

No-one used my software

A tale of quantum software development

Aleksander Wennersteen

Quantum computing devroom @ FOSDEM

Brussels 02/02/2025

Overview

- Background
- Motivation
- Why no one uses your software
- Open-source ecosystem

Lessons learned from building quantum software

- 3.5 years building quantum software professionally
 - First at pure quantum software startup Qu & Co
 - Then at full-stack hardware startup -> scaleup Pasqal
- In the past: high performance computing (HPC) and machine learning
- Today:
 - i. Integrating quantum computing with classical HPC systems
 - ii. Programming languages and libraries
 - iii. Analog and digital quantum computing

How did I end up here?

CFP main ideas

- Quantum programming languages and tools
- Compilation, transpilation, and optimization of quantum programs
- Error mitigation, correction, and making noisy qubits work
- Too much for me on a Sunday

"Off the beaten track topics"

- **Lessons learned from building quantum software**
- **Insights on building open quantum communities**
- **Surprising and fun** uses of quantum hardware or software
- Long-shot ideas and ambitious **visions for the future of quantum software**

Help, my talk was accepted

- Google: "why no one used my software"

Why no one uses your software [1]

1. Your employees have too many software tools
2. Your users don't see your tool's added value
3. Your employees want a practical software training
4. Your employees want a tailor-made training
5. Your teams need to be supported in real time
6. Your employees don't feel heard

Holds just as much for quantum software as enterprise software!

Too many software tools with no added value

- How many SDKs do we really need?
 - Every company, big lab, and quite possibly your neighbour has their own
- What is the value-add?

Cf. [qosf/awesome-quantum-software](https://qosf.org/awesome-quantum-software), Unitary foundation survey

According to users, only a few are worth using

My first SDK

- Built "on top of" mainly Qiskit
- Abstract re-usable components of our algorithm libraries

```
class Backend:
    ...
    @cache
    def dfdx(self, x: float, circuit: Circuit, obs: Observable) -> float:
        ...

    def expectation(self, state: State, obs: Observable) -> float:
        ...

    def run(self, state: State, circuit: Circuit) -> State:
        ...
```

My first SDK

```
class Backend(ABC):  
    ...  
  
class LocalBackend(Backend):  
    ...  
  
class RemoteBackend(Backend):  
    ...
```

How multiple inheritance killed my SDK

```
class LocalBitstringBackend(Backend, LocalBackend, BitstringBackend):  
    ...  
  
class LocalWaveFunctionBackend(Backend, LocalBackend, WFBackend):  
    ...
```

How multiple inheritance killed my SDK

```
class LocalQiskitBitstringBackend(Backend, LocalBackend, QiskitBackend, BitstringBackend):  
    ...
```

- This was getting out of hand.
- Hard to onboard people on, required solid understanding of OOP
- Hard to extend
- Slow for our QML workloads

We needed better performance

- So we wrote the numerical backend in Julia
- We wanted "GPU goes BRRRR", so we did the required CUDA.jl work
- We weren't happy enough with Julia AutoDiff so we
 - Made it a PyTorch function
 - With a custom AutoDiff override
 - Didn't go well with the aforementioned inheritance pattern...
- Clearly it was time to start from scratch

Qadence

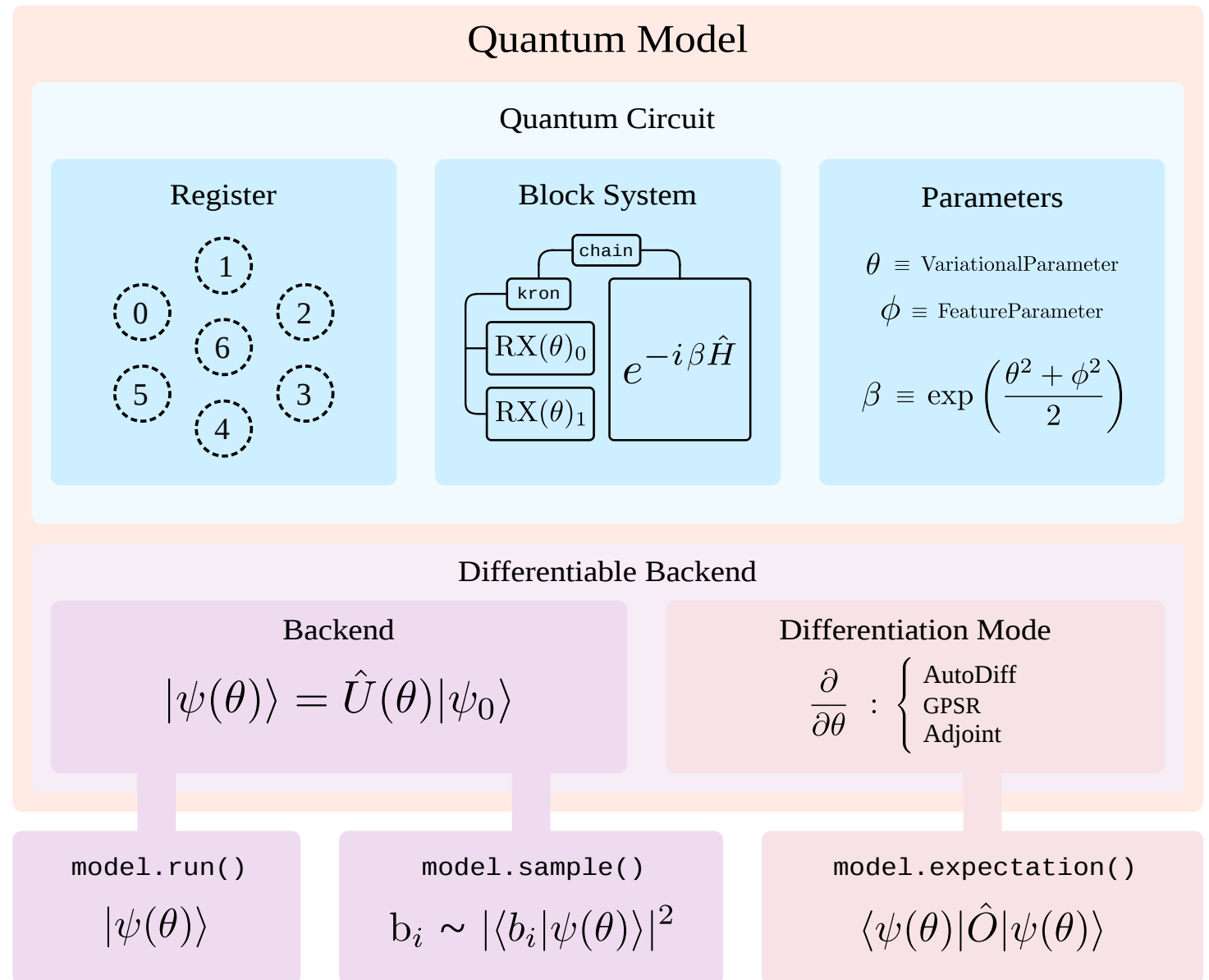
Return of the SDK

A **differentiable** DSL
for analog, digital and
digital-analog paradigms

Geared towards **Rydberg**
atom devices, yet generic

Focused on **hardware-**
realisable programs

Designed according to
algorithm-design needs



Analog programming of Neutral Atom QPUs

$$\hat{H}(\Omega, \delta, \phi; t) = \sum_{\text{atom } i} \left(\underbrace{\frac{\Omega}{2} [\cos(\phi) \hat{\sigma}_i^x - \sin(\phi) \hat{\sigma}_i^y]}_{\text{Programmable laser pulse, global interaction}} - \delta \hat{n}_i + \underbrace{\sum_{j < i} \frac{C_6}{|\mathbf{r}_{ij}|^6} \hat{n}_i \hat{n}_j}_{\text{Programmable register, atom-atom interaction}} \right)$$

In Pulser

```
register = Register.square(3, blockade_radius)

program = seq.add(Pulse(
    amplitude = ConstantWaveform(time, delta),
    detuning = RampWaveform(time, Omega_init, Omega_final),
    phase = 0,
), "rydberg-global")
```

In Qadence

```
register = Register.square(n_qubits=9)

program = chain(
    AnalogRX(pi/2),          # Rotate all qubits, sigma_x term
    AnalogInteraction(time) # Evolve Rydberg system, register term
)

sample(register, program, n_shots=100) # convenience function
```



The GPU strikes back

- The algorithm developers said:
 - Need more performance -> GPUs, and auto-differentiable
 - So we wrote PyQTorch and Horqrux in PyTorch and Jax, respectively
 - Qubit count too low
 - So we wrote a tensor network backend, also in PyTorch (Internal)
 - Because AutoDiff and GPU!
- Most day-to-day work too small-scale for GPU speed-up
- Many are scared of approximate methods like tensor networks

Lessons learnt

- Requirements gathering remains important
 - Algorithm developers will ask for an exact, arbitrary noise, tensor network backend with GPU acceleration
 - But they may not need it, and it may not be possible
 - As engineers we should push back when appropriate
- Crazy OOP - bad, but no contracts enforced also bad

Bespoke, practical training, and real-time support

- Software a key ingredient for scaling
 - Workforce/Communities and Companies
- The training, documentation, and support must be up to par
- Documentation, examples
- Slack channels, office hours, GitHub/Gitlab
 - Try to always show all the steps required

```
$ srun -N1 -G1 -c32 --pty bash
$ source .venv/bin/activate
$ cd qadence/docs/tutorials/low_level_api.ipynb
$ python3 -m nbconvert --to python low_level_api.ipynb
$ python3 low_level_api.py
```

Not listening to users / not giving the users what they want

- Application library that didn't fit users need
 - Clean, DRY code, written for software engineers - not algorithm researchers
 - New attempt, beginning with releasing **lean** standalone "Solvers" like github.com/pasqal-io/quantum-evolution-kernel
- One internal library that only became popular after removing "features" aka bloat
 - Users are typically smart - they prefer flexibility
- Julia based tensor network emulator arXiv: 2302:05253
 - Great for engineers, hard to deploy, hard to modify for users
 - PyTorch based emulators at github.com/pasqal-io/emulators
- AWS Batch job submitter / Convenience Script for on-prem cluster
 - Documentation, examples, cloud-platform for emulators

When my never used software became super popular

- Cloud platform tensor network emulator: arXiv: 2302.05253
 - Didn't get traction at all
 - Until suddenly it did
 - And it crashed and burnt
- Turns out we got the requirements right, but not the UX
 - But the need wasn't there until much later -> bit rot
 - Julia issues - hard for scientists, hard to install in HPC centers
 - PyTorch based emulators at github.com/pasqal-io/emulators to the rescue

Open-source ecosystem desires

- More modular, re-usable blocks
 - Testing software which makes little/no assumptions about my SDK
 - All the endianness bugs in alpha-release Qadence
 - HPC/Classical computations integration building blocks
- Solve common pain-points
- EC and EuroHPC workshop on quantum software
 - Focus on European open-source software
- No need to end up with "left-pad" type ecosystem

Summary

- User adoption of Quantum Software just like any other product
 - Training, documentation, value-add, listen to users
- Realistic requirements gathering
 - Push back on requirements is necessary
 - Focus not just on technical requirements, also UX
- More re-usable, widespread, QFOS please

Thank you

Aleksander Wennersteen

Quantum software technical lead @ Pasqal

aleksander.wennersteen@pasqal.com

github.com/pasqal-io, community.pasqal.com

Sign up for Pasqal community launch webinar