QFlux: Quantum Circuit Implementations of Molecular Dynamics

Victor S Batista
Yale University, Department of Chemistry and Yale Quantum Institute

Part II: Closed Quantum Systems

From physical Hamiltonians to executable quantum circuits, showing how real-time dynamics can be implemented, validated, and benchmarked on qubit-based hardware

https://qflux.batistalab.com

JCTC_II.ipynb

QFlux: Quantum Circuit Implementations of Molecular Dynamics

Victor S Batista

Yale University, Department of Chemistry and Yale Quantum Institute

Part II: Closed Quantum Systems

This tutorial is based on the manuscript

QFlux: Quantum Circuit Implementations for Molecular Dynamics

Part II - Closed Quantum Systems

Authors:

Delmar G. A. Cabral, Brandon C. Allen, Cameron Cianci, Alexander V. Soudackov, Xiaohan Dan, Nam P. Vu, Rishab Dutta, Sabre Kais, Eitan Geva, and Victor S. Batista

Motivation: Why Quantum Dynamics on Quantum Computers?

Why this matters ?

- Real-time dynamics → transport, tunneling, spin exchange
- Classical cost grows exponentially
- Quantum hardware natively follows **unitary evolution**

- **Key message:** dynamics is the shared strategy across methods

Position in the QFlux Series

Everything later—variational algorithms, Lindblad dynamics, non-Markovian effects—depends on the circuit primitives introduced here

Tutorial roadmap

- **Part I**: Classical dynamics & validation
- **Part II (this talk)**: Closed-system quantum circuits
- Part III: State preparation & unitary decomposition
- Part IV–VI: Open systems, variational dynamics, memory effects

Core QFlux Philosophy (Reframed for Quantum)

Same workflow, new backend

- Same Hamiltonian
- Same observables
- Same validation strategy
- Backend = circuit execution

From Hamiltonians to Circuits: The Central Problem

How to compile the exponential of a Hamiltonian into gates for quantum hardware?

$$|\psi(0)\rangle = \mathrm{e}^{-\mathrm{i}\mathcal{H}t}\,|\psi(t)\rangle$$

- Hardware supports only certain conditional or unconditional gates (e.g., rotations, Pauli gates)
- Need systematic decomposition

Pauli-String Encoding of Hamiltonians

Pauli decomposition:

$$\mathcal{H} = \sum_j a_j P_j \qquad P_j = \bigotimes_{k=1}^{n} \sigma_k^{(j)}$$

$$a_j = \frac{1}{2^n} \mathrm{Tr}(P_j \mathcal{H}) \qquad \sigma_k^{(j)} \in \{I, X, Y, Z\}$$

Key properties:

- Exact, orthogonal basis
- Sparse for physical Hamiltonians

Trotterization: Making Time Evolution Executable

Product formula

$$\mathrm{e}^{-\mathrm{i}\mathcal{H}t} \approx \left(\prod_j \mathrm{e}^{-\mathrm{i}a_j P_j t/m} \right)^m$$

Tradeoff

More steps (larger *m*) → higher accuracy

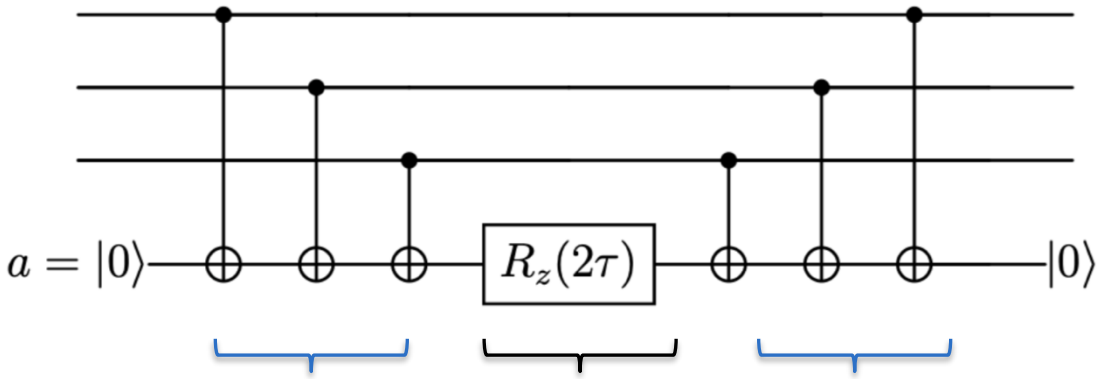
More steps (larger *m*) → deeper circuits

All-Z Hamiltonians via Parity Circuits

Example: Script S.2.3, JCTC_II.ipynb

$$\mathcal{H} = Z \otimes Z \otimes Z.$$

$$e^{-i\tau\, Z\otimes Z\otimes Z}$$



Implementation

Collect parity on ancilla Apply single Rz Uncompute parity

Arbitrary Pauli Strings via Basis Rotation

Key identities:

$$Y = R_x(-\pi/2) Z R_x(\pi/2)$$

$$X = HZH$$

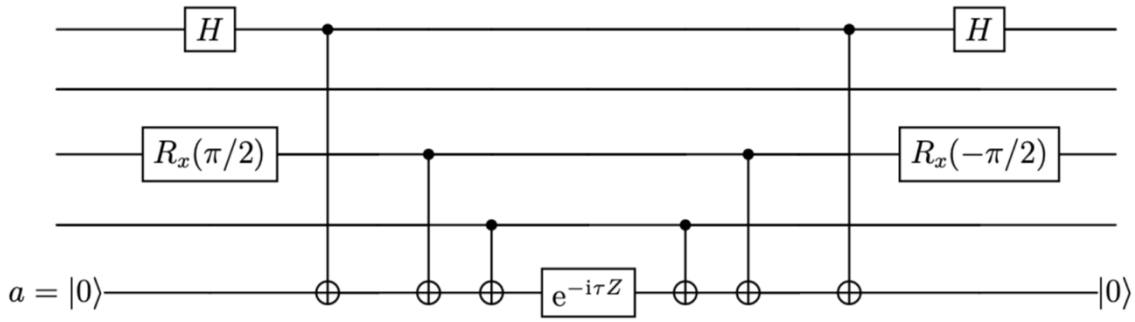
Example: **Script S.2.4,** JCTC_II.ipynb

$$\mathcal{H} = X \otimes I \otimes Y \otimes Z$$

$$e^{-i\tau(X \otimes I \otimes Y \otimes Z)}$$

Workflow:

1. Rotate into Z-basis
2. Apply parity evolution
3. Rotate back



General Hamiltonians: Automated in QFlux

What QFlux does

- Pauli decomposition
- Circuit generation
- Trotter layering
- Backend execution (Qiskit)

Example: Script S.2.5, JCTC_II.ipynb

Inputs:

- hamiltonian: dictionary like { "XZ": 0.5, "YY": -1.2 } Keys = Pauli strings, values = coefficients.
- quantum_register: optional register to run on.
- control_qubit: optional control (for controlled time evolution).
- t: total simulation time.
- trotter_number: number of Trotter steps.

General Hamiltonians: Automated in QFlux

Script S.2.5, JCTC_II.ipynb

```
for pauli_str, coeff in hamiltonian.items():
    term_circuit = exp_pauli(pauli_str, quantum_register, control_qubit, coeff * delta_t)
    circuit.compose(term_circuit, inplace=True)
full_circuit = QuantumCircuit(quantum_register)
for _ in range(trotter_number):
    full_circuit.compose(circuit, inplace=True)
return full_circuit
```

For each Hamiltonian term $c_k P_k$:

- Calls `exp_pauli(...)` to build a circuit implementing $e^{-i\, c_k\, P_k\, \Delta t}$

  (this helper is assumed to exist elsewhere).
- Appends it to `circuit`.

After this loop, `circuit` represents:

$$\prod_k e^{-ic_k P_k \Delta t}$$

which is one first-order Trotter step.

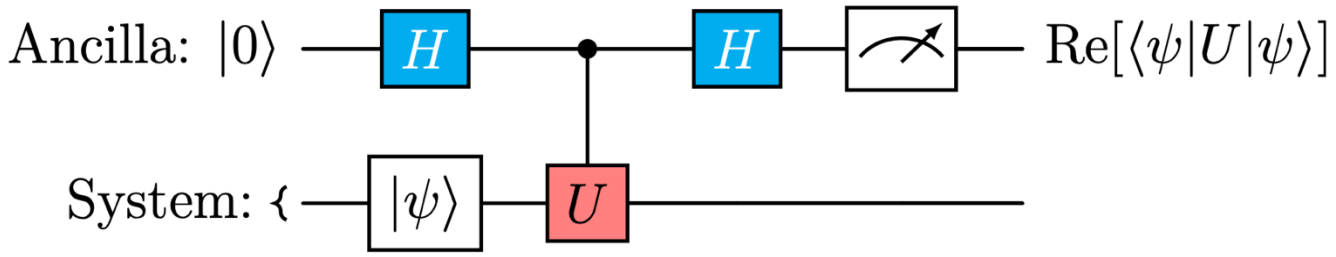
Measuring Dynamics: The Hadamard Test

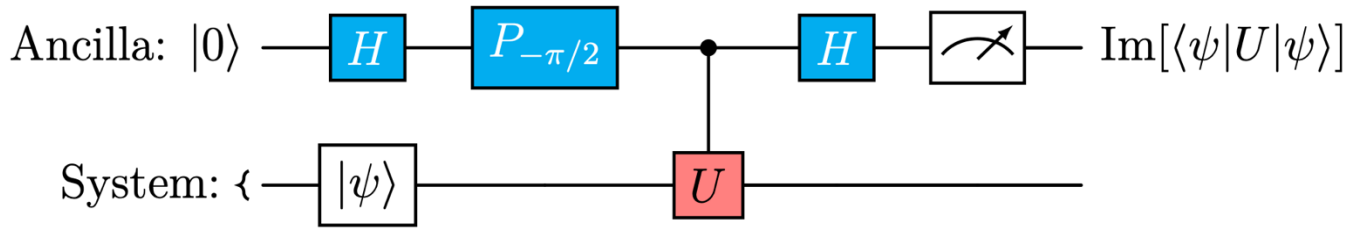
Goal: Compute expectation values and correlation functions

$$\langle U \rangle = \langle \psi | U | \psi \rangle$$

Ancilla-based interference

- Real part
- Imaginary part





Measuring Dynamics: The Hadamard Test

Script S.2.32, JCTC_II.ipynb

```
# Hadamard test on the ancilla qubit
qc_hadamard.h(0)
if imag_expectation:
    qc_hadamard.p(-np.pi/2, 0) # qc_hadamard.s(0).inverse() may be equivalent

# iterates over the number of times the control operation should be added
for i in range(control_repeats):
    qc_hadamard.append(control_operation, qr_hadamard[:])
qc_hadamard.h(0)
qc_hadamard.barrier()

# Measuring the ancilla
qc_hadamard.measure(0,0)
```

- Creates superposition on an ancilla
- Conditionally applies U^k
- Interferes the paths
- Measures the ancilla

to extract either the real or imaginary part of $\langle\psi|U^k|\psi\rangle$.

Expectation Values Without Ancillas

Shallow Circuits

Alternative

- Decompose observable O in Pauli strings
- Measure Pauli strings directly:
  - Basis rotation + sampling
- Average the measurements weighted by the Pauli decomposition coefficients a_j

$$\langle O \rangle = \langle \psi | O | \psi \rangle$$

$$O = \sum_j c_j P_j$$

$$c_j = \frac{1}{2^n} \mathrm{Tr}(P_j O)$$

$$P_j = \bigotimes_{k=1}^{n} \sigma_k^{(j)}$$

$$\sigma_k^{(j)} \in \{I, X, Y, Z\}$$

When to use

- Variational algorithms
- Static observables

$$\langle O \rangle = \sum_j c_j \langle \psi | P_j | \psi \rangle$$

Case Study I: Two-Spin Heisenberg Model

Hamiltonian

$$\mathcal{H} = \frac{1}{2}\left(h_0\sigma_0^z + h_1\sigma_1^z\right) + \frac{J}{4}\left(\sigma_0^x\sigma_1^x + \sigma_0^y\sigma_1^y + \sigma_0^z\sigma_1^z\right)$$

Script S.2.6- S.2.14, JCTC_II.ipynb

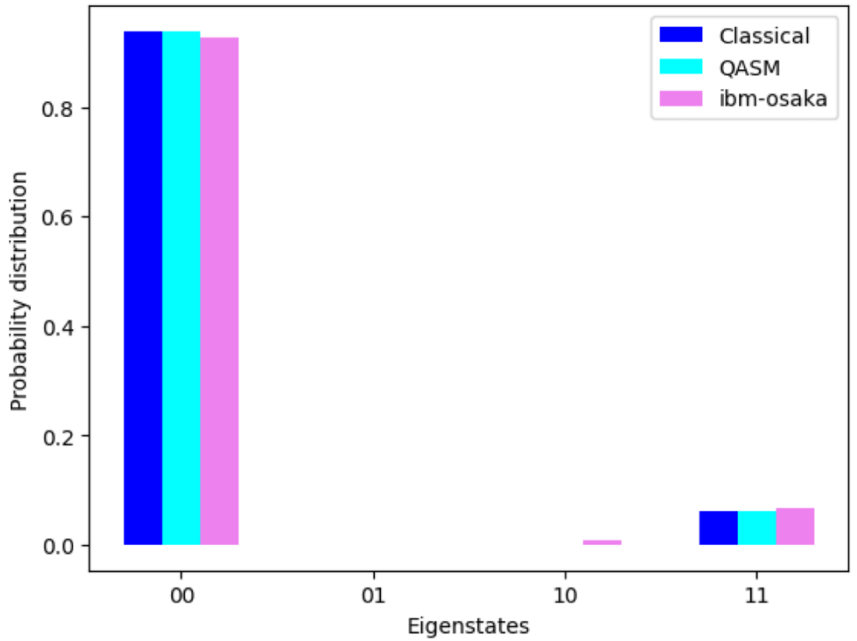
Initialization: Script S.2.9

Propagation: Script S.2.10

Measurement: Script S.2.11

Benchmarks

- Classical exact
- QASM simulator
- IBM hardware



Hardware Results and Noise

Observation

- Simulator = exact
- Hardware = deviations

Origin

- Decoherence
- Gate errors
- Readout noise

Scaling Up: N-Spin Heisenberg Chains

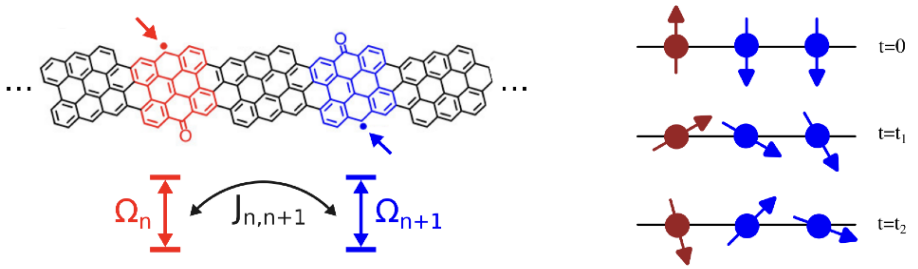
$$\mathcal{H} = \sum_{n=0}^{N-1} \Omega_n \sigma_n^z - \frac{1}{2} \sum_{n=0}^{N-2} \left(J_{n,n+1}^x \sigma_n^x \sigma_{n+1}^x + J_{n,n+1}^y \sigma_n^y \sigma_{n+1}^y + J_{n,n+1}^z \sigma_n^z \sigma_{n+1}^z \right)$$

Statics Dynamics

General form

On-site fields

Nearest-neighbor couplings



Applications

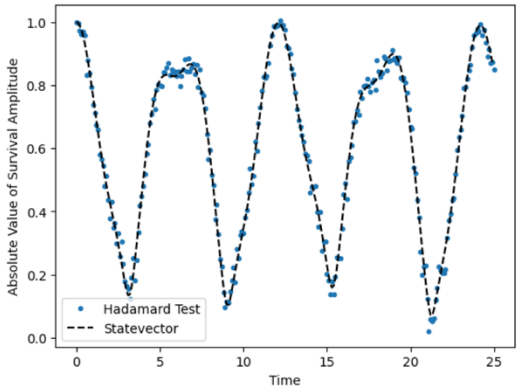
Spin transport

Molecular radicals

Solid-state chains

Script S.2.17

JCTC_II.ipynb



Efficient Trotter Circuit Design

Optimizations

Group one- vs two-qubit terms

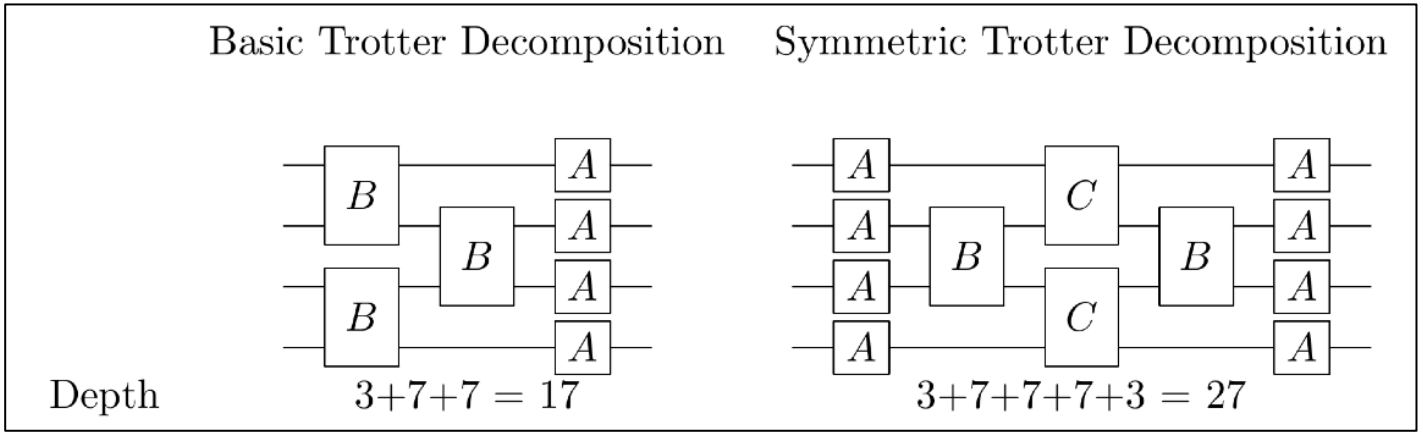
Even–odd layering

Symmetric Trotter schemes

Result

Reduced circuit depth

Script S.2.23, JCTC_II.ipynb



Q-SOFT: Quantum Split-Operator Fourier Transform

Quantum analog of classical SOFT

- Potential: diagonal in position
- Kinetic: diagonal in momentum
- QFT ↔ FFT

Q-SOFT Circuit Structure

One time step:

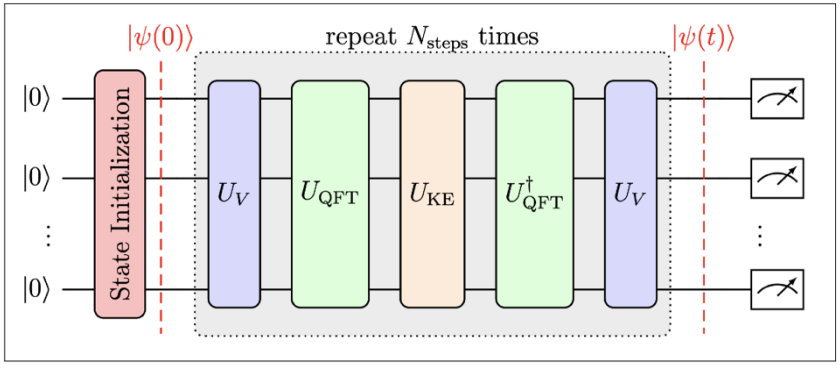
$$U_V \; U_{\mathrm{QFT}} \; U_T \; U_{\mathrm{QFT}}^{\dagger} \; U_V$$

$$\psi(x, t_{i+1}) = \mathrm{e}^{\frac{-\mathrm{i}V(x)\tau}{2\hbar}} \mathcal{F}^{-1}\left[\mathrm{e}^{\frac{-\mathrm{i}p^2\tau}{2m\hbar}} \mathcal{F}\left(\mathrm{e}^{\frac{-\mathrm{i}V(x)\tau}{2\hbar}} \psi(x, t_i)\right)\right]$$

Script S.2.40, JCTC_II.ipynb

```
qc.initialize(psi_0, q_reg[:],normalize=True)

for k in trange(iterations):
    V_op = Operator(VVd_prop)
    qc.append(V_op, q_reg)
    qc.append(QFT(nqubits,do_swaps=True,inverse=False),q_reg)
    K_op = Operator(KEd_prop)
    qc.append(K_op, q_reg)
    qc.append(QFT(nqubits,do_swaps=True,inverse=True),q_reg)
    qc.append(V_op, q_reg)
```



Case Study II: Proton Transfer in DNA

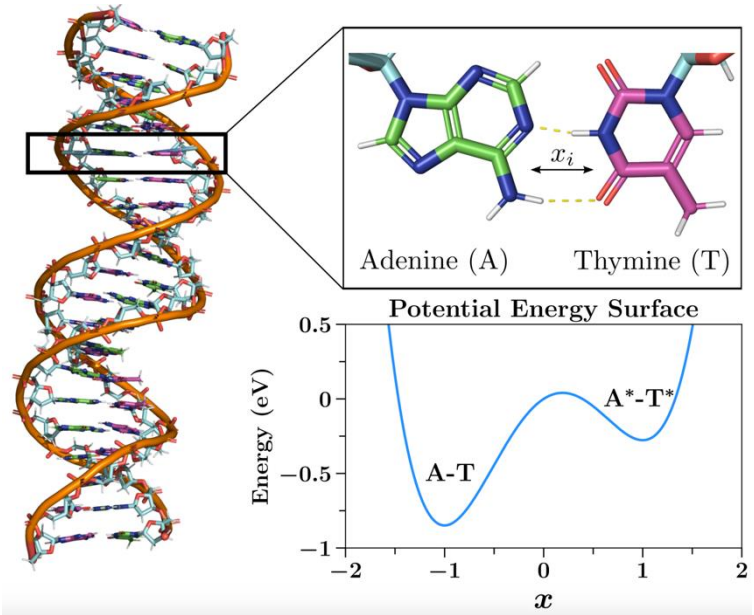
Script S.2.36-S.2.41, JCTC_II.ipynb

Model

- Asymmetric double well
- Proton tunneling
- A–T base pair tautomerization

Result

- Quantum = classical SOFT



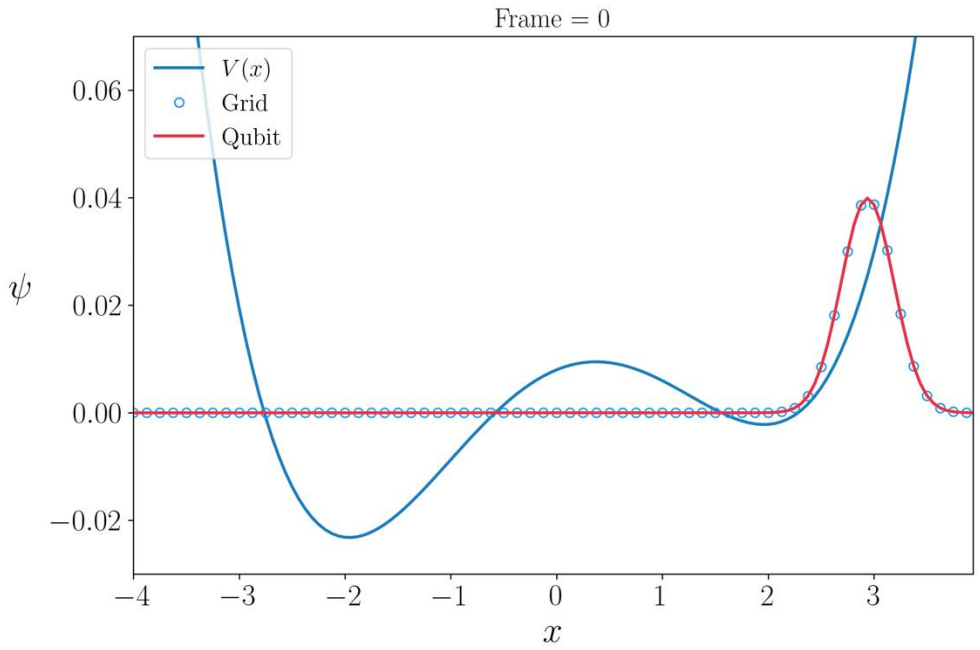
$$V(x) = \alpha\left[0.429\left(\frac{x}{x_0}\right) - 1.126\left(\frac{x}{x_0}\right)^2 - 0.143\left(\frac{x}{x_0}\right)^3 + 0.563\left(\frac{x}{x_0}\right)^4\right] \qquad x_0 = 1.9592$$

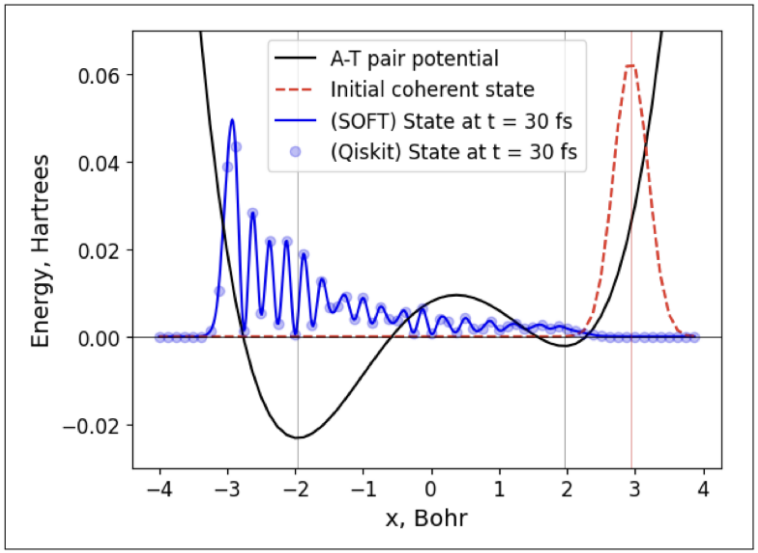
$$\psi_0(x) = \frac{1}{(\pi\sigma_x^2)^{1/4}}\exp\left[-\frac{(x-x_c)^2}{2\sigma_x^2} + ip_c(x-x_c)\right]$$

Validation Against Classical Dynamics

Key result

- Probability densities overlap
- Phase conventions consistent
- Dynamics reproduced quantitatively





Key Takeaways

Quantum Computing

- Hamiltonians → Pauli strings → circuits
- Classical methods guide quantum design
- Validation is non-negotiable
- QFlux provides a full workflow

Outlook

Quantum Computing

Next steps:

- State preparation (Part III)
- Open systems (Part IV)
- Variational & noisy hardware (Part V–VI)