Open source variational quantum eigensolver (OpenVQE)

Extension of the quantum learning machine (QLM) for quantum chemistry

Tutorial: OpenVQE training session

Presented by: Huy Binh TRAN

What is OpenVQE ?

Open Source Variational Quantum Eigensolver package for Quantum Chemistry that based on the tools provided in MyQLM-fermion package.

Why is OpenVQE ?

The combined OpenVQE/ myQLM-fermion libraries facilitate the implementation, testing and development of variational quantum algorithms.

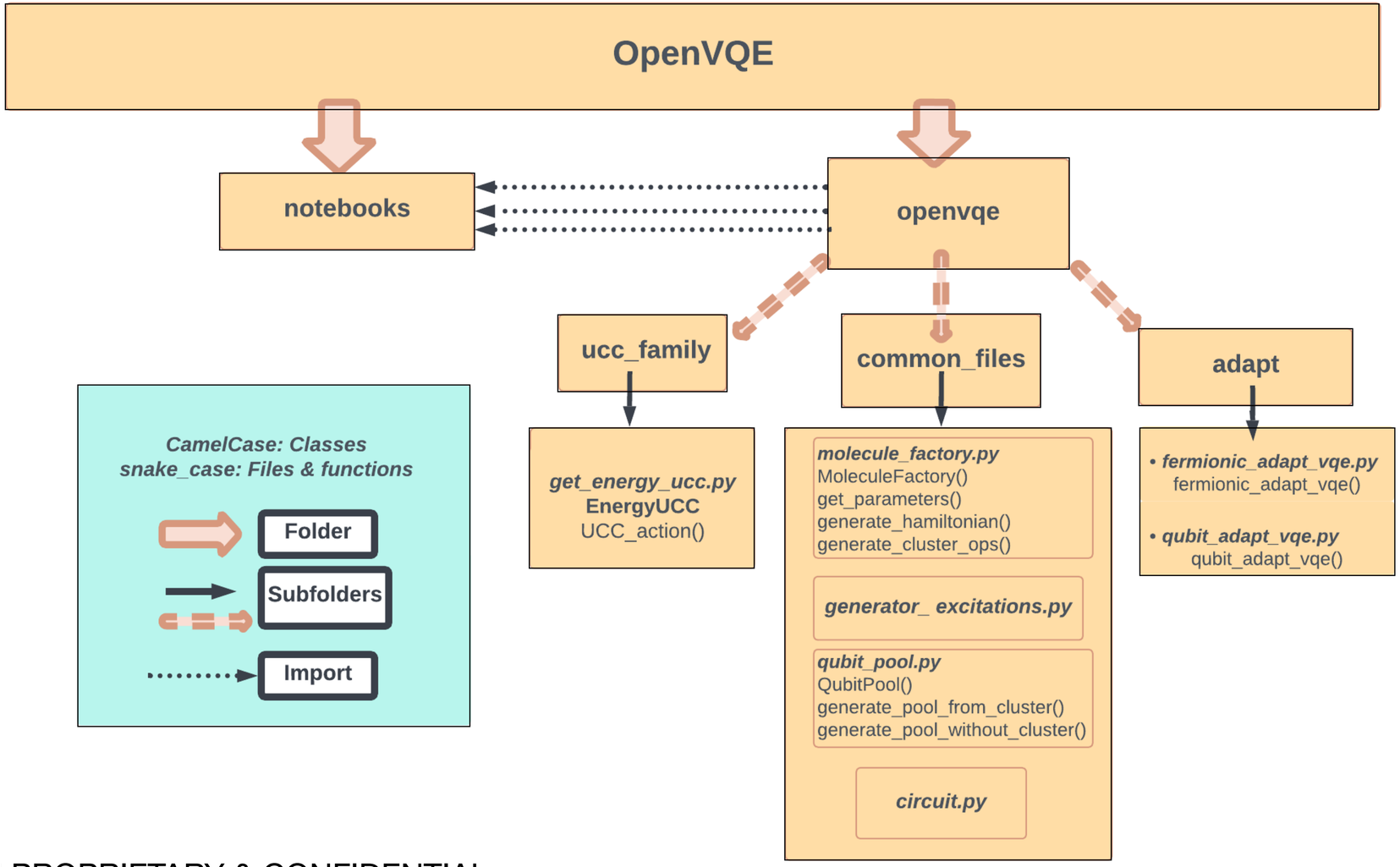
Interoperability of Open-Source Packages with MyQLM

MyQLM library provides binders to connect with the other Python-based quantum frameworks: MyQLM_interoperability

FrameWork		Qiskit	OpenQasm	PyQuit(no py 3.6)	Project Q	Cirq
Circuit translation	to QLM	Yes	Yes	Yes	Yes	Yes
	From QLM	Yes	No	Yes	No	Yes
QPU connection	to QLM	Yes	N/A	Yes	No	No
	From QLM	Yes		No	No	No

Flowchart of the OpenVQE Package

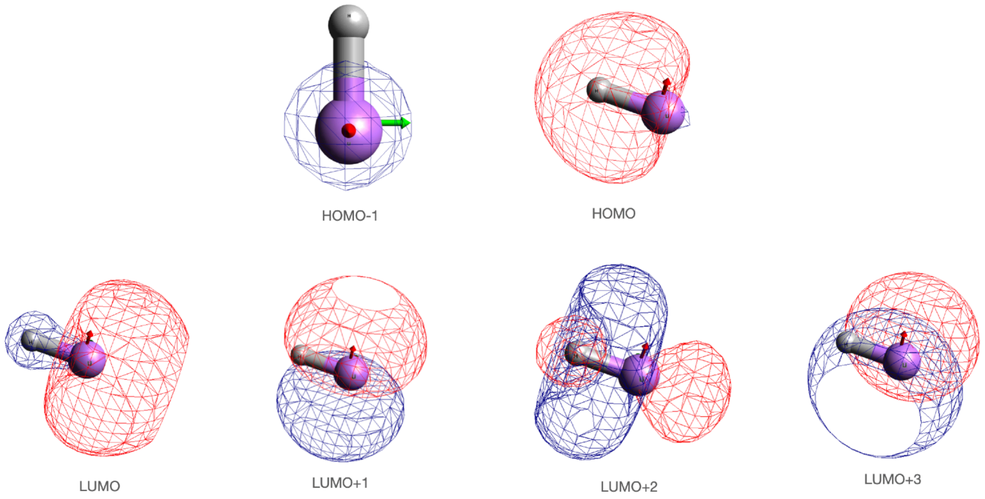
The code is given in our Github repository and documentation



Initializing geometry, charge, spin, basis of molecules

```python
def get_parameters(molecule_symbol):
    if molecule_symbol == "LIH":
        r = 1.45
        geometry = [("Li", (0, 0, 0)), ("H", (0, 0, r))]
        charge = 0
        spin = 0
        basis = "sto-3g"
    elif molecule_symbol == "H2":
        r = 0.75
        geometry = [("H", (0, 0, 0)), ("H", (0, 0, r))]
        charge = 0
        spin = 0
        basis = "sto-3g"
    return r, geometry, charge, spin, basis
```

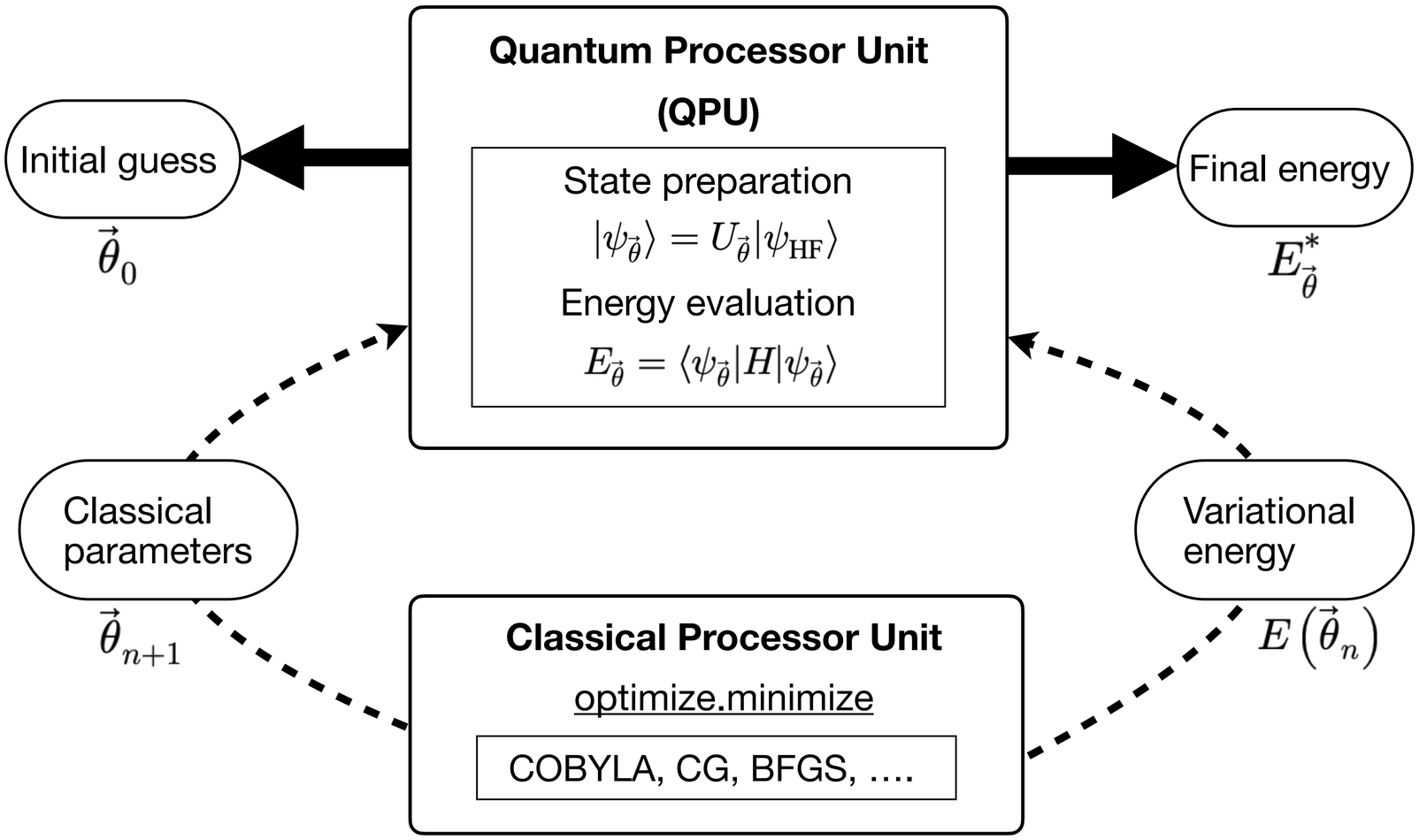
Getting spin representation of Hamiltonian and UCC ansatz

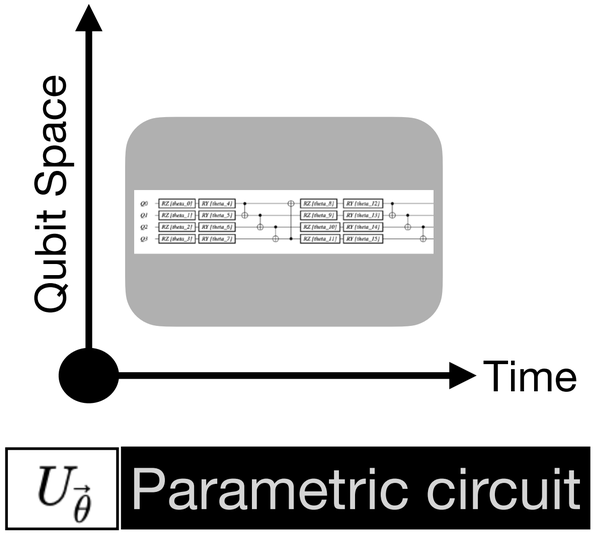


Visualization of spatial orbitals in LiH molecule uisng STO-3G basis-set: 2 HOMOs and 4 LUMOs.

```python
def compute_circuit_and_hamiltonian(molecule_name):
    """Compute the quantum circuit and Hamiltonian for a given molecule."""
    r, geometry, charge, spin, basis = get_parameters(molecule_name)

    # Perform quantum chemistry computation
    rdm1, orbital_energies, nuclear_repulsion, nels, one_body_integrals,
two_body_integrals, info = perform_pyscf_computation(
        geometry=geometry, basis=basis, spin=spin, charge=charge, run_fci=True
    )

    #print(f"Number of qubits before active space selection: {rdm1.shape[0] * 2}")
    #print("RDM1:", rdm1)
    print("Computation info:", info)

    # Convert integrals to fermionic Hamiltonian
    hpq, hpqrs = convert_to_h_integrals(one_body_integrals, two_body_integrals)
    H = ElectronicStructureHamiltonian(hpq, hpqrs, constant_coeff=nuclear_repulsion)

    # Get cluster operators for UCC ansatz
    noons, basis_change = np.linalg.eigh(rdm1)
    noons = list(reversed(noons))
    noons_full, orb_energies_full = [], []
    for ind in range(len(noons)):
        noons_full.extend([noons[ind], noons[ind]])
        orb_energies_full.extend([orbital_energies[ind], orbital_energies[ind]])

    cluster_ops, theta_0, hf_init = get_cluster_ops_and_init_guess(nels, noons_full,
orb_energies_full, H.hpqrs)

    # Transform Hamiltonian and cluster operators to qubit representation
    H_sp = transform_to_jw_basis(H)
    nqbits = H_sp.nbqbits
    print("Number of qubits", nqbits)
    print("Spin representaion of Hamiltonian", H_sp)
    cluster_ops_sp = [transform_to_jw_basis(t_o) for t_o in cluster_ops]
    hf_init_sp = recode_integer(hf_init, get_jw_code(H_sp.nbqbits))

    # Construct UCC ansatz quantum circuit
    qprog = construct_ucc_ansatz(cluster_ops_sp, hf_init_sp)
    circ = qprog.to_circ()
    qpu = get_default_qpu()

    return H_sp, circ, qpu, nqbits, theta_0
```

Flowchart of the VQE algorithm





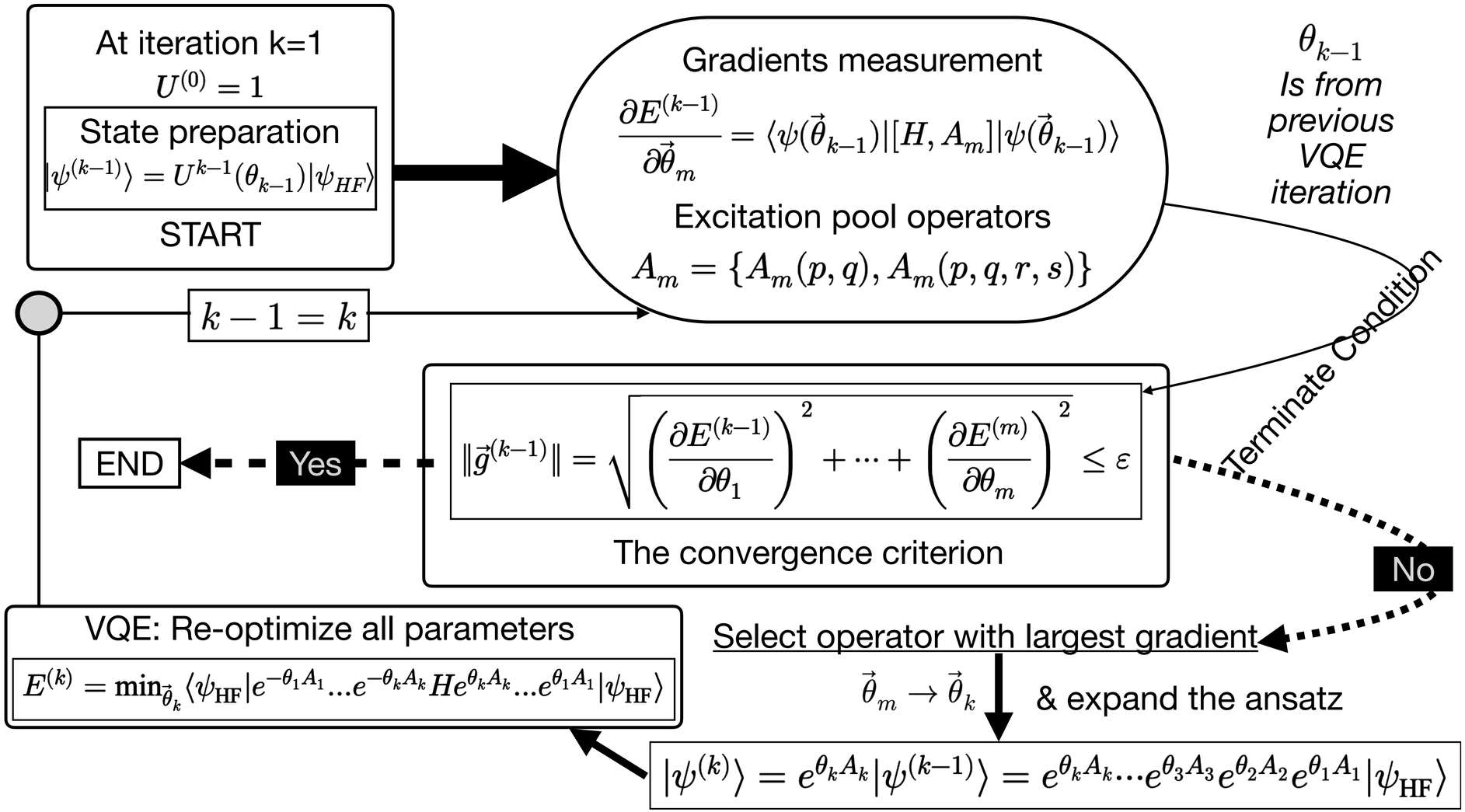
Optimisation function from imported circuit and observables

```python
def get_optimization_func(circ, qpu, H_sp, method, nqbits, psi0, energy_list,
fid_list):

    def my_func(x):

        circ1 = circ.bind_variables(
            {k: v for k, v in zip(sorted(circ.get_variables()), x)}
        )
        res0 = qpu.submit(circ1.to_job(observable=H_sp))
        energy = res0.value
        energy_list[method].append(energy)

        res = qpu.submit(circ1.to_job())
        psi = np.zeros((2**nqbits,), complex)
        for sample in res:
            psi[sample.state.int] = sample.amplitude
        fid = abs(psi.conj().dot(psi0)) ** 2
        fid_list[method].append(fid)

        return energy

    return my_func
```

Gradient-based minimisation function - PMRS

```python
def get_grad_func(circ, qpu, H_sp):
    # here I show a gradient-based minimization strategy
    def my_grad(x):
        grads = circ.to_job(observable=H_sp).gradient()
        grad_list = []
        for var_name in sorted(circ.get_variables()):
            list_jobs = grads[var_name]
            # list_jobs contains jobs to compute E(theta+pi/2) and E(theta-pi/2)
            # the gradient w.r.t theta is then 0.5 (E(theta+pi/2) - E(theta-pi/2))
            grad = 0.0
            for ind in range(len(list_jobs)):
                circ1 = list_jobs[ind].circuit.bind_variables(
                    {k: v for k, v in zip(circ.get_variables(), x)}
                )
                job = circ1.to_job(observable=list_jobs[ind].observable)
                res = qpu.submit(job)
                grad += 0.5 * res.value
            grad_list.append(grad)
        return grad_list

    return my_grad
```

Flowchart of the ADAPT VQE algorithm



State of the art

Empowering impactful projects via OpenVQE.

- More than **35** contributors from different countries: Europe, US, Asia
- Noiseless Schrödinger-style dense simulator can reach up to **41** qubits for any circuit
- **13** citations from the published paper
- Non-profit organisation, aim for education

Wiley review https://doi.org/10.1002/wcms.1664