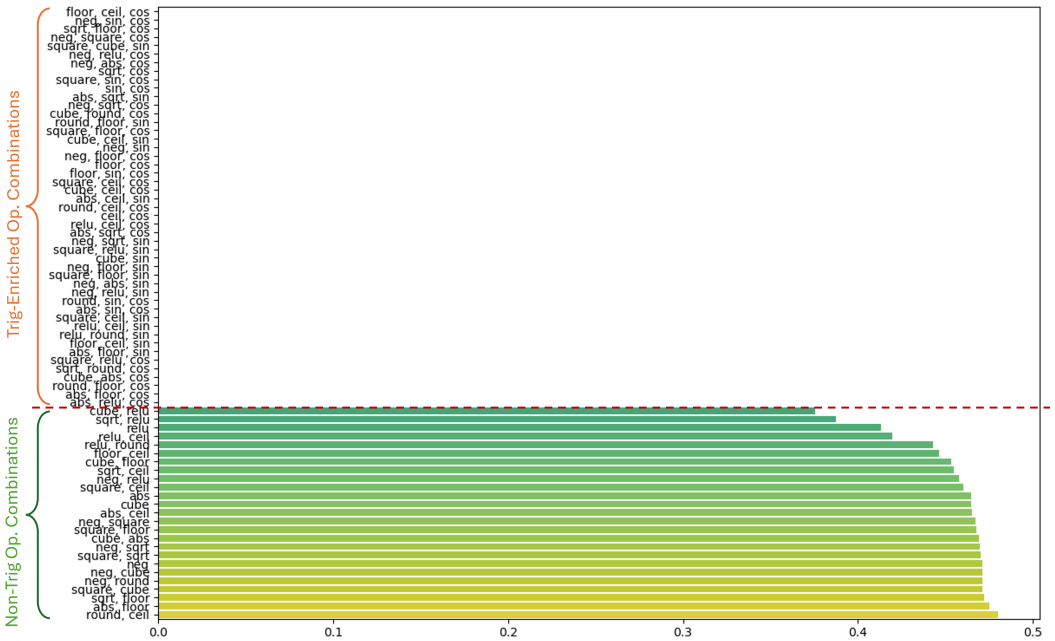

Figure 4: Trig-enriched (top) vs. Non-trig (bottom) operator combinations.

A Operator-Set Study

A preliminary sweep determined which unary functions should enter the search space. To systematically explore the symbolic design space before scaling to full DNN regression, we constructed a synthetic $2 \to 1$ regression task ($\mathbf{x} \in \mathbb{R}^2 \mapsto y$) whose ground-truth function blends polynomial and sinusoidal terms. This task serves as a proxy to mirror the kinds of nonlinearities seen in neural networks, allowing us to evaluate how different operator combinations in PySR navigate the trade-off between interpretability and accuracy. Two broad categories of unary operators were explored:

1. **Non-trig:** {`neg`, `square`, `cube`, `abs`, `sqrt`, `relu`, `round`, `floor`, `ceil`, `sign`}
2. **Trig-enriched:** all of the non-trig plus {`sin`, `cos`}

We swept through the space of all valid 3-element and 2-element subsets of candidate unary operators (i.e., $\binom{|\mathcal{F}|}{3}$ + $\binom{|\mathcal{F}|}{2}$), where $\mathcal{F}$ is the operator pool). For each subset, we trained a PySRRegressor using the fixed binary operator set {+, -, *}, with 40 iterations and 10 independent populations.

The **trig-enriched** subsets consistently achieved validation losses around $\mathcal{L}_2 \approx 10^{-11}$, while the best-performing **non-trig** configuration (`square` + `relu`) stalled at approximately $\mathcal{L}_2 \approx 10^{-1}$. Hence, even under identical tree-depth and complexity budgets, incorporating `sin`/`cos` closes a $\sim$5-order-of-magnitude accuracy gap. Within the non-trig family, sets that retained `relu` always outperformed those that did not, which aligns with prior findings on ReLU's effectiveness in deep neural network training [8].

Guided by these findings, we designed three sets of operators to be used in the NeuSym-HLS framework, to meet various model compression and accelerator optimization needs.

1. **SCE** (Sin–Cos–Exp): $\{+, -, *, \sin, \cos, \exp\}$ – highest expressiveness; used when accuracy is paramount.
2. **SRL** (Square–ReLu): $\{+, -, *, \text{square}, \text{relu}\}$ – no trig; captures piece-wise linearity with good hardware efficiency.
3. **POL** (Polynomial): $\{+, -, *, /\}$ – pure arithmetic baseline.

B Distillation Strategy

Symbolic regression is appealing for interpretability, yet GP-based methods scale poorly to high-dimensional data because their stochastic, discrete tree search lacks gradient guidance [6].

Our LeNet study confirms this: turning the whole network into one symbolic expression either failed to converge or produced unwieldy formulas, dropping accuracy from 98% to $\approx 70\%$. Full-network symbolic distillation is hence impractical for high-dimensional vision data.

To mitigate this, we adopt a targeted distillation strategy. Rather than distilling the entire network, we identify specific layers or submodules where symbolic expressions can be injected with minimal impact on end-to-end accuracy. Criteria for selection include **computational intensity** and **input/output dimensionality** of the layer (low-dimensional layers preferred). We therefore tested symbolic replacement of the final one or two dense layers, with the detailed accuracy–resource trade-off reported in TABLE 3.

C Evaluation Setup

DNN Models and Tasks. We evaluate our proposed NeuSym-HLS framework on three DNNs across two tasks, as shown in TABLE 4. These DNNs vary layer types, number of layers, and number of neurons in the layers.

Model Training and Quantization. All DNN models are first trained with standard cross-entropy loss for 10 epochs (Adam, learning rate $1\mathrm{e}{-3}$, batch size 64). In some evaluations, the model goes through quantization-aware training (QAT) using FINN [9] as well as fine-tuning, where each network is fine-tuned for 5 epochs using 2-bit activations and weights.

Table 4: Neural Network Benchmarks Used in This Study

Task & Model	DNN Architecture Details (# of layers and neurons)
MNIST MLP	Input: $784 \rightarrow 128 \rightarrow 64 \rightarrow 10$ output classes
MNIST LeNet	Two conv-pool blocks ($6 \times 5 \times 5$ and $16 \times 5 \times 5$ kernels, 2×2 max-pool), followed by $120 \times 5 \times 5$ conv and one FC-10 output layer
SVHN MLP	Input: $3072 \rightarrow 512 \rightarrow 128 \rightarrow 1$ (binary; digits "1" vs "7")

Symbolic Regression Engine. We leverage an open-source tool, PySR [7], to build the symbolic regression engine inside NeuSym-HLS. We use PySR v1.5.0[1] where search parameters `niterations` and `populations` are set to 40 and 10 respectively. NeuSym-HLS framework explores the non-linear operator selection space on top of a combination of *POLY*, *SCE*, and *SRL* operator sets. The default ℓ_2 regression loss on DNN layer logits is used.

Hardware Accelerator Prototyping Platform. We use an AMD XC7Z020 FPGA (Zynq-7000 series) as our hardware accelerator prototyping and evaluation platform. We use Vitis HLS (v2022.2) [1] to generate hardware designs, and target 100 MHz clock rate in the synthesis (post-place and route timing met in all designs).

Evaluation Metrics. We evaluate the effectiveness of NeuSym-HLS with three metrics, (i) inference **accuracy** (%) of compressed DNN model on the full validation set; (ii) inference **latency** (in clock cycles and in microseconds at 100 MHz) of generated hardware accelerator; (iii) **resource usage** of generated hardware accelerator on FPGA, including look-up table (LUT), flip-flop (FF), and DSP counts, expressed both in absolute numbers and as utilization percentage of the targeting FPGA device capacity.

We present end-to-end hardware synthesis results that answer three experimental questions:

1. **Operator relevance.** Which *unary* operators matter most for symbolic regression on image data?
2. **Pure-SR feasibility.** How well can a *fully* symbolic model approximate MNIST without any MACs?
3. **Hybrid efficiency.** When only the cost-critical layers of a DNN are replaced by symbolic expressions, how do accuracy, latency, and FPGA utilization compare with floating-point and quantized baselines?

To answer these, the section proceeds in five steps:

- *Operator-set search.* A 220-way grid sweep identifies `square+relu` as the most accurate non-trigonometric set and confirms the importance of keeping `relu`.
- *End-to-end SR on MNIST.* We train a pure symbolic regressor on the full dataset and report accuracy and resource cost against BL-MLP and Q-MLP baselines.
- *LeNet-5 hybrid on MNIST.* Replacing only the final fully connected layer with a symbolic expression (NeuSym-HLS) cuts LUT, FF, and DSP usage by up to $3.3\times$ at iso-accuracy and reduces latency by $1.5\times$.
- *Fully-connected MLP.* The same partial-distillation recipe is applied to a three-layer MLP, showing that gains generalize beyond convolutional architectures.
- *Same experiments on another SVHN Dataset.* Expanded SVHN experiments which include two-layer replacement and alternative operator sets.

Model ID Naming Scheme: `BL` = baseline (floating-point, no symbolic or quantization); `SR` = symbolic regression (last layer(s) replaced); `Q` = quantized (QAT, 2-bit weights, no symbolic); `Q-SR` = quantized + symbolic regression; `1L/2L` = number of layers replaced; `SCE`, `SRL`, `POL` = operator sets (see Sec. A).

[1] https://github.com/MilesCranmer/PySR

D NeuSym-HLS on MNIST: A Generality Study

To evaluate the effectiveness and generality of our NeuSym-HLS symbolic distillation approach, we apply NeuSym–HLS to a fully connected multilayer perceptron (MLP) and a LeNet model trained on MNIST task. Fig. 5 shows the evaluation results for the MLP model.

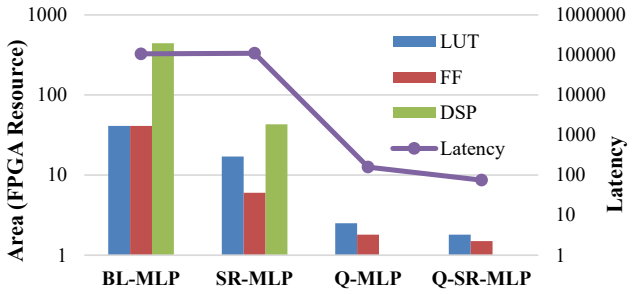

Figure 5: Hardware Accelerators for MLP Model on the MNIST Task.

The Q–SR–MLP version delivers the most aggressive compression while still preserving accuracy. Compared to SR-MLP, the Q-SR-MLP variant reduces latency by more than an order of magnitude and lowers LUT/FF usage to under 1% of the original baseline. These results confirm that NeuSym–HLS works effectively: symbolic final-layer replacement consistently reduces hardware cost, when applied to either convolutional models or MLPs.

D.1 SVHN Binary Classifier: Depth and Operator-Set Exploration

We now evaluate NeuSym-HLS on the SVHN dataset using a compact binary classifier architecture. Our analysis explores two axes: (1) the depth of symbolic replacement—whether 1 or 2 layers are replaced—and (2) the symbolic operator family used.

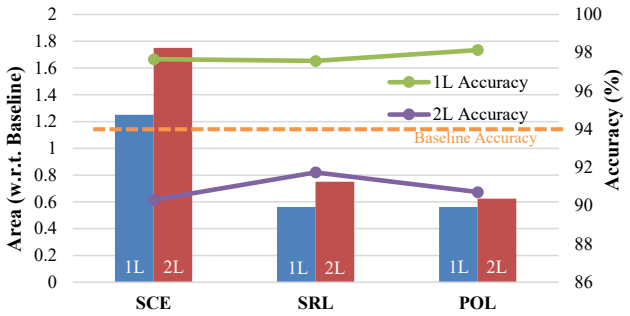

Figure 6: 1-layer vs. 2-layer Symbolic Distillation, in Area and Accuracy.

Fig. 6 shows the evaluation results. Among various versions, SR-1L-POL achieves the highest accuracy (98.2%) while maintaining the lowest LUT usage (5%). SR-1L-SRL trails slightly at 97.6% but with nearly identical hardware usage, whereas SR-1L-SCE lags in both accuracy and LUT cost. This confirms that operator set selection has a tangible impact on hardware efficiency. SCE incurs the highest DSP cost at 6.8%, while SRL and POL remain below 3%.

We next examine the effect of replacing two layers instead of one. Evaluation results show that 2-layer symbolic models consistently degrade in accuracy across all operator families. For instance, accuracy drops by 7.4 points for SCE and 6.9 points for POL compared to their 1-layer counterparts. Resource costs also increase, confirming that aggressive symbolic replacement compromises both accuracy and efficiency.

Finally, TABLE 3 presents detailed synthesis metrics for all designs. NeuSym-HLS achieves better accuracy than both the quantized baseline (Q–MLP) and prior SR accelerators (e.g., SR-Paper [5]) while reducing LUT and DSP usage by over 10× in some configurations. Notably, Q–SR-1L-POL matches the performance of SR-1L-POL but executes in just 65k cycles, enabling low-latency inference without multipliers.

These results confirm that symbolic regression is most effective when applied selectively and targeting a single output layer using low-complexity, hardware-friendly operators.

E Discussion and Future Work

E.1 Resource–Latency Trade-offs

Our results demonstrate that even replacing a **single fully connected layer** with a symbolic expression preserves accuracy within 1% of the floating-point baseline while reducing LUT and flip-flop usage by an order of magnitude and eliminating almost all DSP requirements (Tables 2, 3). Quantization excels at raw speed, achieving up to $675\times$ latency reductions on MNIST, but can inflate logic usage on larger inputs (e.g., 90% LUT for SVHN). On SVHN, symbolic regression alone can offer the best Pareto trade-off: `SR-1L-POL` achieves 98.2% accuracy with only 5% LUT and 7% DSP, surpassing both baseline and quantized variants. Combining QAT with symbolic regression (`Q-SR`) is most beneficial for small networks (e.g., MNIST MLP), but provides diminishing returns for deeper architectures due to accumulator overhead. For example, on MNIST, quantized symbolic models achieve up to 98% LUT savings relative to the baseline with less than 1% drop in accuracy. On SVHN, replacing one layer with symbolic expressions delivers strong accuracy-resource trade-offs, while two-layer replacements begin to significantly degrade accuracy, highlighting a practical limit to symbolic compression depth.

E.2 Operator Set and Depth Ablations

Our operator-set ablation shows that the polynomial-only (`POL`) search yields the best balance of accuracy and DSP cost. Depth ablation confirms that symbolic regression is most effective for the last, low-dimensional layers; replacing multiple layers can halve logic again, but with up to 8 percentage points accuracy drop, indicating diminishing returns for high-dimensional layers.

E.3 Current Limitations

Some practical limitations remain. Currently, symbolic partitioning and formula integration require manual intervention (copy-pasting PySR outputs, editing typecasts), which can be streamlined. Also, symbolic regression search can be computationally intensive, particularly as the number of candidate operators or target layers and dimensions increase. This is expected, as the search space for equations grows combinatorially with the number of available parameters. Recent work formally proves that symbolic regression is an NP-hard problem [10], confirming the intractability of exact solutions in general cases. Finally, while symbolic methods are effective for fully connected layers, direct support for convolutional and high-dimensional layers remains an open challenge.

E.4 Future Work

To further improve applicability and automation, future directions include: **Automated partition selection:** Heuristics or learning-based methods to choose layers that yield optimal trade-offs. **CNN layer support:** Extending symbolic regression to convolutional or depth-wise layers, possibly by constraining expression growth. **Operator Complexity Study**: An algorithm to learn hardware-specific cost models (latency and resource) for each operator type, and adaptively promote or demote operators during training so that the final expression set simultaneously minimizes resource cost and maintains accuracy performance.