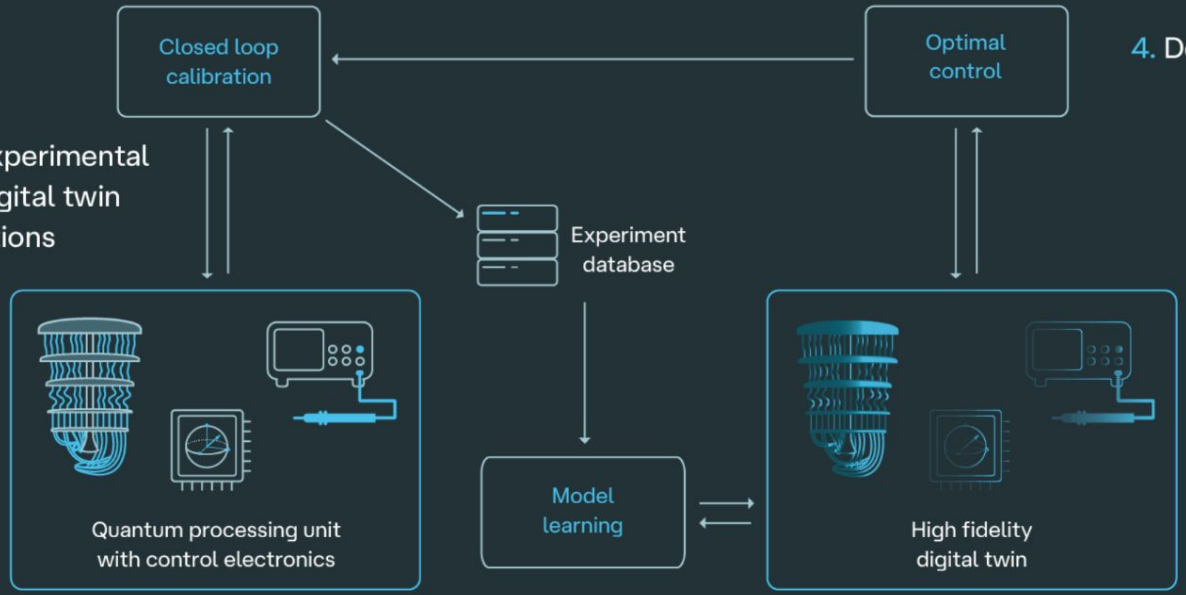
qruise-toolset: differentiable quantum simulation toolbox

Yousof Mardoukhi - Qruise GmbH

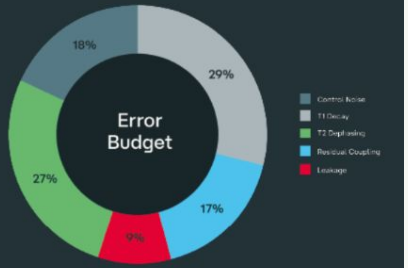
28th EuroAD Workshop - CERN

How do we learn?



2. Compare experimental data with digital twin predictions

4. Develop deep new insights into hardware behaviour!



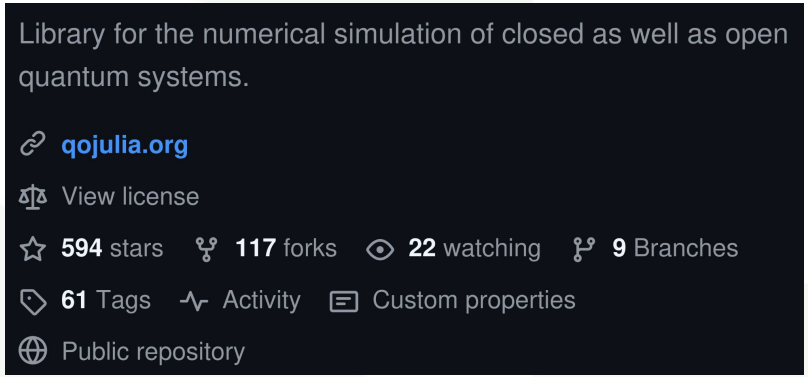
3. Define a loss function and update parameters with ML algorithm

1. Design parameterised and differentiable digital twin

qruise

Overview

- It is similar to QuTiP, but it is differentiable.
- Seamless integration with QuTiP.
- Has a Python interface but a Julia backend for simulation.
- Is tailored specifically for for quantum optimal control problems.
- Provides a differentiable digital twin for closed-loop control.



qruise

Optimal control and bilinear systems

- A bilinear system is described in the state-space representation as

$$\frac{dx}{dt} = \left[A + \sum_k u_k(t;\{p\})B_k\right]x$$

- *x(t)* is the state of system at time *t*.
- *A* and B_k are linear operators.
- u_k are time-dependent parameterised control fields.

Optimal control and bilinear systems

- For a BLS defined over time interval $[t_0, t_1]$ and a set of parameters $\{p\}$ defined the objective function as

$$J = \Phi[x(t_1)]$$

- Seek a set of optimised parameters that minimises the objective function J.

Schrödinger equation

$$\frac{d\psi}{dt} = -iH(t;\{p\})\psi$$

$$H(t;\{p\}) = \overbrace{H_0}^{\text{stationary part}} + \underbrace{u(t;\{p\})}_{\text{control field}} \overbrace{H_t}^{\text{quantum channel}}$$

$$J = 1 - ||\langle\psi_t|\psi(t_1)\rangle||^2$$

qruise

The core issue: interfacing Python with Julia

- From the perspective of data structures:
- A and B_k are linear operators → array representation
- x is the state of the system → scalar or array representation

BUT

- $u_k(t, \{p\})$ are functions: first-class objects in Python – singleton type in Julia

Possible solutions

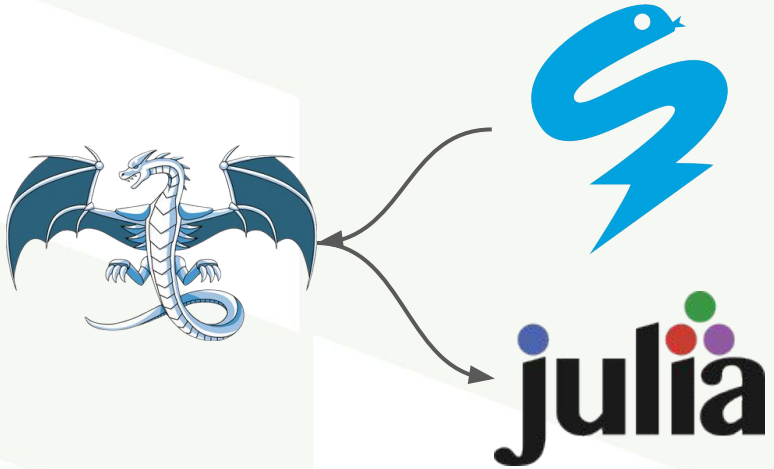
- PythonCall.jl
  - Call a Python function in Julia
  - Custom type conversion rules
  - Non-differentiable
  - Still has interpretation overhead sometimes ~5x
- SymPy
  - Symbolic representation
  - Translation to native Julia functions
  - More complicated functions does not emit correct Julia function
  - Algebra for quantum operators is still immature

Numba & LLVM IR

```
from numba import cfunc, types

sig = types.float64(
    types.float64,
    types.float64,
    types.float64
)

@cfunc(sig)
def func(t, a, b):
    return a * t + b
```



- Native hardware compilation
- LLVM analysis and transform passes
- Optimisation
- Thanks to **Enzyme**, it is differentiable

Summary

- We made Python functions differentiable through LLVM IR + Enzyme
- An AD quantum simulation toolbox with a Python interface and Julia backend
- Does not rely on JAX, PyTorch or TensorFlow for AD

References

1. Moses, W. and Churavy, V., 2020. **Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients**. *Advances in neural information processing systems*, 33, pp.12472-12485.
2. Moses, W.S., Churavy, V., Paehler, L., Hückelheim, J., Narayanan, S.H.K., Schanen, M. and Doerfert, J., 2021, November. **Reverse-mode automatic differentiation and optimization of GPU kernels via Enzyme**. I*n Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1-16).
3. Lattner, C. and Adve, V., 2003. **A compilation framework for lifelong program analysis and transformation**. *In CGO* (Vol. 4, p. 75).
4. Lam, S.K., Pitrou, A. and Seibert, S., 2015, November. **Numba: A LLVM-based Python JIT compiler**. *In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (pp. 1-6).