Real-time Quantum Control with Reinforcement Learning

Integrating QPU and GPU using DGX Quantum

Dean Poulos[1] , Ramon Szmuk[1], Oded Wertheim[1], Avishai Ziv[1], Benedikt Dorschner[2] , Sam Stanwyck[2] , Jin-Sung Kim[2] , Yonatan Cohen[1]

[1]**Quantum Machines,** [2]**NVIDIA**

1 Facilitating advancements in quantum computing

The **tremendous rate of scaling of QPUs,** along with **improvements** in **gate and readout fidelities**, opens the path to fault-tolerant computations using quantum error correction (QEC) with ensembles of qubits. These improvements require complementary *classical* compute in the form of FPGA-based pulse-processing instrumentation and high-powered compute like GPUs.

2 Why do we need *real-time* hybrid compute?

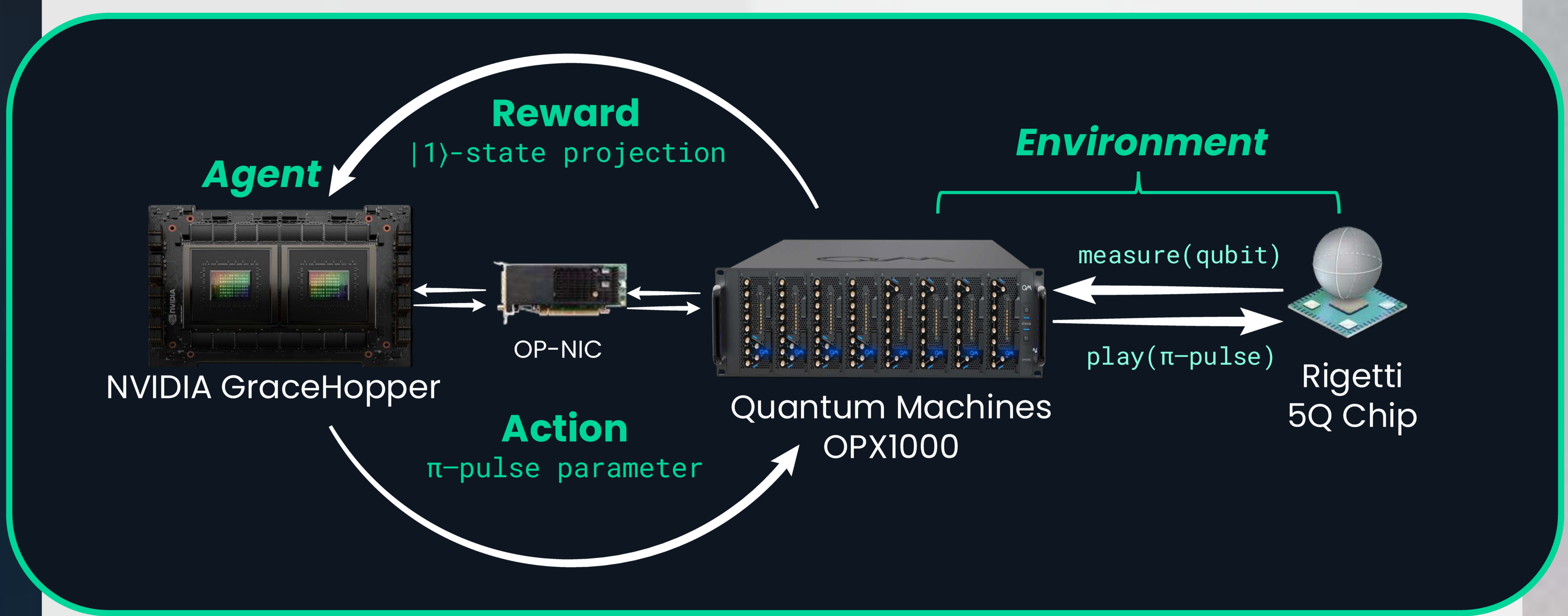
Fault-tolerant quantum computation requires real-time, mid-circuit decoding of logical qubit states by observing ancilla qubit states to eliminate errors over the lifetime of a quantum algorithm, necessitating the tight integration of classical and quantum compute. In addition to QEC, multi-qubit systems require precise control over numerous degrees of freedom to maintain fidelities below error-correction thresholds, including repeated re-calibration and low-latency feedback routines to combat high-frequency noise.

So, real-time classical compute plays a critical role in enabling both:

1. Low-latency, feedback-based **optimal control**.
2. Real-time decoding for **error-correction** of logical qubit states.

3 Model-free reinforcement learning for quantum optimal control in hardware

We used a control system combining the **Quantum Machines** OPX controller, with an **NVidia** Grace Hopper Superchip to optimize the control of quantum NOT gate on a **Rigetti** 5Q Novera Quantum Computer, in the **IQCC** (Israeli Quantum Computing Center).



Low-latency communication between the GH and OPX systems was achieved via the **DGX Quantum platform**, enabling data transfer from the FPGA-based pulse-programming language QUA to the GH superchip in ***under 4 microseconds, round-trip!***

We use the **TD3 algorithm** from StableBaselines3, making use of OpenAI's open-source RL implementations in **Python** and **CUDA**, to optimize the continuous parameter space of our qubit. The implementation of TD3 that was used strikes a strong balance between exploration and exploitation by applying random action noise and delaying network updates for early iterations.

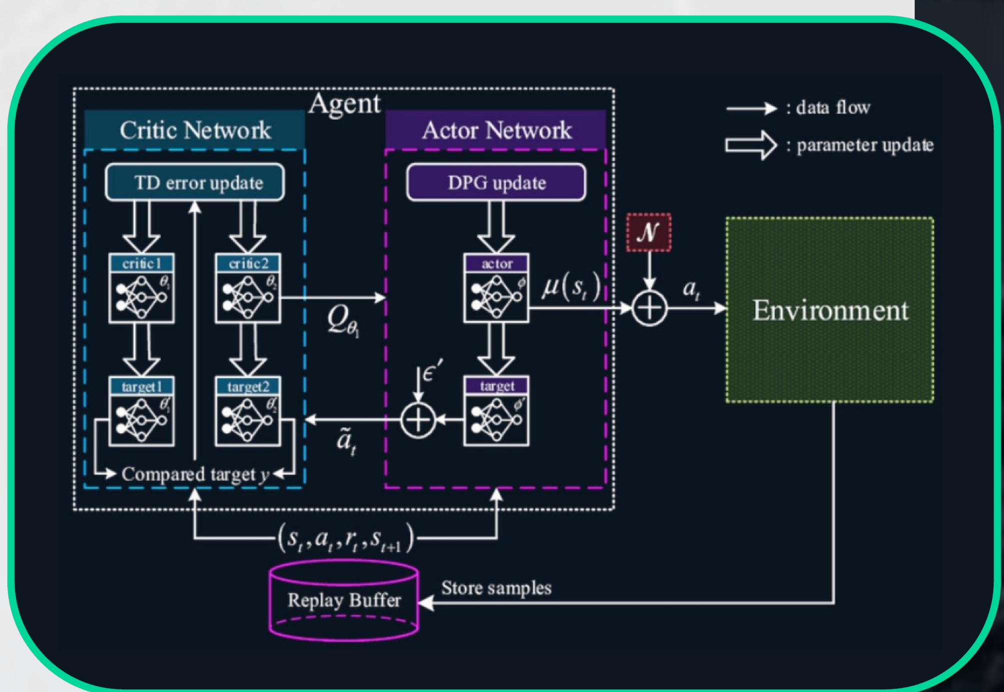
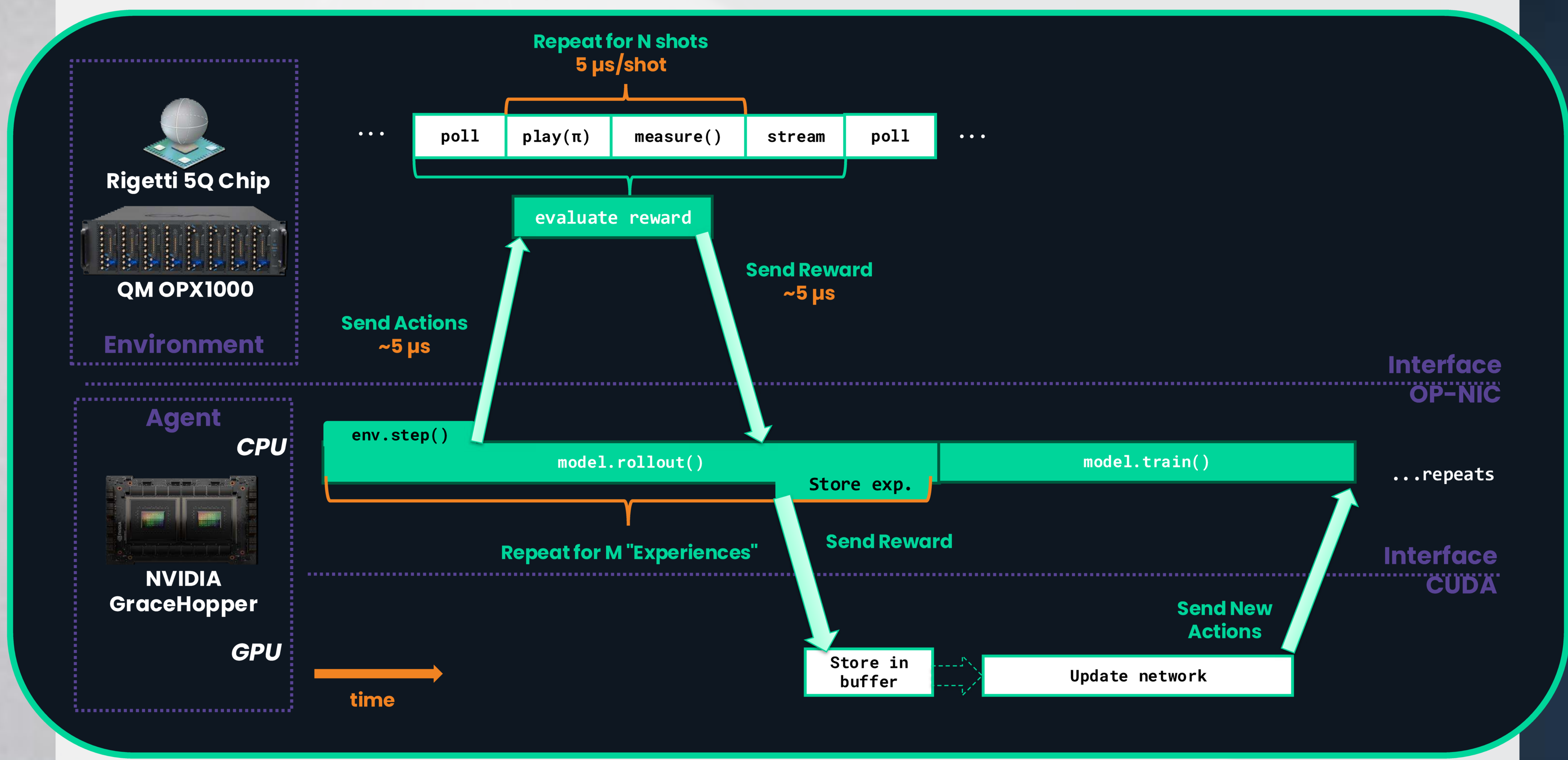


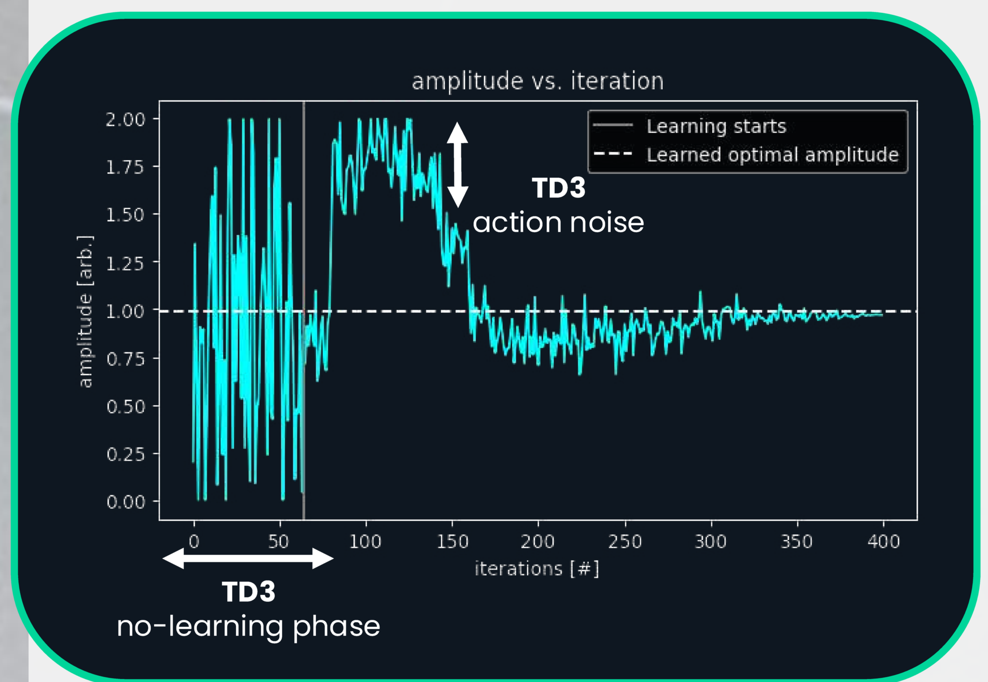
Figure by Liu et. al, 2022

4 Real-time multi-parameter optimization of a quantum NOT-gate using RL

Using the model-free reinforcement learning protocol described, **any continuous qubit parameter can be optimized**, the most straightforward being the amplitude of the gate-pulse. In the following demonstration, the agent provides the OPX with a normalized amplitude, which is then used to modulate a NOT-gate pulse waveform in real-time on the FPGA. Then, the qubit is measured, and the state-projection reward is returned to the agent to update the policy and learn.

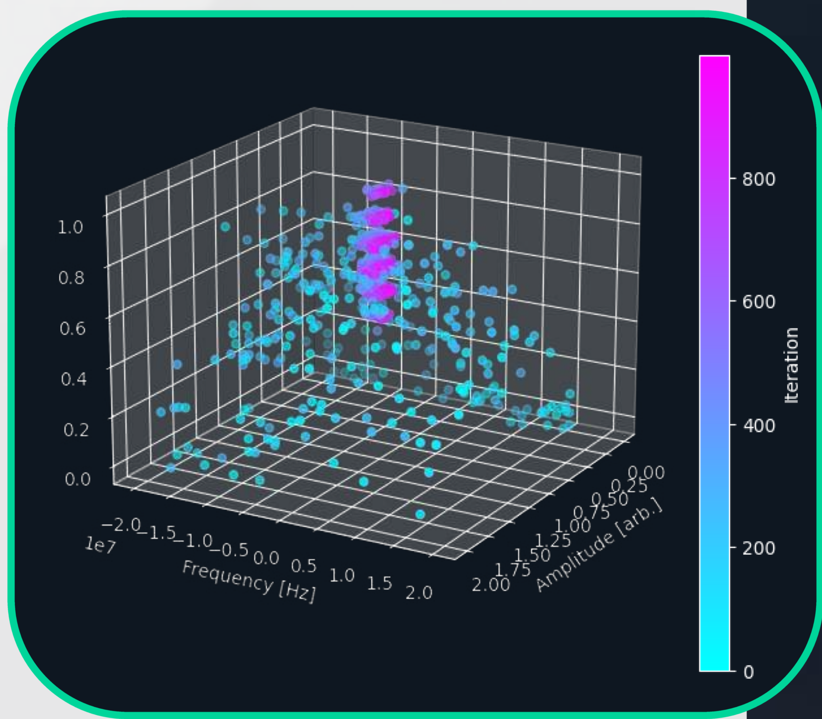


The communication overhead of the DGX is *so low* (~4µs round-trip) that it is *no longer* the bottleneck of the calibration routine, **enabling ultra-fast feedback**. Each "*experience*", including action/reward communication between the GH and OPX, qubit operation, readout, and policy network update, takes on average ***~2.5ms*** using the open-source RL protocol. This is encouragingly *less* than the *total* expected communication overhead using ethernet.



The model is initialized with a random initial amplitude, and all other qubit parameters are calibrated. The agent is constrained to explore between 0-2 times the known, correct NOT-gate amplitude. With network updates only every 8 iterations, the model successfully converges to the known amplitude in under 400 iterations, totaling ~1s.

The action space was expanded to include the qubit frequency as well. The multi-dimensional optimization successfully converged in ~1000 iterations to achieve the maximum possible reward. Despite adding another dimension, the NVIDIA Hopper GPU remains heavily *under-utilized,* indicating the ability to scale to *much larger* parameter spaces. The remaining bottleneck is the communication between the CPU and GPU of the GH due to the open-source protocol's implementation in Python.



5 Larger-scale applications

Future applications of the DGX quantum platform aim to leverage the full power of the GH superchip and benefit fault-tolerant quantum computing at larger scales, including:

- General, real-time single- and **two-qubit gate optimization.**
- Real-time **stabilizer round optimization** for quantum error correction.