Fermilab

LUNA: LUT-Based Neural Architecture for Fast and Low-Cost Qubit Readout

arXiv:2512.07808

LUNA: LUT-Based Neural Architecture for Fast and Low-Cost Qubit Readout

Muhammad Ali Farooq
Arizona State University
Tempe, Arizona, USA
mafarooq19@asu.edu

Giuseppe Di Guglielmo
Fermi National Accelerator
Laboratory
Batavia, Illinois, USA
gdg@fnal.gov

Abhi Rajagopala
University of Arkansas
Fayetteville, Arkansas, USA
abhi@uark.edu

Nhan Tran
Fermi National Accelerator
Laboratory
Batavia, Illinois, USA
ntran@fnal.gov

Vidya Chhabria
Arizona State University
Tempe, Arizona, USA
vachhabr@asu.edu

Aman Arora
Arizona State University
Tempe, Arizona, USA
aman.kbm@asu.edu

Abstract

Qubit readout is a critical operation in quantum computing systems, which maps the analog response of qubits into discrete classical states. Deep neural networks (DNNs) have recently emerged as a promising solution to improve readout accuracy . Prior hardware implementations of DNN-based readout are resource-intensive and suffer from high inference latency, limiting their practical use in low-latency decoding and quantum error correction (QEC) loops.

This paper proposes **LUNA**, a fast and efficient superconducting qubit readout accelerator that combines low-cost integrator-based preprocessing with Look-Up Table (LUT) based neural networks for classification. The architecture uses simple integrators for dimensionality reduction with minimal hardware overhead, and employs LogicNets (DNNs synthesized into LUT logic) to drastically reduce resource usage while enabling ultra-low-latency inference. We integrate this with a differential evolution based exploration and optimization framework to identify high-quality design points.

Our results show up to a **10.95× reduction in area** and **30% lower latency** with **little to no loss in fidelity** compared to the state-of-the-art. LUNA enables scalable, low-footprint, and high-speed qubit readout, supporting the development of larger and more reliable quantum computing systems.

CCS Concepts

• **Computer systems organization → Quantum computing**.

Keywords

Qubit Readout, Quantum Computer Architecture, Quantum Control Hardware

1 Introduction

Scalable quantum computing demands fast, accurate, and resource-efficient qubit readout. In superconducting platforms [7], the readout operation converts microwave responses from resonator-coupled qubits into digital I/Q traces, which are then processed and discriminated as $|0\rangle$ or $|1\rangle$. This digital signal processing is typically performed on FPGA or RFSoC-based controllers [8, 19, 20], which handle demodulation, integration, and classification in real time (Figure 1).
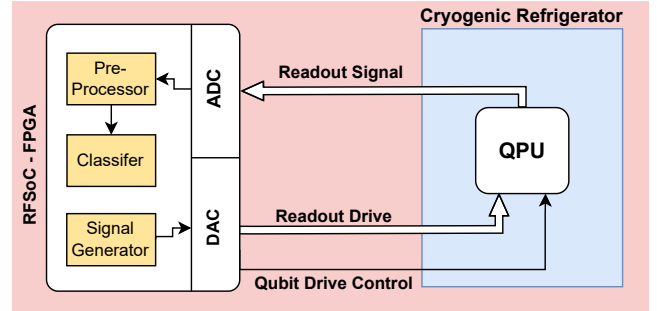

Figure 1: Simplified superconducting-qubit readout chain. RFSoC-FPGA is responsible for qubit drive and readout.

As quantum processors scale to hundreds or thousands of qubits, readout systems face three constraints: (*i*) **latency**, since mid-circuit measurement and feedback must occur within coherence windows; (*ii*) **accuracy**, as assignment errors directly degrade algorithmic and QEC performance; and (*iii*) **hardware footprint**, as limited FPGA resources must perform many parallel tasks.

Recent work demonstrates that neural-network discriminators can significantly improve readout fidelity by compensating for system nonidealities [9, 10]. However, large models and complex preprocessing stages often make such implementations resource-heavy and slow, limiting scalability and mid-circuit usability. While FPGA-accelerated ML classifiers have shown nanosecond-scale inference [5, 6, 17], the trade-offs in cost (latency and area) and fidelity remain underexplored.

This work aims to push qubit-readout acceleration toward the ultra-fast, resource-efficient regime. We co-design preprocessing and classification for FPGA-based qubit readout and make two key observations. First, simple *integrators* can be used in lieu of expensive matched filters [15][6] with minimal to no fidelity loss, with low preprocessing cost and reducing model footprint through dimensionality reduction. Second, *LUT-based neural networks* map efficiently to FPGA primitives, providing ultra-low-latency inference with minimal area, for qubit classification. We jointly optimize both these components via a combined design-space exploration (DSE) and neural architecture search (NAS) framework using differential evolution.

Table 1: Representative prior FPGA-based readout efforts. "DSE/NAS" indicates whether a formal architecture search was performed.

Work	Qubits	Preprocessing	Classifier	FPGA/RFSoC?	DSE/NAS?
Lienhard [9]	5	Matched filter	Fully connected DNN	✗	✗
Vora [17]	1	Matched filter	Shallow NN	✓	✗
Di Guglielmo [5]	1	—	hls4ml DNN	✓	Partial
Guo [6]	1	Matched filter + averaging	Distilled DNN	✓	Compression only
This work	1	Integrator (co-designed)	LogicNet (LUT-DNN)	✓	DE-based DSE/NAS

We implement and evaluate our superconducting qubit readout discrimination accelerator on an AMD/Xilinx FPGA. Our results demonstrate significant reductions in FPGA resource usage and inference latency with no degradation in fidelity, compared to the SOTA implementation [5].

We make the following contributions in this work:

(1) We introduce the first ever use of **LUT-based neural networks (LogicNets [16])** for FPGA-based zero DSP qubit state discrimination acceleration.

(2) We develop a **lightweight integrator-based preprocessing pipeline** that maintains fidelity at far lower cost resource cost.

(3) We perform a joint **DSE+NAS** across preprocessing and LogicNet architectures using differential evolution to explore and evaluate trade-offs, marking the first structured NAS attempt using LUT based DNNs.

(4) We present an **FPGA implementation that is compatible with the Quantum Instrumentation and Control Kit (QICK) [12]**, demonstrating upto a 10.95× reduction in area and a 30.9% reduction in latency at competitive fidelities.

The LUNA flow is fully automated, and is available at *blinded*.

2 Background and Related Work

2.1 Qubit readout for superconducting devices

Single-shot readout of superconducting qubits converts a quantum state into a classical microwave response. A readout pulse probes a resonator coupled to the qubit; the reflected/transmitted waveform is amplified, down-converted, and digitized to yield time-series in-phase (I) and quadrature (Q) traces. The digital front end, typically a Radio Frequency System on Chip (RFSoc) FPGA, performs demodulation, preprocessing, and classification to produce a binary assignment ($|0\rangle$ or $|1\rangle$).

Classical preprocessing choices are matched filtering [15] sliding window averaging. Matched filters maximize signal-to-noise ratio (SNR) under Gaussian noise but require multipliers and memory, which scale poorly when replicated across many channels. Sliding window averaging is hardware-efficient (adders and shifts only) but slightly suboptimal in discrimination performance [6]. The front-end design thus exposes a fidelity vs. resource tradeoff critical to scalable FPGA-based readout systems. Keeping this tradeoff in mind, we adopt a novel integrator based strategy for preprocessing.

Thresholding methods for classification degrade under crosstalk, nonstationary noise, and device nonlinearities. Learned discriminators (e.g., Deep Neural Networks (DNN)) improve fidelity, especially across multiplexed channels [9]. These models can compress and denoise time-series inputs while adapting to experimental non-idealities that fixed templates cannot capture.

Several recent works demonstrate real-time ML-based readout on FPGA/RFSoC hardware. Di Guglielmo et al. present an end-to-end workflow that couples QICK [12] with hls4ml [4], achieving low-nanosecond inference but consuming tens of thousands of LUTs and hundreds of DSPs[5], with no preprocessing or dimensionality reduction. Vora et al. deploy RFSoC-integrated discriminators, utilizing digital local oscillator (DLO) as multiplication strategy for deploying matched filters [17] and integrated it into the QubiC framework [19]. Guo et al. use distillation to compress large readout networks into smaller FPGA-friendly models [6]. They utilize both matched filter and window averaging as pre-processors to form feature vectors for classification. These works validate ML inference on FPGA platforms but tend to rely on multiplier-heavy layers (incurring DSP/BRAM cost) or limited/non-structured design-space search for the preprocessing + classifier co-design.

We use the work of Di Guglielmo et al. [5] as a baseline due to the ready availability of the readout data in the form of [3]. Given that qubit response varies from device to device, we do not draw performance comparisons with other works.

2.2 LUT-based networks

Mapping inference directly into FPGA LUTs eliminates multiplier/DSP dependence by treating quantized neurons as small Boolean truth tables that can be implemented as native K-input LUT primitives. Early work (LUTNet [18]) showed that trained, binarized operators can be hardened into LUT configurations to yield very area-efficient, low-latency inference engines; subsequent efforts developed toolchains to convert trained networks into LUT masks and to unroll operators for maximal parallelism. LogicNets extends this idea with an explicit hardware–software co-design of a LUT mapped neural network: during training it constrains connectivity (low fan-in), encourages sparsity, and uses quantization so that each neuron's function can be extracted as one (or a small number of) LUT truth tables, producing a directly deployable, highly-pipelined FPGA netlist [16]. Weightless neural networks (WNNs) and RAM-based classifiers (WiSARD variants and recent LogicWiSARD work) take the table-lookup idea further toward memory-centric, lookup-only inference and can achieve extremely low latency and energy, but they typically trade off accuracy or require different memory/BRAM tradeoffs compared with LUT-mapped quantized networks [11, 14].

Practically, LUT-based DNN implementation forces a set of co-design tradeoffs. Limiting neuron fan-in (commonly to device LUT sizes, e.g., 6–8 inputs) avoids exponential truth-table growth but often requires deeper or wider DNN topologies or input-partitioning

3 The LUNA Approach

3.1 Overview and design flow

Figure 2 summarizes our co-design flow. Our approach couples a hardware-aware neural-network design methodology (LogicNets) with an evolutionary architecture search (differential evolution) and automated FPGA synthesis. The flow is as follows: From a parameterized design space ❶, we apply light pruning heuristics to remove infeasible design points ❷, then run a search and optimization loop driven by DE❸. We then fully implement the resulting design points ❹ and implement and evaluate them ❺.

3.2 Architecture

The accelerator comprises two tightly-coupled blocks: (i) an **integrator preprocessor** that performs low-cost dimensionality reduction, and (ii) a **LogicNet classifier** where each neuron is mapped directly into FPGA LUTs for ultra-low-latency inference [16]. Data flows in a short, pipelined path: digitized I/Q samples are captured, routed into the integrator windows, reduced to a compact feature vector, then fed into the LogicNet for immediate classification. A high level overview of this architecture is given Fig. 3.

3.2.1 Integrator Based Preprocessor. Integrators compress the raw I/Q trace by partitioning the ADC samples into num_filter fixed, non-overlapping windows and computing an accumulation per window. In each window, the incoming 14-bit ADC samples are optionally right-shifted by $shift_m$ (pre-accumulation) to discard LSB noise and reduce precision, summed in an adder-tree, and then right-shifted by $shift_n$ (post-accumulation) to normalize dynamic range and produce a quantized scalar I and Q feature per window. These scalar pairs are concatenated to form the input feature vector for the classifier. The integrator is parametrizable by the start ADC sample index/total number of ADC samples, number of windows, and shift amounts; its hardware is an adder-tree with pipeline registers at each stage. We adopt this approach to reduce input dimensionality in order to reduce the cost of the classifier, and to do so in a manner that does not consume expensive DSPs.

3.2.2 LogicNet classifier. The feature vector from the integrator is presented to a LUT-based neural network implemented via LogicNets, where each neuron is synthesized as a Boolean function implemented directly in FPGA LUT primitives [16]. A LogicNet is defined by its layer widths $[\ell_1, \ldots, \ell_{k-1}]$, per-layer fanin ($\gamma$) and bitwidth ($\beta$). Each Neuron Equivalent (NEQ) consumes $X = \beta \cdot \gamma$ input bits and produces $Y = \beta$ output bits, effectively implementing an $X : Y$ LUT operation. These parameters determine both model capacity and FPGA cost. Because all neurons are purely combinational and the network is fully pipelined (typically only 1–2 LUTs between registers) the classifier achieves very low latency suitable for tight QEC timing budgets.

3.3 Design space

The design space of the integrator and LogicNet is defined by various parameters of each stage, listed in Table 2. We model a candidate design point as a fixed-length integer vector v consisting of a value of each parameter. Each parameter's value can have a large range, leading to a vast design space, which is difficult to explore. We apply the following heuristics to prune the ranges for each parameter:
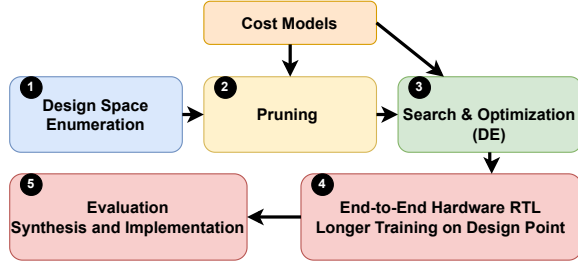

Figure 2: LUNA co-design flow: (1) enumerate, (2) prune, (3) DE-based NAS, (4) generate RTL (integrator + LUT-DNN), (5) implement design on target device and evaluate fidelity on test set.

(decomposing a large receptive field across multiple LUTs) to preserve accuracy. Input-packing and carefully chosen quantization schemes are used to amortize LUT cost. The standard toolflow is to design a fan-in-aware topology, apply sparsification/pruning and constrained retraining, then extract neuron truth tables for direct LUT instantiation. The main benefits of LUT-based DNNs are removal of multiplier/DSP resources, single-cycle (or deeply pipelined) per-layer latency, and highly predictable timing.

We adopt the LogicNets approach in this work because it explicitly co-optimizes topology, sparsity, and quantization to produce directly mappable LUT truth-tables with a controllable fan-in vs. accuracy tradeoff. Compared with LUTNet [18], LogicNets [16] provides a clearer training-to-netlist pipeline and tighter topology constraints for predictable area–accuracy tradeoffs, and compared with WNNs [11, 14], it preserves closer compatibility with modern quantized DNN accuracy while still delivering the LUT-native, multiplier-free hardware advantages we require.

2.3 DSE and NAS

Automated techniques, like reinforcement learning, Bayesian optimization, and evolutionary algorithms, have been applied to DSE and NAS problems. Differential Evolution (DE) is a simple, gradient-free evolutionary optimizer that adapts readily to mixed encodings and costly black-box evaluations and has been used successfully for NAS-style problems [2, 13]. We adopt DE since it's population-generation and selection mechanism fits naturally with our search; candidate evaluations (DNN training + resource cost prediction) are expensive but can be easily parallelized, as compared to other meta-heuristic approaches like Simulated Annealing.

2.4 LUNA vs. prior work

Table 1 summarizes prior FPGA-based readout works that use ML. Prior efforts demonstrated ML feasibility on FPGA/RFSoC platforms but generally rely on multiplier-based layers, limited preprocessing co-design, or offline compression. To our knowledge, none combine (i) a low-cost integrator front end, (ii) LUT-mapped LogicNet classifiers, and (iii) an automated DSE-NAS that jointly optimizes preprocessing and LUT-DNN topology for area/latency/fidelity tradeoffs.
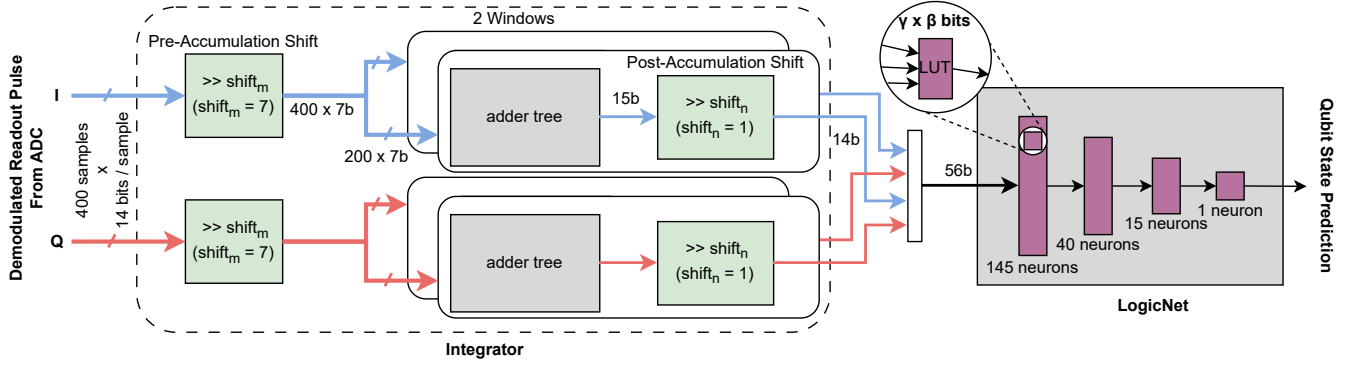
Figure 3: A high level overview of the LUNA architecture. Values shown are demonstrative; taken from our `fidelity-optimized` solution shown in Table 3.

Table 2: Search-space parameters and allowed ranges.

Parameter	Description	Range
start time	Start sample index (ADC); end fixed at 500.	{0, 50, 100}
# windows	Signal partitions before preprocessing.	{1, 2, 3, 4}
shift_m	Pre-accum. right shift (ignore LSB noise).	{2, 3, …, 7}
shift_n	Post-accum. right shift (scale integrator output).	{0, 1, …, 6}
ℓ_0	NEQs in input layer.	{25, 30, …, 145}
# layers	Hidden layer count.	{2, 3}
$\ell_1, \ell_2, \ldots$	NEQs per hidden layer.	{5, 10, …, 45}
β_i, β, β_o	NEQ input bitwidth (input, hidden, output).	{1, 2}
γ_i	NEQ fan-in (input layer).	{6, 7}
γ, γ_o	NEQ fan-in (hidden/output layers).	$\begin{cases}\{6, \ldots, 16\}, & \beta = 1 \\ \{6, 7, 8\}, & \beta = 2\end{cases}$

(1) **Area-based pruning:** A conservative LUT estimate is made using a simple model (Section 3.5.2) and designs with estimated LUTs > $A_{\max}$ = 20,000 (empirically chosen) are discarded.

(2) **Full cost-based pruning:** A number of random design points are created after the area-based pruning and fully evaluated according to the criteria in Section 3.5. These results are used to further refine our design space.

3.4 Search and Optimization

The design space of the preprocessing and classifier architecture is very large, due to the large number of variables that govern the architecture (eg. signal window size, normalization, and sparsity settings. See Section 3.3.) The search and optimization of our preprocessing and classifier architecture is unique, as it requires both design space exploration (DSE) for the preprocessor and neural architecture search (NAS) for the classifier. Since isolated optimization of either can ignore better performing points, we are motivated to perform joint optimization of the spaces. We therefore fold the DSE+NAS problem into a single search and optimization problem.

We use Differential Evolution (DE) [13] to search the pruned, design space. DE is a population-based optimization algorithm that we adapt for our mixed-discrete design space. It works by maintaining a population of candidate solutions, which evolve over generations through mutation, crossover, and selection. Mutation follows the classical DE rule, where for each member of the population $v^{(i)}$, we draw three distinct population members $v^{(a)}, v^{(b)}, v^{(c)}$ and form a

mutant vector m:

$$m = v^{(a)} + F_{\text{DE}} \cdot \left(v^{(b)} - v^{(c)}\right) \quad (1)$$

where F_{DE} is the mutation rate. Crossover then constructs a trial 'offspring' vector o by combining the mutant m with the parent $v^{(i)}$. Each element in o is determined as:

$$o_j = \begin{cases} m_k, & \text{if } k = j, \\ m_j, & \text{if } u_j < CR, \\ v_j^{(i)}, & \text{otherwise,} \end{cases} \quad (2)$$

where k is a random index selected from the length of the vector, u_j is a random number sampled uniformly in $[0, 1]$, and CR is the crossover rate. The parameter values of the resulting offspring are clipped and snapped to the nearest valid configuration before evaluation. After evaluation, the offspring replaces its parent if it exhibits a lower cost thereby surviving to the next generation. This process repeats for $G_{\max}$ generations, allowing us to arrive at a viable solution point. Early stopping terminates the search when no improvement is observed for a fixed patience window P. With DE, evaluation of each candidate design point becomes independent and therefore fully parallelizable, making DE an attractive optimization strategy for our use case. With full parallelization across all evaluations within a generation, we are able to evaluate a generation within $\approx$ 4 minutes.

3.5 Composite cost and metric estimation

To evaluate each design point, we use a composite cost C that is defined as a weighted combination of normalized area, latency, and fidelity metrics. Weights w_a, w_l, w_f define the importance of the three metrics.

$$C = w_a \widetilde{A} + w_l \widetilde{L} + w_f \widetilde{\mathcal{F}}, \text{ where:} \quad (3)$$

$$\widetilde{A} = \frac{\text{area}}{A_{\max}}, \quad \widetilde{L} = \frac{\text{latency}}{L_{\max}}, \quad \widetilde{\mathcal{F}} = \frac{1 - \text{fidelity}}{1 - 0.90}$$

and where $A_{\max}$ and $L_{\max}$ are empirically chosen (based on the random exploration in Section 4.4) to be 20,000 LUTs and 14 cycles. This does not hinder our search process, and allows us to have a meaningful combined cost metric.

The ideal evaluation of a design point would entail training the LogicNet fully on the available data for fidelity evaluation, and

complete synthesis and implementation of the end-to-end solution (integrator and LogicNet) on the target device for area and latency. The cost of that design point would then be calculated using the equation above. However, this would be extremely time consuming, and would make exploration of the large design space prohibitively expensive. As such, we develop inexpensive methods for quick and accurate estimates.

3.5.1 Latency estimation. Latency (cycles) is estimated analytically as latency = integrator_cycles + LogicNet_stages, with integrator cycles approximated by the adder-tree pipeline depth $\lceil \log_2(N) \rceil$ for N inputs, and each LogicNet layer contributing 1 pipelined cycle.

3.5.2 Area estimation. Area is approximated by the number of LUTs (flip-flops (FF) are ignored; number of FFs generally tracks with the number of LUTs). LUT count is predicted using a hybrid model:

- **Integrator LUTs** are estimated using a regression model trained on a few synthesized adder-tree RTL designs (features: number of inputs, bitwidth).
- **LogicNet LUTs** are estimated using analytical per-neuron LUT-equivalents using the analytical formula from [16] and a correction factor obtained through a lightweight regression trained on a few implemented models.

This cost model is calibrated using a few end-to-end implementations, and is able to predict end-to-end area within ±20%.

3.5.3 Fidelity evaluation. Fidelity $\mathcal{F}$ is obtained by training the LogicNet on the full dataset and reporting the classification metric $\mathcal{F} = 1 - 0.5 \cdot \big(P(0 \mid 1) + P(1 \mid 0)\big)$ on the test set for parity with prior work for a reduced number of epochs (see Sec. 4.3).

3.6 FPGA implementation

For our final FPGA implementation, we utilize our hand-coded, parameterized RTL templates for the preprocessing segment. For the classifier segment, we use the LogicNets framework [16] to generate RTL from the trained model. These are synthesized and placed-and-routed on the target FPGA. For each design point, we extract post-implementation resource utilization (LUTs, FFs, DSPs) and timing reports (worst negative slack, reported clock period). Similar to [5], our end to end implementation takes a fixed 2 cycles to save the discrimination prediction to memory; as such, we add a fixed 2 cycles to our final implementation results.

4 Methodology

4.1 Toolchains and platforms

Our DE implementation and orchestration scripts are implemented in Python 3.8.19. We use the LogicNets framework from AMD/Xilinx [1] for creating our LUT-based DNN. Parameterized RTL (adder trees and integrators) is generated from Mako templates (mako 1.3.10), enabling a single template family to emit RTL for many design points. Xilinx Vivado 2022.2 performs synthesis and implementation, for timing extraction and to obtain final resource usage.

4.2 Dataset

We use the publicly available superconducting-qubit readout dataset (time-series I/Q traces, labeled $|0\rangle / |1\rangle$) used by Guglielmo et al. [3]. The dataset and experimental conventions (readout lengths,

sampling, and QICK-based capture) follow the end-to-end readout workflows previously reported. [5, 12]

For the experiments reported here, we use a fixed split of 90,000 training samples and 10,000 test samples. During the search phase (DE evaluation), we train and validate candidate LogicNets on the full split described above to obtain robust fidelity estimates (see next subsection for training protocol).

4.3 Training protocol

To balance search throughput and fidelity estimation reliability, we use a two-stage training protocol:

(1) **Search-time training:** During DE search each candidate LogicNet is trained *and* evaluated on the full dataset split for **5 epochs** using a **batch size of 512**. This reduced training budget is a trade-off: it yields stable, comparable fidelity estimates across many candidates while keeping per-candidate wall-clock time manageable.

(2) **Final training:** Once the DE procedure selects a final candidate, that architecture is re-trained from scratch on the same training split for **30 epochs** using a **batch size of 1024**. This final training run uses identical preprocessing and quantization settings as during search but with the larger epoch count and batch size to realize the best achievable fidelity prior to FPGA conversion and deployment.

The use of a full-dataset training during search (rather than a proxy subset) helps reduce variance across fidelity estimates when comparing candidate designs.

4.4 Differential evolution parameters

We use a population size of $NP = 75$ and adopt DE settings of scale factor $F_{\text{DE}} = 0.7$ and crossover rate $CR = 0.8$. The maximum number of generations is $G_{\text{max}} = 150$, and early termination is triggered after $P = 40$ generations without improvement. These values were chosen empirically to balance exploration and convergence speed.

4.5 Baseline

The end-to-end readout work by Guglielmo et al. [5] is used as a state-of-the-art reference for ML-based FPGA readout systems, especially since it uses the same dataset [3]. As such, we perform the FPGA implementation for the $XCZU49DR$ device as well.

5 Results

5.1 Search and Optimization Results

We perform search and optimization using our described flow under three optimization targets for separate LUNA architectures:

- **Area-Optimized**: prioritize area efficiency (w_a=0.8,w_l=0.1,w_f=0.1)
- **Latency-Optimized**: prioritize inference time (w_a=0.1,w_l=0.8,w_f=0.1)
- **Fidelity-Optimized**: prioritize fidelity (w_a=0.1,w_l=0.1,w_f=0.8)

Each configuration reports the best architecture discovered, including integrator and LogicNet parameters, along with estimated performance metrics. The results for each run are given in Table 3.

Table 3: Search and Optimization Results. For each optimization target, the top-performing configuration and estimated metrics are listed. Latency is quoted in cycles, Area is quoted in LUTs

Target	Configuration											Estimated Results		
	Start Sample	# Windows	$shift_m$	$shift_n$	Layers	β_i	β	β_o	γ_i	γ	γ_o	Area	Latency	Fidelity (%)
Fidelity	100	2	7	1	145, 40, 15, 1	1	2	2	7	6	8	8226	12	95.995
Area	100	1	9	0	25, 5, 5, 1	1	1	2	6	6	11	6754	13	95.929
Latency	100	2	9	1	145, 35, 15, 1	1	1	2	6	6	7	7311	12	95.885

Table 4: FPGA Implementation Summary. Results include logic utilization, latency, and fidelity. Resource usage is quoted as percentages of available resources on the $XCZU49DR$ device.

Architecture	Target	LUTs	Flip Flops	DSPs	Period (ns)	Latency (ns)	Fidelity
LUNA	Fidelity	10,642 (2.50%)	13,206 (1.55%)	0 (0.00%)	1.751	24.51	96.059%
	Area	6,095 (1.43%)	9,688 (1.14%)	0 (0.00%)	1.588	23.82	95.924%
	Latency	6,205 (1.46%)	9,809 (1.15%)	0 (0.00%)	1.579	22.10	95.969%
Baseline [5]	—	66,600 (15.66%)	34,369 (4.04%)	481 (11.26%)	3.220	32.00	96.000%

The fidelity-optimized model uses a wider network and moderate integrator shifts, reaching $\approx 96.0\%$ fidelity with a 12-cycle latency and 8226 LUTs. The area-optimized design uses a much smaller LogicNet and more aggressive pre-accumulation, reducing area to 6754 LUTs while retaining 95.93% fidelity. The latency-targeted run also settles at 12 cycles (7311 LUTs, 95.89%). The fidelities are tightly clustered while the three points emphasize different area/latency tradeoff — the search did not produce lower-latency candidates with a lower objective cost, so both latency- and fidelity-targeted runs converged to 12-cycle designs.

The cost of the best performing candidate per generation is tracked in Figure 4, demonstrating that improvement does occur over the course of DE.

Figure 4: Best cost trajectory across generations for each target objective: (a) Fidelity-optimized, (b) Latency-optimized, and (c) Area-optimized. Each curve shows the best individual's cost per generation.
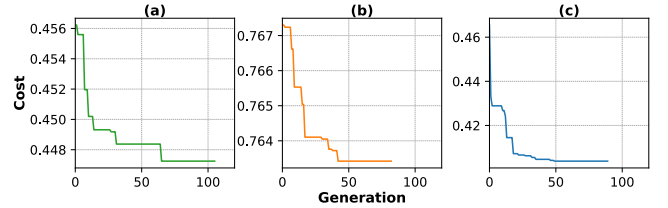
5.2 FPGA Implementation Results

As summarized in Table 4, all three target variants achieve large LUT and FF reductions relative to the baseline accelerator while maintaining competitive fidelity and, most notably requiring **zero DSP blocks**. This demonstrates that both the integrator frontend and the classifier can be mapped entirely onto LUT fabric without sacrificing readout accuracy.

The fidelity optimized LUNA architecture improves upon SOTA fidelity by 0.059% fidelity, with a 6.3× reduction in LUT usage and a 23.4% reduction in latency. The area optimized architecture achieves a 10.95× reduction in LUT usage and a 25.6% improvement in latency with only a 0.076% loss of fidelity. The latency optimized architecture improves upon latency by 30.9% with a 10.74× reduction in LUT use and a fidelity loss of only 0.031%.

These hardware results validate the trends observed during DE-based search and confirm that the flow yields consistently compact and efficient implementations.

6 Discussion

This work shows that combining an integrator-based dimensionality reduction stage with LUT-mapped LogicNet classifiers enables substantial FPGA savings while preserving state-of-the-art qubit-state discrimination fidelity. The integrator compresses raw I/Q trajectories into a small set of aggregate features using simple shift-and-accumulate operations, reducing classifier input dimensionality with negligible compute cost. When paired with LogicNets, the resulting LUNA flow achieves order-of-magnitude LUT reductions, lower latency, and zero DSP usage, freeing arithmetic resources for qubit control and signal-generation tasks.

Reducing per-qubit area is increasingly important as quantum systems move toward mid-circuit measurement and quantum error correction (QEC), where large numbers of concurrent readout paths are required. A smaller hardware footprint directly increases the number of readout engines that can be instantiated on a single device, supporting the scaling needs of future processors.

Several avenues for future work follow. Evaluating LUNA on additional datasets, including multi-qubit or higher-dimensional readout traces, will help assess generality. Extending the architecture to multi-qubit joint classifiers could capture crosstalk and correlated noise, a limitation of per-qubit pipelines noted in prior work. Finally, richer search strategies may provide improved coverage of the design space and may uncover even better solutions.

7 Conclusion

This paper presents LUNA, a hardware–software co-design framework that couples integrator-based dimensionality reduction with LUT-based neural network classifiers to produce highly efficient FPGA implementations for qubit-state readout. Using differential evolution to jointly optimize preprocessing and classifier structure, LUNA identifies compact designs that maintain high discrimination fidelity while drastically reducing hardware footprint. Across the studied dataset, LUNA reaches fidelities near 96% with up to a 10.95× LUT reduction and up to a 30% latency improvement, all with zero DSP utilization. These resource savings directly support scalable quantum processors, particularly scenarios requiring large numbers of parallel mid-circuit measurements and QEC operations.

References

[1] 2020. *LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications.* Retrieved November 01, 2025 from https://github.com/Xilinx/logicnets/tree/master

[2] Noor Awad, Neeratyoy Mallik, and Frank Hutter. 2020. Differential evolution for neural architecture search (dehb). In *International Conference on Machine Learning (ICLR) Neural Architecture Search (NAS) Workshop.*

[3] Batao Du. 2024. *Data for "End-to-end workflow for machine learning-based qubit readout with QICK and hls4ml.* Retrieved November 11, 2025 from https://doi.org/10.5281/zenodo.14427490

[4] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, Dylan Rankin, Manuel Blanco Valentin, Josiah Hester, Yingyi Luo, John Mamish, Seda 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 Co-Design Workflow to Empower Scientific Low-Power Machine Learning Devices. In *Research Symposium on Tiny Machine Learning.* https://openreview.net/forum?id=YBFqldo30vG

[5] Giuseppe Di Guglielmo, Botao Du, Javier Campos, Alexandra Boltasseva, Akash Dixit, Farah Fahim, Zhaxylyk Kudyshev, Santiago Lopez, Ruichao Ma, Gabriel N. Perdue, Nhan Tran, Omer Yesilyurt, and Daniel Bowring. 2025. End-to-End Workflow for Machine-Learning-Based Qubit Readout With QICK and hls4ml. *IEEE Transactions on Quantum Engineering* 6 (2025), 1–10. doi:10.1109/TQE.2025.3604712

[6] Xiaorang Guo, Tigran Bunarjyan, Dai Liu, Benjamin Lienhard, and Martin Schulz. 2025. KLiNQ: Knowledge Distillation-Assisted Lightweight Neural Network for Qubit Readout on FPGA. In *2025 62nd ACM/IEEE Design Automation Conference (DAC).* 1–7. doi:10.1109/DAC63849.2025.11132854

[7] Johannes Heinsoo, Christian Kraglund Andersen, Ants Remm, Sebastian Krinner, Theodore Walter, Yves Salathé, Simone Gasparinetti, Jean-Claude Besse, Anton Potočnik, Andreas Wallraff, and Christopher Eichler. 2018. Rapid High-fidelity Multiplexed Readout of Superconducting Qubits. *Phys. Rev. Appl.* 10 (Sep 2018), 034040. Issue 3. doi:10.1103/PhysRevApplied.10.034040

[8] Keysight Technologies. 2024. Quantum Control System. https://www.keysight.com/us/en/products/modular/pxi-products/quantum-control-system.html. Accessed: 2024-03-05.

[9] Benjamin Lienhard, Antti Vepsäläinen, Luke C.G. Govia, Cole R. Hoffer, Jack Y. Qiu, Diego Ristè, Matthew Ware, David Kim, Roni Winik, Alexander Melville, Bethany Niedzielski, Jonilyn Yoder, Guilhem J. Ribeill, Thomas A. Ohki, Hari K. Krovi, Terry P. Orlando, Simon Gustavsson, and William D. Oliver. 2022. Deep-Neural-Network Discrimination of Multiplexed Superconducting-Qubit States. *Phys. Rev. Appl.* 17 (Jan 2022), 014024. Issue 1. doi:10.1103/PhysRevApplied.17.014024

[10] Satvik Maurya, Chaithanya Naik Mude, William D. Oliver, Benjamin Lienhard, and Swamit Tannu. 2023. Scaling Qubit Readout with Hardware Efficient Machine Learning Architectures. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23).* Association for Computing Machinery, New York, NY, USA, Article 7, 13 pages. doi:10.1145/3579371.3589042

[11] 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

[12] Leandro Stefanazzi, Kenneth Treptow, Neal Wilcer, Chris Stoughton, Collin Bradford, Sho Uemura, Silvia Zorzetti, Salvatore Montella, Gustavo Cancelo, Sara Sussman, Andrew Houck, Shefali Saxena, Horacio Arnaldi, Ankur Agrawal, Helin Zhang, Chunyang Ding, and David I. Schuster. 2022. The QICK (Quantum Instrumentation Control Kit): Readout and control for qubits and detectors. *Review of Scientific Instruments* 93, 4 (04 2022), 044709. arXiv:https://pubs.aip.org/aip/rsi/article-pdf/doi/10.1063/5.0076249/19817152/044709_1_online.pdf doi:10.1063/5.0076249

[13] Rainer Storn and Kenneth Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization* 11, 4 (01 Dec 1997), 341–359. doi:10.1023/A:1008202821328

[14] 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. 2023. Weightless Neural Networks for Efficient Edge Inference. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques* (Chicago, Illinois) *(PACT '22).* Association for Computing Machinery, New York, NY, USA, 279–290. doi:10.1145/3559009.3569680

[15] G. Turin. 1960. An introduction to matched filters. *IRE Transactions on Information Theory* 6, 3 (1960), 311–329. doi:10.1109/TIT.1960.1057571

[16] 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).* 291–297. doi:10.1109/FPL50879.2020.00055

[17] Neel R Vora, Yilun Xu, Akel Hasim, Neelay Fruitwala, Nam Nguyen, Haoran Liao, Jan Balewski, Abhi Rajagopala, Kasra Nowrouzi, Qing Ji, K. Brigitta Whaley, Irfan Siddiqi, Phuc Nguyen, and Gang Huang. 2024. QubiCML: ML-Powered Real-Time Quantum State Discrimination Enabling Mid-Circuit Measurements. In *2024 IEEE International Conference on Quantum Computing and Engineering (QCE),* Vol. 02. 414–415. doi:10.1109/QCE60285.2024.10332

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

[19] Yilun Xu, Gang Huang, Jan Balewski, Ravi Naik, Alexis Morvan, Bradley Mitchell, Kasra Nowrouzi, David I. Santiago, and Irfan Siddiqi. 2021. QubiC: An Open-Source FPGA-Based Control and Measurement System for Superconducting Quantum Information Processors. *IEEE Transactions on Quantum Engineering* 2 (2021), 1–11. doi:10.1109/TQE.2021.3116540

[20] Zurich Instruments. 2024. Quantum Readout. https://www.zhinst.com/americas/en/quantum-computing-systems/qubit-readout. Accessed: 2024-03-05.