Quantum Computing: Concepts and Applications

Jared Edgerton

What Quantum Computing Is

Quantum computing is a model of computation that uses:

- Qubits (quantum bits)
- Quantum operations (gates)
- Measurement

It is not "faster computing" in general. It is different computing for specific problem classes.

Classical Bits vs Qubits

Classical bit:

- Takes values 0 or 1

Qubit:

- Can be in a quantum state that is a combination of 0 and 1
- Produces classical outcomes only when measured

Quantum states carry information differently than classical states.

The Key Differences

Three core differences from classical computing:

- **Superposition** (state combinations)
- **Entanglement** (correlated states beyond classical dependence)
- **Interference** (amplitudes can reinforce/cancel)

Quantum advantage comes from engineering interference patterns.

Why It Can Be Powerful

Quantum algorithms can provide:

- Large speedups for certain linear-algebraic problems
- Exponential state space representation (with caveats)
- Efficient simulation of quantum systems (chemistry, materials)

Power depends on:

- The algorithm
- The hardware
- The error model

What It Does *Not* Do

Quantum computing does not:

- Make every computation faster
- Replace statistical modeling or ML broadly
- Eliminate the need for data quality or identification

It is a specialized tool, not a universal accelerator.

Where Quantum Advantage Is Discussed

Canonical application areas include:

- Factoring / cryptography (Shor)
- Search (Grover-style quadratic speedup)
- Optimization heuristics (QAOA)
- Sampling and simulation

Many claims depend on hardware maturity and noise tolerance.

How You Would Use It (Practically)

A useful mental model:

- Identify a subproblem with known quantum-style benefit
- Map it to a circuit/sampler/optimizer formulation
- Compare to strong classical baselines

In most social-science settings, quantum is exploratory or hybrid.

Quantum as an Optimization Lens

Many social-science tasks can be framed as optimization:

- Matching and assignment
- Clustering / community detection
- Combinatorial model selection
- Portfolio / allocation problems

Quantum approaches often target hard combinatorial cores.

Quantum as Sampling / Inference

Another angle:

- Sampling from complex distributions
- Estimating partition functions
- Approximate inference in large graphical models

The core question is whether quantum sampling beats classical MCMC/VI.

Hardware Reality Check

Current constraints:

- Noise and decoherence
- Limited qubit counts and connectivity
- Error correction is expensive

Most near-term workflows are:

- Simulation
- Small circuits
- Hybrid classical–quantum methods

A Minimal Workflow

Workflow (conceptual):

1. Choose a problem and encoding
2. Build a circuit (or quantum program)
3. Run on simulator or device
4. Measure outcomes
5. Post-process classically
6. Evaluate against classical baseline

Pseudocode: Circuit-Based Workflow

```
# 1) Define problem → encode into circuit parameters
problem = load_problem_data()
# 2) Build a circuit (e.g., parameterized ansatz)
circuit = QuantumCircuit(n_qubits)
circuit.initialize(superposition=True)

params = initialize_parameters()
for layer in range(L):
    circuit.apply_entangling_layer()
    circuit.apply_parameterized_rotations(params[layer])
# 3) Run (simulator or hardware)
backend = get_backend(simulator=True)
counts = backend.run(circuit, shots=2000)
# 4) Post-process to produce solution candidate(s)
solution = decode_counts(counts)
score = evaluate(solution, problem)
# 5) Iterate (hybrid loop) to improve parameters
while not converged:
    params = update_params(params, score)
    counts = backend.run(circuit, shots=2000)
    solution = decode_counts(counts)
    score = evaluate(solution, problem)
```

Pseudocode: QAOA-Style Optimization

```
# Encode objective as a Hamiltonian / cost operator
cost = build_cost_operator(problem)
mixer = build_mixer_operator(n_qubits)

# Initialize parameters (gamma, beta)
gamma, beta = init_qaoa_params(p=3)

# Build QAOA circuit
qc = QuantumCircuit(n_qubits)
qc.h(range(n_qubits))

for k in range(p):
    qc.apply(cost, gamma[k])
    qc.apply(mixer, beta[k])

# Measure and compute expected cost
counts = run(qc, shots=5000)
obj = expected_cost(counts, problem)

# Classical optimizer updates parameters
(gamma, beta) = classical_optimize((gamma, beta), obj)
```

How to Evaluate Claims of Advantage

For any "quantum" result, ask:

- What is the classical baseline?
- Are we comparing to the best-known classical algorithm?
- Are we measuring wall-clock time or asymptotics?
- Does noise change the conclusion?

In practice, baselines and measurement define the story.

What We Emphasize in Practice

- Treat quantum as a specialized computational model
- Separate conceptual advantage from hardware feasibility
- Use hybrid workflows and strong baselines
- Be explicit about what is and is not gained

Discussion

- Which parts of your research are optimization or sampling problems?
- What would a fair classical baseline be?
- When is "better" about time, quality, or feasibility?