Realizing a deep reinforcement learning agent
discovering real-time feedback control strategies for a quantum system

Kevin Reuer,[1, *] Jonas Landgraf,[2, 3] Thomas Fösel,[2, 3] James O'Sullivan,[1] Liberto Beltrán,[1]
Abdulkadir Akin,[1] Graham J. Norris,[1] Ants Remm,[1] Michael Kerschbaum,[1]
Jean-Claude Besse,[1] Florian Marquardt,[2, 3] Andreas Wallraff,[1, 4] and Christopher Eichler[1, †]

[1] *Department of Physics, ETH Zurich, CH-8093 Zurich, Switzerland*
[2] *Max Planck Institute for the Science of Light, Staudtstraße 2, 91058 Erlangen, Germany*
[3] *Physics Department, University of Erlangen-Nuremberg, Staudtstraße 5, 91058 Erlangen, Germany*
[4] *Quantum Center, ETH Zurich, CH-8093 Zurich, Switzerland*
(Dated: November 1, 2022)

To realize the full potential of quantum technologies, finding good strategies to control quantum information processing devices in real time becomes increasingly important. Usually these strategies require a precise understanding of the device itself, which is generally not available. Model-free reinforcement learning circumvents this need by discovering control strategies from scratch without relying on an accurate description of the quantum system. Furthermore, important tasks like state preparation, gate teleportation and error correction need feedback at time scales much shorter than the coherence time, which for superconducting circuits is in the microsecond range. Developing and training a deep reinforcement learning agent able to operate in this real-time feedback regime has been an open challenge. Here, we have implemented such an agent in the form of a latency-optimized deep neural network on a field-programmable gate array (FPGA). We demonstrate its use to efficiently initialize a superconducting qubit into a target state. To train the agent, we use model-free reinforcement learning that is based solely on measurement data. We study the agent's performance for strong and weak measurements, and for three-level readout, and compare with simple strategies based on thresholding. This demonstration motivates further research towards adoption of reinforcement learning for real-time feedback control of quantum devices and more generally any physical system requiring learnable low-latency feedback control.

Future quantum information processing devices will rely on the ability to continuously monitor their state via quantum measurements and to act back on them, on timescales much shorter than the coherence time, conditioned on prior observations. Such real-time feedback control of quantum systems, which offers applications e.g. in qubit initialization [1–3], gate teleportation [4, 5] and quantum error correction [6–8], typically relies on an accurate model of the underlying system dynamics. With the increasing number of constituent elements in quantum processors such accurate models are generally not available. Model-free reinforcement learning [9] promises to overcome such limitations by learning feedback-control strategies without prior knowledge of the quantum system.

Reinforcement learning, a subfield of machine learning, has had outstanding success in tasks ranging from board games [10] to robotics [11]. Reinforcement learning, however, has only very recently been started to be applied to complex physical systems, with training performed either on simulations [12–18] or directly in experiments [19–26], for example in laser [19, 22, 26], particle [20, 21], soft-matter [23] and quantum physics [24, 25]. Specifically in the quantum domain, during the past few years, a number of theoretical works have pointed out the great

promises of reinforcement learning for tasks covering state preparation [27–31], gate design [32], error correction [33–35] and circuit optimization/compilation [36, 37], making it an important part of the machine learning toolbox for quantum technologies [38–40]. In first applications to quantum systems, reinforcement learning was experimentally deployed, but training was mostly performed based on simulations, specifically to optimize pulse sequences for atoms and spins [14, 15, 18]. Beyond that, there are two pioneering works demonstrating the training directly on experiments [24, 25] which was used to optimize pulses for quantum gates [24] and to accelerate the tune-up of quantum dot devices [25]. However, in none of these experiments [14, 15, 18, 24, 25] real-time quantum feedback was required.

Here, we realize a reinforcement learning agent which interacts with a quantum system on a sub-microsecond timescale. This rapid response time enables the agent's use for real-time quantum feedback control. We implement the agent using a novel low-latency neural network architecture, which processes data concurrently to data acquisition, on a field-programmable gate array (FPGA). As a proof of concept, we train the agent using model-free reinforcement learning to initialize a superconducting qubit into its ground state without relying on a prior model of the quantum system. The training is performed directly on the experiment, i.e. by acquiring experimental data with updated network parameters in every training step. In repeated cycles, the trained agent acquires measurement data, processes it and applies pre-calibrated

---

pulses to the qubit conditioned on the measurement outcome until the agent terminates the initialization process. We study the evolution of the agent's performance during training and demonstrate convergence in less than three minutes wall clock time and based on less than 30,000 episodes of training data. Furthermore, we explore the agent's strategies in more complex scenarios, i.e. when performing weak measurements or when resetting a qutrit.
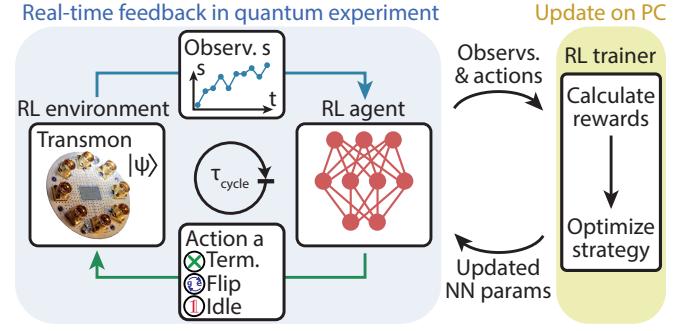

Figure 1. Concept of the experiment. A reinforcement learning (RL) agent, realized as a neural network (red) on a field-programmable gate array (FPGA), receives observations **s** (blue trace) from a quantum system, which constitutes the reinforcement learning (RL) environment. Here, the quantum system is realized as a transmon qubit coupled to a readout resonator fabricated on a chip (see photograph). The agent processes observations on sub-microsecond timescales to decide in real time on the next action a applied to the quantum system. The update of the agent's parameters is performed by processing experimentally obtained batches of observations and actions on a PC.

I. REINFORCEMENT LEARNING FOR A QUANTUM SYSTEM

In model-free reinforcement learning, an agent interacts with the world around it, the so-called reinforcement learning environment (see Fig. 1). In repeated cycles, the agent receives observations **s** from the environment and selects actions a according to the respective observation **s** and its policy π. In the important class of policy-gradient methods [9], this policy is realized as a conditional probability distribution $\pi_\theta(a|\mathbf{s})$, which can be modelled as a neural network with parameters θ. To each sequence of observation-action pairs, called an episode, one assigns a cumulative reward R. The goal of reinforcement learning is to maximize the reward $\bar{R}$ averaged over multiple episodes, by updating the parameters θ e.g. via gradient ascent $\Delta\theta \sim \nabla_\theta \bar{R}$ [9]. Such a policy-gradient procedure is able to discover an optimal policy even without access to an explicit model of the reinforcement learning environment's dynamics.

In the present work, our goal is to use reinforcement learning to learn strategies for real-time control of quantum systems. Here, observations are obtained via quantum measurements, actions are realized as unitary gate operations, and the reward is measured in terms of the speed and fidelity of initializing the quantum system into a target state, see schematic in Fig. 1. In our experiment the quantum system is realized as a transmon qubit with ground $|g\rangle$, excited $|e\rangle$, and second excited state $|f\rangle$ dispersively coupled to a superconducting resonator (see App. A for details). We probe the qubit with a microwave field, which scatters off the resonator and gets amplified and digitized to result in an observation vector $\mathbf{s} = (\mathbf{I}, \mathbf{Q})$, where $\mathbf{I}$ and $\mathbf{Q}$ are time traces of the two quadrature components of the digitized signal [41–43] (see App. B for details and App. C for averaged time traces). Depending on **s** the agent selects, according to its policy π, one of several discrete actions in real time. In the simplest case, it either *idles* until the next measurement cycle, it performs a *bit-flip* as a unitary swap between $|g\rangle$ and $|e\rangle$ or it *terminates* the initialization process.

To train the agent, we transfer batches of episodes to a personal computer (PC) serving as a reinforcement learning trainer. The reinforcement learning trainer computes the associated reward for each episode and updates the agent's policy accordingly (see App. D for details), before sending back the updated network parameters θ to the FPGA.

II. IMPLEMENTATION OF THE NEURAL-NETWORK-BASED REAL-TIME AGENT

We implemented this scheme in an experimental setup, in which the agent, for each episode, is able to perform multiple measurement cycles j, in each of which it receives a qubit-state-dependent observation $\mathbf{s}^j$ and selects an action a^j, until it terminates the episode, see Fig. 2(a). If the agent selects the flip action, a π-pulse is applied after a total latency of $\tau_{\mathrm{EL,tot}} = 451$ ns, dominated by analog-to-digital and digital-to-analog converter delays. The agent's neural network contributes only $\tau_{\mathrm{NN}} = 48$ ns to the total latency as it is evaluated mostly during qubit readout and signal propagation (see App. B for detailed discussion of the latency). To provide the agent with a memory about past cycles we feed downsampled observations $(\mathbf{s}^{j-1}, ..., \mathbf{s}^{j-l})$ and actions $(a^{j-1}, ..., a^{j-l})$ from up to $l = 2$ previous cycles into the neural network. To characterize the performance of the agent, we perform a verification measurement $\mathbf{s}^{\mathrm{ver}}$ after termination.

Any neural-network agent used for real-time system control greatly benefits from short latencies in the signal processing. For our FPGA implementation we therefore introduce a novel network architecture, which aims to keep latencies at a minimum, see Fig. 2(b). First, we implement the agent as a feedforward neural network [44] on the FPGA, rather than a more resource-demanding recurrent neural network, like a long short-term memory network (LSTM) [45]. Second, we process information from previous cycles in a two-layer *pre-processing network* before the start of the current cycle, thus not contributing to the latency. Third, and most importantly, we implement a novel *low-latency network* architecture, in
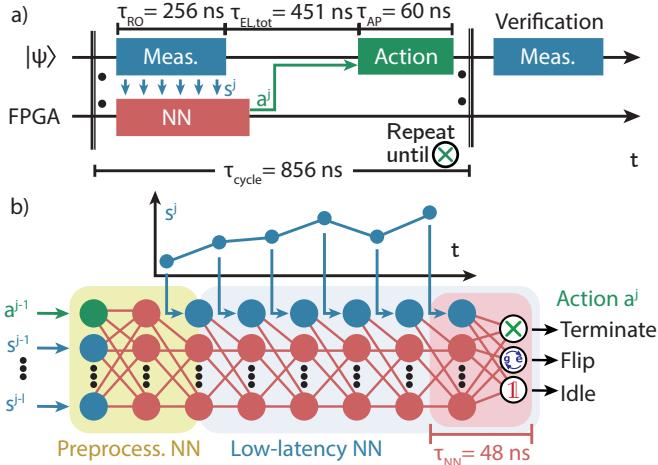
Figure 2. Schematic of neural-network-based real-time feedback control. (a) Timing diagram of a reinforcement learning episode. In each cycle j, the observation $\mathbf{s}^j$ resulting from a measurement (blue) is continuously fed into a neural network (red) which decides on the next action a^j (green). Once the agent terminates after several cycles, a verification measurement is performed. (b) Schematic of the neural network implemented on an FPGA. The neural network consists of fully-connected (red lines) layers of feed-forward neurons (red dots) and input neurons (blue dots for observations, green dots for actions). The first layers form the preprocessing network (yellow background). During the evaluation of the low-latency network (blue background), new data points from the signal trace $\mathbf{s}^j$ are fed into the network as they become available. The network outputs the action probabilities for the three actions. Only the execution of the last layer (red background) contributes to the overall latency.

which new measurement data is processed as soon as it becomes available. More specifically, we sequentially feed elements I_k^j, Q_k^j of the digitized time trace $\mathbf{s}^j = (\mathbf{I}^j, \mathbf{Q}^j)$ into each layer of the neural network concurrent with its evaluation, see Fig. 2(b) and App. E. As a result, only the execution of the last layer contributes to the total latency while all other layers are evaluated in parallel with the data acquisition. For the experiments presented in the following, we use a network with 7 hidden layers and 12 neurons per layer. The output layer has only three neurons, corresponding to the three actions. However, as the exact neural network structure may in general depend on the properties of the specific quantum system and the agent's task, the width and depth of the neural network are adjustable parameters in our FPGA design. We have also explored the use of the same type of neural network for quantum state discrimination, in a supervised-learning setting (see App. C) .

III. TRAINING THE AGENT WITH EXPERIMENTAL DATA

To train the agent, we experimentally acquire 1000 observation-action pairs $(\mathbf{s}, a)$ with the FPGA, before transferring them to the reinforcement learning trainer on the PC, see Fig. 1. The reinforcement learning trainer then updates the parameters θ of the agent's policy $\pi_\theta(a|\mathbf{s})$ using a state-of-the-art algorithm (proximal policy optimization, PPO) [46, 47], with the goal to maximize the cumulative reward $R = U_{\mathrm{ver}}/\Delta U - n\lambda$ (see App. D for details). Here, the integrated observation in the final verification measurement $U_{\mathrm{ver}} = \mathbf{w_s}\mathbf{s}^{\mathrm{ver}}$ serves as an indicator for the ground-state population, with a normalization factor $\Delta U = \mathbf{w_s}(\langle\mathbf{s}_g\rangle - \langle\mathbf{s}_e\rangle)$ setting the scale, and the second term penalizes each cycle with a constant amount λ. Thus, λ controls the trade-off between short episode length and high initialization fidelity. The updated parameters θ are then transferred back to the FPGA, and we repeat this procedure until the cumulative reward R is maximized.

We first train the reinforcement learning agent to initialize the qubit using fast, high-fidelity readout. In this regime, an initialization strategy based on weighted integration and thresholding is close-to-optimal, and we can thus easily verify and benchmark the strategies discovered by the reinforcement learning agent. To study the agent's learning process, we monitor the average initialization error $1-P_g$, inferred from a fit to the measured distribution of U_{ver} (see App. B for details), and the average number of cycles $\langle n\rangle$ until termination, see Fig 3(a) and (b). While the initial random policy results in only $P_g \sim 50\%$, the agent quickly learns how to initialize the qubit for both prepared initial states, which we choose to be the equilibrium state (red) and its counterpart with the population inverted by a π-pulse (dark blue). The initialization error $1-P_g$ has already converged to about $0.2\,\%$ after training with about 30,000 episodes, which includes 100 parameter updates by the reinforcement learning trainer on the PC, and takes only three minutes wall clock time. The short training duration, limited mainly by data transfer between the PC and FPGA, enables frequent readjustment of the neural network parameters and thus allows to account for drifts in experimental parameters. The average number of cycles $\langle n\rangle$ converges to about 1.1 for the initial equilibrium, and to about 2.2 for the inverted equilibrium state, indicating that for strong measurements the agent only needs an additional cycle to terminate for about 10 %, or respectively 20 %, of the episodes.

IV. POLICY AND PERFORMANCE FOR STRONG MEASUREMENTS

After the training has been completed, we analyze the policy. Instead of directly investigating the (high-dimensional) dependence of the agent's policy $\pi(a|\mathbf{s})$ on the full time trace $\mathbf{s}$, we simply extract the probabilities $P(a)$ for the agent to select the possible actions depending
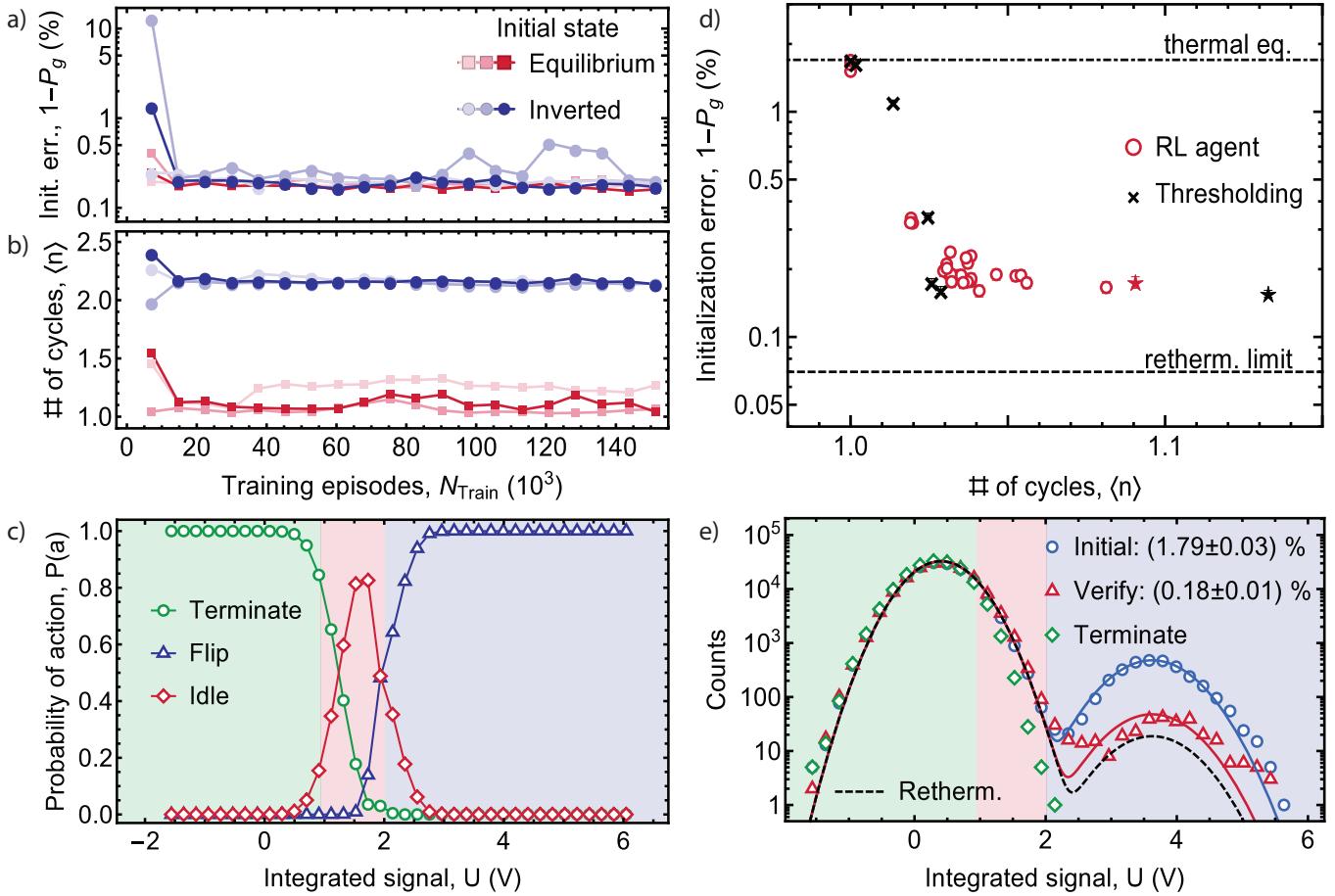
Figure 3. Experimental data for reinforcement learning with a network-based real-time agent. (a) Initialization error $1 - P_g$ and (b) average number of cycles $\langle n \rangle$ until termination vs number of training episodes N_{train}, when preparing an equilibrium state (red squares) and when inverting the population with a π-pulse (dark blue circles) for three independent training runs (solid and transparent points). Each datapoint is obtained from an independent validation data set with $\sim 180,000$ episodes. (c) Probability of choosing an action $P(a)$ vs the integrated measurement signal U. Actions chosen by the threshold-based strategy are shown as background colors (also for (e)). (d) Initialization error $1 - P_g$ vs average number of cycles $\langle n \rangle$ until termination for an equilibrium state for the reinforcement learning agent (red circles) and the threshold-based strategy (black crosses). Stars indicate the strategies used for the experiments in (c) and (e). (e) Histogram of U in equilibrium (blue circles), for the measurement in which the agent terminates (green diamonds) and for the verification measurement (red triangles). Lines are bimodal Gaussian fits, from which we extract ground state populations as shown in the inset. The dashed black line indicates the rethermalization limit (see main text).

on the integrated signal U, see Fig. 3(c). We compare the agent's selection of actions to a simple strategy in which the process is terminated (green background) for U values below an acceptance threshold and a flip (blue background) is applied for U larger than a state discrimination threshold. We observe that for U far below the acceptance threshold the agent nearly always terminates, while the agent predominantly selects the flip action for U far above the state discrimination threshold. This is expected as, in both cases, the agent has high certainty about the qubit state. Between the two thresholds where uncertainty is large, the agent is more likely to idle. The transitions of the individual probabilities are smooth. This is not due to some deliberate randomization of action choices, but rather a sign that the agent's policy depends on additional

information beyond the integrated signal U shown here: the agent has access to the full measured time trace.

To evaluate the agent's performance we analyze the tradeoff between initialization error $1 - P_g$ and average cycle number $\langle n \rangle$ as a function of the control parameter λ. As expected, we find that an increase in $\langle n \rangle$, controlled by lowering λ, results in a gain of initialization fidelity until $1 - P_g$ converges to about 0.18% (for $\langle n \rangle \geq 1.1$ cycles), see Fig 3(d). We attribute the remaining infidelity mostly to rethermalization of the qubit between the termination and the verification cycle, and, possibly, state mixing during the final verification readout. In our experiment, this rethermalization rate is $N_{\text{eq}}/T_1 \approx 1$ kHz with $N_{\text{eq}} = 1.4\%$, contributing $\sim 0.07\%$ to the infidelity. As anticipated, the agent's performance matches the perfor-

mance of simple, close-to-optimal, thresholding strategies, where we vary the acceptance threshold to control the average cycle number $\langle n \rangle$ (black crosses). This indicates that the strategies discovered by the agent are also close-to-optimal. We also deduce that the agent's performance is limited mostly by rethermalization by analyzing the bimodal Gaussian distribution of the integrated qubit readout signal U_{ver} in the verification measurement (red triangles in Fig. 3(e)). While the integrated signal in the termination cycle (green diamonds) has only very few counts above the state discrimination threshold, the number of such instances rises to about 0.18% in the verification measurement, indicating transitions into the excited state occuring between the two cycles. Compared to the equilibrium state (blue circles) the excited state fraction is reduced by about a factor 10 by using the reinforcement learning initialization scheme.

V. WEAK MEASUREMENTS AND QUTRIT READOUT

The observations until this point demonstrate that our real-time agent performs well and trains reliably on experimentally obtained rewards. Next, we discuss regimes where good initialization strategies are more complex. As a first example, we investigate the agent's strategy and performance when only weakly measuring the qubit. We reduce the power of the readout tone, while keeping its duration and frequency unchanged, such that bimodal Gaussian distributions of a prepared ground and excited state overlap by 25 % (see App. B). In this case, we find that the agent profits from memory, if it is permitted access to information from l previous cycles. In that case, its strategy not only depends on the current signal U_t, but also on the signal U_{t-1} from the previous cycle, see Fig. 4(a). Whenever the current measurement hints at the same state as the previous measurement (upper right and lower left in each panel) the agent gains certainty about the state and thus becomes more likely to terminate the process (green region in lower left corner) or swap the $|g\rangle$ and the $|e\rangle$ state (blue region in upper right corner). As for strong measurements, we find a trade-off between $\langle n \rangle$ and $1 - P_g$ when varying λ, see Fig. 4(b). Importantly, we observe that agents making use of memory ($l = 2$, red circles) require fewer rounds $\langle n \rangle$ to reach a certain initialization error than agents without memory ($l = 0$, green triangles) or a thresholding strategy (black crosses). This indicates that the observed dependence of the strategy on U_{t-1} does result in a performance improvement; the reinforcement learning agent can exploit the possibility of measuring multiple times, in contrast to the simple thresholding strategy.

In addition, we have studied the performance of the agent when also considering the second excited state $|f\rangle$, which we have neglected so far. The $|f\rangle$ state is populated with a certain probability due to undesired leakage out of the computional states $|g\rangle$ and $|e\rangle$ during single-qubit,

two-qubit and readout operations [48]. Thus, schemes which also reset $|f\rangle$ into $|g\rangle$ are required. For this purpose, we enable the agent to also swap $|f\rangle$ and $|g\rangle$ states by adding a fourth action, and train the agent on a qutrit mixed state with one third $|g\rangle$, $|e\rangle$ and $|f\rangle$ population, prepared by idling the qubit, swapping the qubits $|g\rangle$ and $|e\rangle$ state, and swapping the qubits $|g\rangle$ and $|f\rangle$ state, with probability 1/3 respectively. For this qutrit system, state assignment typically processes two different projections of the measurement trace $U = \mathbf{w_U} \mathbf{s}^{\text{ver}}$ and $W = \mathbf{w_W} \mathbf{s}^{\text{ver}}$, where $\mathbf{w_U}$ and $\mathbf{w_W}$ form an orthonormal set of weights. Here, we use U and W to visualize the agent's strategy. We find that, if the qubit would be classified to be in the $|f\rangle$ state, the agent is most likely to swap $|f\rangle$ and $|g\rangle$ (orange), as expected, see Fig. 4(d). Similarly, if the qubit is classified to be in $|g\rangle$, the agent most likely terminates, while the agent swaps $|e\rangle$ and $|g\rangle$, if it is classified to be in $|e\rangle$. Around the state discrimination threshold between $|g\rangle$ and $|e\rangle$, however, the agent mostly idles, as the qubit state is uncertain. Interestingly, along the state discrimination threshold between $|e\rangle$ and $|f\rangle$, the agent almost never idles, as it is more advantageous either to apply a flip from $|e\rangle$ to $|g\rangle$ or a flip from $|f\rangle$ to $|g\rangle$. After the operation, the qubit will be in the ground state if, by chance, the correct action was applied, while it would never be in the ground state if the agent chose to idle.

To study the performance of the reinforcement learning agent, we trained the agent on a qutrit mixed state. We find that an agent that can swap $|f\rangle$ to $|g\rangle$, in addition to the other actions, efficiently resets the transmon from a qutrit mixed state with an initialization error $1 - P_g \approx 0.2\%$ for $\langle n \rangle \approx 2$ (blue squares), see Fig. 4(d). In contrast, an agent which cannot access the gf-flip action needs significantly more rounds till termination to reach a similar initialization error, as the agent needs to rely on decay from the $|f\rangle$ level, which in our setup had a lifetime of $T_1^{(f)} = 6$ µs.

These examples demonstrate the versatility of the reinforcement learning approach to discovering state initialization strategies under a variety of circumstances.

VI. CONCLUSION

In conclusion, we have implemented a real-time neural-network agent with a sub-microsecond latency enabled by a network design which accepts data concurrently with its evaluation. This is about 100 times faster than in the fusion control experiments of Ref. [17], which is, to our knowledge, the fastest reinforcement learning agent deployed in a physics experiment so far. The need for such optimized real-time control will increase due to the ever more stringent requirements on the fidelities of quantum processes as quantum devices grow in size and complexity. We have successfully trained the agent using reinforcement learning in a quantum experiment and demonstrated its ability to adapt its strategy in different scenarios, includ-
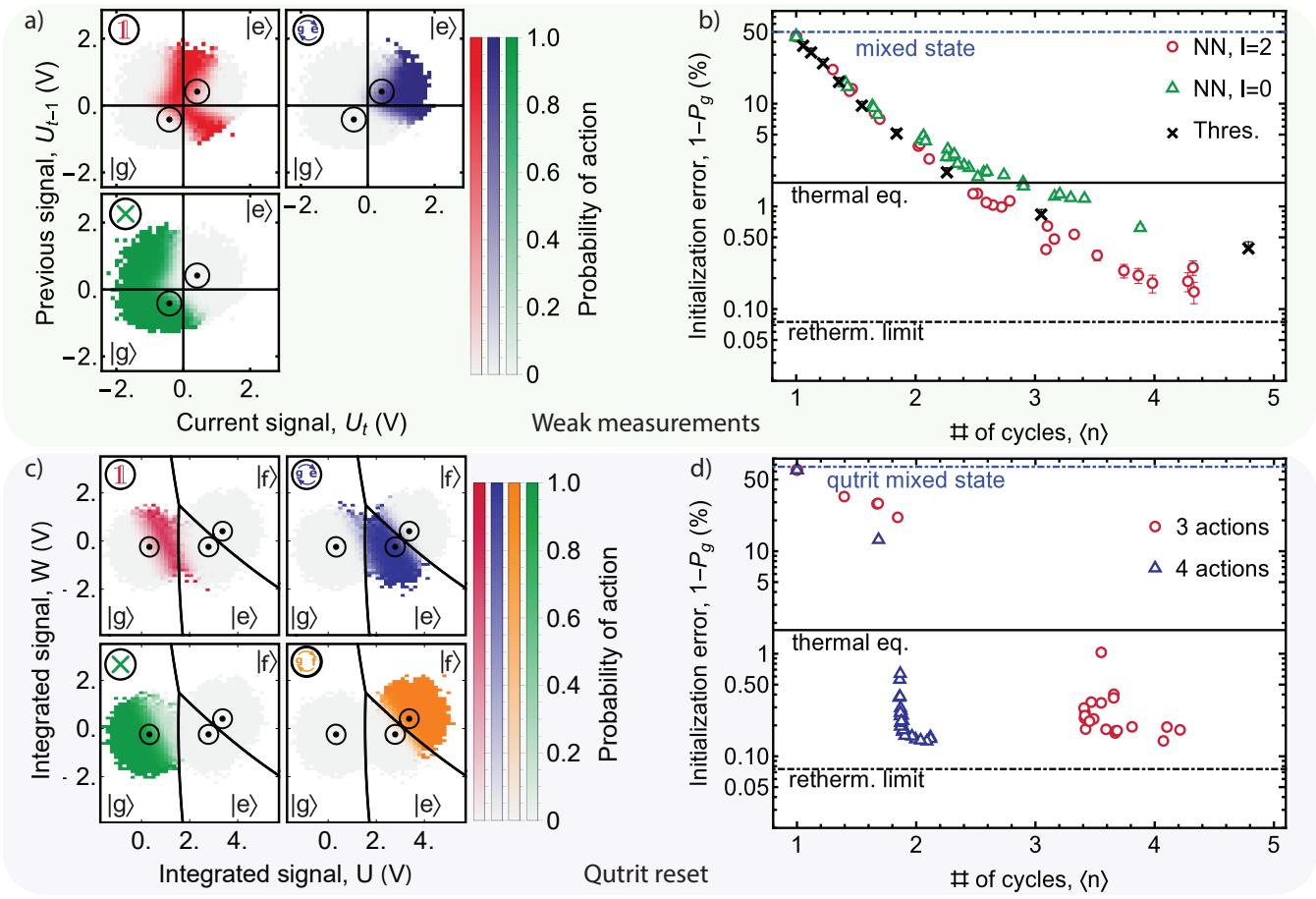
Figure 4. Reinforcement learning results for weak measurements and three-level systems. (a) Probability $P(a)$ of choosing the action indicated in the top left corner vs the signal of the current U_t and the previous U_{t-1} cycle for $l = 2$. The radii of the black circles indicate the standard deviation around the means (black dots) of the fitted bi-modal Gaussian distribution. Black lines are the state discrimination thresholds (normalized to 0, see App. B). $P(a)$ is shown for each bin with at least a single count. Empty bins are colored white (also for (d)). (b) Initialization error $1 - P_g$ vs $\langle n \rangle$ for weak measurements for an initially mixed state for $l = 2$ (red circles), $l = 0$ (green triangles) and a thresholding strategy (black crosses). Performance for $l = 1$ (not shown) is similar to $l = 2$. (c) Probability $P(a)$ of choosing the action indicated in the top left corner vs U and W. Black circles indicate the standard deviation ellipse around the means (black dots) of the fitted tri-modal Gaussian distribution. Black lines are the state discrimination thresholds (see App. B). (d) Initialization error $1 - P_g$ for a completely mixed qutrit state vs $\langle n \rangle$ when the agent can select to *idle*, *flip* and *terminate* (red circles), and when the agent can in addition perform a gf-flip (blue triangles).

ing those for which memory is beneficial. Our experiments are a first example of reinforcement learning of real-time feedback control on a quantum platform. Applying this method to larger systems will enable the discovery of new strategies for tasks like quantum error correction [33–35] and many-body feedback cooling [28–31].

DATA AVAILABILITY STATEMENT

The data produced in this work is available from the corresponding authors upon reasonable request.

AUTHOR CONTRIBUTIONS

K.R. and J.O. prepared and calibrated the experimental setup. K.R., L.B., and A.A. implemented the neural network on the FPGA. F.M. and C.E. conceived the idea for the experiment. J.L., T.F., and F.M. simulated the system and the network and determined the network structure, training algorithm and reward function, with input from K.R. and C.E.. G.J.N., M.K., A.R., and J.-C.B. fabricated the device. K.R. and J.O. carried out the experiments and analyzed the data, with support from J.L.. K.R., J.L., F.M., and C.E. wrote the manuscript with input from all co-authors. F.M., A.W., and C.E. supervised the work.

COMPETING INTERESTS

The authors declare no competing interests.

Appendix A: Experimental setup and device calibration

For the experiments, we use a transmon qubit coupled to a readout resonator on the chip shown in Fig. 5. The chip is mounted on the base temperature stage (20 mK) of a dilution refrigerator and housed inside three magnetic shields, two made from cryoperm, one from aluminum, see sketch of the experimental setup in Fig. 6. We apply microwave pulses to the chip via charge lines with 20 dB attenuation each on the 4 K, 100 mK and base temperature stage for signal conditioning [49]. To adjust the qubit frequency, we change the magnetic flux in its superconducting quantum interference device (SQUID) loop by generating currents in an inductively coupled flux line.

To readout the qubit, we generate a 256-ns-long microwave pulse at the readout frequency ω_{ro} with a microwave generator (MWG) and apply it to the readout resonator combined with its Purcell filter through the input line. The response of the resonator is then amplified by a traveling wave parametric amplifier (TWPA) with

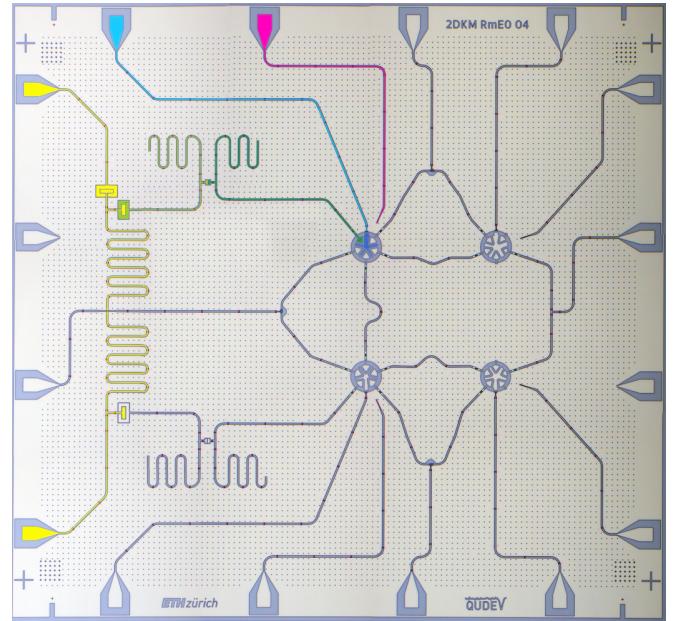|	
g-e frequency, $\omega_{ge}/2\pi$ [GHz]	6.524
e-f frequency, $\omega_{ef}/2\pi$ [GHz]	6.316
anharmonicity, $\alpha/2\pi$ [MHz]	-209
lifetime of $|e\rangle$, $T_1^{(e)}$ [µs]	13
lifetime of $|f\rangle$, $T_1^{(f)}$ [µs]	6
dephasing time of $|e\rangle$, $T_2^{\star(e)}$ [µs]	2
dephasing time of $|f\rangle$, $T_2^{\star(f)}$ [µs]	3
equilibrium excited state population, P_{therm} [%]	1.4
readout frequency, $\omega_{ro}/2\pi$ [GHz]	7.259
dispersive shift, $\chi/2\pi$ [MHz]	10.4

Table I. Measured device parameters.

Figure 5. False color optical micrograph of the sample of the used transmon qubit (blue). Depicted are the readout resonator (green) coupled to the feed line (yellow) via a Purcell filter (light green). A flux line (cyan) and a charge line (pink) couple to the qubit. Uncolored parts of the chip are not used.

20 dB gain, a high-electron mobility transistor (HEMT) and a room-temperatur amplifier (blue line in Fig. 6). We down-convert the readout signal to 250 MHz, using a local oscillator and an IQ mixer. For image rejection, we re-combine the I and Q channels of the IQ mixers using an IQ combiner, which adds a 90° phase shift to the Q channel. After further filtering and amplification, we digitize the signal with an analog-to-digital converter (ADC) and forward it to an FPGA. In the reinforcement learning approach for initializing the qubit (see main text), the agent on the FPGA then selects an action, and if *flip* is chosen, triggers an arbitrary waveform generator (AWG) (dashed green line). The AWG then plays a pre-programmed derivative removal by adiabatic gate (DRAG) pulse [50], which is up-converted to the qubit frequency using a local oscillator and an IQ mixer. We then combine this conditional pulse with a periodically-triggered pulse channel used for preparation pulses and apply it to the qubit via a charge line (green line).

Using this setup, we achieve a feedback latency, defined as the time between the end of the readout pulse and the start of the conditional π-pulse at the qubit, of $\tau_{\mathrm{EL,tot}} \approx$ 451 ns. Main contributors to the latency are the ADC ($\tau_{\mathrm{ADC}} = 160$ ns, including the delay to generate a trigger for the AWG), the AWG ($\tau_{\mathrm{AWG}} = 107$ ns) and the FPGA ($\tau_{\mathrm{FPGA}} = 144$ ns). In addition, there is $\tau_{\mathrm{G}} \approx 40$ ns delay due the signal propagation from the room-temperature electronics to the sample and back through in total about 6 meters of coaxial cable. The latency of the neural network $\tau_{\mathrm{NN}} = 48$ ns is included in the FPGA latency
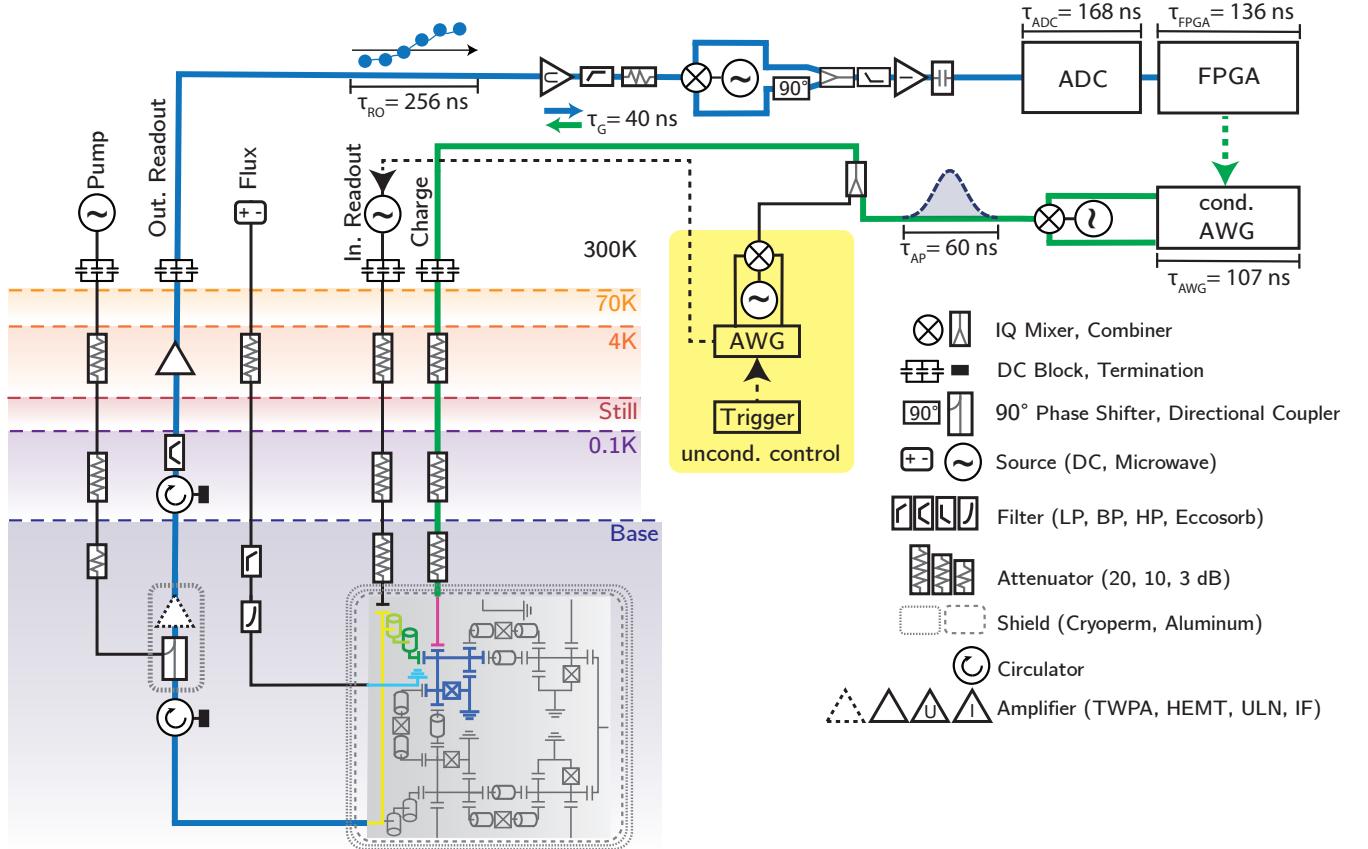
Figure 6. Experimental setup for operating the transmon qubit. For details, see text.

τ_{FPGA}; inefficient signal pre-processing is responsible for 88 ns.

Considering the readout pulse duration $\tau_{\text{RO}} = 256$ ns and the duration of the $\pi-$pulse flipping the qubit $\tau_{\text{AP}} = 60$ ns, we obtain a minimum cycle duration $\tau_{\text{cycle,min}} = \tau_{\text{RO}} + \tau_{\text{AP}} + \tau_{\text{EL,tot}} \approx 767$ ns. When including a *gf-flip*, implemented as a π-pulse on the $|f\rangle$-$|e\rangle$ manifold, followed by a π-pulse on the $|e\rangle$-$|g\rangle$ manifold ($\tau_{\text{AP}} = 112$ ns), we find $\tau_{\text{cycle,min}} = 819$ ns. For the presented experiments, we have chosen a cycle time of 856 ns for both cases, leaving room for further optimization of the cycle time by about 40 ns in future experiments.

We measure the basic device parameters presented in Table I by performing single- and two-tone spectroscopy, as well as Rabi, Ramsey and coherence measurements for the ground state to excited state and excited to second excited state transitions, as presented in Ref. [51].

Appendix B: Readout characterization and population extraction

To characterize the readout detection chain, we measure the dephasing $\beta = |\rho_{01,\text{on}}(T)|/|\rho_{01,\text{off}}(T)|$ induced by a readout pulse of length T, where $\rho_{01,\text{on}}(t)$ is time dependent off-diagonal element of the qubit's density matrix with the readout pulse present, while $\rho_{01,\text{off}}(t)$ is for without the readout pulse. We then compare β to the signal-to-noise ratio (SNR) of the processed readout signal [52], see Fig. 8(a). As, for a given dephasing β, $\sqrt{4\beta}$ is the maximum possible SNR, we define the quantum efficiency η as $\eta = \text{SNR}^2/4\beta$. We measure β in a Ramsey-like experiment, applying a readout pulse of varying amplitude in between the two $\pi/2$ pulses. For these amplitudes, we then also evaluate the SNR of the measurement signal, by preparing the qubit in $|g\rangle$ and $|e\rangle$, creating a histogram of the integrated signal U and fitting a bimodal Gaussian distribution $a_g\mathcal{N}(\mu_g, \sigma_g^2) + a_e\mathcal{N}(\mu_e, \sigma_e^2)$, with means μ_g, μ_e, variances σ_g^2, σ_e^2 and amplitudes a_g, a_e to U. The SNR is then defined as $\text{SNR}^2 = |\mu_g - \mu_e|^2/\sigma_g^2$. We observe, as expected, a linear dependence between SNR^2 and 4β, see Fig. 7, and we obtain the quantum efficiency of $\eta = 15.2\%$ from a linear fit to the data, likely due to losses before the TWPA and added noise by amplifiers after the TWPA, whose gain was not large enough to overcome other noise sources.

Furthermore, we evaluate the performance of the readout in different regimes by extracting the readout infidelity $1 - \mathcal{F}$. For this purpose, we prepare the qubit in $|g\rangle$, $|e\rangle$ (and second excited $|f\rangle$) states, after heralding the ground state with a pre-selection readout pulse, and fit a bimodal (trimodal) Gaussian distribution to the combined
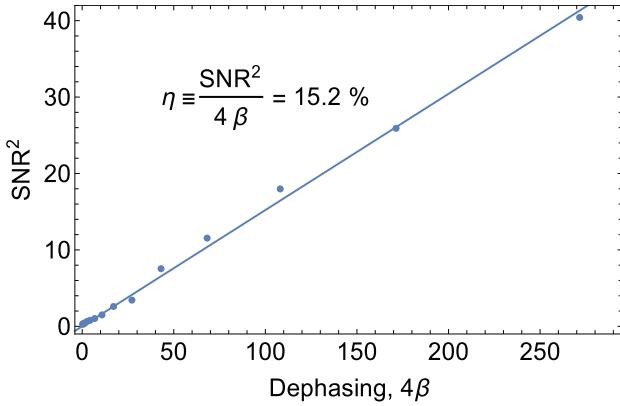
Figure 7. Squared signal-to-noise ratio SNR^2 vs four times the measurement induced dephasing 4β for different readout powers. Line is a linear fit to the data, η is extracted from the fit.

histogram of both (all) prepared states, see Fig. 8. For two-level-readout, we define a threshold $t = (\mu_g + \mu_e)/2$ and assign shots with $U < t$ to $|g\rangle$ and with $U > t$ to $|e\rangle$, see Fig. 8(a,b). By counting the missassigned shots, we extract $P(g|e)$, the probability to assign a prepared excited state to $|g\rangle$, and $P(e|g)$ the probability to assign a prepared ground state to $|e\rangle$, and obtain the readout infidelity of $1 - \mathcal{F} = \frac{1}{2}\left(P(g|e) + P(e|g)\right) = 1.95$ % for strong and 13.9 % for weak measurements. The strong measurements are limited by the decay of the excited state into the ground state during the 256 ns-long readout pulse, while overlap errors dominate in the weak measurement case. For three-level readout, we define three assignment regions based on the fitted Gaussian distributions and obtain an infidelity of $1 - \mathcal{F} = 11.3$ %, see Fig. 8(c). We note that the readout was optimized for two-level readout, resulting in the comparatively large error [48, 53] when choosing to distinguish between all three states.

These finite readout infidelities $1 - \mathcal{F}$ will lead to errors in the extraction of the initialization error $1 - P_g$ when using thresholding. Therefore we use a a different method: We first obtain the means μ_g and μ_e and variances σ_g^2 and σ_e^2 from a fit of a bimodal Gaussian distribution to the histogram of the initial equilibrium state. For the weak measurement case, we facilitate the fitting by assuming $\sigma_g^2 = \sigma_e^2$, as this is expected for low readout powers. In the three-level case, we fit a two-dimensional tri-modal Gaussian distribution $a_g \mathcal{N}(\mu_g, \Sigma_g) + a_e \mathcal{N}(\mu_e, \Sigma_e) + a_f \mathcal{N}(\mu_f, \Sigma_f)$ to the two-dimensional histogram of U and W of the initial equilibrium state, and obtain means μ_g, μ_e and μ_f and covariance matrices Σ_g, Σ_e and Σ_f. In a second step, we then fit the amplitudes a_g and a_e (a_f) to the histogram of U (and W) from the verification measurement, using the previously obtained means and variances (covariance matrices). The extracted populations are then given by the amplitude ratios $P_g = a_g/(a_g + a_e)$ and $P_e = a_e/(a_g + a_e)$ ($P_g = a_g/(a_g + a_e + a_f)$, $P_e = a_e/(a_g + a_e + a_g)$ and

$P_f = a_e/(a_g + a_e + a_f)$). We note that the binning of the shots in the histogram leads to Poissonian noise, making standard least squares fitting procedures inaccurate. Instead we use a maximum likelihood procedure as described in Ref. [54] to fit the bi-/tri-modal Gaussian distributions.

Appendix C: State discrimination with neural networks

Since qubit state initialization relies on the distinguishability between the two states given an observation $\mathbf{s}$, we also study the ability of the neural network to accomplish this task. We compare its performance with the one of a standard classifier, which integrates $\mathbf{s}$ with a set of optimal filter coefficients $\mathbf{w_s}$ to obtain $U = \mathbf{s} \cdot \mathbf{w_s}$ and assigns a state by thresholding U [43, 55]. To train the neural network in assigning the correct state, we use supervised learning [56] on a labelled data set consisting of 8212 individual time-traces in which we prepare ground and excited states after heralding an initial ground state with a pre-selection readout. The performance of the neural network classifier is evaluated based on an independent validation data set, which was interleaved with the training data set.

The two example time-traces for prepared $|g\rangle$ and $|e\rangle$ states (dashed blue and orange lines in Fig. 9(a)) become distinguishable on a timescale of about 50 ns. The fluctuations around their respective average response $\langle \mathbf{s}_g \rangle$ and $\langle \mathbf{s}_e \rangle$ (solid orange and blue lines) are dominated by Gaussian noise added during the amplification process. In addition, there are few instances in which the time-dependent signal suddenly changes its amplitude (black trace in Fig. 9(a)), indicating possible state transition events during the measurement mostly due to decay from $|e\rangle$ to $|g\rangle$.

For short measurement times τ up to 200 ns the readout infidelity $1 - \mathcal{F} = \frac{1}{2}\left(P(g|e) + P(e|g)\right)$, where $P(i|j)$ is the fraction of states prepared in state $|j\rangle$ and assigned to $|i\rangle$, decreases for both classifiers when increasing τ, see Fig. 9(b). The neural network's performance matches the readout fidelity of the standard classifier, which is known to be optimal for integration times much shorter than the qubit lifetime $\tau \ll T_1$. For longer measurement times τ, the readout infidelity of the standard classifier starts to increase because of state transitions during the measurement, which the linear filtering technique cannot resolve. In contrast, the neural network's performance further improves up until $\tau \approx 300$ ns and stays constant afterwards as the neural network is able to detect such state transition events. Thus the neural network outperforms the standard classifier in this regime. For example, the neural network classifier correctly assigned the time-trace shown in black in Fig. 9(a), while it was misclassified by the standard classifier. We note that for the two outliers in the performance of the neural network (around $\tau = 0.75$ µs and $\tau = 1.8$ µs in Fig. 9(b)) the training algorithm most likely converged to non-optimal
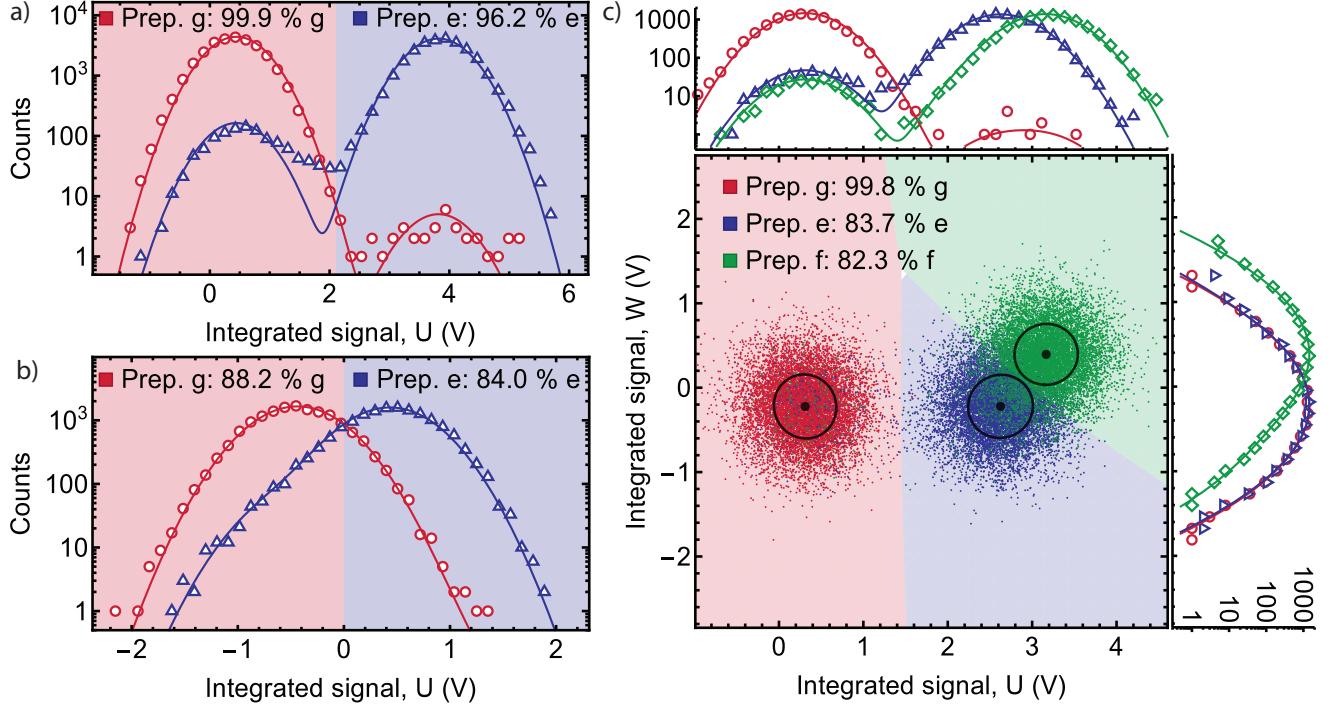
Figure 8. (a,b) Histogram of the integrated readout signal U, when preparing a ground (g, red) and excited state (e, dark blue) for (a) strong and (b) weak measurements, after heralding the ground state with a pre-selection readout pulse. Lines are a bimodal Gaussian fit to the data. (c) Two-dimensional histogram of the integrated readout signals U and W (see main text), when preparing a ground (g, red), excited (e, dark blue) or second excited (f, green) state after heralding the ground state with a pre-selection readout pulse. Black points and circles indicate the fitted means and standard deviation ellipses. Marginal distributions with the corresponding fits are shown in the top and right subpanel. For each panel, all prepared states are fitted with the same means and variances, but different amplitudes. Assignment regions are shown as background colors.

network parameters.

Appendix D: Experimental reinforcement learning

Algorithm1 Training algorithm

Require: Initial network parameters θ and ζ of the policy
    and critic network π_θ and V_ζ
    for training step=1,2,...,N_{steps} **do**
        Transfer θ to the FPGA
        Record episodes in the quantum system
        Transfer episodes to the PC
        Evaluate the critic network V_ζ
        Calculate the rewards (see Eq. (D1))
        Update θ and ζ with PPO [46, 47]
    end for

To train the network, we perform 500 training steps, i.e. 500 updates of the neural network parameters θ, in around 8 min, starting from a random policy. During each training step, the FPGA records initialization episodes with a cycle time of 856 ns and a repetition rate of 10 KHz until 1000 measurements have been carried out. The measurement outcomes and the chosen actions are transferred

Hyperparameter	Value
Adam parameter η	5×10^{-4}
Adam parameter β_1	0.98
Adam parameter β_2	0.999
Discount rate γ	0.92
Entropy coefficient	0.01
Cliprange	0.04
λ of the generalized advantage estimation	0.98
Number of training minibatches per update	1
Number of epochs for surrogate optimization	8
Maximum value for gradient clipping	∞

Table II. Hyperparameters used for training, for definition of the hyperparameters see [46, 47]

to a personal computer (PC) in around 0.6 s. The PC updates the parameters θ of the policy network π_θ within 0.1 s, and the updated parameters θ are transferred back to the FPGA in around 0.3 s.

The only accessible information during the initialization procedure is the measurement results during the feedback loop and the verification measurements; thus, the reward function can only be based on them. We choose the
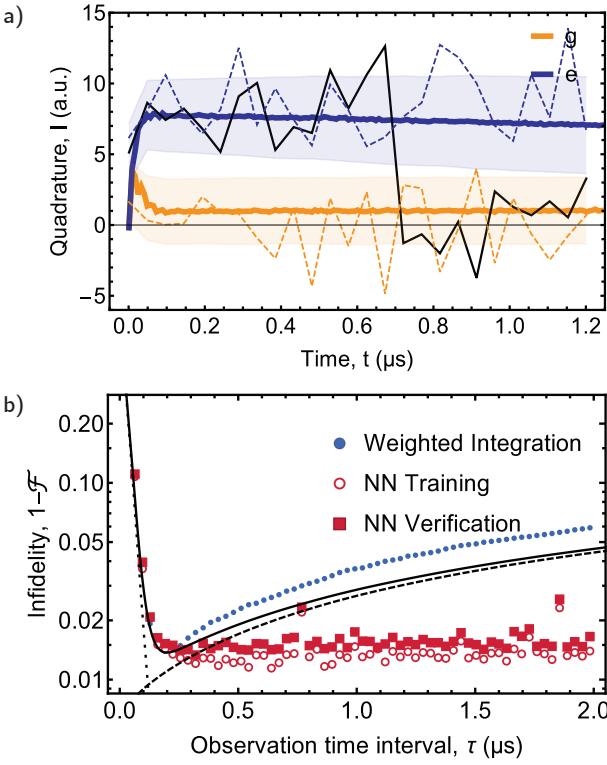
Figure 9. State discrimination via neural network. (a) Measured average (solid lines, $\pm\sigma$ standard deviation shaded) and single-shot (dotted) quadrature I (single-shot down-sampled with a six point boxcar filter) for the qubit being in the ground (orange) or the excited (dark blue) state, as well as a single-shot during a potential decay event (black). (b) Readout infidelity $1 - \mathcal{F}$ with respect to the prepared state vs observation time τ, when assigning the states with the standard classifier (blue dots) and with a trained neural network classifier for a verification data set (red squares) and a training data set (red circles), respectively. Simulated readout infidelity of the standard classifier (solid black line), considering overlap errors (dotted) and errors due to decoherence (dashed).

reward r^t within time step $t \in \{1, ..., n\}$ as

$$r^t = \frac{U_{t+1} - U_t}{U_g - U_e} - \lambda \tag{D1}$$

with the projected measurement result of the tth iteration U_t, the projected verification measurement U_{n+1} and a control parameter λ. U_g and U_e are the average projected readout signals if the qubit is prepared in the ground or excited state. $U_{t+1} - U_t$ provides the progress of the initialization compared to the previous round and gives direct information if the action a_t resulted in a quantum state closer to the target state. The parameter λ penalizes every action and thus controls the trade-off between average episode length and initialization fidelity.

The network parameters θ are modified in every update step to maximize the averaged cumulative reward $\langle R \rangle$,

defined as

$$\langle R \rangle = \left\langle \sum_{t=1}^{n} r^t \right\rangle \tag{D2}$$

where $\langle \cdot \rangle$ denotes the average over all possible episodes. With our chosen reward function, the average cumulative reward equals

$$\langle R \rangle = \frac{\langle U_{\text{ver}} \rangle - U_e}{U_g - U_e} - \lambda \langle n \rangle + \text{const.} \tag{D3}$$

The first term approximates the initialization fidelity, the second one penalizes long episodes and the constant is independent of the agent's policy.

For the training step we use the Proximal Policy Optimization (PPO) algorithm [46] from the Python library Stable Baselines [47]. In addition to the policy network on the FPGA, the PPO algorithm makes use of a second network, the so-called critic network V_ζ with its parameters ζ. Based on the current observation, the critic estimates the expected future cumulative reward given the current policy π_θ. By comparing the cumulative reward for each observation collected on the FPGA to the expectation of the critic, the PPO algorithm identifies action sequences that perform better than expected and modifies the agent's policy such that the agent is more likely to select these action sequences. The critic is only required for the update step while it is not required for the decision-making process. Therefore, the critic network is only running on the PC. We implement the critic network as a feedforward network with two hidden layers with 64 neurons per layer. All additional hyperparameters of the PPO algorithm are listed in Table II.

The whole training loop described above is summarized in Algorithm 1.

Appendix E: Low-latency neural network implemented on the FPGA

In implementing the agent's policy as a neural network on an FPGA (see Fig. 3), we aimed for an architecture which achieves high initialization fidelities while keeping processing latencies at a minimum. In the following, we discuss design considerations of the neural network architecture to reach this goal by making optimal use of the available FPGA resources.

1. Implementation of dense layers

Our network is a feedforward network and consists of multiple dense layers, each of which transforms the values of N input neurons $\mathbf{y}^{(\text{in})}$ into the values of M output neurons $\mathbf{y}^{(\text{out})}$ according to

$$y_j^{(\text{out})} = f\left(\sum_{k=0}^{N-1} w_{jk} y_k^{(\text{in})} + b_j \right) \tag{E1}$$

with $j \in \{0, ..., M-1\}$. Here, f is the nonlinear activation function, w the $M \times N$-dimensional kernel matrix and $\mathbf{b}$ the M-dimensional bias vector (where w and $\mathbf{b}$ are different for different layers).

To compute the values of all $y_j^{(\text{out})}$ in parallel and with minimum latency on the FPGA we multiply all $y_k^{(\text{in})}$ with their respective weights w_{jk} within one clock cycle of duration $\tau_{\text{clock}} = 8\,\text{ns}$ and then add them up sequentially in subsequent clock cycles, see Fig. 10. Since two subsequent additions are performed within one clock cycle, the number of summands gets reduced in each clock cycle by at most a factor $2^2 = 4$, such that the total number of clock cycles required to perform the summation is $\lceil \log_4(N+1) \rceil$, where the "+1" accounts for the bias b_j. We evaluate the nonlinear activation function f chosen to be the Rectified Linear Unit (ReLU) function in the last clock cycle. Therefore, the execution time τ_{dense} of a single dense layer is given by

$$\tau_{\text{dense}} = \tau_{\text{clock}} \left(1 + \lceil \log_4(N+1) \rceil\right), \qquad \text{(E2)}$$

In our specific experiment, we choose $N = 20$ for each layer of the low-latency network resulting in an execution time of $32\,\text{ns}$ per layer.

2. Implementation of the preprocessing network

We equip our network with a memory of the past by providing it in each cycle t with the readout signals s^j and actions a^j from l previous rounds. As this information is already available after the previous action was selected, this input is evaluated in a *pre-processing network* (see Fig. 3) while the agent is waiting to receive the most recent readout signal $\mathbf{s}^t$ of the current cycle. Thus, no additional latency is introduced to the feedback loop. To reduce the amount of data to be processed, we apply a 32-point boxcar filter to the previous measurement results before feeding them into the network. Each of the previous actions is expressed by a three-bit string. A fourth bit is added if the gf-flip action is considered. The preprocessing network consists of two layers with 12 neurons per layer.
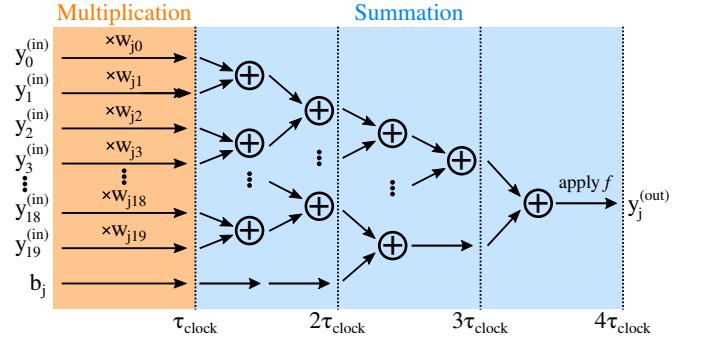

Figure 10. Evaluation for $N = 20$ input neurons on the FPGA. All output neurons are evaluated in parallel. In the first clock cycle, the inputs are multiplied with their respective weights. In the following cycles, these products and the bias are summed pairwise. Two summations are performed per clock cycle.

3. Implementation of the low-latency network

We start to evaluate the network as soon as the first element of the signal arrives. The first layer processes this information together with the output of the preprocessing network. Layer by layer, the most recent measurement data is fed into the network until the whole signal is processed.

The measurement signal, recorded with a time resolution of $1\,\text{ns}$, is down-sampled with an eight-point boxcar filter, introducing a latency of $16\,\text{ns}$. We feed four elements of the in-phase and out-of-phase component of the down-sampled signal and the output of 12 neurons from the previous layer into the subsequent layer resulting in an input size of $N = 20$.

The last layer has one output neuron per action and each neuron value encodes the probability of choosing its corresponding action. In order to sample an action, we use the Gumbel-max trick [57] which does not introduce any additional latencies.

The network execution adds a latency of $48\,\text{ns}$, where $16\,\text{ns}$ result from the eight-point boxcar filter and $32\,\text{ns}$ from the execution of the last layer.

[1] D. Ristè, C. C. Bultink, K. W. Lehnert, and L. DiCarlo, Feedback control of a solid-state qubit using high-fidelity projective measurement, Phys. Rev. Lett. **109**, 240502 (2012).

[2] Y. Salathé, P. Kurpiers, T. Karg, C. Lang, C. K. Andersen, A. Akin, S. Krinner, C. Eichler, and A. Wallraff, Low-latency digital signal processing for feedback and feedforward in quantum computing and communication, Phys. Rev. Appl. **9**, 034011 (2018).

[3] V. Negnevitsky, M. Marinelli, K. K. Mehta, H.-Y. Lo, C. Flühmann, and J. P. Home, Repeated multi-qubit readout and feedback with a mixed-species trapped-ion

register, Nature **563**, 527 (2018).

[4] L. Steffen, Y. Salathe, M. Oppliger, P. Kurpiers, M. Baur, C. Lang, C. Eichler, G. Puebla-Hellmann, A. Fedorov, and A. Wallraff, Deterministic quantum teleportation with feed-forward in a solid state system, Nature **500**, 319 (2013).

[5] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. H. Devoret, L. Jiang, and R. J. Schoelkopf, Deterministic teleportation of a quantum gate between two logical qubits, Nature **561**, 368 (2018).

[6] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. M. Girvin, L. Jiang, M. Mirrahimi, M. H. Devoret, and R. J. Schoelkopf, Extending the lifetime of a quantum bit with error correction in superconducting circuits, Nature **536**, 441 (2016).

[7] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, J. Heinsoo, J.-C. Besse, M. Gabureac, A. Wallraff, and C. Eichler, Entanglement stabilization using ancilla-based parity detection and real-time feedback in superconducting circuits, npj Quantum Information **5**, 69 (2019).

[8] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, Realization of real-time fault-tolerant quantum error correction, Phys. Rev. X **11**, 041058 (2021).

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction* (MIT press, 2018).

[10] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, Mastering the game of Go with deep neural networks and tree search, Nature **529**, 484 (2016).

[11] J. Kober, J. A. Bagnell, and J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research **32**, 1238 (2013).

[12] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, Autonomous navigation of stratospheric balloons using reinforcement learning, Nature **588**, 77 (2020).

[13] M. Praeger, Y. Xie, J. A. Grant-Jacob, R. W. Eason, and B. Mills, Playing optical tweezers with deep reinforcement learning: in virtual, physical and augmented environments, Machine Learning: Science and Technology **2**, 035024 (2021).

[14] S.-F. Guo, F. Chen, Q. Liu, M. Xue, J.-J. Chen, J.-H. Cao, T.-W. Mao, M. K. Tey, and L. You, Faster state preparation across quantum phase transition assisted by reinforcement learning, Phys. Rev. Lett. **126**, 060401 (2021).

[15] M.-Z. Ai, Y. Ding, Y. Ban, J. D. Martín-Guerrero, J. Casanova, J.-M. Cui, Y.-F. Huang, X. Chen, C.-F. Li, and G.-C. Guo, Experimentally realizing efficient quantum control with reinforcement learning, Science China Physics, Mechanics & Astronomy **65**, 250312 (2022).

[16] E. Kuprikov, A. Kokhanovskiy, K. Serebrennikov, and S. Turitsyn, Deep reinforcement learning for self-tuning laser source of dissipative solitons, Scientific Reports **12**, 1 (2022).

[17] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, *et al.*, Magnetic control of tokamak plasmas through deep reinforcement learning, Nature **602**, 414 (2022).

[18] P. Peng, X. Huang, C. Yin, L. Joseph, C. Ramanathan, and P. Cappellaro, Deep reinforcement learning for quantum hamiltonian engineering, Phys. Rev. Applied **18**, 024033 (2022).

[19] H. Tünnermann and A. Shirakawa, Deep reinforcement learning for coherent beam combining applications, Optics Express **27**, 24223 (2019).

[20] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino, Sample-efficient reinforcement learning for CERN accelerator control, Physical Review Accelerators and Beams **23**, 124801 (2020).

[21] S. Hirlaender and N. Bruchon, Model-free and Bayesian Ensembling Model-based Deep Reinforcement Learning for Particle Accelerator Control Demonstrated on the FERMI FEL, arXiv:2012.09737 (2020).

[22] Q. Yan, Q. Deng, J. Zhang, Y. Zhu, K. Yin, T. Li, D. Wu, and T. Jiang, Low-latency deep-reinforcement learning algorithm for ultrafast fiber lasers, Photonics Research **9**, 1493 (2021).

[23] S. Muiños-Landin, A. Fischer, V. Holubec, and F. Cichos, Reinforcement learning with artificial microswimmers, Science Robotics **6**, eabd9285 (2021).

[24] Y. Baum, M. Amico, S. Howell, M. Hush, M. Liuzzi, P. Mundada, T. Merkh, A. R. Carvalho, and M. J. Biercuk, Experimental deep reinforcement learning for error-robust gate-set design on a superconducting quantum computer, PRX Quantum **2**, 040324 (2021).

[25] V. Nguyen, S. B. Orbell, D. T. Lennon, H. Moon, F. Vigneau, L. C. Camenzind, L. Yu, D. M. Zumbühl, G. A. D. Briggs, M. A. Osborne, D. Sejdinovic, and N. Ares, Deep reinforcement learning for efficient measurement of quantum devices, npj Quantum Information **7**, 100 (2021).

[26] Z. Li, S. Yang, Q. Xiao, T. Zhang, Y. Li, L. Han, D. Liu, X. Ouyang, and J. Zhu, Deep reinforcement with spectrum series learning control for a mode-locked fiber laser, Photonics Research **10**, 1491 (2022).

[27] C. Chen, D. Dong, H.-X. Li, J. Chu, and T.-J. Tarn, Fidelity-Based Probabilistic Q-Learning for Control of Quantum Systems, IEEE Transactions on Neural Networks and Learning Systems **25**, 920 (2013).

[28] M. Bukov, A. G. R. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, Reinforcement learning in different phases of quantum control, Phys. Rev. X **8**, 031086 (2018).

[29] S. Borah, B. Sarma, M. Kewming, G. J. Milburn, and J. Twamley, Measurement-Based Feedback Quantum Control with Deep Reinforcement Learning for a Double-Well Nonlinear Potential, Physical Review Letters **127**, 190403 (2021).

[30] V. V. Sivak, A. Eickbusch, H. Liu, B. Royer, I. Tsioutsios, and M. H. Devoret, Model-Free Quantum Control with Reinforcement Learning, Phys. Rev. X **12**, 011059 (2022).

[31] R. Porotti, A. Essig, B. Huard, and F. Marquardt, Deep Reinforcement Learning for Quantum State Preparation with Weak Nonlinear Measurements, Quantum **6**, 747 (2022).

[32] M. Y. Niu, S. Boixo, V. N. Smelyanskiy, and H. Neven, Universal quantum control through deep reinforcement learning, npj Quantum Information **5**, 1 (2019).

[33] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, Reinforcement learning with neural networks for quantum feedback, Phys. Rev. X **8**, 031084 (2018).

[34] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215 (2019).

[35] R. Sweke, M. S. Kesselring, E. P. van Nieuwenburg, and J. Eisert, Reinforcement learning decoders for fault-tolerant quantum computation, Machine Learning: Science and Technology **2**, 025005 (2020).

[36] Y.-H. Zhang, P.-L. Zheng, Y. Zhang, and D.-L. Deng, Topological Quantum Compiling with Reinforcement Learning, Physical Review Letters **125**, 170501 (2020).

[37] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, Quantum circuit optimization with deep reinforcement learning, arXiv:2103.07585 (2021).

[38] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Machine learning and the physical sciences, Review of Modern Physics **91**, 045002 (2019).

[39] A. Dawid, J. Arnold, B. Requena, A. Gresch, M. Płodzień, K. Donatella, K. A. Nicoli, P. Stornati, R. Koch, M. Büttner, R. Okuła, G. Muñoz-Gil, R. A. Vargas-Hernández, A. Cervera-Lierta, J. Carrasquilla, V. Dunjko, M. Gabrié, P. Huembeli, E. van Nieuwenburg, F. Vicentini, L. Wang, S. J. Wetzel, G. Carleo, E. Greplová, R. Krems, F. Marquardt, M. Tomza, M. Lewenstein, and A. Dauphin, Modern applications of machine learning in quantum sciences, arXiv:2204.04198 (2022), arXiv:2204.04198 [quant-ph].

[40] M. Krenn, J. Landgraf, T. Foesel, and F. Marquardt, Artificial intelligence and machine learning for quantum technologies, arXiv:2208.03836 (2022).

[41] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation, Phys. Rev. A **69**, 062320 (2004).

[42] A. Wallraff, D. I. Schuster, A. Blais, L. Frunzio, J. Majer, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, Approaching unit visibility for control of a superconducting qubit with dispersive readout, Phys. Rev. Lett. **95**, 060501 (2005).

[43] J. Gambetta, W. A. Braff, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, Protocols for optimal readout of qubits using a continuous quantum nondemolition measurement, Phys. Rev. A **76**, 012325 (2007).

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) http://www.deeplearningbook.org.

[45] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Computation **9**, 1735 (1997).

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal Policy Optimization Algorithms, arXiv:1707.06347 [cs] (2017).

[47] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, Stable baselines, https://github.com/hill-a/stable-baselines (2018).

[48] P. Magnard, P. Kurpiers, B. Royer, T. Walter, J.-C. Besse, S. Gasparinetti, M. Pechal, J. Heinsoo, S. Storz, A. Blais, and A. Wallraff, Fast and unconditional all-microwave reset of a superconducting qubit, Phys. Rev. Lett. **121**, 060502 (2018).

[49] S. Krinner, S. Storz, P. Kurpiers, P. Magnard, J. Heinsoo, R. Keller, J. Lütolf, C. Eichler, and A. Wallraff, Engineering cryogenic setups for 100-qubit scale superconducting circuit systems, EPJ Quantum Technology **6**, 2 (2019).

[50] F. Motzoi, J. M. Gambetta, P. Rebentrost, and F. K. Wilhelm, Simple pulses for elimination of leakage in weakly nonlinear qubits, Phys. Rev. Lett. **103**, 110501 (2009).

[51] J.-C. Besse, K. Reuer, M. C. Collodo, A. Wulff, L. Wernli, A. Copetudo, D. Malz, P. Magnard, A. Akin, M. Gabureac, G. J. Norris, J. I. Cirac, A. Wallraff, and C. Eichler, Realizing a deterministic source of multipartite-entangled photonic qubits, Nat. Commun. **11**, 4877 (2020).

[52] C. C. Bultink, B. Tarasinski, N. Haandbæk, S. Poletto, N. Haider, D. J. Michalak, A. Bruno, and L. DiCarlo, General method for extracting the quantum efficiency of dispersive qubit readout in circuit qed, Appl. Phys. Lett. **112**, 092601 (2018).

[53] S. Krinner, N. Lacroix, A. Remm, A. D. Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, Realizing repeated quantum error correction in a distance-three surface code, Nature **605**, 669 (2022).

[54] T. A. Laurence and B. A. Chromy, Efficient maximum likelihood estimator fitting of histograms, Nature Methods **7**, 338 (2010).

[55] T. Walter, P. Kurpiers, S. Gasparinetti, P. Magnard, A. Potočnik, Y. Salathé, M. Pechal, M. Mondal, M. Oppliger, C. Eichler, and A. Wallraff, Rapid, high-fidelity, single-shot dispersive readout of superconducting qubits, Phys. Rev. Appl. **7**, 054020 (2017).

[56] B. Lienhard, A. Vepsäläinen, L. C. G. 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. Applied **17**, 014024 (2022).

[57] I. A. M. Huijben, W. Kool, M. B. Paulus, and R. J. G. van Sloun, A Review of the Gumbel-max Trick and its Extensions for Discrete Stochasticity in Machine Learning, arXiv:2110.01515 [cs, stat] (2021).