COE 530
Quantum Computers and Architecture

Lecture 11

Quantum Circuit Synthesis

References:

Ch 6 Quantum Computing Systems

Quantum Circuit Synthesis and Compilation Optimization: Overview and Prospects, https://arxiv.org/pdf/2407.00736

Review Quantum Circuit Synthesis for Grover's Algorithm Oracle, https://www.mdpi.com/1999-4893/17/9/382

Quetschlich, N., Burgholzer, L., & Wille, R. (2024). "Quantum Architecture Search: A Systematic Review." *arXiv preprint arXiv:2406.06210.* https://arxiv.org/abs/2406.06210

Outline

- Examples of Gate Synthesis
- Oracle synthesis
- AI/ML approached for circuit synthesis
- Quantum Architecture Search

Generic Quantum Compilation Flow

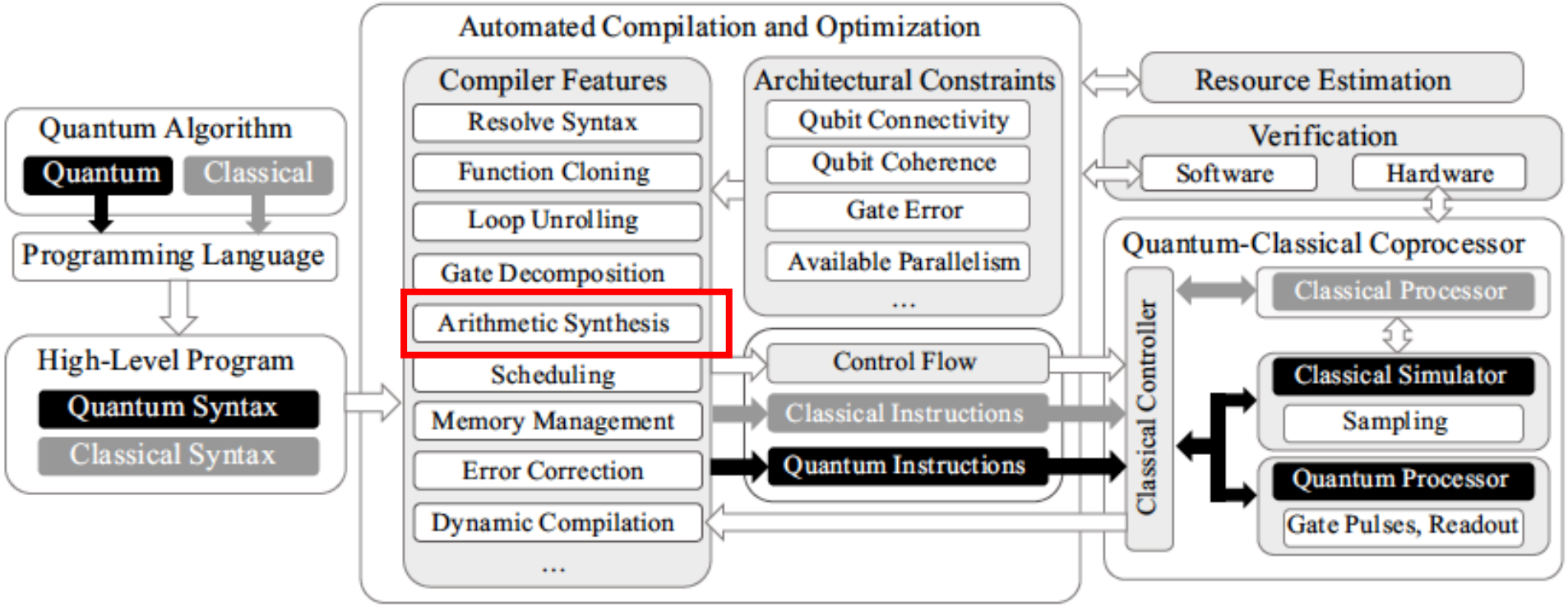


Figure 4.2: A detailed quantum compilation flow outlining the transformations and optimizations involved in a generic compiler.

Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers

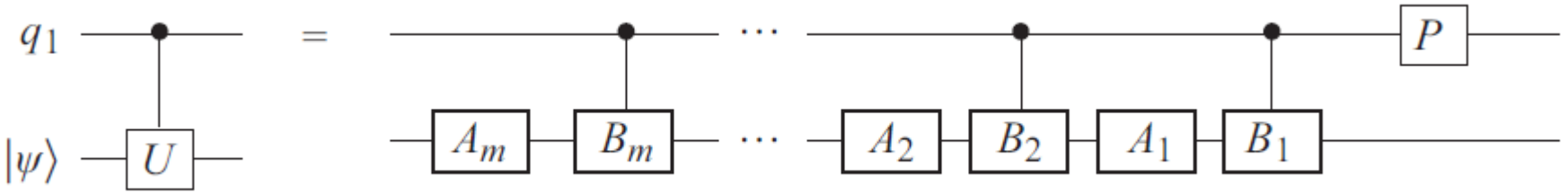
Example: Synthesizing C-U Gates

Any controlled-unitary ($C-U$) gate can be decomposed into simpler single-qubit and CNOT gates

Theorem: For any unitary U, there exist single-qubit gates $A_1, \ldots, A_m$ and $B_1, \ldots, B_m$ phase α such that:

$$C - U = e^{i\alpha} A_m B_m \ldots A_2 B_2 A_1 B_1$$

Satisfying $A_m \ldots A_1 = I$ and B_i are chosen to be simpler than $C-U$, i.e., $B_i = X$



Example: Synthesizing C-U Gates

Example:

It can be shown that an arbitrary unitary U can be decomposed into $U = e^{i\alpha}CXBXA$ where X is the Not gate and $CBA = I$

Synthesis Procedure:

1. Decompose U into A, B, C matrices satisfying $ABC = I$
2. Apply C to the target qubit
3. Apply $CNOT$ with control qubit toggling the target
4. Apply B, then another $CNOT$, then A to the target qubit
5. Apply the global phase $e^{i\alpha}$ as a controlled phase gate on the control qubit

Synthesizing Quantum Oracles

- Most quantum algorithms use quantum oracles as black-box subroutines (Deutsch-Jozsa, Simon, Bernstein-Vazirani, Grover)
- Quantum oracles can be implemented in two steps:
  1. Find an efficient implementation of the desired function using reversible logic gates
     - Map the classical Boolean function f(x) to a reversible circuit using Toffoli, Fredkin, or Peres gates
     - Ancilla qubits may be needed to ensure reversibility (initialized to $|0\rangle$ and uncomputed after use)
  2. Implement each reversible logic gate using quantum gate(s)
     - Toffoli gate decomposes into 6 CNOTs + single-qubit gates ($H, T, T^\dagger$)
     - Goal: minimize total gate count and circuit depth for NISQ compatibility

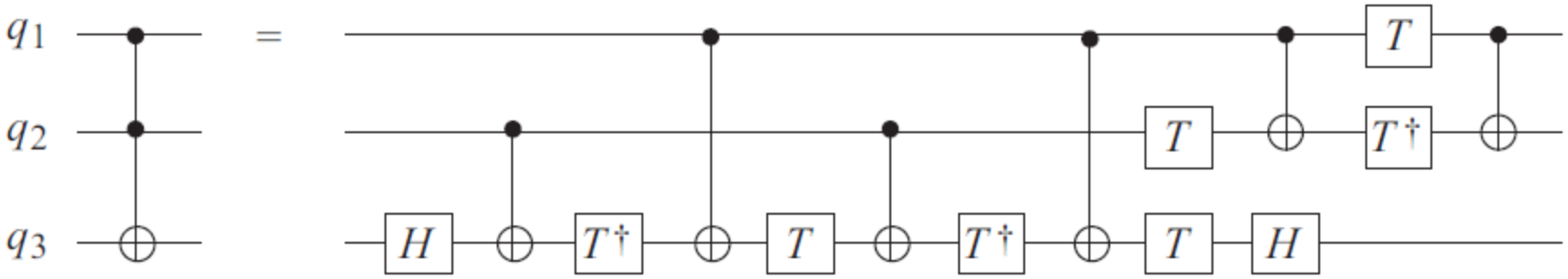
Example: Synthesizing Toffoli gates

What is the Toffoli Gate?

- Also called CCNOT (Controlled-Controlled-NOT) — a 3-qubit gate
- Flips the target qubit only when both control qubits are $|1\rangle$
- Universal gate for classical reversible computation

Why Decompose?

- Quantum hardware only supports native 1- and 2-qubit gates
- Toffoli must be broken into CNOT + single-qubit rotations ($H, T, T^\dagger$)
- Standard decomposition requires 6 CNOTs and several $T/T^\dagger/H$ gates



https://algassert.com/quirk#circuit={%22cols%22:[[1,1,%22H%22],[1,%22%E2%80%A2%22,%22X%22],[1,1,%22Z^-%C2%BC%22],[%22%E2%80%A2%22,1,%22X%22],[1,1,%22Z^%C2%BC%22],[1,%22%E2%80%A2%22,%22X%22],[1,1,%22Z^%C2%BC%22],[%22%E2%80%A2%22,1,%22X%22],[1,%22Z^%C2%BC%22,%22Z^%C2%BC%22],[1,1,%22H%22],[%22%E2%80%A2%22,%22X%22],[%22Z^%C2%BC%22,%22Z^-%C2%BC%22],[1,%22X%22],[%22%E2%80%A2%22]]

Why Toffoli Decomposition Matters?

- **Hardware Constraints**
  - Real quantum processors only support native 1-qubit and 2-qubit gates
  - Any multi-controlled gate must be decomposed before execution

- **Gate Cost**
  - Toffoli is relatively expensive: ~6 CNOT gates per decomposition
  - Key target for optimization in circuit compilation

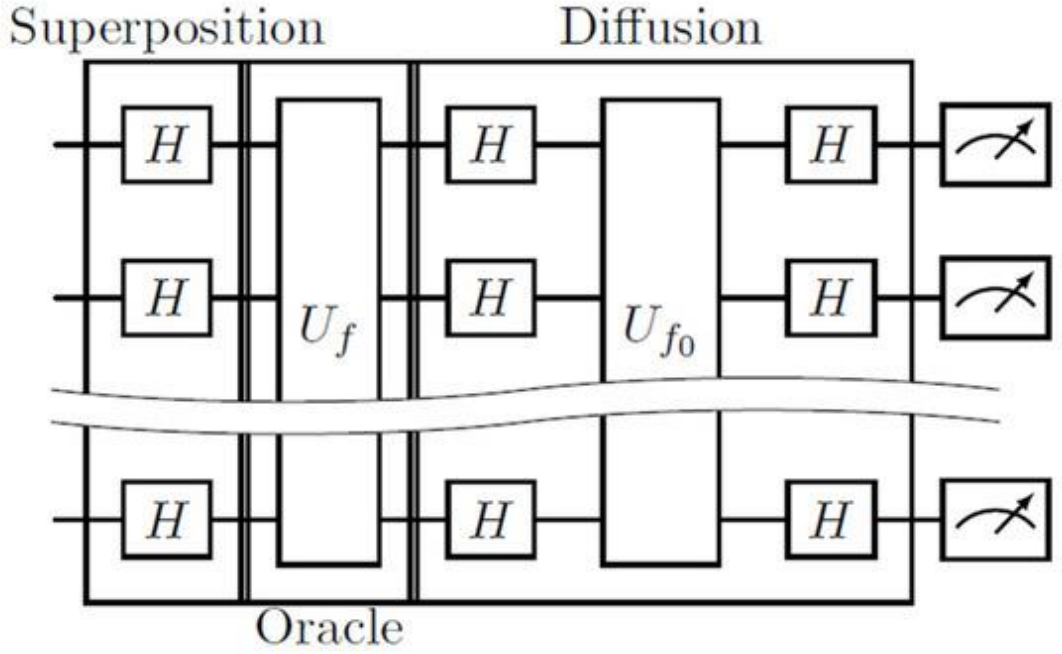
- **Universality**
  - Toffoli + Hadamard = universal for quantum computation
  - Toffoli alone is universal for classical reversible computation

Synthesizing Quantum Oracles

- **Example:** Deutsch Algorithm oracle implements $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$
  - Constant f: oracle is either Identity $(f = 0)$ or X gate on output qubit $(f = 1)$
  - Balanced f: oracle is a $CNOT$ gate with input as control, output as target

Grover's Algorithm & The Oracle Problem

- Grover's algorithm achieves $O(\sqrt{N})$ search vs. classical $O(N)$
- **Three steps of the algorithm:**
  1. **Superposition:** Put all N items into equal probability using Hadamard gates
  2. **Oracle:** Flip the phase of the target item (marks it without revealing it)
  3. **Diffusion:** Amplify the marked state's probability



Grover's Algorithm & The Oracle Problem

- **The oracle mathematically:** $U_f|s\rangle = -|s\rangle$ if $s = \omega$ (target), and $+|s\rangle$ otherwise
- **The core challenge:** The oracle must be custom-built for every new search problem
- Three families of solutions: **Classical Synthesis**, **Metaheuristics**, and **Neural Networks**

Classical Synthesis Methods

- **Approach**
  - Converts a Boolean function into a reversible quantum circuit using algebraic decomposition
  - Key technique: **Zhegalkin** polynomials — represent Boolean functions in a vector space

- **Key Idea:** Every Boolean function $f(x)$ can be represented as a unitary quantum transformation

- For n-bit input: U_f is a $2^{n+1} \times 2^{n+1}$ block-diagonal matrix of I and X blocks
- Example (1-bit functions):
  - $f_0 = 0 \rightarrow U_0 = diag(I, I) — identity$
  - $f_1 = x \rightarrow U_1 = diag(I, X) — CNOT\ gate$
  - $f_2 = \bar{x} \rightarrow U_2 = diag(X, I) — CNOT + NOT$
  - $f_3 = 1 \rightarrow U_3 = diag(X, X) — NOT\ gate$
- Qubits required: n + 1 + a (n = input, a = ancilla for intermediate results)

- Limitations: Requires Boolean function construction; ancilla qubit overhead; limited entanglement use

From Zhegalkin Polynomial to Quantum Circuit*

- Zhegalkin polynomial = Algebraic Normal Form (ANF) of a Boolean function
- Every Boolean function can be uniquely written as XOR of AND terms:
  - $f(x_1, x_2) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_{12} x_1 x_2$
- Basis functions built from tensor products:
  - $e_0 = (1 - x)$ and $e_1 = x$ are the 1-bit basis

- Each term in the Zhegalkin polynomial maps to a quantum gate:
  - Constant term 1 → X gate (NOT) on output qubit
  - Single variable x_j → CNOT (qubit j controls output)
  - Product $x_{j1} \cdot x_{j2}$ → Toffoli / CCNOT (two controls)
  - Product of m variables → $C^m NOT$ gate (m controls)
- Example:

$$f(x_1, x_2) = 1 \oplus x_1 \oplus x_2 \oplus x_1 x_2$$

- Circuit: $X + CNOT(x_1 \to y) + CNOT(x_2 \to y) + CCNOT(x_1, x_2 \to y)$

Oracle Synthesis Algorithm: Step-by-step procedure (Truth Table → Quantum Circuit)*

- Step 1: Start with the truth table of Boolean function f
- Step 2: Compute Zhegalkin polynomial coefficients
  - Build P matrix from p-column vectors
  - Coefficient vector a = $P^{\otimes n} \cdot f$
  - (Tensor product of P matrices applied to truth table column)
- Step 3: Read off the quantum circuit from polynomial terms
  - Each non-zero coefficient → corresponding gate
  - Constant → X, linear → CNOT, quadratic → CCNOT, etc.
- Step 4: Compose gates sequentially on the output qubit
  - Reverse transform: a → f via Q matrix ($Q = P^{-1}$)

Metaheuristic Methodologies

- Approach
  - Use optimization algorithms (Ant Colony, Genetic Programming) to search for optimal circuits
  - Circuits are evolved/searched iteratively using fitness metrics to converge on correct oracles

- Mathematical Foundation
  - Fidelity: $\mathcal{F}(|y\rangle, |x\rangle) = |\langle y|x\rangle|$ (overlap between expected and obtained states)
  - Mean Square Fidelity: $MSF(A, X, Y) = \frac{\Sigma_{i=1}^{|X|} \mathcal{F}(|y\rangle, A|x\rangle)^2}{|X|}$
    - A = realized unitary matrix, X = all inputs, Y = all expected outputs

- Key Methods
  - QAP (Quantum Ant Programming): graph-based pheromone trail reinforcement
  - Genetic Programming: tree-structured circuits with mutation (insert/delete/replace subtrees)

- Limitations: Poor fidelity for large circuits; expensive runtime; does not scale linearly

Neural Network Methodologies

- Approach
  - Use neural networks as classical encoding/decoding systems to generate quantum circuit components
  - Networks learn to map unitary matrices or input-output pairs to gate sequences

- Mathematical Foundation
  - Matrix decomposition (GRU network): $U \approx U_1 \cdot U_2 \ldots U_N$
  - Lie algebra coefficients (RELU network):
  - = coefficients, τ = Lie algebra generators → directly translatable to quantum gates

- Key Architectures
  - GRU + RELU — matrix → simpler matrices → Lie algebra coefficients → gates (Swaddle)
  - Seq2Seq — encoder-decoder on Open QASM; generates + optimizes circuits from text (Cruz-Benito)

- Limitations: Data-dependent; limited scalability (tested up to 3 qubits); non-deterministic results

https://www.sciencedirect.com/science/article/pii/S0375960117308009?ref=pdf_download&fr=RR-2&rr=9e4f0c6609955278

AI/ML Approaches for Circuit Synthesis

- Traditional circuit synthesis relies on deterministic decomposition methods that often produce circuits with excessive gate counts and depths
- AI/ML methods offer data-driven alternatives that can discover more efficient circuit implementations

- Key motivations:
  1. Reducing circuit depth to minimize decoherence effects on NISQ devices
  2. Optimizing gate counts to reduce error accumulation in noisy quantum hardware
  3. Discovering novel circuit topologies that classical heuristics may miss
  4. Automating hardware-aware compilation for specific quantum processor architectures

Machine Learning Techniques for Quantum Circuits

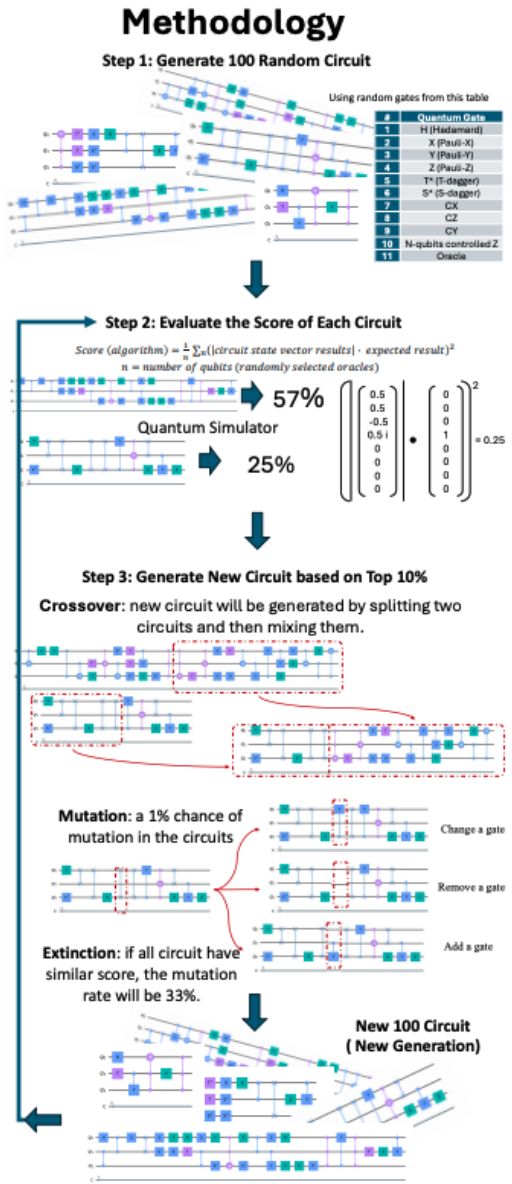
- **Supervised Learning**
  - Neural networks trained on known optimal decompositions to predict efficient gate sequences for new unitaries
  - Graph neural networks (GNNs) that learn circuit structure representations and propose transformations

- **Generative Models**
  - Variational autoencoders (VAEs) that learn latent representations of circuit families and generate new candidates
  - Generative adversarial networks (GANs) for producing circuits that approximate target unitary matrices

- **Evolutionary Algorithms**
  - Genetic programming that evolves circuit structures through crossover and mutation of gate sequences
  - Multi-objective optimization balancing circuit depth, gate count, and fidelity simultaneously

Discovery of Quantum Algorithms Using Genetic Algorithms: Exponential Speedup via Random Sampling*

Methodology



Results

Our experiments used a genetic algorithm to discover oracle-based quantum algorithms in circuits with 3 to 8 qubits. By reducing evaluation time complexity from $O(2^n)$ to $O(n)$, we efficiently optimized larger circuits.

•**Grover's Algorithm Discovery**: The genetic algorithm identified the first iteration of Grover's algorithm in multiple experiments, proving the effectiveness of our approach. Sample of found circuits is shown in **Figure 1**.

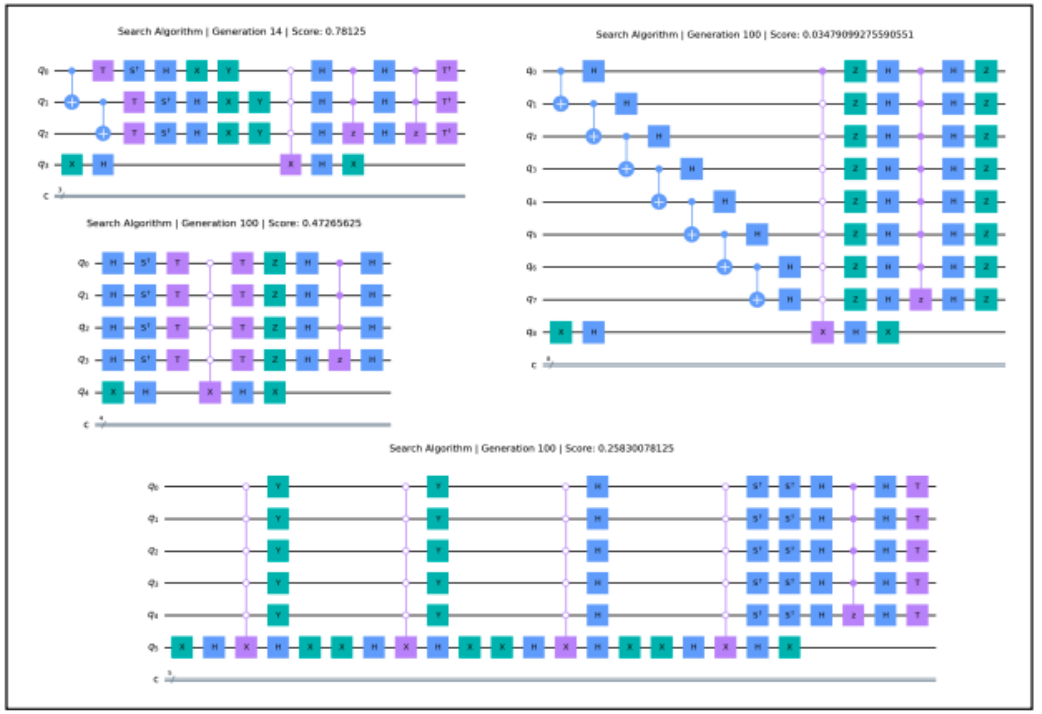


Figure 1: Sample of the best circuits found

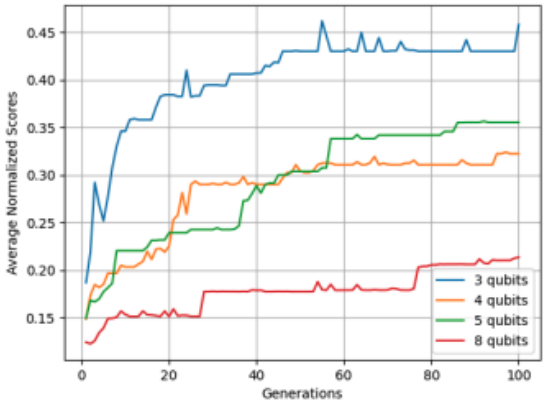


Figure 2: Average Normalized Scores Over Generations

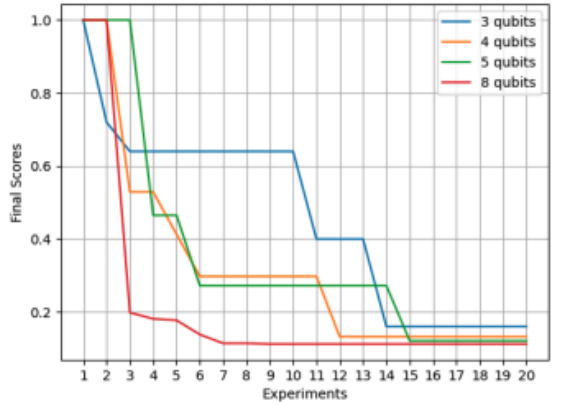


Figure 3: Final Normalized Scores of Each Experiment

PI: Tariq Almuqbil – MX Student

*Demo paper – IEEE QCE 2024

Reinforcement Learning for Gate Optimization

- RL formulates circuit synthesis as a sequential decision-making problem:
  1. **State:** The current partial circuit (sequence of gates applied so far)
  2. **Action:** Selecting the next quantum gate and its target qubits from the allowed gate set
  3. **Reward:** Fidelity of the synthesized circuit to the target unitary, penalized by circuit depth

- Common RL approaches used:
  - **Deep Q-Networks (DQN):** Learn value functions over discrete gate selection actions
  - **Policy Gradient Methods:** Directly optimize parameterized policies for continuous gate parameters (e.g., rotation angles)
  - **Monte Carlo Tree Search (MCTS):** Explore gate sequence trees with rollouts guided by learned value estimates

Quantum Architecture Search (QAS)

- Quantum Architecture Search (QAS) is inspired by Neural Architecture Search (NAS) in classical deep learning

- **Goal:** Automatically discover optimal variational quantum circuit (VQC) ansatz structures for a given task

- **Why QAS matters:**
  - Manual ansatz design requires deep quantum expertise and often fails to balance expressivity vs. trainability
  - Barren plateaus and quantum noise make generic ansatz patterns unreliable on NISQ hardware
  - QAS can automatically tailor circuits to both the computational problem and the physical device

- **The four dimensions of any QAS method:**
  1. **Search Space:** what gates/layers are available (gate-wise, layer-wise, block-wise)
  2. **Encoding:** how a circuit is represented (list, DAG, tensor, graph)
  3. **Search Strategy:** how the space is explored (RL, EA, differentiable, etc.)
  4. **Performance Evaluation:** how candidates are scored (full training, predictors, zero-cost proxies)

Variational Quantum Circuits (VQC)

What is a VQC?

- Parameterized quantum circuit with tunable rotation angles θ
- Classical optimizer updates θ to minimize a cost function
- Hybrid quantum-classical optimization loop

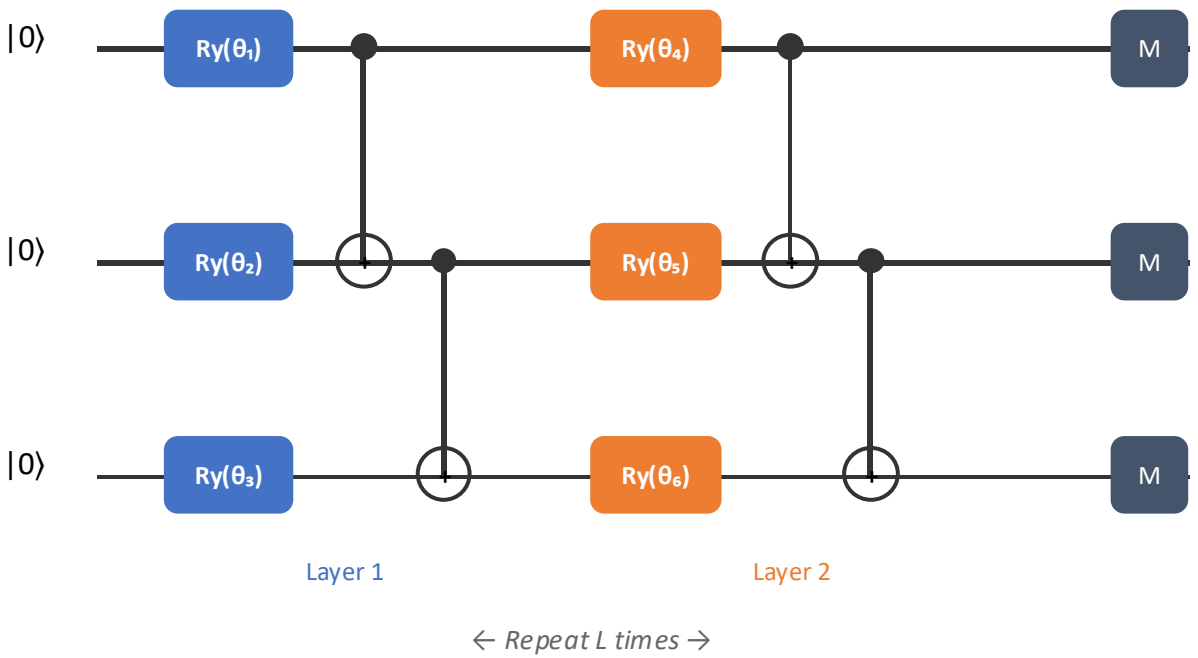
Ansatz Structure

- Layer of single-qubit rotations $Ry(\theta)$
- Entangling layer of CNOT gates
- Repeated L times (circuit depth)

Applications (more on this in COE531)

- VQE
- QAOA
- QML

Example: 3-Qubit Hardware-Efficient Ansatz



https://pqc-prototype-xzxvt65tsamxivpntvqrka.streamlit.app/

Classical Optimization Loop
Measure → Evaluate cost C(θ) → Update θ → Re-run circuit
Converges to min C(θ*)

QAS Methods & Frameworks

- **Search Space Definition**
  - Gate pool: single-qubit rotations (Rx, Ry, Rz), entangling gates (CNOT, CZ), and parametric gates
  - Topology constraints: which qubit pairs can interact, defined by the hardware connectivity graph
  - Depth budget: maximum number of circuit layers allowed

- **Search Strategies**
  - **Differentiable QAS (DQAS):** Relaxes discrete gate choices into continuous parameters optimized via gradient descent
  - **RL-based QAS:** An RL agent sequentially places gates, receiving reward based on task performance
  - **Evolutionary QAS:** Populations of candidate circuits evolve through selection, crossover, and mutation
  - **Bayesian Optimization:** Surrogate models guide efficient exploration of the architecture space

Search Strategies: RL, Evolutionary & Differentiable

- **1. Reinforcement Learning (RL)**
  - Models circuit construction as a Markov Decision Process — agent adds gates sequentially, rewarded by circuit performance
  - Algorithms: PPO, DDQN, A3C; CRLQAS uses curriculum learning to gradually increase task difficulty
  - Weakness: sample-inefficient, requiring millions of environment interactions

- **2. Evolutionary Algorithms (EA)**
  - Population of circuit candidates evolves via mutation and crossover
  - Notable: MoG-VQE (multi-objective), EVQE (identity-initialized), QCEAT (multi-species with robustness)
  - Some implementations avoid crossover — combining two circuits doesn't guarantee inherited performance

- **3. Differentiable QAS**
  - Relaxes discrete gate selection into continuous probability distributions optimized via gradient descent
  - Key implementations: DQAS (Monte Carlo gradients), QuantumDARTS (Gumbel-softmax)
  - Jointly optimizes architecture probabilities and gate parameters in one pass

Search Strategies: Adaptive, MCTS, BO & Generative

- **4. Adaptive / Greedy Algorithms**
  - Iteratively add or remove gates using gradient signals or heuristic rules
  - **ADAPT-VQE:** greedily appends the fermionic operator with the largest gradient — highly popular for chemistry
  - **VAns:** stochastically inserts gates, then prunes via rule-based classical logic

- **5. Monte Carlo Tree Search (MCTS)**
  - Encodes the search space as a tree; nodes are gates, paths are circuits
  - Balances exploration vs. exploitation via Upper Confidence Bounds; MCGS extends this to a graph

- **6. Bayesian Optimization (BO)**
  - Surrogate probabilistic model (Gaussian Processes, Tree Parzen Estimators) guides efficient sampling
  - Best suited for small search spaces where each circuit evaluation is very expensive

- **7. Generative Models**
  - **GPT-QE:** GPT-2-style transformer generates gate sequences, framing circuit design as language generation
  - Supports pre-training and fine-tuning for transfer learning across tasks; GANs/VAEs also applicable

Open Challenges & Key Takeaways

- **Open Challenges:**
  - **Scalability:** most methods tested on ≤10 qubits; scaling to larger circuits remains unsolved
  - **Hardware adaptability:** re-calibrated QPUs may require expensive QAS retraining
  - **Benchmarks:** no standardized benchmark suite exists, making method comparisons difficult
  - **Noise modeling:** better noise-aware evaluation metrics needed beyond simple expectation value perturbations
  - **Transfer learning:** pre-trained QAS models for one problem type rarely transfer cleanly to another

- **Key Takeaway:**
  - QAS sits at the intersection of quantum computing, AutoML, and hardware-aware optimization
  - Most mature techniques today: RL-based and adaptive methods (especially ADAPT-VQE)
  - Differentiable methods and generative models are gaining ground as scalability bottlenecks are addressed