RESEARCH ARTICLE | MAY 12 2025

Machine learning enhanced quantum state tomography on a field-programmable gate array

Hsun-Chung Wu ; Hsien-Yi Hsieh ; Zhi-Kai Xu ; Hua Li Chen ; Zi-Hao Shi ; Po-Han Wang ; Popo Yang ; Ole Steuernagel ; Te-Hwei Suen; Chien-Ming Wu ; Ray-Kuang Lee

APL Quantum 2, 026117 (2025)
https://doi.org/10.1063/5.0262942

Machine learning enhanced quantum state tomography on a field-programmable gate array

Cite as: APL Quantum 2, 026117 (2025); doi: 10.1063/5.0262942
Submitted: 3 February 2025 • Accepted: 20 April 2025 •
Published Online: 12 May 2025

Hsun-Chung Wu,[1] Hsien-Yi Hsieh,[1] Zhi-Kai Xu,[2] Hua Li Chen,[3] Zi-Hao Shi,[1] Po-Han Wang,[1] Popo Yang,[1] Ole Steuernagel,[1] Te-Hwei Suen,[1] Chien-Ming Wu,[1] and Ray-Kuang Lee[1,2,3,4,5,a)]

AFFILIATIONS

[1] Institute of Photonics Technologies, National Tsing Hua University, Hsinchu 30013, Taiwan
[2] Department of Electrical Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan
[3] Department of Physics, National Tsing Hua University, Hsinchu 30013, Taiwan
[4] Center for Theory and Computation, National Tsing Hua University, Hsinchu 30013, Taiwan
[5] Center for Quantum Science and Technology, Hsinchu 30013, Taiwan

[a)] **Author to whom correspondence should be addressed:** rklee@ee.nthu.edu.tw

ABSTRACT

Machine learning techniques have opened new avenues for real-time quantum state tomography (QST). In this work, we demonstrate the deployment of machine learning-based QST on edge devices, specifically utilizing field-programmable gate arrays (FPGAs). Our implementation uses the *Vitis AI Integrated Development Environment* provided by AMD® Inc. Compared to graphics processing unit-based machine learning QST, our FPGA-based approach reduces the average inference time by an order of magnitude, from 38 to 2.94 ms, but only suffers an average fidelity reduction by about 1% (from 0.99 to 0.98). This FPGA-based QST system offers a highly efficient and precise tool for diagnosing quantum states, marking a significant advancement in the practical applications for quantum information processing and quantum sensing.

I. INTRODUCTION

Accurately characterizing the full information in a quantum system is a critical challenge in the presence of unavoidable environmental noises. Quantum state tomography (QST), particularly through balanced homodyne measurements, has become an essential tool for reconstructing quantum states, serving as a diagnostic method in many applications.[1,2] Starting with the maximum likelihood estimation,[3–5] QST has been widely applied not only to discrete and continuous variables but also to hybrid variables in optical, atomic, ionic, semiconductor, and superconducting systems.[6–13]

Using recent advances in artificial intelligence (AI), dedicated computational hardware has enabled the application of machine learning (ML) techniques to various sub-fields both in the classical and quantum world. To enhance the efficiency in applying QST, ML-enhanced QST has provided solutions to overcome limitations of conventional QST algorithms, such as overfitting and long run-time problems.[14–16] For example, in Ref. 17, we have demonstrated the potential of using ML to improve QST by framing it as a feature parameter estimation problem, reducing the model complexity by skipping the computationally expensive process of reconstructing density matrices. Even though hybrid quantum–classical neural networks or fully quantum neural networks are still evolving toward promising quantum machine learning architectures,[18–20] edge devices (located at the "edge of a network") are already available for applications in resource-limited computational environments.

In this work, we implement ML-enhanced QST on edge computing devices, namely, on field-programmable gate arrays (FPGAs) using AMD®'s (*ZCU104*) *Evaluation Board*, to deploy a

parameter-estimation neural network model. As flexible, configurable integrated circuits with high signal processing speed and parallel processing capabilities, FPGAs have been widely used in various industrial sectors, such as telecommunication, automotive industry, and aerospace. Benchmarking with our Graphics Processing Unit (GPU)-based ML-enhanced QST,[15,17] a significant reduction in the average inference time is demonstrated, dropping from 38 to 2.94 ms. At the same time, our FPGA lowers the output fidelity merely from 0.99 to 0.98, certifying it is a trustable reconstruction model. With these experimental results, our ML-enhanced QST on FPGA leverages simplicity and efficiency to facilitate real-time quantum state analysis in resource-limited computational environments.

This paper is organized as follows: in Sec. II, we introduce the flow charts of our implementation of ML-enhanced QST onto a FPGA device. Then, in Sec. III, comparisons between GPU-based and FPGA-based ML-enhanced QST for average fidelity, run-time costs, and power consumption are reported. Some further perspectives are provided before we, finally, conclude this paper in Sec. IV.

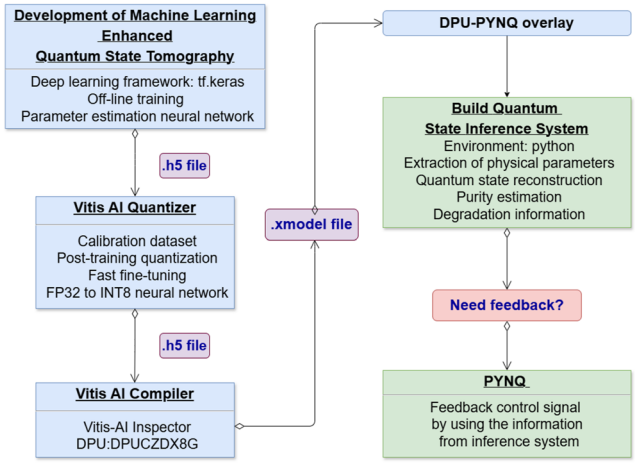

FIG. 2. Flow chart for ML-enhanced QST implementation on FPGA.

II. IMPLEMENTATION FLOW CHART

Figure 1 illustrates the entire execution process of FPGA-based quantum state tomography. The AMD (*ZCU104*) *Evaluation Board* is our FPGA device, into which the quadrature sequences from homodyne measurements are read. From left to right, the quadrature sequence signals are fed through the interface into the board. A pre-trained neural network processes these data and extracts three key physical parameters using a pre-trained inference system residing on the FPGA. We utilize these parameters to estimate the quantum state density matrix, the Wigner distribution, and compute both the average photon number of the pure part component and the average photon number of the non-pure component. This enables the analysis of quantum state degradation.

The flow chart in Fig. 2 describes some more details of the computational implementation: In edge devices, computational resource limitations often lead to challenges in maintaining precision. For the implementation, we first utilize the package "tf.keras" to convert the pre-trained neural network we reported in Ref. 17 from a 32 floating-point digit precision (FP32) format into a ".h5"-formatted file. To support this conversion, we optimized our 8-bit integer (INT8) machine learning model using *Vitis AI* provided by AMD Inc.

Subsequently, we employ the *Vitis AI Integrated Development Environment* to convert the neural network from 32-bit floating-point numbers (FP32) to the INT8 format and generate a ".xmodel file" for the FPGA's ML algorithm. The *Vitis AI Integrated Development Environment* provides AI inference on AMD (and Xilinx®)

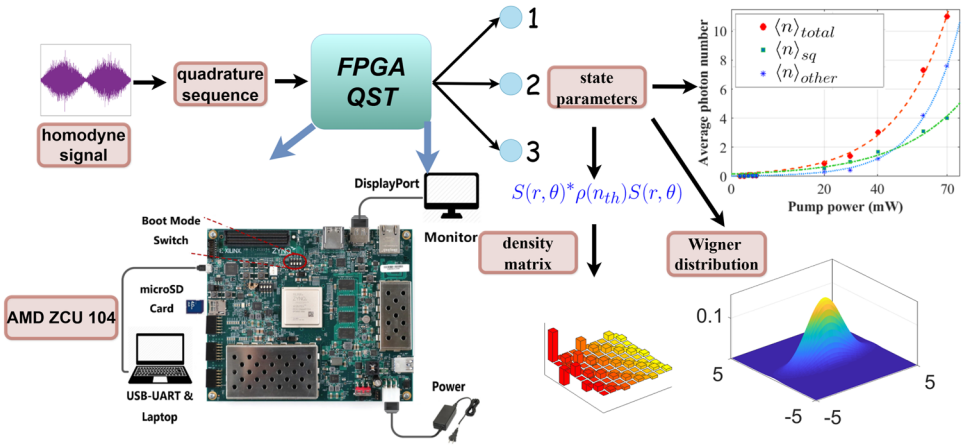

FIG. 1. Schematic of our FPGA quantum state tomography (QST): the quadrature sequences from homodyne measurement are the inputs from which parameters are generated to reconstruct density matrices, Wigner distributions, or characteristics (such as the average photon numbers). Here, the boot mode switch is present to select self-test or normal operating mode for the evaluation board *ZCU104*. The microSD card is used to load the *PYNQ* operating system, including the Python environment. A laptop can be connected for real-time control or program modification. (The demonstration-sketches are only for illustration and do not represent observed data.)

hardware platforms.[21] With the *Vitis AI Quantizer*, our machine learning model is converted into the INT8 format for efficient storage and computation. Subsequently, the *Vitis AI Compiler* is employed to transform the quantized model into the ".xmodel" file format, which is compatible with FPGA deployment. The compiler optimizes the model based on the hardware architecture, as well as the layout of the FPGA, invoking appropriate Intellectual Property (IP) cores for computation.

We now, briefly, describe the framework comprising the *Vitis AI Development Environment*, which includes the *Vitis AI Quantizer*, the *Vitis AI Compiler*, and the Deep learning Processing Unit (DPU).

A. Vitis AI Quantizer

During the training phase of our neural network, FP32 weights and activation values are changed. The function of the *Vitis AI Quantizer* is the conversion of weights and activation values from FP32 to INT8 format. This reduces overheads while maintaining prediction accuracy and gives us a fixed-point network model requiring less memory bandwidth and providing greater speed and power efficiency than flexible floating-point models can provide. Additionally, the *Vitis AI Quantizer* can be executed in common layers of neural networks, such as convolutional layers, pooling layers, and fully connected layers.

During the conversion process, also known as "quantization," we prepare 1000 quadrature sequences as a calibration dataset, with each sequence containing 2048 quadrature values. We then apply the so-called "post-training quantization" to fine-tune and optimize model performance.

B. Vitis AI Compiler

The *Vitis AI Compiler* takes a "quantized" INT8-neural network model as input and compiles it into a format that a specific DPU can read [the IP for our edge device (*ZCU104*) *Evaluation Board* is *DPUCZDX8G*]. The compiler accepts quantized models processed by the relevant language: *Caffe*, *TensorFlow*, or *PyTorch*. Finally, it is converted into a compiled ".xmodel" file.

C. Deep learning Processing Unit (DPU)

The DPU is an IP core specifically designed to accelerate deep learning inference on FPGAs. Here, an IP core is a pre-designed, pre-verified module used in electronic circuit design. Its IP is usually written in *Verilog*® or *VHDL*® languages. By writing in these programming languages, one can control the programmable logic embedded in FPGAs into specific logic configurations, such as adders or decoders. IP cores provide specific functions or tasks to be integrated directly on a FPGA. Using such IP cores helps designers save time and focus on the unique aspects of their projects.

After using the *Vitis AI Integrated Development Environment* to convert the INT8 model into a specialized ".xmodel" format, we can use this to off-load the corresponding DPU and transform our integer model into an appropriate FPGA computing unit for execution. We want to remark that before this conversion, we also utilized the *Vitis AI* inspector to verify that each component and structure of our neural network is supported by the DPU.

Different FPGA boards have different architectures; hence, different DPU modules correspond to each architecture. The *Vitis AI Integrated Development Environment* automatically invokes the appropriate hardware design, generating configuration files based on the FPGA board used, along with the selected DPU model. These files contain pre-designed hardware descriptions and configuration data.

Specifically, we use the edge device (*ZCU104*) *Evaluation Board* as an embedded deep learning processing unit to develop the deep neural networks. The (*ZCU104*) *Evaluation Board* is based on AMD (and Xilinx) FPGA chip, Zynq® UltraScale+TM MPSoC. The *ZCU104* can run the *Python* language (also in a *Jupyter Notebook* environment). Additionally, AMD has developed an operating system called *PYNQ* (*DPU-PYNQ* is one of their flavors), which starts the development board by installing *PYNQ* on a Secure Digital (SD) card. This SD card can be directly inserted into the development board, allowing the latter to operate independently of a host computer.

For (*ZCU104*) *Evaluation Boards*, the *Vitis AI Integrated Development Environment* provides us with an IP core: *DPUCZDX8G*, which allows developers to focus on high-level deep learning model design without delving into the intricacies of underlying hardware implementations.

After uploading the ".xmodel" file to the FPGA (*ZCU104*) *Evaluation Board*, we use *DPU-PYNQ*[22] for loading the model and employing overlay technology to operate the model. The *DPU-PYNQ* package is provided by *Xilinx*, which facilitates the deployment and execution of deep learning models on FPGA platforms by using the *PYNQ* framework. It leverages the DPU "overlay" to enable an efficient inference from neural networks. This package allows users to integrate hardware acceleration into their *Python*-based workflows, enhancing computational performance for machine learning tasks.

In *DPU-PYNQ,* an "overlay" is a high-level concept in FPGA design, representing a hardware configuration that can be dynamically loaded onto the FPGA. An overlay includes predefined hardware, specific IP cores, and connection logic, allowing users to perform specific tasks. On the *PYNQ* platform, overlays consist of a bitstream file and software drivers, enabling control through *Python* code. This simplifies the development process as users can deploy and run deep learning inference workloads without deep hardware knowledge. Overlays offer flexibility, simplified development, and quick deployment, making it easy to add new features or update existing ones. Once the model is successfully loaded onto the FPGA, we develop and use a *Python*-based program to implement a quantum state inference system.

For the deployment and execution of the model on the FPGA, we leverage the *DPU-PYNQ* package,[22] which facilitates the integration and loading from the input model onto the hardware. This package processes the experimental data, providing input quadrature sequences through a machine learning model to estimate three key parameter types: the degradation of the target quantum states (present in the quantum state reconstruction), physical parameters, and purity estimation.[15,17] These parameters are crucial for characterizing the degradation of squeezed states and can also be used to help with the reconstruction of density matrices or Wigner distributions of input quantum states. To visualize the density matrix, we can utilize an on-board *Python* or *NumPy* environment to construct the

squeezing operator and represent it in the Fock basis. Last but not least, after deploying the model on such edge devices, if feedback control is desired, this can be achieved through *PYNQ* by reading information obtained from quantum state tomography and controlling the FPGA board to put out signals, e.g., for feedback control purposes.

III. PERFORMANCE AND RESULTS

We tested our implementation using 100,000 quadrature-sequence signals, representing mock data for degraded squeezed states with various average photon numbers and different squeezing factors. Figure 3 compares the average output fidelity (solid curves) and corresponding standard deviation (shaded regions) between GPU-based (original model used in Refs. 15 and 17) and the FPGA-based Int8 (*ZCU 104 Evaluation Board*) ML-enhanced QST-model introduced here. These curves, in Fig. 3, demonstrate that the estimation of the output's average fidelity is slightly degraded by ~1% (from 0.99 to 0.98).

For each squeezing level, there are in total 100,000 mock-states (20,000 states per squeezing level ×5 sampling squeezing levels). The corresponding standard deviation is calculated by averaging the fidelity over 20,000 states.

After confirming that this fidelity reduction is acceptable, we further evaluate the output performance by loading 10,000 quadrature sequence datasets into our custom inference system to measure the average time required for quantum state inference, as shown in Fig. 4. The results demonstrate a significant, more than ten-fold, reduction of the inference times (from 38 to 2.94 ms). This clearly demonstrates the main benefit of deploying FPGA-based ML-enhanced QST.

Here, we use FGPA-based QST to directly generate characteristic parameters. Specifically, we calculate the average photon number of the pure state, the total average photon number, and the average photon number of the non-pure state (arising from the environment), which enables a clear distinction between contributions from the quantum system and environment. To visualize the density matrix, we also utilize the on-board *Python* and *NumPy* environment to construct the squeezing operator in the Fock basis. Subsequently, one can generate the corresponding density matrix of

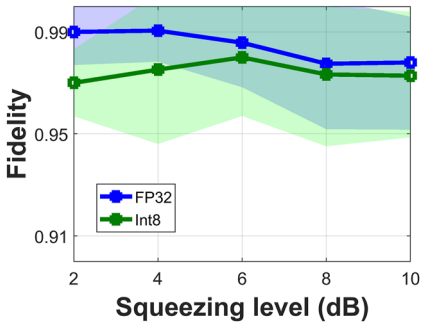

FIG. 3. Squeezed state fidelities (squeezing levels in units of dB) estimated either by the GPU-based models of Refs. 15 and 17 or the FPGA-based model used here. This comparison shows good agreement between the two approaches; the standard deviation corridors are displayed by shaded regions.

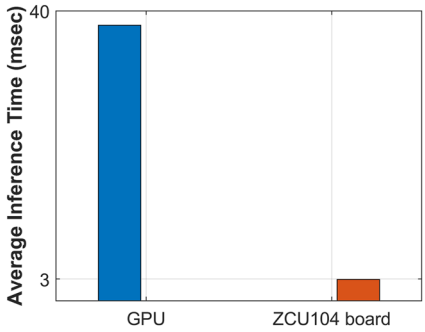

FIG. 4. Comparison of the average inference time cost, in milliseconds (ms), between GPU-based and FPGA-based (on *ZCU 104 Evaluation Board*) ML-enhanced QST.

lossy squeezed states. Performing such numerical computations on the FPGA board tends to be slow and is generally not recommended.

Figure 5 reports the power consumption in our FPGA-based QST; it is measured using an on-chip current sensor. It shows that before our QST starts, the FPGA's idle mode power consumption is at a level just below 14 W (black line). Any program written in *Verilog* or *HDL* using the Vivado IDE (Integrated Development Environment) must be in the "bitstream" format in order for the FPGA to load and execute. The corresponding green curve represents the power consumption during this process of downloading a test bitstream file (provided by AMD Inc) into the programmable logic region on the FPGA board; a slight increase in the power consumption to 14 W is recorded. When our custom tomography program starts to execute, the power consumption jumps to 17 W, while loading 10,000 quadrature sequence datasets into the FPGA, it reaches 18 W. This takes about 30 s (in agreement with the average inference time of 2.94 ms multiplied by 10,000 instances, yielding ≈30 s in total). Upon its completion, the power consumption drops back to 17 W (see Fig. 5).

Our neural network model, i.e., the ".xmodel" file compiled by a *Vitis AI Compiler*, has also been loaded by a *Python* program running on a separate computer in a *Jupyter Notebook* environment to perform AI operations. Built-in I/O connectors allow the *ZCU 104 Evaluation Board* to interface with an external computer.

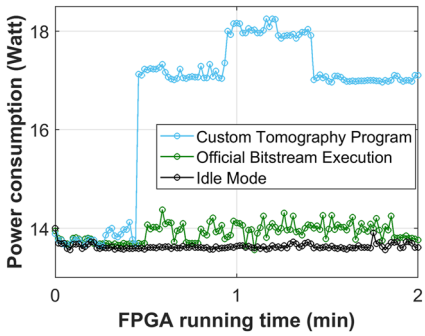

FIG. 5. FPGA power consumption either in idle mode, or performing our custom tomography program, or when loading 10,000 datasets.

In this way, one may use *PYNQ* to perform the Proportional-Integral-Derivative (PID) feedback control on the board. We may also connect ADC/DAC cards or sensors (for temperature, light, etc.) and obtain signals through *Python* commands. In addition, the *Python-Matplotlib* library can display data streams on screens in real time. This helps us to generate FPGA-based ML-enhanced QST data to perform real-time PID feedback control.

The FPGA's parallel architecture reduces data delays in real-time control. The power consumption of an FPGA is much lower than that of GPUs and CPUs, which allows for FPGA-use based on batteries to perform remote control when needed. The FPGA board can also be connected to a computer server for monitoring and control, or it can be directly connected to desired targets for preset real-time control.

In our case, the FPGA's embedded *PYNQ* operating system is stored on an SD card on-board. The Jupyter Notebook interface in *PYNQ* also provides the Python language for sending commands or programming flows to control and monitor the performance of FPGA boards on edge devices.

In general, a fixed-point network model requires less bandwidth than a floating-point network model, thus providing faster running speeds: due to the hardware architecture of FPGAs (configurable flip-flops and parallel structures in logic blocks), for 8-bit integer precision, the FPS (Flash Per Second) of an FPGA is faster than that of GPU under the same conditions (such as a 28 nm semiconductor process). Nevertheless, in some algorithms, for 8-bit integer precision, the GPU's FPS can achieve better performance while maintaining higher accuracy.

IV. CONCLUSIONS

We have implemented machine learning-based quantum state tomography on a *ZCU 104 Evaluation Board* with a *Vitis AI Integrated Development Environment* FPGA-device. We report on the effectiveness of this approach for achieving real-time, high-precision quantum state analysis on resource-constrained devices and find a slight degradation in the inferred output fidelity, dropping from 0.99 to 0.98. The signal processing speed, when tomographically reconstructing quantum states, is demonstrated to be an order of magnitude greater, for instance, an inference time of 2.94 ms, instead of 38 ms previously.[15,17] Our FPGA-based QST is an efficient, flexible, and precise tool for real-time diagnostics of quantum states, offering parallel processing capabilities.

We use a GPU with 32-bit floating point numbers to train the network for high accuracy and convert it to 8-bit integers for deployment on an FPGA. The main advantages of FPGAs over GPUs are timing latency, power consumption, and interface scalability (see Figs. 3–5).

In the literature, a scalable and self-analyzing digital locking system, for use on quantum optics experiments,[23] was reported. An open-source platform, *NQontrol*, also demonstrated digital control-loops,[24] as well as a reconfigurable QST solver on a FPGA.[25] Squeezed states have been used as true applications from quantum metrology,[26] advanced gravitational wave detectors,[27,28] and quantum information manipulation using continuous variables.[29] FPGA-based ML-enhanced QST can be used as in-line diagnostic toolboxes for various applications that rely on the use of squeezed states.[30–32]

In addition to application to Gaussian states, as illustrated here, this technology paves the way to dealing with more general quantum states, including non-Gaussian states and multi-partite quantum states at high throughput speeds.

ACKNOWLEDGMENTS

This work was partially supported by the Ministry of Science and Technology of Taiwan (Grant Nos. 112-2123-M-007-001, 112-2119-M-008-007, and 112-2119-M-007-006), Office of Naval Research Global, the International Technology Center Indo-Pacific (ITC IPAC) and Army Research Office under Contract No. FA5209-21-P-0158, and the collaborative research program of the Institute for Cosmic Ray Research (ICRR) at the University of Tokyo.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Hsun-Chung Wu: Data curation (equal); Software (equal); Validation (equal); Writing – original draft (equal). **Hsien-Yi Hsieh**: Data curation (equal); Formal analysis (equal); Methodology (equal); Software (equal); Validation (equal); Writing – original draft (equal). **Zhi-Kai Xu**: Data curation (equal); Methodology (equal); Software (equal). **Hua Li Chen**: Data curation (equal); Methodology (equal); Validation (equal). **Zi-Hao Shi**: Data curation (equal); Methodology (equal); Software (equal). **Po-Han Wang**: Data curation (equal); Methodology (equal); Software (equal). **Popo Yang**: Investigation (equal); Resources (equal). **Ole Steuernagel**: Methodology (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Te-Hwei Suen**: Methodology (equal); Software (equal). **Chien-Ming Wu**: Data curation (equal); Methodology (equal); Project administration (equal); Resources (equal); Software (equal); Supervision (equal); Validation (equal); Writing – original draft (equal). **Ray-Kuang Lee**: Conceptualization (equal); Funding acquisition (equal); Project administration (equal); Resources (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

Data underlying the results presented in this paper are not publicly available at this time but may be obtained from the author upon reasonable request.

REFERENCES

[1] U. Leonhardt, *Measuring the Quantum State of Light* (Cambridge University Press, 1997).
[2] A. I. Lvovsky and M. G. Raymer, "Continuous-variable optical quantum-state tomography," Rev. Mod. Phys. **81**, 299 (2009).
[3] K. Banaszek, "Maximum-likelihood estimation of photon-number distribution from homodyne statistics," Phys. Rev. A **57**, 5013 (1998).
[4] Z. Hradil, "Quantum-state estimation," Phys. Rev. A **55**, R1561(R) (1997).

[5]A. I. Lvovsky, "Iterative maximum-likelihood reconstruction in quantum homodyne tomography," J. Opt. B: Quantum Semiclassical Opt. **6**, S556 (2004).
[6]U. L. Andersen, J. S. Neergaard-Nielsen, P. van Loock, and A. Furusawa, "Hybrid discrete-and continuous-variable quantum information," Nature Phys. **11**, 713 (2015).
[7]D. Barredo, S. de Léséleuc, V. Lienhard, T. Lahaye, and A. Browaeys, "An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays," Science **354**, 1021 (2016).
[8]M. Endres, H. Bernien, A. Keesling, H. Levine, E. R. Anschuetz, A. Krajenbrink, C. Senko, V. Vuletic, M. Greiner, and M. D. Lukin, "Atom-by-atom assembly of defect-free one-dimensional cold atom arrays," Science **354**, 1024 (2016).
[9]J. Zhang, G. Pagano, P. W. Hess, A. Kyprianidis, P. Becker, H. Kaplan, A. V. Gorshkov, Z.-X. Gong, and C. Monroe, "Observation of a many-body dynamical phase transition with a 53-qubit quantum simulator," Nature **551**, 601 (2017).
[10]N. Friis, O. Marty, C. Maier, C. Hempel, M. Holzäpfel, P. Jurcevic, M. B. Plenio, M. Huber, C. Roos, R. Blatt, and B. Lanyon, "Observation of entangled states of a fully controlled 20-qubit system," Phys. Rev. X **8**, 021012 (2018).
[11]D. Cogan, G. Peniakov, Z.-E. Su, and D. Gershoni, "Complete state tomography of a quantum dot spin qubit," Phys. Rev. B **101**, 035424 (2020).
[12]K. Takeda, A. Noiri, T. Nakajima, J. Yoneda, T. Kobayashi, and S. Tarucha, "Quantum tomography of an entangled three-qubit state in silicon," Nat. Nanotechnol. **16**, 965 (2021).
[13]B. Vlastakis, G. Kirchmair, Z. Leghtas, S. E. Nigg, L. Frunzio, S. M. Girvin, M. Mirrahimi, M. H. Devoret, and R. J. Schoelkopf, "Deterministically encoding quantum information using 100-photon Schrödinger cat states," Science **342**, 607 (2013).
[14]E. S. Tiunov, V. V. Tiunova Vyborova, A. E. Ulanov, A. I. Lvovsky, and A. K. Fedorov, "Experimental quantum homodyne tomography via machine learning," Optica **7**, 448 (2020).
[15]H.-Y. Hsieh, Y.-R. Chen, H.-C. Wu, H. L. Chen, J. Ning, Y.-C. Huang, C.-M. Wu, and R.-K. Lee, "Extract the degradation information in squeezed states with machine learning," Phys. Rev. Lett. **128**, 073604 (2022).
[16]S. Ahmed, C. Sánchez Muñoz, F. Nori, and A. F. Kockum, "Quantum state tomography with conditional generative adversarial networks," Phys. Rev. Lett. **127**, 140502 (2021).
[17]H.-Y. Hsieh, J. Ning, Y.-R. Chen, H.-C. Wu, H. L. Chen, C.-M. Wu, and R.-K. Lee, "Direct parameter estimations from machine learning-enhanced quantum state tomography," Symmetry **14**, 874 (2022).
[18]J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," Nature **549**, 195 (2017).
[19]S. Lohani, B. T. Kirby, M. Brodsky, O. Danaci, and R. T. Glasser, "Machine learning assisted quantum state estimation," Mach. Learn.: Sci. Technol. **1**, 035007 (2020).
[20]A. Melnikov, M. Kordzanganeh, A. Alodjants, and R.-K. Lee, "Quantum machine learning: From physics to software engineering," Adv. Phys.: X **8**, 2165452 (2023).
[21]AMD®, Inc., Vitis AI User Guide, UG1414 (v3.0), 24 February 2023.
[22]AMD®, Inc., DPUCZDX8G for Zynq®UltraScale+MPSoC, PG338 (v4.1), 23 January 2023.
[23]B. M. Sparkes, H. M. Chrzanowski, D. P. Parrain, B. C. Buchler, P. K. Lam, and T. Symul, "A scalable, self-analyzing digital locking system for use on quantum optics experiments," Rev. Sci. Instrum. **82**, 075113 (2011).
[24]C. Darsow-Fromm, L. Dekant, S. Grebien, M. Schröder, R. Schnabel, and S. Steinlechner, "NQontrol: An open-source platform for digital control-loops in quantum-optical experiments," Rev. Sci. Instrum. **91**, 035114 (2020).
[25]N. E. Miller, B. Chakraborty, and S. Mukhopadhyay, "A reconfigurable quantum state tomography solver in FPGA," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Bellevue, WA, 2023), pp. 1412–1421.
[26]A. P. Alodjants, D. V. Tsarev, D. A. Kuts, S. A. Podoshvedov, and S. P. Kulik, "Quantum optical metrology," Phys.-Usp. **67**, 668 (2024).
[27]Y. Zhao *et al.*, "Frequency-dependent squeezed vacuum source for broadband quantum noise reduction in advanced gravitational-wave detectors," Phys. Rev. Lett. **124**, 171101 (2020).
[28]D. Ganapathy *et al.*, "Broadband quantum enhancement of the LIGO detectors with frequency-dependent squeezing," Phys. Rev. X **13**, 041021 (2023).
[29]H. Aghaee Rad *et al.*, "Scaling and networking a modular photonic quantum computer," Nature **638**, 912 (2025).
[30]Y.-R. Chen, H.-Y. Hsieh, J. Ning, H.-C. Wu, H. L. Chen, Y.-L. Chuang, P. Yang, O. Steuernagel, C.-M. Wu, and R.-K. Lee, "Experimental reconstruction of Wigner phase-space current," Phys. Rev. A **108**, 023729 (2023).
[31]Y.-R. Chen, H.-Y. Hsieh, J. Ning, H.-C. Wu, H. L. Chen, Z.-H. Shi, P. Yang, O. Steuernagel, C.-M. Wu, and R.-K. Lee, "Generation of heralded optical cat states by photon addition," Phys. Rev. A **110**, 023703 (2024).
[32]H.-Y. Hsieh, Y.-R. Chen, M.-M. Huang, J. Ning, H.-C. Wu, H. L. Chen, Z.-H. Shi, P.-H. Wang, O. Steuernagel, C.-M. Wu, and R.-K. Lee, "Neural-network-enhanced Fock-state tomography," Phys. Rev. A **110**, 053705 (2024).