# Probabilistic Inference for Quantum Programs

Charles Yuan
MIT CSAIL
Cambridge, MA, USA

## Abstract

The design of quantum computers and quantum algorithms requires the ability to reason about uncertainty due to the inevitability of quantum noise on today's hardware. Meanwhile, probabilistic inference can be used to build informative models that learn latent parameters of interest from observations. We propose a probabilistic and quantum programming language featuring automatic inference of error models from empirical measurement data, streamlining and generalizing our ability to program on noisy systems.

## 1 Introduction

Designing algorithms to execute on quantum computers requires grappling with inherent quantum noise, as foreseeable quantum hardware does not perfectly isolate qubits from their environment and can introduce e.g. random bit flip errors at rates that are orders of magnitude higher than classical digital circuits [14]. Though quantum error-correcting codes [17] have been devised to mitigate the destructive effect of noise in computation, they generally require too many physical qubits to be deployed extensively today. Consequently, quantum hardware does not preserve the semantics of idealized constructs in existing programming languages.

Developing a quantum program that satisfies its specification in the presence of noise requires an error model that describes possible errors and their likelihood of occurrence. For example, an error model may specify that any quantum gate is followed by a bit flip error with probability $p$. Commonly modeled errors include bit flips, represented by application of the Pauli-X gate, and phase flips by the Pauli-Z gate. The theory of quantum channels models more complex errors, such as depolarization and amplitude damping [14]. Given an error model, a developer can simulate the potential effects of errors or statically reason about the reliability of their program. Models can be imported into commercially-available simulators [11], which then generate a distribution of output measurements reflecting appropriate noise. There has also been recent interest in using error models to statically quantify the reliability of quantum programs [10].

There are several contemporary methodologies for developing error models. Quantum process tomography is the most general technique for determining the density matrix, and thereby error characteristics, of a quantum circuit. However, ascribing circuit errors to individual gate errors is difficult and scales exponentially with the number of qubits [4]. Randomized benchmarking is a popular alternative to determine gate-level error that executes randomized circuits and fits their average fidelity to a statistical model that predicts average error per gate. Unlike process tomography, it scales polynomially in the number of qubits and is robust to errors in state preparation and measurement [4].

## 2 Research Problem

The standard quantum error quantification workflow requires the manufacturer or some other experimenter to specify a error model for programmers to use, which has several limitations. First, the model may not account for more sophisticated forms of noise such as correlated qubit error [14]. Second, programmers must either depend on another party to provide an accurate model or invest heavily in benchmarking techniques before they can be productive. Third, programmers may want embrace noise in programs, as is done for variational quantum circuits [16], and avoid overestimation of noise by a conservative error model.

As an alternative, we propose *synthesizing* an error model for a quantum circuit on a hardware device, by executing the circuit on the noisy device and simultaneously feeding observed outputs into a probabilistic inference system. The same quantum program will specify both a hardware circuit and a probabilistic model over unknown error probabilities. In this way, the error model becomes *programmable*, freeing the programmer from the restrictions of the traditional workflow and opening the door to writing programs supporting a broader range of errors and hardware devices.

Given the observed outputs of a system, *probabilistic inference* can determine latent parameters and output new predictions or compute outcome probabilities conditioned on specific events. Probabilistic programming languages [2, 5–7, 13, 15] enable inference through primitive operations that build, manipulate, and sample from probability distributions. These languages can utilize sampling algorithms like Sequential and Markov Chain Monte Carlo to compute and sample from probability distributions efficiently.

A first step in our project is to extend a standard quantum programming language, such as the `while`-language of [18], to include probabilistic programming operators such as `observe`. As an illustrative example, consider the program in Figure 1, which constructs a Bell state, then applies bit flip error to each bit with probability `x_err_p`. In this program, the `observe` statement effectively measures `q0` and post-selects for only the outcome `obs_q0`. This program defines a probability distribution over the remaining unobserved `q1`. We may sample from this distribution and determine that `q1` is likely equal to `obs_q0`, with some probability of discrepancy

```
f :: Float -> Bool -> Quantum Bool
f x_err_p obs_q0 = do
    q0 <- qubit          -- initialize to zero
    q1 <- qubit
    hadamard q0
    qif q0 (not q1) q1 -- establish Bell state
    prob_gate not x_err_p q0 -- bit flip error
    prob_gate not x_err_p q1
    observe q0 obs_q0      -- observation data
    return q1              -- unobserved qubit
```

**Figure 1.** Probabilistic observation of a Bell pair.

based on `x_err_p`. Explicitly specifying `obs_q1` would allow us to compute the likelihood of that outcome and enable ML estimation of the parameter `x_err_p`.

The classical meaning of `observe` is to establish a probability distribution conditioned on its argument being true. In a quantum language, we may define it as a measurement followed by conditioning, or post-selection, on the measurement result being $|1\rangle$. The quantum `while`-language possesses a denotational semantics based on density matrices given in [18], and we may extend the semantics with the usual definition of quantum measurement (Appendix A).

Suppose we have $\theta$, error model parameters such as gate probabilities, and $\mathcal{D}$, observation samples for a subset of qubits. Then, the program defines a probability distribution over unobserved measurement outcomes, which we can use to evaluate the likelihood function $p(\theta \mid \mathcal{D})$. The program is then amenable to standard techniques for inference and learning, such as maximum likelihood estimation. As we continue executing the program and obtaining more observational data, we build an increasingly accurate error model. At convergence, the probabilistic model should be able to accurately simulate program execution on the noisy hardware.

## 3 Preliminary Results

We have implemented a proof-of-concept of the several parameter learning scenarios in Mathematica and used its symbolic optimization facilities to perform ML estimation of the classical error parameter. Specifically, we studied two scenarios, the first assuming that we have access to the *full reconstructed state* at program termination (i.e. through tomography) and the second assuming we only have access to *measurements* in some basis.

In the first scenario, we modeled a two-qubit system initialized in the Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ with an independent phase flip error with probability $p$ on each qubit. Each phase error flips the sign of $|11\rangle$, and two errors cancel each other out, so there are effectively two possible outcomes, $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and $\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$. The probability of the first is the probability of zero or two errors, $p^2 + (1-p)^2$, and the second is the probability of one error, $2p(1-p)$, from which we compute a binomial likelihood function of observing $m$ instances of the first and $n$ instances of the second. Computing the MLE for $p$ at various values of $m$ and $n$ yields results consistent with our expectations. When $m = n = 50$, we infer $p = 50\%$, and increasing $m$ to 1000 results in $p = 2.44\%$ and increasing to 2000 results in $p = 1.24\%$, consistent with the understanding that observing fewer instances of sign flips indicates a less likely occurrence of error.

In the second scenario, we modeled a similar two-qubit Bell state subject to various independent errors on each qubit, and observed computational basis measurements of both qubits. In the case of bit flip error with probability $p$, we computed the probability of observing either $|00\rangle$ or $|11\rangle$ to be $\frac{1}{2} - p + p^2$ and $|01\rangle$ or $|10\rangle$ to be $p(1-p)$. Again, ML estimation produced estimates for $p$ consistent with our understanding that more instances of differing qubits indicates more likely error. When we set the ratio of instances of $|00\rangle$ or $|11\rangle$ to $|01\rangle$ or $|10\rangle$ to be 100:1, we inferred a small $p = 0.50\%$. Decreasing the ratio to 5:1 yielded $p = 9.18\%$, 5:3 yielded $p = 25\%$, and 1:1 yielded $p = 50\%$.

We also performed the second experiment on phase flip, amplitude damping, and depolarization error models. As expected, we were unable to infer the value of $p$ for phase flips because measurements in the computational basis are uncorrelated with phase error. For the other error models, we obtained similar results as the bit flip case, with the additional observation that the impact of observing $|11\rangle$ outcomes differed from $|00\rangle$ outcomes for amplitude dampening, since it affects excited $|1\rangle$ states differently from ground $|0\rangle$ states.

## 4 Discussion

A more realistic implementation of this programming language would require additional constructs that bridge quantum states and classical probability distributions. The language runtime should automate the computation of likelihood functions, possibly building off an existing density matrix simulator such as [11]. We intend to evaluate the language by implementing circuits of larger size and with a diverse set of possible errors. Other potential applications include simulating circuits with parameters themselves sampled from probability distributions, such as learning variational quantum circuits from data, or possibly expressing classical post-processing of quantum algorithms that require repeated re-sampling such as Shor's algorithm.

Numerous quantum operations have an intuitive connection to classical probability theory. For example, measurement corresponds to sampling from a distribution, and density matrix reduction corresponds to marginalization and conditioning. By integrating the two, we hope to apply knowledge about efficient probabilistic inference algorithms to quantum programming and to lower the conceptual difficulty of quantum by connecting it to probability.

Remaining challenges include optimizing the likelihood function, which is possibly non-convex, as well as scaling program execution as qubits, observation data, and program complexity increase. We would also like to develop the theoretical underpinning of probabilistic primitives in quantum programming. For example, post-selection was shown in [1] to strictly strengthen the computational power of quantum computers, a fact we must reconcile with our goals.

## 5 Related Work

The probabilistic nature of quantum programs is well-known, but generally only at the coarse level of the final joint distribution output by a simulator. Inference as a central tool of quantum simulation has recently become of interest, as in [9] who compiled quantum circuits to probabilistic graphical models that could be optimized and re-parameterized efficiently [3]. proposed a simulation technique for quantum circuits with error that leverages probabilistic sampling. The field of variational circuits studies optimization of classical parameters in circuits, applying techniques like automatic differentiation to quantum programs [12]. Probability as a natural tool for studying noise has been developed by, among others, [10], who devised a program logic to bound the accumulated error of a program under an error model. Finally, probabilistic inference is a broadly applicable tool in fields of science like quantum physics, where [8] has made available a software suite enabling inference and learning of quantum systems. We hope that the natural connection between probabilistic inference and physical experimentation will also lead to interesting applications in quantum computing.

## References

[1] Scott Aaronson. 2005. Quantum Computing, Postselection, and Probabilistic Polynomial-Time. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 461 (2005).

[2] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research* 20, 28 (2019), 1–6.

[3] Himanshu Chaudhary, Biplab Mahato, Lakshya Priyadarshi, Naman Roshan, Utkarsh, and Apoorva D. Patel. 2019. A Software Simulator for Noisy Quantum Circuits.

[4] J. M. Chow, J. M. Gambetta, L. Tornberg, Jens Koch, Lev S. Bishop, A. A. Houck, B. R. Johnson, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf. 2009. Randomized Benchmarking and Process Tomography for Gate Errors in a Solid-State Qubit. *Physical Review Letters* 102, 9 (Mar 2009).

[5] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. 2019. Gen: A General-Purpose Probabilistic Programming System with Programmable Inference. In *Conference on Programming Language Design and Implementation.*

[6] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. In *International Conference on Artificial Intelligence and Statistics.*

[7] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic Programming. In *Future of Software Engineering.*

[8] Christopher Granade, Christopher Ferrie, Ian Hincks, Steven Casagrande, Thomas Alexander, Jonathan Gross, Michal Kononenko, and Yuval Sanders. 2017. QInfer: Statistical inference software for quantum applications. *Quantum* 1 (2017).

[9] Yipeng Huang, Steven Holtzen, Todd Millstein, Guy Van den Broeck, and Margaret Martonosi. 2021. Logical Abstractions for Noisy Variational Quantum Algorithm Simulation. In *Architectural Support for Programming Languages and Operating Systems.*

[10] Shih-Han Hung, Kesha Hietala, Shaopeng Zhu, Mingsheng Ying, Michael Hicks, and Xiaodi Wu. 2019. Quantitative robustness analysis of quantum programs. In *Symposium on Principles of Programming Languages.*

[11] IBM. 2018. Qiskit. https://qiskit.org.

[12] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. 2020. Yao.jl: Extensible, Efficient Framework for Quantum Algorithm Design. *Quantum* 4 (Oct 2020).

[13] P. Narayanan, J. Carette, Wren Romano, Chung chieh Shan, and Robert Zinkov. 2016. Probabilistic Inference by Program Transformation in Hakaru (System Description). In *International Symposium on Functional and Logic Programming.*

[14] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press.

[15] A. Pfeffer. 2009. Figaro : An Object-Oriented Probabilistic Programming Language.

[16] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018).

[17] Peter W. Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* 52 (Oct 1995). Issue 4.

[18] Mingsheng Ying. 2012. Floyd–Hoare Logic for Quantum Programs. *ACM Trans. Program. Lang. Syst.* 33, 6 (2012).

## A Syntax and Semantics

We may add the `observe` operator to the quantum `while`-language:

$$\begin{aligned} s ::= &\ \texttt{skip} \mid q := |0\rangle \mid q := U[q] \mid s_1; s_2 \\ &\mid \texttt{case}\ M[q]\ \texttt{of}\ \ldots \mid \texttt{while}\ M[q]\ \texttt{do}\ s \mid \texttt{observe}(q) \end{aligned}$$

$$[\![\texttt{observe}(q)]\!]\rho = \frac{M_q \rho M_q^\dagger}{\mathrm{tr}(\rho M_q)}$$

where $M_q$ projects qubit $q$ into the subspace corresponding to a measurement of $|1\rangle$.