# Goose: an OCaml environment for quantum computing

Denis Carnier[1], Arthur Correnson[2], Christopher McNally[3], and Youssef Moawad[4]

[1] imec-DistriNet, KU Leuven
[2] Ecole Normale Supérieure de Rennes
[3] Massachusetts Institute of Technology
[4] University of Glasgow

**Abstract**

Quantum computing is an emerging model of computation that exploits non-classical effects like superposition and entanglement to achieve algorithmic speedups. A radical break from classical computation, the model presents new challenges for programming languages research. In this presentation, we showcase Goose: an OCaml library to model, simulate and compile low-level quantum programs. Goose is designed to support research into quantum programming languages through an emphasis on ease of use and extensibility. The library is compatible with the OpenQASM standard, and targets a variety of backends with a minimalistic circuit-based IR.

## 1 Introduction

**Context** *Quantum computing* [8] ... Despite these performance promises and recent advances in the hardware implementation of quantum computers, widespread access remains limited. Furthermore, the noisy, intermediate-scale quantum (NISQ [?]) quantum systems available today are error-prone and can only maintain their coherence for a limited time. In addition, they typically do not have enough resources (*qubits*, or quantum bits) to run meaningful algorithms and there may be further restrictions in terms of allowed operations [6].

1: What is coher-
ce? Will readers
ow?

1: What does 'mean-
ful' mean?

1: such as?

1: We didn't talk
out the *unpre-
tability* of errors
ove.

1: What is the trade-
for classical simula-
n? Will it be obvi-
s?

As an alternative to physical quantum computers, simulation on classical computers allows researchers to study quantum algorithms without such restrictions and with predictable errors. Additionally, simulators enable researchers to artificially simulate errors to predict the behaviour of their algorithms on NISQ-era quantum computers . Running and simulating quantum algorithms is only one half of the big picture. The other half is to try and find expressive ways to design and program quantum algorithms. This presents new challenges for programming language researchers. For this reason, the programming language community needs accessible tools for compilation and classical simulation of quantum programs.

In this presentation, we showcase Goose: an OCaml library to model, simulate and compile quantum programs. This library is designed to act as a playground to experiment with quantum programming in the OCaml ecosystem.

**Related Work** The past decade has seen the development of several quantum Programming Languages (QPLs). These languages are designed to be used by researchers to write quantum programs that can be executed on a variety of backends.

OpenQASM [4] is a quantum assembly language designed to be a common intermediate representation for quantum computing. We support a subset of OpenQASM in our parser. Qiskit [1] is a python-based toolchain and eDSL for specifying and compiling quantum circuits for simulation and deployment on quantum hardware. Other tools and QPLs include Cirq [3], Quil [9], and Silq [2]. A number of Functional QPLs have also been developed, including

Quipper [5], written in Haskell, VOQC [7], in Coq, and Q# [10], in F#. We are contributing to this ecosystem with the development of a toolchain with similar functionality in OCaml.

To our knowledge there are no other actively-maintained OCaml libraries for quantum programming. The OCaml ecosystem is well-suited to the development of such a library, as it provides a rich set of libraries for numerical computation, and a powerful type system that can be used to ensure correctness of quantum programs .

CM: This makes sens
for a shallow embed-
ding, but are we real
do that? The type sy
tem of the metalan-
guage helps ensure th
correctness of the int
preter, but not neces
sarily the programs i
interprets, no?

## 2 The Goose Library

**What is Goose?** Goose is an OCaml library for quantum computing. In its core, it features a simple intermediate representation to model quantum circuits. The library is built around this IR and provides a modular API to build and customize new tools such as simulators, compilers and static analyzers for quantum programs.

**Why?** Quantum computing is not accessible to a wide audience. This can be explained by several factors. (1) Maths/Physics are scary, (2) availability of quantum computers to non-experts is limited, (3) computer science in itself is perceived as hard. Recent projects [] tried to overcome these limitations by developing more resources to make quantum computing more accessible to computer scientists. Goose follows the same track and try to provide a well-documented playground to start experimenting with the science of quantum computing and the activity of quantum programming. By choosing OCaml as an implementation language, we also hope to create interesting connections with the functional programming language community.

rewrite

rewrite

**Goose's present and future** Goose is open source and freely available at the address `https://www.sillygoose.fr`. Currently, the library's front-end supports the OpenQASM 2.0 circuit representation, a *de facto* standard for interoperability between quantum computing tools, in addition to a textual description of the IR. Goose also implements a customizable simulator for quantum circuits. To demonstrate the benefits of the modular design, we derive a symbolic simulator and a C compiler from the circuit simulator. These tools are distributed as a part of the library.

CM: What does 'cus-
tomizable' mean?

In the future, we could leverage Goose's modularity to connect Goose with other existing projects such as VOQC, a formally verified compiler for quantum circuits that can be extracted to OCaml.

CM: What does it
mean that the library
is "modular?" How
does that *actually ca
out?* We never de-
scribed the architectu
and API or attempt
justify those engineer
ing choices!

## References

[1] Qiskit: An open-source framework for quantum computing.

[2] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: A high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '20, pages 286–300, New York, NY, USA, June 2020. Association for Computing Machinery.

[3] Cirq Developers. Cirq, July 2018.

[4] Andrew W. Cross, Ali Javadi-Abhari, Thomas Alexander, Niel de Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):12, September 2022.

[5] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, June 2013. Association for Computing Machinery.

[6] Laszlo Gyongyosi and Sandor Imre. A survey on quantum computing technology. *Computer Science Review*, 31:51–71, 2019.

[7] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for Quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):37, January 2021.

[8] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[9] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture, 2016.

[10] Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher E. Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, RWDSL '18, page 7, New York, NY, USA, February 2018. Association for Computing Machinery.