End-to-end workflow for machine learning-based qubit readout with QICK and hls4ml

Giuseppe Di Guglielmo,[1, *] Botao Du,[2] Javier Campos,[1] Alexandra Boltasseva,[3] Akash V. Dixit,[4] Farah Fahim,[1] Zhaxylyk Kudyshev,[3] Santiago Lopez,[2] Ruichao Ma,[2] Gabriel N. Perdue,[1] Nhan Tran,[1] Omer Yesilyurt,[3] and Daniel Bowring[1]

[1] *Fermi National Accelerator Laboratory, Batavia, IL 60510, USA*
[2] *Department of Physics and Astronomy, Purdue University, West Lafayette, IN 47907, USA*
[3] *Elmore Family School of Electrical and Computer Engineering, Birck Nanotechnology Center and Purdue Quantum Science and Engineering Institute, Purdue University, West Lafayette, IN 47907, USA*
[4] *University of Chicago, Chicago, IL 60637, USA*

(Dated: January 27, 2025)

We present an end-to-end workflow for superconducting qubit readout that embeds co-designed Neural Networks (NNs) into the Quantum Instrumentation Control Kit (QICK). Capitalizing on the custom firmware and software of the QICK platform, which is built on Xilinx RFSoC FPGAs, we aim to leverage machine learning (ML) to address critical challenges in qubit readout accuracy and scalability. The workflow utilizes the `hls4ml` package and employs quantization-aware training to translate ML models into hardware-efficient FPGA implementations via user-friendly Python APIs. We experimentally demonstrate the design, optimization, and integration of an ML algorithm for single transmon qubit readout, achieving 96% single-shot fidelity with a latency of 32 ns and less than 16% FPGA look-up table resource utilization. Our results offer the community an accessible workflow to advance ML-driven readout and adaptive control in quantum information processing applications.

Keywords: Quantum Computing; Superconducting Qubit Readout; Machine Learning; Radio Frequency System-on-Chip (RFSoC)

I. INTRODUCTION

Quantum technologies hold the promise to transform a range of applications from computation and communication to sensing. Realizing these quantum advantages, however, requires scaling from current small-scale prototypes to large-scale quantum processors with increasingly complex qubit arrays. As the number of qubits grows, so does the need for a commensurately scalable and efficient classical co-processing infrastructure. The software, firmware, and hardware responsible for controlling and reading out the quantum states must not only support the expanding qubit counts but also maintain the high fidelity and low latency critical for successful quantum operations.

Superconducting (SC) qubits have emerged as a leading platform for building large-scale quantum processors [1]. The rapid progress is thanks to the scalability of the lithographically fabricated superconducting circuits, the instrumentation ecosystem available at the operating frequencies of SC qubits, and advances in cryogenic systems to reach the millikelvin temperatures required for their quantum operation. Noisy Intermediate-Scale Quantum (NISQ) devices with more than a hundred qubits are now in operation, showing significant progress towards computational quantum advantages [2–4]. As these devices increase in complexity, there is a growing need for integrated control and readout solutions that combine functionality, flexibility, ease of deployment, and cost-effectiveness.

Recent developments in Radio Frequency System-on-Chip (RFSoC) technologies are accelerating this effort [5–8]. By combining the capabilities of Field-Programmable Gate Arrays (FPGAs) with RF data converters, RFSoC-based systems provide real-time signal generation via direct digital synthesis in the microwave domain, efficient readout via fast analog-to-digital conversion, and low-latency signal processing. In particular, the Quantum Instrumentation Control Kit (QICK) [5, 9] has seen growing adoption among laboratories developing superconducting qubits, as well as other experimental platforms.

In parallel to this hardware development, there is a crucial need to integrate the classical and quantum hardware with a robust and scalable software stack. Software challenges include device calibration and tune-up, multiplexed readout, and fast adaptive control. These challenges span both classical and quantum control in the presence of noise, crosstalk, and other errors. For future fault-tolerant quantum machines, fast, high-fidelity measurements across large systems and real-time, low-latency adaptive feedback control are essential [10, 11].

The rise of machine learning (ML) tools in classic computing provides a powerful toolset to develop adaptive, heuristic algorithms for qubit readout and control. In particular, ML can be used in readout to account for effects that are difficult to compute analytically, such as non-linear behaviors in the time-evolution of signal traces including noise, multi-qubit correlations and crosstalk, and the evolution of the system dynamics over time due

* gdg@fnal.gov

FERMILAB-PUB-24-0925-ETD-PPD

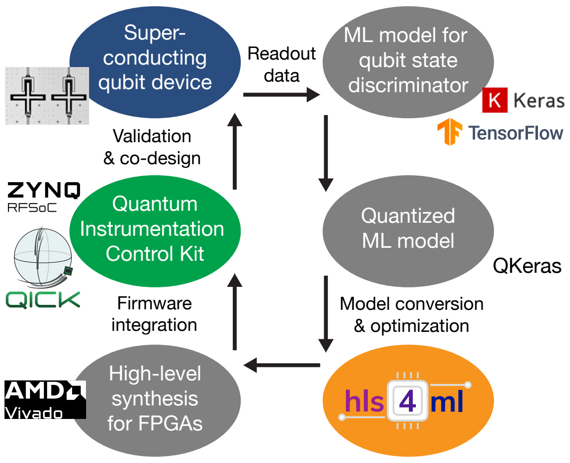
Super-conducting qubit device
Readout data
ML model for qubit state discriminator
Keras
TensorFlow
Validation & co-design
ZYNQ RFSoC
Quantum Instrumentation Control Kit
QICK
Quantized ML model
QKeras
Firmware integration
Model conversion & optimization
AMD Vivado
High-level synthesis for FPGAs
hls 4 ml

FIG. 1. Overview of the QICK and `hls4ml` workflow.

to external effects [12–16]. Ultimately, an active learning ML approach can provide a path to high-fidelity autonomous and adaptive qubit readout control systems. Toward this goal, it is important to develop a platform for researchers to study, train, and deploy ML algorithms, bringing together expertise in quantum systems, machine learning, and readout electronics that meet the constraints of state-of-the-art experiments.

In this paper, we present an open-source workflow based on QICK and `hls4ml` to advance ML-based quantum control. The open-source `hls4ml` package [17, 18] enables users to translate ML models into low-level logic descriptions using High-Level Synthesis tools through accessible Python APIs. This allows for algorithm and experiment design without requiring detailed firmware development knowledge for FPGAs. Specifically, hls4ml supports the optimization of neural network designs in hardware with techniques such as quantization-aware training [19] and pruning [20], which minimize hardware resources and latency by tuning high-level parameters.

We demonstrate an ML-based qubit readout algorithm on QICK, integrating a neural network designed through `hls4ml`. This includes developing and deploying an end-to-end workflow for ML-based superconducting qubit readout and providing open-source tools for others to use in their research. Our initial demonstration focuses on the readout of a single superconducting transmon qubit. The ML implementation achieves a single-shot readout fidelity of 96% with a latency of 32 ns and the maximum FPGA resource usage of Look-Up Tables (LUTs) of 16%. The open-source end-to-end workflow for designing and integrating NN models into QICK is illustrated in Figure 1. The code is made publicly available at Ref. [21].

This paper is organized as follows. In Section 2, we describe the setup for the superconducting qubit system under study, visualize the readout data, and describe non-ML-based readout methods. In Section 3, we detail the NN model training and optimization with `hls4ml` for the available resources within the QICK readout system. Section 4 describes the integration of the `hls4ml` neural network intellectual property (IP) block with the QICK firmware and hardware. We conclude in Section 5, describing the results of this study and what future capabilities this will enable.

II. SUPERCONDUCTING QUBIT READOUT

A. Background

In superconducting (SC) qubits, quantum information is encoded as excitations of electromagnetic resonances formed by SC circuit elements. The transmon qubit [22], a workhorse in contemporary SC quantum processors, can be described as an anharmonic microwave resonator, where the SC Josephson junction provides the anharmonicity. The computational basis of the transmon consists of its two lowest energy levels, denoted as $|0\rangle$ and $|1\rangle$, which correspond to zero and one excitation (microwave photon) in the transmon, respectively. The anharmonicity of the transmon ensures unequal level spacing, enabling coherent control within the computational subspace using resonant microwave pulses.

The quantum state of a transmon qubit is measured by coupling the transmon to a linear microwave resonator [23, 24]. The dispersive interaction between the readout resonator and the qubit induces a frequency shift in the resonator that depends on the qubit state. Consequently, the qubit states can be distinguished by probing the readout resonator at a frequency near its resonance and monitoring the transmitted or reflected quadrature signals. In a typical projective measurement, the quantum state collapses into one of the basis states, and the readout produces a single binary outcome.

Scaling SC quantum processors toward fault-tolerant operation and quantum error correction requires high-fidelity single-shot readout with minimal measurement duration. Achieving this fidelity depends on several design considerations that influence the distinguishability of the qubit states $|0\rangle$ and $|1\rangle$ [23]. These factors include the linewidth of the readout resonator, the magnitude of the dispersive shift, noise from downstream amplifiers, and the number of photons collected during the measurement. The latter is determined by the integration time and the readout power.

B. Control and readout hardware

The experiments in this work are conducted on a frequency tunable multi-transmon device, as described in [25]. The hardware setup for qubit control and readout is illustrated in Fig. 2. The ML-based qubit readout is performed on a single qubit of frequency $\omega_q = 2\pi \times 4.50\,\text{GHz}$, with all other qubits far-detuned in frequency and effectively decoupled. The corresponding readout resonator has frequency $\omega_r = 2\pi \times 6.32\,\text{GHz}$,

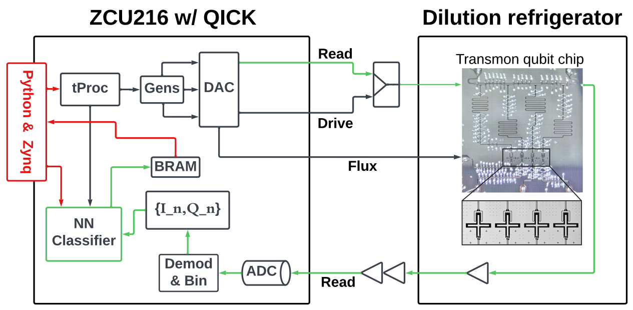

FIG. 2. Experimental setup. The Xilinx RFSoC ZCU216 runs on custom QICK firmware and software for qubit control and readout. The qubit readout and drive pulses are combined before entering the fridge; the flux pulse is used for qubit cooling. The readout signal is amplified before being digitalized by the ADC.

and linewidth $\kappa_r = 2\pi \times 1.5\,\mathrm{MHz}$ from its coupling to the readout transmission line. The qubit-resonator coupling is $g = 2\pi \times 65\,\mathrm{MHz}$, giving a qubit-state-dependent dispersive shift of $2\chi = 0.4\,\mathrm{MHz}$. At the operating frequency, the transmon qubit has a relaxation time of $T_1 = 32\,\mu\mathrm{s}$, and a dephasing time $T_2^* = 1.7\,\mu\mathrm{s}$.

The Xilinx ZCU216 RFSoC evaluation board is running a modified version of the QICK firmware with an integrated ML classifier. We utilize a mixer-free setup without any external local oscillators [9]: The microwave pulses for qubit control and readout are directly synthesized with the ZCU216's 6.88 GS/s digital to analog converters (DACs). rly, the readout signal from the SC qubit device is directly digitized by the ZCU216's analog to digital converters (ADCs) running at 6.88 GS/s and demodulated on the FPGA to generate the time-series quadrature signals $I(t)$ and $Q(t)$.

C. Single-shot qubit readout

To characterize the readout fidelity, we first prepare the transmon qubit in the computational basis of ground state $|0\rangle$ and excited state $|1\rangle$.

In our device, the transmon qubit has an excited state population of approximately 6% at thermal equilibrium due to coupling to environmental noise. To reduce state preparation error from the thermal population, we perform a cooling step at the beginning of each experiment which utilizes the readout resonator as a dissipative cold reservoir. By applying an AC signal of frequency $\omega_r - \omega_q$ on the flux control line to modulate the transmon frequency, the transmon parametrically couples to the resonator and relaxes towards a lower excited state population [25]. Here, we use a flux modulation pulse of duration 2.3 μs, and an amplitude that modulates the qubit frequency by approximately $\pm 36\,\mathrm{MHz}$. This corresponds to an effective resonant qubit-resonator coupling of $g_{\mathrm{eff}} = 2\pi \times 0.65\,\mathrm{MHz}$. The modulation pulse, with a frequency at the qubit-resonator detuning of approximately 1.8 GHz, is generated using a separate DAC channel on the QICK-controlled ZCU216. With the cooling sequence, we prepare the qubit ground state with less than 1.6% thermal population, limited by the thermal population in the readout resonator.repare the excited state, we first perform the cooling step and then apply a resonant microwave π-pulse at the qubit frequency with a Gaussian width $\sigma = 46\,\mathrm{ns}$ and total length 4σ. The π-pulse error is $\leq 0.4\%$, characterized by measuring qubit population after applying repeated π-pulses.

Immediately following the initial state preparation, we send a 2.5 μs square-shaped readout pulse to the readout resonator. After interacting with the coupled resonator-qubit system, the readout pulse returned to the ZCU216 and is digitized by the ADC to a discrete-time series of quadrature values $\{I_n, Q_n\}$, where n labels the discrete time steps. In our QICK setup, the readout pulse length of 2.5 μs corresponds to 770 ADC clock cycles, i.e. $n = \{1, ..., 770\}$. The experiment is repeated with a cycling period of 100 μs to collect statistics. For the labeled training data, we used 500,000 shots each for qubits prepared in $|0\rangle$ and $|1\rangle$. In the standard QICK firmware, the time series can either be saved in the on-board memory for offline processing, or time-averaged in real-time and saved as average values for each readout $\{\bar{I}, \bar{Q}\}$. Additionally, the time-averaged values can be compared to a pre-configured threshold in real-time on the FPGA to yield a binary output and be used as logic input for conditional qubit control. In our current implementation of ML-based readout, the time-series $\{I_n, Q_n\}$ is streamed to the NN classifier block on the FPGA and the NN prediction outputs are saved in the programmable logic block memories (BRAM), as shown in Fig. 2.

D. Readout Fidelity from threshold methods

Figure 3(a) shows representative single-shot readout data and the time-series I/Q trajectories averaged over all training shots. The trajectories for states $|0\rangle$ and $|1\rangle$ separate within the first 100 clock cycles (approximately 325 ns), limited by the linewidth of the resonator. Beyond this initial period, the photon number in the resonator stabilizes, resulting in a near-constant separation between the average $|0\rangle$ and $|1\rangle$ trajectories.

We plot the time-averaged readout signal in the I/Q plane in Fig. 3(b), revealing two distinct Gaussian-like distributions. A simple threshold can be applied to the time-averaged I/Q signal to predict the qubit state and quantify readout fidelity. By projecting the I/Q signal along the axis connecting the average values of the $|0\rangle$ and $|1\rangle$ state distributions, we generate the histogram shown in Fig. 3(c), with the optimal threshold indicated by the gray dashed line.

The single-shot readout fidelity is defined as $\mathcal{F} \equiv 1 - \frac{1}{2}(P(0|1) + P(1|0))$. Here, $P(0|1)$ is the probability of a qubit prepared in the excited state being misclassified as the ground state, and $P(1|0)$ is the probability of the

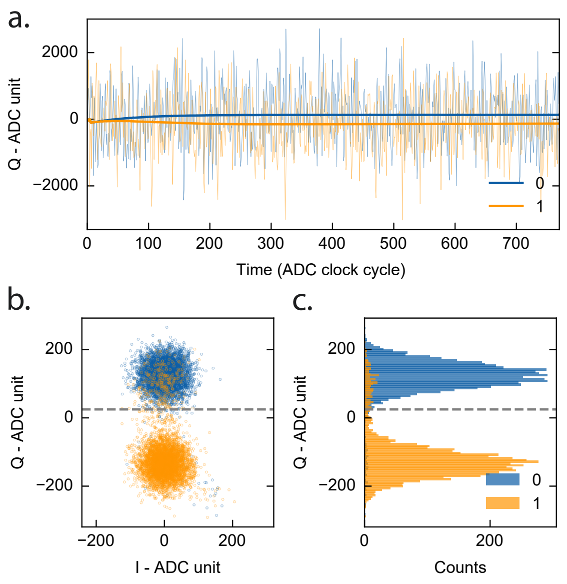

FIG. 3. Typical single shot readout signal. The readout phase is adjusted to optimize the signal in Q quadrature. (a) Single-shot readout trajectory and averaged readout trajectory. (b)(c) Time-averaged single-shot readout signal in the I-Q plane and its histogram (showing 5000 shots for better visibility). The gray dashed line represents the readout threshold for optimized readout fidelity.

reverse classification error.

Using the full readout data (770 clock cycles), the thresholding (TH) method yields a single-shot fidelity of $\mathcal{F} = 95.80 \pm 0.03\%$, with $P(0|1) \approx 6.05 \pm 0.04\%$ and $P(1|0) \approx 2.34 \pm 0.02\%$. Error bars represent the standard error of the mean, based on the training dataset size. The distinguishability of the qubit states is initially limited by the resonator bandwidth, and diminishes at longer times due to qubit relaxation. Therefore, a weighted time average of the I/Q signal can enhance performance when the threshold is applied. Using the optimal weights given by the matched-filter (MF) method [26] with the full readout window, we obtain $\mathcal{F} = 95.76 \pm 0.03\%$, comparable to the simple thresholding (TH) method.

The observed single-shot infidelity results from several factors, including residual thermal population, qubit relaxation, and signal-to-noise ratio (SNR) of the amplifier chain. Residual thermal population ($\approx 1.6\%$) contributes to errors in both $P(0|1)$ and $P(1|0)$. Additionally, qubit relaxation during the readout window (T_1 decay) accounts for $\approx 4\%$ contribution to $P(0|1)$. These effects are evident in the histogram as asymmetric tails of the Gaussian distributions. The remaining infidelity ($< 1\%$) arises from finite SNR in the readout amplifier chain and potential readout-induced qubit transitions not captured by T_1 decay. Higher readout fidelity can be achieved by employing quantum-limited amplifiers to enhance SNR [27], incorporating Purcell filters to reduce qubit relaxation and allow faster readout [28, 29], and optimizing readout pulse shaping to minimize state-changing errors and readout duration [30].

To utilize the data more efficiently and reduce the size of the neural network (NN) classifier, we analyze the readout fidelity with truncated single-shot data, limiting the readout window to 400 clock cycles starting at different locations within the readout period, as shown in Fig. 7. Using the threshold (TH) method, we achieve an optimized fidelity of $\mathcal{F} = 96.04 \pm 0.03\%$ with $P(0|1) \approx 5.56 \pm 0.04\%$ and $P(1|0) \approx 2.35 \pm 0.02\%$ when truncating the window to [100,500] clock cycles. The matched-filter (MF) method yields a comparable fidelity of $\mathcal{F} = 96.01 \pm 0.03\%$ for the same truncation window. A detailed discussion of truncation window optimization for the NN method is provided in Sec. III B.

III. NEURAL NETWORK MODEL CO-DESIGN

A. Design strategy

The primary objective of the neural network model is to facilitate efficient and accurate qubit readout within the firmware design pipeline. Recently, various neural network architectures have found direct applications with transmon qubit systems. Recurrent neural networks (RNNs), designed for sequential data, have been utilized to infer individual quantum trajectories of a superconducting qubit's evolution [31, 32]. Autoencoders are used for superconducting qubits by pretraining on qubit readout signals to extract relevant features, which are then used for enhanced classification performance in a supervised manner [33]. For direct qubit state classification, convolutional neural networks (CNNs) and multilayer perceptrons (MLPs) have been proposed due to their simplicity and ability to mitigate cross-talk effects in multi-qubit configurations [34], [35].

In line with previous comparisons between NN models [34, 35], we found that different types of neural network architectures perform similarly for the qubit classification problem, with MLPs having a slight edge in classification accuracy. Beyond accuracy, the choice of NN architecture is highly constrained by the system requirements, available FPGA resources, and strict latency constraints. Co-design of the algorithm is a Pareto optimization between algorithm fidelity and NN resources and latency. Dense MLPs, due to their straightforward design, often require fewer computational resources than more complex architectures like RNNs, CNNs, or autoencoders. Considering these factors, we selected an MLP for binary classification to distinguish between the ground and excited states of qubits. The model's simplicity ensures effective integration into the QICK system without overwhelming computational resources.

To that end, we start with a simple 2-layer MLP neural network architecture as illustrated in Fig. 4. We have explored other MLP and CNN architectures, including more and less hidden layers, and we find a 2-layer ar-

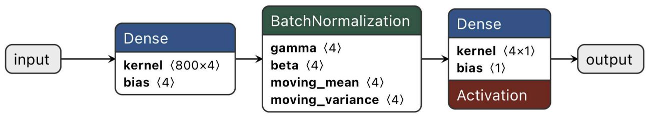

FIG. 4. Visualization of a two-layer neural network architecture with an 800-dimensional input, a dense layer with batch normalization followed by another dense layer with a sigmoid activation function.

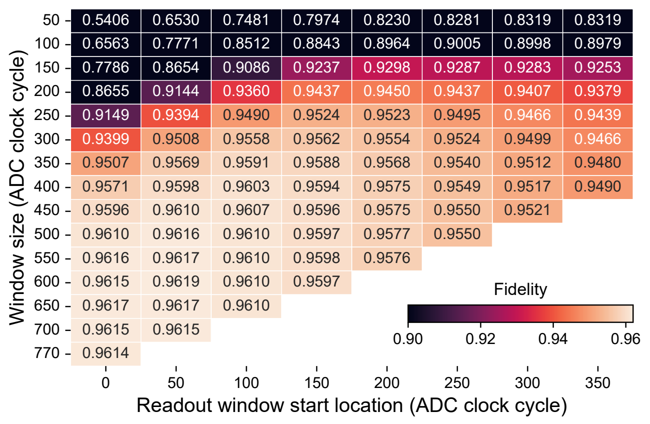

FIG. 5. Analysis of NN with four hidden neurons performance as a function of the starting time of the readout window (in clock cycles) on the x axis and the size of the readout window on the y axis. The z axis is the fidelity of the model.

chitecture is sufficient for this task. The optimal architecture may be different for other tasks such as different experimental setups or multiple qubits. To demonstrate our general co-design principles to minimize FPGA resources while maximizing fidelity performance, we focus on the following:

- Hyperparameter design space exploration: Reducing the overall network size, i.e. weights and computations, through hyperparameter design space exploration – this includes optimizing the readout window start time and size and the number of neurons in the hidden layers.
- Hardware optimization: *Quantization* of the NN through quantization-aware training (QAT) methods – the resources of the NN approximately scale quadratically with the precision of the operations and embedding quantization into the training process with QAT often yields an overall lower precision model. Further hardware optimization can be done at the implementation level by tuning the amount of parallelization of the model computations in hardware. For such low-latency applications, we generally try to “unroll” (parallelize) the computations as much as is allowed while balancing FPGA resources.

The model consists of dense and batch normalization layers. A batch normalization layer is valuable for such an architecture given the bit width of inputs and the use of fixed-point calculations in quantized neural networks to optimize FPGA resources – described in more detail below. Among other benefits of batch normalization, scaling the logits prevents computational overflows. We utilized the Adam optimizer [36] and a binary cross-entropy loss function for optimization, and processed the network’s output with a sigmoid function to produce a probability distribution over the classes. We employed gradient descent with a learning rate of 10^{-4}. Each training iteration was completed in approximately 29 seconds on NVIDIA A100 GPU.

To design the NN classifier, we adopt `hls4ml` [17, 18], an open-source software framework that bridges the gap between high-level machine learning models and low-level hardware implementation. `hls4ml` converts machine learning algorithms, especially neural networks, from frameworks like TensorFlow or PyTorch into hardware descriptions in C++ for high-level synthesis (HLS) tools such as AMD/Xilinx Vivado HLS [37]. In particular, we will use the QKeras [38] front-end that interfaces with `hls4ml` in order to perform QAT. The `hls4ml` tool generates a dataflow architecture on FPGA, which is well-suited for neural network computations’ parallel and pipelined nature. In this architecture, each neural network layer can be implemented as a separate hardware module, and data moves sequentially from one module (layer) to the next with minimum buffering. This allows for continuous data processing and minimizes latency.

B. Hyperparameter Design Space Exploration

Figure 5 shows the fidelity of an optimized neural network with four hidden neurons (hn) using varying readout window sizes and starting locations. The smallest window size is 50 clock cycles (CLK) and increments by 50 CLKs up to 700 CLKs, followed by an evaluation on the full readout window. This process is repeated for starting locations, beginning at 0 CLK and increasing in 50 CLK increments up to 350. The results indicate that starting the readout window later generally improves performance, especially with smaller window sizes. However, this improvement plateaus at around 100-150 CLKs. Similarly, larger windows lead to better performance, though the benefits diminish beyond approximately 400 CLKs. Based on these findings, we focus on a starting location of 100 CLKs, as it provides high fidelity with the smallest effective window size (400 CLKs). We repeated this design space exploration for the second NN version with 64 hidden neurons, as well as for thresholding and matched filtering methods, observing similar trends as shown in Figure 5.

Figure 6 presents the design space exploration of both 2-layer NN variants, comparing their performance based on the number of trainable parameters and test fidelity

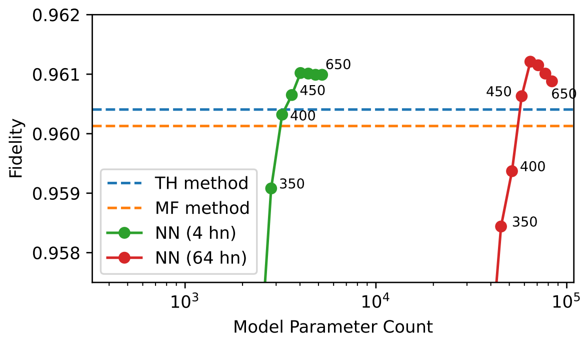

FIG. 6. Model size (in parameter count) versus fidelity. Each point on the graph corresponds to a different window size ranging from 350 to 650 clock cycles.

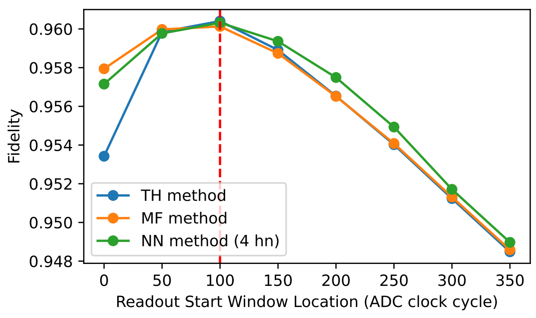

FIG. 7. Optimal selected neural network hyperparameters (red line) with 400 clock cycle readout window starting at 100 clock cylces compared against the threshold and match filter methods.

across different readout window sizes. All results are based on data starting at 100 CLKs. The exploration includes window sizes starting from 350 CLKs and increasing in 50 CLK increments, up to 650 CLKs. As seen in Figure 5, the most critical features for state discrimination appear after the first 100 ADC clock cycles. Therefore, we exclude the first 100 ADC units in our exploration. The graph indicates that test accuracy improves with the number of parameters, reaching a peak at a readout window size of 400 CLKs, beyond which additional parameters provide minimal improvement. Both NN variants show similar performance within this range, demonstrating that smaller NNs can achieve the same fidelity as larger architectures, making them more efficient. This may suggest that the smaller NN is less prone to overparameterization and overfitting, making it a more efficient choice for state discrimination and FPGA integration.

Figure 7 illustrates how fidelity varies as we change the starting position of the 400-CLK readout window of the 2-layer NN with 4 hidden neurons, thresholding, and match filtering methods. All methods show a similar trend: fidelity initially increases with the readout start window location, peaks around the 100–150 ADC clock cycle mark, and then gradually declines. This is consistent with the time it takes to populate the readout resonator, seen previously in the average trajectory data in Fig. 3. When the readout window starts too late, the decrease in fidelity can be associated with qubit T_1 decay. The NN method generally has a slightly higher fidelity than the TH and MF methods across most locations. Both methods achieve the highest fidelity when the readout window begins around 100 ADC CLKs. Moving forward, we will focus on a 400 CLK readout window starting at 100 CLKs.

C. Hardware optimization

A co-design approach is taken to optimize neural networks for FPGA implementation. Typically, all parameters and data are represented using fixed-point precision, as opposed to the standard 32-bit floating-point notation used during the training and evaluation phases. For FPGA deployment, all data is quantized to fixed-point, which results in some level of performance degradation compared to floating-point notation. To address this, we employ quantization aware training (QAT). This approach helps improve the accuracy of the quantized models on FPGAs by simulating quantization during the training process [39, 40]. Neural networks learn to operate within the constraints of lower precision (e.g., 8-bit integers). The benefits of QAT are two-fold: it minimizes the accuracy loss due to quantization and significantly reduces model size, making it easier to deploy on devices with limited memory and hardware resources. Quantized models can achieve faster inference times due to the reduced computational complexity of arithmetic operations, which are crucial for real-time processing applications. All parameters and activations are quantized using uniform symmetric quantization. Data from the ADC units are fed directly to the network as 14-bit unsigned integers.

In future work, normalizing the ADC data should be studied for robustness against drifts in the readout signal. We explore different quantization schemes to evaluate the impact of precision on fidelity. The two NN models we tested have $800\times64\times1$ and $800\times4\times1$ parameters, and we explore their performance when using full precision (32-bit floating-point) and reduced precision, including 6-bit fixed-point, 3-bit fixed-point, and ternary (2-bit) representations. Both models show stable performance using 32-bit floating point (32FP). 6-bit and 3-bit quantization shows a slight drop compared to 32FP, but remains close. Ternary quantization performance, restricted to -1, 0, or 1, is similar to 6-bit and 3-bit performance for this single quibit classification task. All quantization schemes followed the same pattern as 32FP, peaking at 100 CLKs around 96% fidelity, then declining at the same rate. We opt to proceed with the $800 \times 4 \times 1$ NN using ternary weights to minimize both hardware footprint anc latency.

With a resource-optimized neural network implementation, we can fully parallelize the hardware implementation by unrolling the matrix multiplications. This enable us to minimize the computational latency of the ternary neural network. We can tune the FPGA resources of the neural network in `hls4ml` by configuring this parallelization factor, but we choose to fully unroll the hardware implementation for the lowest latency. After synthesis we find the hardware implementation takes 10 clock cycles in total – 8 clock cycles for the computation itsel and 2 clock cycles to store the results in BRAM. Witl a 3.22 ns clock cycle, the total latency is 32 ns. The 8 clock cycles is driven by the multiplication latency anc the number of hidden layers in the neural network.

IV. MODEL INTEGRATION: QICK+hls4ml

A. QICK firmware

The QICK system adopts a software-based approach in which users can access the system remotely using Jupyter notebooks. QICK includes, in addition to software applications, *Processing System* (PS) and *Programmable Logic* (PL), as shown in Fig. 2. The PS in the AMD/Xilinx UltraScale+ device integrates a Zynq system and DDR4 memory, while Linux OS runs on the multicore ARM processor. QICK uses PYNQ software libraries and drivers to simplify the software-firmware interaction and, in particular, provides the PL with an easy solution for direct memory access (DMA). The firmware on the PL integrates *Readout* and *Signal Generator* blocks, which are controlled by the timed processor (*tProc*). The *tProc* implements custom instructions to produce, for example, pulses to control and readout qubits via the Signal Generator blocks. In QICK, data flows between firmware components and software applications through the *AXI Interconnect*, which is also the backbone for integrating our NN classifier IP.

B. Neural network classifier integration

Our NN classifier IP follows the loosely coupled model for hardware accelerators [41]. A tightly coupled accelerator would be designed and integrated closely with the *tProc*, increasing its complexity, sharing its caches, and possibly stalling the computation. Meanwhile, loosely coupled accelerators can be designed separately, easily maintained, and independently integrated. They operate on larger data sets and alternate coarse-grain computation with data transfer phases. In QICK, we integrated the IP as a device managed with Linux device drivers running on the processing system's ARM cores. To interface the classifier with the rest of the QICK system, we encapsulated the `hls4ml`-generated NN implementation

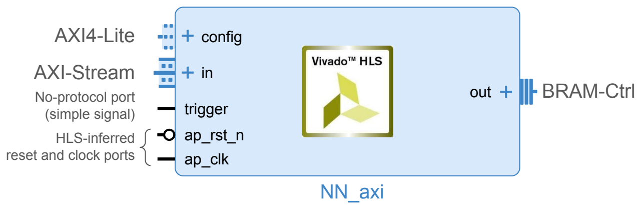

FIG. 8. The interface of the NN classifier as a Vivado IP integrated into the QICK firmware. AXI interfaces configure and stream readout data; a BRAM interface connects the IP with an external buffer on the programmable logic.

C. IP interface

Figure 8 shows the interface of the top-level wrapper (`NN_axi`) and NN classifier synthesized with AMD/Xilinx Vivado HLS as an IP for the integration in the QICK Vivado project. Four main ports are explicitly defined in the wrapper specifications for HLS:

- Port `config` receives the configuration information from the software via memory-mapped registers, which are bundled together in the AXI4-Lite protocol. For example, the user can configure the readout window offset and scaling factor at runtime.

- Port `in` receives input data from the QICK *Readout* block as a discrete-time series of quadrature values (I_n, Q_n). The series is defined as a 32-bit wide stream and uses the AXI-Stream (AXIS) protocol with no `TREADY` or `TLAST` side-channel signals. Each 32-bit word packs two 14-bit I and Q values with four bits of zero padding. Typically, with an AXIS protocol, the `TREADY` signal allows the receiver to control the pace of the data transfer, indicating to the transmitter when it is ready to accept more data (i.e., backpressure). In QICK, the lack of backpressure towards the *Readout* block means that the receiver must always be ready to accept data.

- Port `out` transfers the classifier output logit values to a memory buffer on the programmable logic block memories (BRAM). The buffer size is 128KB, which allows the storage of 16,384 consecutive predictions. The BRAMs are mapped to the main memory via AXI4-Lite and function as high-speed, local memory within the PL, allowing the NN to save the prediction in a buffer with minimum delay. The AXI4-Lite interface provides a communication link between the software running on the

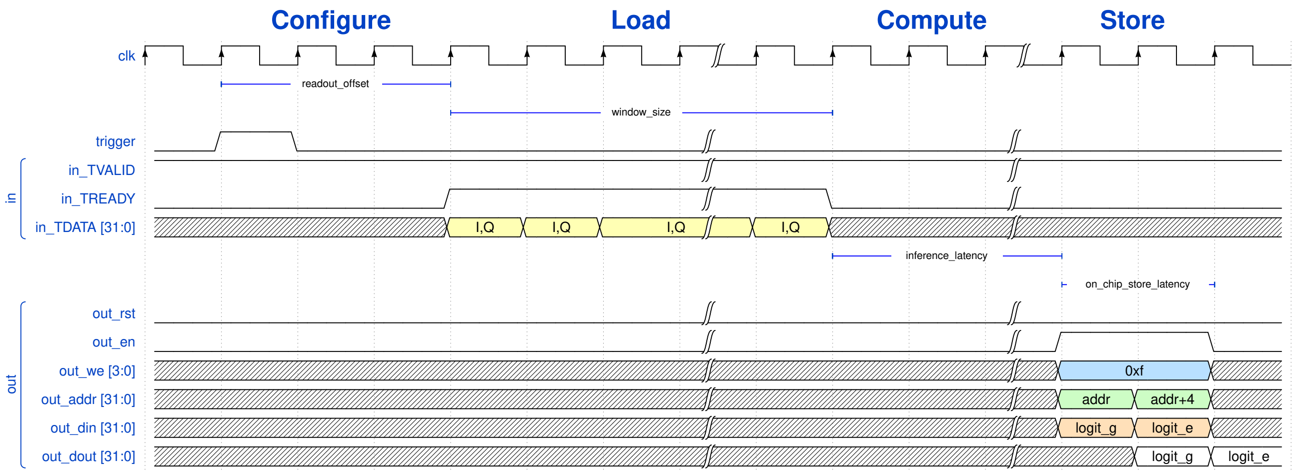

FIG. 9. Waveforms at the classifier IP interface for a single readout.

RFSoC processor and the BRAMs, enabling access to the prediction results at a lower throughput. The BRAMs on the PL and their corresponding regions in the main memory operate independently, with the PL logic directly managing the data in the BRAMs, while the AXI4-Lite interface allows the processor to interact with this data in a controlled manner following a loosely coupled paradigm.

- Port `trigger` receives the trigger signal from the *tProc* that controls the classifier execution. No HLS-generated protocol is specified for this port, and the associated signal is used directly in the control logic of the IP.

Vivado HLS automatically infers additional ports for clock and reset signals and adds them to the final IP interface, as shown in Figure 8.

D. Behavior and Timing

The classifier IP execution has four phases: configuration, data loading, NN computation, and storage of the prediction results. First, the software application configures the accelerator via memory-mapped registers. Readout data is then streamed over an AXI-stream interface during the loading phase and transferred to the IP's private local memory. The NN prediction results are saved locally to BRAM in the PL and finally transferred to the system's main memory through an AXI4-Lite interface.

Figure 9 shows the behavior of the classifier as waveforms at the IP interface for a single readout trace. The clock period is 3.25 ns. After a pulse of the `trigger` signal, the classifier loads the input data from the `in` channel, which has a data lane (`in_TDATA`) that is 32-bits wide to pack two 14-bit I and Q values. Data constantly stream from the ADC to the readout block and into the classifier; thus the valid signal (`in_TVALID`) is mostly high. Since the classifier IP does not push back, the ready signal (`in_TREADY`) is only shown as a reference. The user can configure an additional delay (the `readout_offset`) counted in clock cycles from the rising edge of the trigger pulse. The duration of the load phase in clock cycles is the `window_size` of the trained neural network. The `inference_latency` depends on the complexity of the neural network, and varies between 5 and 20 clock cycles for the different NN models we tested. Finally, the IP takes two more clock cycles (`on_chip_store_latency`) to store the inference results locally in the programmable logic in BRAMs through a simple memory interface `out`.

E. Software API and Usage

We defined a comprehensive Python API that runs on the RFSoC processor and interacts with the NN classifier integrated with the rest of QICK system on the programmable logic. These functions enable users to reset, configure, and retrieve predictions from the classifier, as well as debug and inspect the classifier's internal state.

Once the bitstream is loaded on the FPGA, the first function to be called is `reset_classifier`. This function ensures that the classifier is in a clean state by resetting its configuration and state registers. Depending on the needs of the application, a deep reset may be performed to zero out specific memory locations, providing a fresh start for subsequent operations. The parameters `index_lo` and `index_hi` allow for selective resetting of a range of memory indices.

After resetting, the classifier is configured using the `configure_classifier` function. This step involves setting parameters such as the `readout_offset` or the `scaling_factor`, which define the operational characteristics of the classifier. Additionally, the `debug` parameter can be set to true to enable detailed logging of the configuration process, aiding in debugging and fine-tuning.

After the initial state preparation, we use a QICK program that runs on the *tProc* and sends a square-shaped

	Memory Resources		Computational Resources	
	FF	BRAM	LUT	DSP
ZCU216	850,560	1,080	425,280	4,272
QICK	89,783 (10.56%)	309 (28.61%)	60,057 (14.12%)	481 (11.26%)
QICK+NN	124,152 (14.60%)	341 (31.57%)	126,726 (29.80%)	481 (11.26%)
NN	+4.04%	+2.96%	+15.60%	+0.00%

TABLE I. Overall FPGA resources available on Zynq UltraScale+ RFSoC ZCU216 and resource utilization for QICK and its ML-enhanced version. The memory resources are flip-flops (FF) and block RAMs (BRAM); the computational resources are look-up tables (LUT) and data-signal processors (DSP).

readout pulse to the readout resonator. The ADC digitizes the returned readout signal to a discrete-time series of quadrature values I_n, Q_n that feeds our classifier. At this point, to monitor the classifier's performance, the `get_classifier_prediction_count` function returns the total number of predictions made by the classifier. This count starts at zero upon initialization or after a reset, providing a clear indication of how many pulses have been sent and predictions have been processed.

The `get_classifier_prediction` functions allow users to retrieve the classifier's output for a specific index or a range of indices. These outputs are returned as tuples containing logits for the ground and excited states, which can be further analyzed or used in subsequent processing stages.

Finally, to inspect the classifier's internal state, the `print_classifier_buffer` function prints the contents of the classifier's buffer for a specified range of indices. This is particularly useful for verifying that the classifier has processed the data correctly and can be an essential tool for debugging.

F. Performance and Implementation

In summary, our final implementation of the NN model performs single-qubit state discrimination with a readout pulse length of 500 clock cycles (1.6 μs). Using an ADC offset of 100 clock cycles, the last 400 clock cycles of the readout signal are streamed to the NN block. The NN takes 8 clock cycles to perform the inference and an additional 2 clock cycles to store the inference result in a memory buffer, corresponding to a total latency of 10 clock cycles (32 ns) following the end of the readout window.

With the NN integrated readout, we performed 500,000 single-shot readout experiments for both the ground- and excited-state readouts. To directly assess the NN discriminator's performance, we record both the NN predicted outputs and the raw time-series I/Q data for each single-shot experiment. The NN state discriminator resulted in a readout fidelity of $\mathcal{F} = 96\%$, comparable to the performance of the simple thresholding or matched filter methods over the same readout window. For a single-qubit system, we expect similar performance and will now be able to deploy this end-to-end flow in more complex and dynamic systems – discussed in more detail in Section V.

Table I shows the resource requirements for implementing the model in FPGA. Over the columns, we report the memory resources as flip-flops (FF), basic memory elements used to store binary data, and Block RAMs, dedicated memory blocks that can store up to 36 kilobits on the programmable logic; the computational resources are look-up tables (LUT), configurable logic blocks used for implementing logic functions, and digital signal processors (DSPs), specialized hardware units for efficient computation of operations like multiplications and additions. The row denoted by ZCU216 shows the overall resources available on the chip of our target development board (Zynq UltraScale+ RFSoC ZCU216). The second row shows the resources required by the QICK platform, while the third row details the combined resources for the QICK plus the NN IP; in both cases, we report the absolute value and the percentage of usage for each resource (in parentheses). Finally, the last row indicates the overhead of the NN IP alone. The NN IP requires additional resources of 4.04% for FFs, 2.96% for BRAMs, and 15.6% for LUTs. Finally, in terms of performance, once the readout data has been loaded, the algorithm latency is ten clock cycles: eight cycles for the NN inference and two cycles to store the results in the external BRAMs.

V. SUMMARY AND OUTLOOK

This study introduces a comprehensive workflow for enhancing the readout of superconducting qubits by incorporating co-designed NN into the QICK hardware platform. Using `hls4ml` to efficiently co-design NNs on programmable logic, the workflow addresses critical challenges such as improving accuracy, reducing latency, and preserving quantum states during readout. The NN algorithm is optimized using ternary quantization and parallelization methods to run with a low latency of 32 ns following the qubit readout process, consuming less than 16% of the FPGA look-up table resources and less for other resources. Performance evaluations show that this approach achieves fidelity comparable to conventional techniques like thresholding and matched filtering for single qubit readout. This open-source framework demonstrates the feasibility of NN-based readout for superconducting qubits. It provides a valuable tool for researchers to explore innovative quantum computing methodologies that integrate machine learning with high-level synthesis for efficient hardware deployment.

There are increasingly more studies on embedded, real-time, ML-based qubit readout such as in Ref [12, 42, 43]. However, the experiments from which the data come and, thus, the algorithmic approaches, optimization, and implementation vary. To make direct comparisons of different approaches and new methods as they are developed, it is valuable, as a community, to have publicly available benchmarks, including common datasets and reproducible results. To that end, we have made our dataset available on Zenodo [44] and the code to reproduce the algorithm training and implementation available at Ref [21]. We hope that this will be useful for future studies and encourage more datasets and benchmarks to be made available as systems grow in complexity.

One near-term upcoming for our platform for user experimentation is to integrate output of the NN block into the conditional logic of QICK to run readout experiments including real-time ML feedback control. This will also be updated and included in our publicly available code repository.

There are several directions of exploration that follow-on directly from this work towards realizing the ultimate goal of an adaptive and continuously optimized readout system. For example, model-based readout [45] and reinforcement learning methods [46] are promising for continuous and autonomous qubit readout. This could be integrated directly with the QICK and `hls4ml` platforms. While our demonstration algorithm uses a straightforward dense neural network architecture, there is significant potential for developing more performant and resource-optimized algorithms in hardware using additional codesign methods such as pruning [20] or other efficient architectures amenable to time-series data [47]. Relatedly, developing robust algorithms to changing instrument conditions [48] can also aid in improved continuous learning. These directions of exploration are especially valuable as readout systems grow in complexity for multi-qubit systems, and ML approaches are already proving to be powerful [12–15].

In the longer term, we plan to deploy the control and readout logic in the cryostat as a more scalable solution for quantum computers with many thousands of qubits. Placing an SoC that integrates programmable logic as embedded FPGAs in a cryostat for quantum readout is motivated by the need to minimize thermal noise, enhance signal integrity, and reduce latency. The close proximity of the logic to the qubits within the cryostat minimizes signal loss and noise introduction, leading to a more accurate and reliable quantum readout. Additionally, the reduced latency in signal processing is essential for real-time quantum error correction and feedback, where even minor delays can impact system performance. Integrating classical control hardware with quantum hardware in the same cryogenic environment also simplifies system design, improves efficiency, and supports scalability as quantum computing systems become more complex [49, 50].

ACKNOWLEDGEMENTS

We would like to thank Sho Uemura, Leandro Stefanazzi, and Gustavo Cancelo of the QICK development team for their technical assistance.

This document was prepared using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, Office of High Energy Physics HEP User Facility. Fermilab is managed by Fermi Forward Discovery Group, LLC, acting under Contract No. 89243024CSC000002.

G.N.P. was supported for this work by the Department of Energy (DOE), Office of Science, Office of High Energy Physics, QuantISED program grant "HEP Machine Learning and Optimization Go Quantum," identification number 0000240323. N.T. and J.C. are supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under the "Real-time Data Reduction Codesign at the Extreme Edge for Science" Project (DE-FOA-0002501). G.D.G. is supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the Department of Energy (DOE), Office of Science, Office of High Energy Physics. B.D., S.L., R.M., and D.B. are supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Quantum Science Center.

[1] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, Superconducting qubits: Current state of play, Annual Review of Condensed Matter Physics **11**, 369–395 (2020).

[2] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. van den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, and A. Kandala, Evidence for the utility of quantum computing before fault tolerance, Nature **618**, 500 (2023).

[3] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, Nature 10.1038/s41586-024-08449-y (2024).

[4] D. Gao, D. Fan, C. Zha, J. Bei, G. Cai, J. Cai, S. Cao, X. Zeng, F. Chen, J. Chen, *et al.*, Establishing a new benchmark in quantum computational advantage with 105-qubit zuchongzhi 3.0 processor (2024), arXiv:2412.11924 [quant-ph].

[5] L. Stefanazzi, K. Treptow, N. Wilcer, C. Stoughton, C. Bradford, S. Uemura, S. Zorzetti, S. Montella, G. Cancelo, S. Sussman, A. Houck, S. Saxena, H. Arnaldi, A. Agrawal, H. Zhang, C. Ding, and D. I. Schuster, The QICK (quantum instrumentation control kit): Readout and control for qubits and detectors, Rev. Sci. Instrum. **93**, 044709 (2022).

[6] Y. Xu, G. Huang, J. Balewski, R. Naik, A. Morvan, B. Mitchell, K. Nowrouzi, D. I. Santiago, and I. Siddiqi, Qubic: An open-source fpga-based control and measurement system for superconducting quantum information processors, IEEE Transactions on Quantum Engineering **2**, 1 (2021).

[7] M. O. Tholén, R. Borgani, G. R. Di Carlo, A. Bengtsson, C. Križan, M. Kudra, G. Tancredi, J. Bylander, P. Delsing, S. Gasparinetti, and D. B. Haviland, Measurement and control of a superconducting quantum processor with a fully integrated radio-frequency system on a chip, Rev. Sci. Instrum. **93**, 104711 (2022).

[8] K. H. Park, Y. S. Yap, Y. P. Tan, C. Hufnagel, L. H. Nguyen, K. H. Lau, P. Bore, S. Efthymiou, S. Carrazza, R. P. Budoyo, and R. Dumke, ICARUS-Q: Integrated control and readout unit for scalable quantum processors, Rev. Sci. Instrum. **93**, 104704 (2022).

[9] C. Ding, M. Di Federico, M. Hatridge, A. Houck, S. Leger, J. Martinez, C. Miao, D. S. I, L. Stefanazzi, C. Stoughton, S. Sussman, K. Treptow, S. Uemura, N. Wilcer, H. Zhang, C. Zhou, and G. Cancelo, Experimental advances with the QICK (quantum instrumentation control kit) for superconducting quantum hardware, Phys. Rev. Res. **6**, 013305 (2024).

[10] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, and A. Vaschillo, Assessing requirements to scale to practical quantum advantage (2022), arXiv:2211.07629 [quant-ph].

[11] M. Mohseni, A. Scherer, K. G. Johnson, O. Wertheim, M. Otten, N. A. Aadit, K. M. Bresniker, K. Y. Camsari, B. Chapman, S. Chatterjee, G. A. Dagnew, A. Esposito, F. Fahim, M. Fiorentino, A. Khalid, X. Kong, B. Kulchytskyy, R. Li, P. A. Lott, I. L. Markov, R. F. McDermott, G. Pedretti, A. Gajjar, A. Silva, J. Sorebo, P. Spentzouris, Z. Steiner, B. Torosov, D. Venturelli, R. J. Visser, Z. Webb, X. Zhan, Y. Cohen, P. Ronagh, A. Ho, R. G. Beausoleil, and J. M. Martinis, How to build a quantum supercomputer: Scaling challenges and opportunities (2024), arXiv:2411.10406 [quant-ph].

[12] P. K. Gautam, S. Kalipatnapu, S. H, U. Singhal, B. Lienhard, V. Singh, and C. S. Thakur, Low-latency machine learning fpga accelerator for multi-qubit-state discrimination (2024), arXiv:2407.03852 [quant-ph].

[13] P. Duan, Z. F. Chen, Q. Zhou, W. C. Kong, H. F. Zhang, and G. P. Guo, Mitigating crosstalk-induced qubit readout error with shallow-neural-network discrimination, Physical Review Applied **16**, 1 (2021).

[14] E. Magesan, J. M. Gambetta, A. D. Córcoles, and J. M. Chow, Machine learning for discriminating quantum measurement trajectories and improving readout, Phys. Rev. Lett. **114**, 200501 (2015).

[15] S. Maurya, C. N. Mude, W. D. Oliver, B. Lienhard, and S. Tannu, Scaling qubit readout with hardware efficient machine learning architectures, in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23 (Association for Computing Machinery, New York, NY, USA, 2023).

[16] N. R. Vora, Y. Xu, A. Hashim, N. Fruitwala, H. N. Nguyen, H. Liao, J. Balewski, A. Rajagopala, K. Nowrouzi, Q. Ji, K. B. Whaley, I. Siddiqi, P. Nguyen, and G. Huang, Ml-powered fpga-based real-time quantum state discrimination enabling mid-circuit measurements (2024), arXiv:2406.18807 [quant-ph].

[17] J. Duarte *et al.*, Fast inference of deep neural networks in FPGAs for particle physics, JINST **13** (07), P07027, arXiv:1804.06913 [physics.ins-det].

[18] FastML Team, fastmachinelearning/hls4ml (2023).

[19] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference (2017), arXiv:1712.05877 [cs.LG].

[20] H. Cheng, M. Zhang, and J. Q. Shi, A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations, IEEE Transactions on Pattern Analysis and Machine Intelligence **46**, 10558 (2024).

[21] ML Quantum Readout, `https://github.com/fastmachinelearning/ml-quantum-readout` (2024).

[22] J. Koch, T. M. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, Charge-insensitive qubit design derived from the cooper pair box, Phys. Rev. A **76**, 042319 (2007).

[23] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, A quantum engineer's guide to superconducting qubits, Appl. Phys. Rev. **6**, 021318 (2019).

[24] Y. Y. Gao, M. A. Rol, S. Touzard, and C. Wang, Practical guide for building superconducting quantum devices, PRX Quantum **2**, 040202 (2021).

[25] B. Du, R. Suresh, S. López, J. Cadiente, and R. Ma, Probing site-resolved current in strongly interacting superconducting circuit lattices, Phys. Rev. Lett. **133**, 060601 (2024).

[26] C. A. Ryan, B. R. Johnson, J. M. Gambetta, J. M. Chow, M. P. da Silva, O. E. Dial, and T. A. Ohki, Tomography via correlation of noisy measurement records, Phys. Rev. A **91**, 022118 (2015).

[27] C. Macklin, K. O'Brien, D. Hover, M. E. Schwartz, V. Bolkhovsky, X. Zhang, W. D. Oliver, and I. Siddiqi, A near–quantum-limited josephson traveling-wave parametric amplifier, Science **350**, 307 (2015).

[28] E. Jeffrey, D. Sank, J. Y. Mutus, T. C. White, J. Kelly, R. Barends, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, A. Megrant, P. J. J. O'Malley, C. Neill, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and J. M. Martinis, Fast accurate state measurement with superconducting qubits, Phys. Rev. Lett. **112**, 190504 (2014).

[29] J. Heinsoo, C. K. Andersen, A. Remm, S. Krinner, T. Walter, Y. Salathé, S. Gasparinetti, J.-C. Besse, A. Potočnik, A. Wallraff, and C. Eichler, Rapid high-fidelity multiplexed readout of superconducting qubits, Phys. Rev. Appl. **10**, 034040 (2018).

[30] D. T. McClure, H. Paik, L. S. Bishop, M. Steffen, J. M. Chow, and J. M. Gambetta, Rapid driven reset of a qubit readout resonator, Phys. Rev. Appl. **5**, 011001 (2016).

[31] E. Flurin, L. S. Martin, S. Hacohen-Gourgy, and I. Siddiqi, Using a recurrent neural network to reconstruct quantum dynamics of a superconducting qubit from physical observations, Phys. Rev. X **10**, 011006 (2020).

[32] G. Koolstra, N. Stevenson, S. Barzili, L. Burns, K. Siva, S. Greenfield, W. Livingston, A. Hashim, R. K. Naik, J. M. Kreikebaum, K. P. O'Brien, D. I. Santiago, J. Dressel, and I. Siddiqi, Monitoring fast superconducting qubit dynamics using a neural network, Phys. Rev. X **12**, 031017 (2022).

[33] P. Luchi, P. E. Trevisanutto, A. Roggero, J. L. DuBois, Y. J. Rosen, F. Turro, V. Amitrano, and F. Pederiva, Enhancing qubit readout with autoencoders, Phys. Rev. Appl. **20**, 014045 (2023).

[34] B. Lienhard, A. Vepsäläinen, L. C. Govia, C. R. Hoffer, J. Y. Qiu, D. Ristè, M. Ware, D. Kim, R. Winik, A. Melville, B. Niedzielski, J. Yoder, G. J. Ribeill, T. A. Ohki, H. K. Krovi, T. P. Orlando, S. Gustavsson, and W. D. Oliver, Deep-neural-network discrimination of multiplexed superconducting-qubit states, Phys. Rev. Appl. **17**, 014024 (2022).

[35] R. Navarathna, T. Jones, T. Moghaddam, A. Kulikov, R. Beriwal, M. Jerger, P. Pakkiam, and A. Fedorov, Neural networks for on-the-fly single-shot state classification, Applied Physics Letters **119**, 114003 (2021).

[36] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017), arXiv:1412.6980 [cs.LG].

[37] Xilinx, Vivado high-level synthesis (2024).

[38] Google, Qkeras (2020).

[39] J. Chen, Y. Gai, Z. Yao, M. W. Mahoney, and J. E. Gonzalez, A statistical framework for low-bitwidth training of deep neural networks, in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20 (Curran Associates Inc., Red Hook, NY, USA, 2020).

[40] J. Chen, L. Zheng, Z. Yao, D. Wang, I. Stoica, M. W. Mahoney, and J. Gonzalez, Actnn: Reducing training memory footprint via 2-bit activation compressed training, in *Proc. 38th Int. Conf. on Mach. Learn.* (2021) pp. 1803–1813.

[41] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni, An analysis of accelerator coupling in heterogeneous architectures, in *ACM/EDAC/IEEE Design Automation Conference (DAC)* (2015) pp. 1–6.

[42] B. Lienhard, A. Vepsäläinen, L. C. Govia, C. R. Hoffer, J. Y. Qiu, D. Ristè, M. Ware, D. Kim, R. Winik, A. Melville, B. Niedzielski, J. Yoder, G. J. Ribeill, T. A. Ohki, H. K. Krovi, T. P. Orlando, S. Gustavsson, and W. D. Oliver, Deep-neural-network discrimination of multiplexed superconducting-qubit states, Phys. Rev. Appl. **17**, 014024 (2022).

[43] N. R. Vora, Y. Xu, A. Hasim, N. Fruitwala, N. Nguyen, H. Liao, J. Balewski, A. Rajagopala, K. Nowrouzi, Q. Ji, K. B. Whaley, I. Siddiqi, P. Nguyen, and G. Huang, QubiCML: ML-Powered Real-Time Quantum State Discrimination Enabling Mid-Circuit Measurements , in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE Computer Society, 2024) pp. 414–415.

[44] Data for "End-to-end workflow for machine learning-based qubit readout with QICK and hls4ml" (2024).

[45] A. Bengtsson, A. Opremcak, M. Khezri, D. Sank, A. Bourassa, K. J. Satzinger, S. Hong, C. Erickson, B. J. Lester, K. C. Miao, A. N. Korotkov, J. Kelly, Z. Chen, and P. V. Klimov, Model-based optimization of superconducting qubit readout, Phys. Rev. Lett. **132**, 100603 (2024).

[46] A. Chatterjee, J. Schwinger, and Y. Y. Gao, Demonstration of enhanced qubit readout via reinforcement learning (2024), arXiv:2412.04053 [quant-ph].

[47] A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré, How to train your hippo: State space models with generalized orthogonal basis projections (2022), arXiv:2206.12037 [cs.LG].

[48] T. Baldi, J. Campos, B. Hawks, J. Ngadiuba, N. Tran, D. Diaz, J. Duarte, R. Kastner, A. Meza, M. Quinnan, O. Weng, C. Geniesse, A. Gholami, M. W. Mahoney, V. Loncar, P. Harris, J. Agar, and S. Qin, Reliable edge machine learning hardware for scientific applications, in *2024 IEEE 42nd VLSI Test Symposium (VTS)* (2024) pp. 1–5.

[49] S. Chakraborty, D. J. Frank, K. Tien, P. Rosno, M. Yeck, J. A. Glick, R. Robertazzi, R. Richetta, J. F. Bulzacchelli, D. Underwood, D. Ramirez, D. Yilma, A. Davies, R. V. Joshi, S. D. Chambers, S. Lekuch, K. Inoue, D. Wisnieff, C. W. Baks, D. S. Bethune, J. Timmerwilke, T. Fox, P. Song, B. R. Johnson, B. P. Gaucher, and D. J. Friedman, A cryo-CMOS low-power semi-autonomous transmon qubit state controller in 14-nm FinFET technology, IEEE Journal of Solid-State Circuits **57**, 3258 (2022).

[50] D. J. Frank, S. Chakraborty, K. Tien, P. Rosno, T. Fox, M. Yeck, J. A. Glick, R. Robertazzi, R. Richetta, J. F. Bulzacchelli, D. Ramirez, D. Yilma, A. Davies, R. V. Joshi, S. D. Chambers, S. Lekuch, K. Inoue, D. Underwood, D. Wisnieff, C. Baks, D. Bethune, J. Timmerwilke, B. R. Johnson, B. P. Gaucher, and D. J. Friedman, A cryo-CMOS low-power semi-autonomous qubit state controller in 14nm FinFET technology, in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65 (2022) pp. 360–362.