

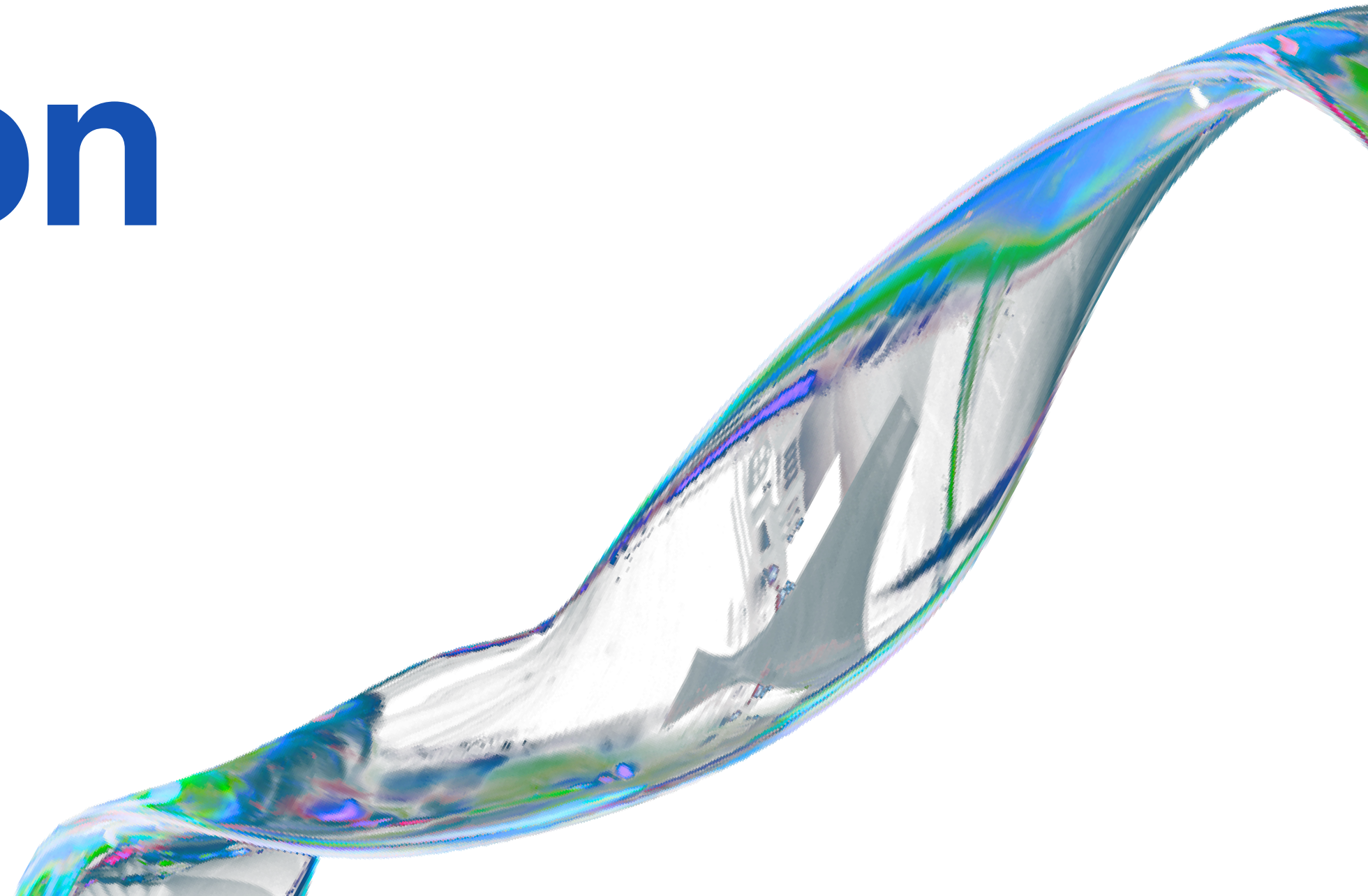


**Team  
Super(position)**

Github: @quantum\_for\_portfolio\_optimization

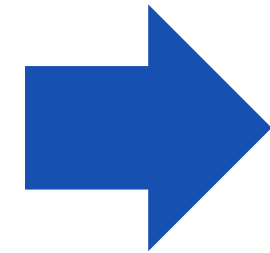
# Portfolio Optimization

Using QAOA

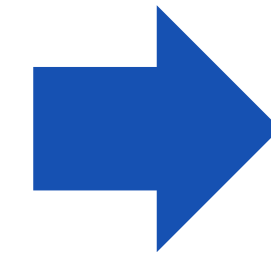




**portfolio  
problem  
QUBO**



**quantum  
Hamiltonian**



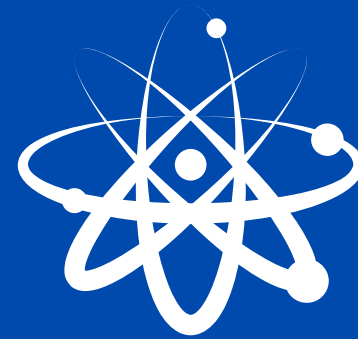
**ground state  
= optimal  
portfolio**

Challenge: Complex portfolio optimization with multiple constraints

- Portfolio size constraints (target N bonds from n available)
- Cash flow bounds (yield requirements)
- Risk characteristic limits (duration/credit targets)



Exponential complexity,  
local optima traps



QAOA for combinatorial  
optimization



## Deliverables

- 1) Review the mathematical formulation provided below, focusing on binary decision variables, linear constraints, and the quadratic objective.
- 2) Convert the binary optimization problem to a formulation that is compatible with a quantum optimization algorithm. For example, convert the constrained problem to an unconstrained problem.
- 3) Write a quantum optimization program for handling problems of the type in (2). An example of such an optimization routine which is used in portfolio optimization is the Variational Quantum Eigensolver (see resources below), however you may pursue what you judge to be the best solution.
- 4) Solve the optimization problem in (1) using your quantum formulation.
- 5) Validate your solution in (4) using a classical optimization routine. Compare the solution quality against the benchmark classical solution in terms of the cost function, and include relevant performance metrics(e.g., convergence of the optimization routine, and scaling properties with problem size).



## The Equations, Parameters, Constraints, and Objective Function

### Input parameters

A set of securities  $C$  with:

- $p_c$  market price
- $m_c$  min trade
- $M_c$  max trade
- $i_c$  basket inventory
- $\delta_c$  minimum increment

A set  $L$  of risk buckets

- $\mathbb{K}_\ell$  are the bonds in bucket  $\ell$  (not mutually exclusive)

A set  $J$  of characteristics with:

- $K_{\ell,j}^{\text{target}}$  target of characteristic  $j$  in risk bucket  $\ell$
- $b_{\ell,j}^{\text{up}}$  and  $b_{\ell,j}^{\text{low}}$  guardrails for characteristic  $j$  in risk bucket  $\ell$
- $\beta_{c,j}$  contribution of a unit of bond  $c$  to the target of characteristic  $j$
- $K_{\ell,j}^{\text{up}}$  and  $K_{\ell,j}^{\text{low}}$  guardrails for characteristic  $j$  in risk bucket  $\ell$  (binary version)

Global parameters:

- $N$  max number of bonds in portfolio
- Min/max residual cash flow of portfolio

### Decision variables

- $y_c$  whether bond  $c$  is included in the portfolio

Here  $x_c$  (how much of bond  $c$  is included in the basket) is not a variable, but is fixed to the average value it is allowed to have if  $c$  is included at all in the portfolio:

$$x_c = \frac{m_c + \min\{M_c, i_c\}}{2 \delta_c} y_c$$

### Constraints

- Maximum number of bonds in basket

$$\sum_{c \in C} y_c \leq N$$

- Residual cash flow of portfolio

$$\frac{\max RC}{MV^b} \leq \sum_{c \in C} \frac{p_c}{100} \frac{\delta_c}{MV^b} x_c \leq \frac{\min RC}{MV^b}$$

- Min/max value of each characteristic  $j$  in each risk group  $\ell$

$$\sum_{c \in \mathbb{K}_\ell} \frac{p_c}{100} \frac{\delta_c}{MV^b} \beta_{c,j} x_c \leq b_j^{\text{up}}, \quad \forall j \in J, \ell \in L,$$

$$\sum_{c \in \mathbb{K}_\ell} \frac{p_c}{100} \frac{\delta_c}{MV^b} \beta_{c,j} x_c \geq b_j^{\text{low}}, \quad \forall j \in J, \ell \in L.$$

- Similar constraint on the binary variable  $y_c$

$$\sum_{c \in \mathbb{K}_\ell} \beta_{c,j} y_c \leq K_j^{\text{up}}, \quad \forall j \in J, \ell \in L,$$

$$\sum_{c \in \mathbb{K}_\ell} \beta_{c,j} y_c \geq K_j^{\text{low}}, \quad \forall j \in J, \ell \in L.$$

Constraints that guarantee consistency between  $x_c$  and  $y_c$  are no longer needed since  $x_c$  is not a variable in this model

### Objective function

Match the value of each characteristic  $j$  in each risk group  $\ell$  to its target (based on the quantity  $x_c$  in the basket)

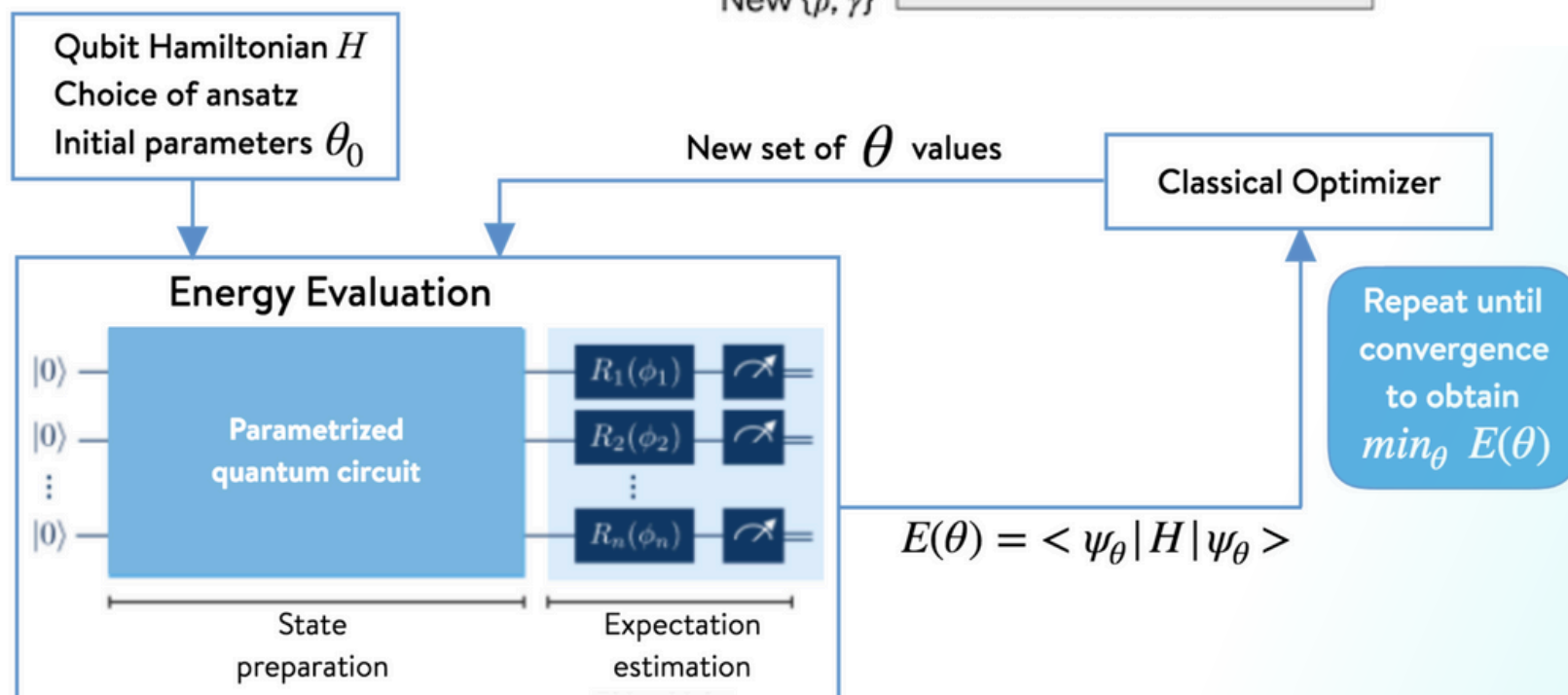
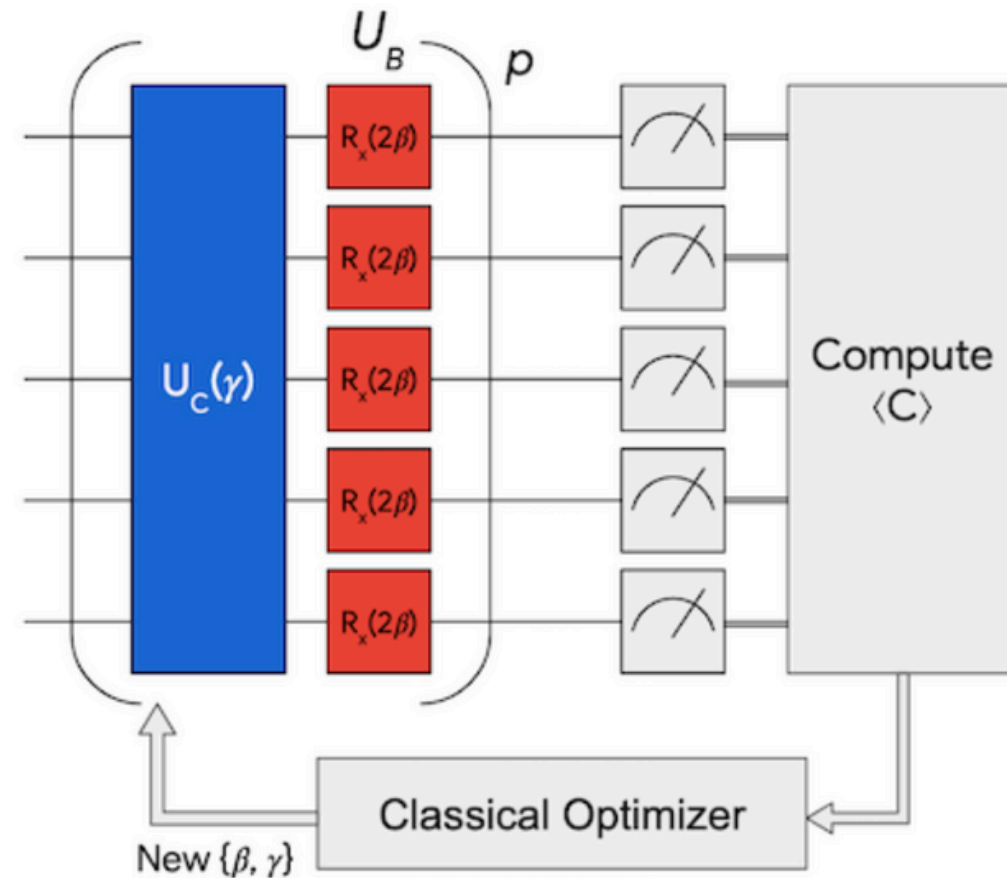
$$\min \sum_{\ell \in L} \sum_{j \in J} \rho_j \left( \sum_{c \in \mathbb{K}_\ell} \beta_{c,j} x_c - K_{\ell,j}^{\text{target}} \right)^2$$





## Overall summary of process

Overview  
diagrams of  
quantum  
circuits



Loading in Data + QUBO Formulation

Building+Solving QAOA

Backend + memory

Post-processing + selecting best solution



## Step 2: Loading in the Vanguard 2600+ Bonds Data (Part I)

### To Load the Vanguard Data:

- Filter for corporate bonds (main focus for VCIT Fund)
- Select top holdings by market value for quantum optimization:
  - Ensures we work best for the most significant positions
- Extract expected returns from bond characteristics:
  - 1.) Credit spread (OAS) - Compensation for credit risk
  - 2.) Duration - Interest Rate sensitivity
  - 3.) Current Yield Environment
  - Make sure risk-free rate is 4.5% to approximate the current 10-year treasury
  - Extract OAS in basis points → Convert to decimal (divide by 10000)
  - Expected Return = Risk-free rate + Credit Spread
- Extract Risk measures from bond duration and spread characteristics:
  - Interest Rate Risk and Credit Risk (measured by duration and spread volatility, respectively)
  - Higher Spreads → Higher credit risk.
  - Combine Risk Measure and ensure reasonable risk bounds to 0.5% to 15% annual volatility



## Step 2: Loading in the Vanguard 2600+ Bonds Data (Part II)

### To Load the Vanguard Data:

- Estimate correlation matrix based on sector and credit quality clustering, duration buckets, and general market factor
  - Loop through the number of assets in  $O(N^2)$  Time
    - Make 0.05 be the base market correlation
      - If same sector correlation boost, add 0.25 to correlation matrix
      - If same credit quality correlation boost, add 0.15 to correlation matrix
      - Otherwise (similar duration), if within 2 years duration, add 0.10
    - Ensure to cap max correlation between under 0.85
    - Ensure Positive Semi-definite matrix and normalize diagonal to 1
- Of course, calculate current weights, get asset identities, and extract risk factors and sector classifications
- And then, you're done! You now can load the vanguard data into your workflow!



## **Step 2.5 : Converting the QUBO problem into the right formation to be used by our VQA with QAOA Algorithms**

**Our QUBO problem consists of expected returns, risk measures, asset names, current weights, and the portfolio information, as well as the correlation matrix for sector and credits**

- Target basket size should be minimum 8 bonds for optimization OR it depends on the bonds (qubits) you're simulating on
- Min Portfolio Yield (3%) & Max Portfolio yield (6%) due to the dataset
- Market value base is normalized (1.0) to avoid any division by 0
- K\_target is based on the portfolio average
- Portfolio size is 1000, cash flow upper and lower bound penalty is 300, and risk characteristic penalty is 200
- Scaled current weights to optimization range (current weights \* bonds \* 2)





## Step 2.5 (cont): Converting the QUBO problem into the right formation to be used by our VQA with QAOA Algorithms

### To build the QUBO Formulation Part 1:

- Prepare a single bond "bucket" indexed by 'l,j' and pulls out its weight and target return
- Prepare 2 structures:
  - Q: an  $n \times n$  matrix capturing all quadratic (pairwise) couplings between decision bits
  - q: A length-n vector capturing all linear biases on each bit
- Prepare Risk-Adjusted Return Term:
  - Penalized or rewards selecting pairs (i,k) in proportion to their joint exposure ( $\beta[i,j] * \beta[k,j]$ ) scaled by the bond weight and any client-specific sizes  $x_c$
  - Adds a real world risk penalty via the covariance between assets i and k



## Step 2.5 (cont): Converting the QUBO problem into the right formation to be used by our VQA with QAOA Algorithms

### To build the QUBO Formulation Part 2:

Apply a linear reward (negative bias) for selecting each asset  $i$ , scaled by how it helps hit the target return + add additional incentive proportional to the asset's standalone expected return

#### basket-size constraint:

- Implement a soft constraint on the total # of bonds selected
  - Uniform positive couplings between all pairs push the solver to pick exactly  $N$  ones
- This ensures a proper penalty

#### risk-contribution bounds:

- Computes each asset's fractional contribution to total risk ( $a_{cf}$ ) (see github code)
- Adds 2 squared-error penalties to ensure each individual contribution stays between  $rc_{min}$  and  $rc_{max}$

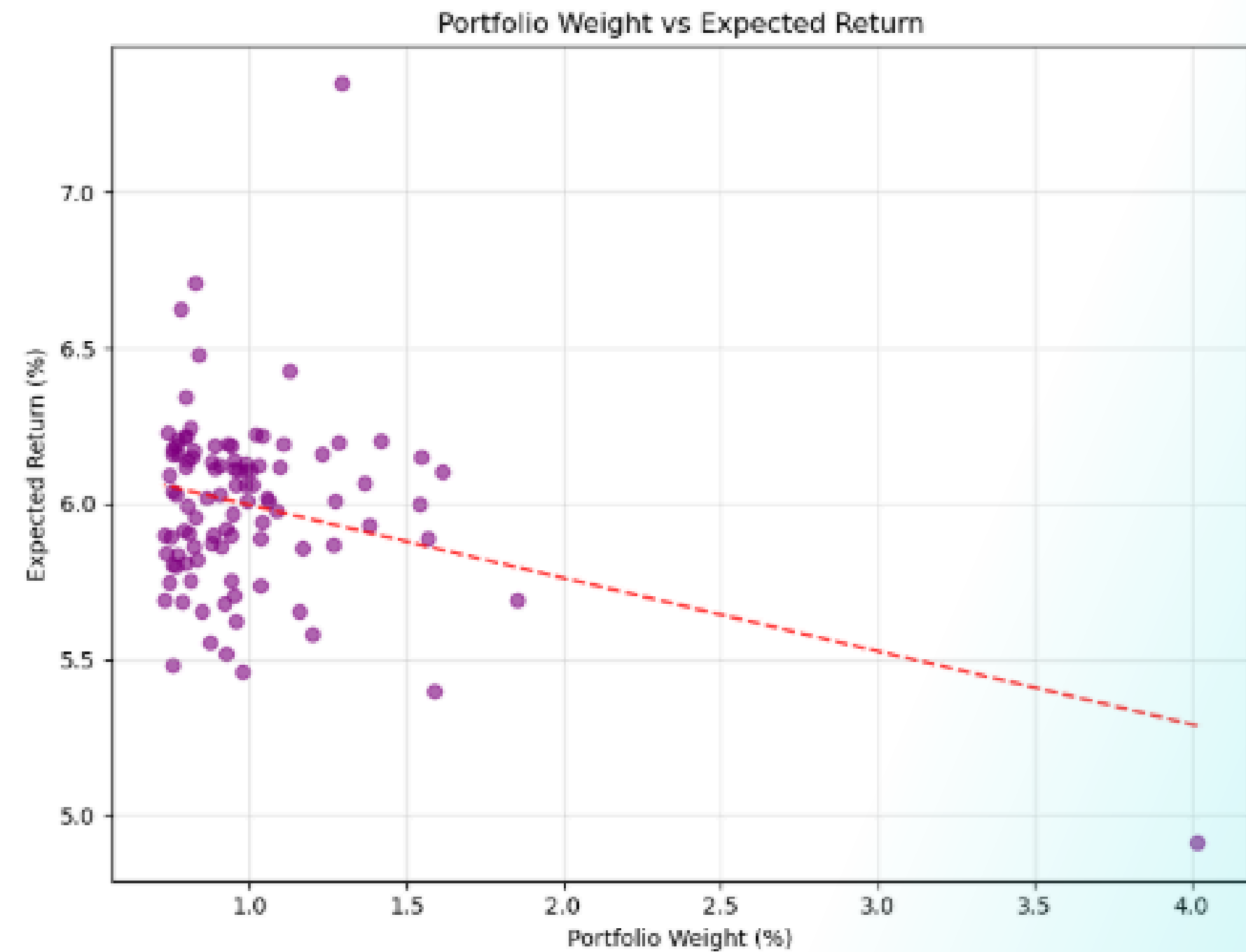
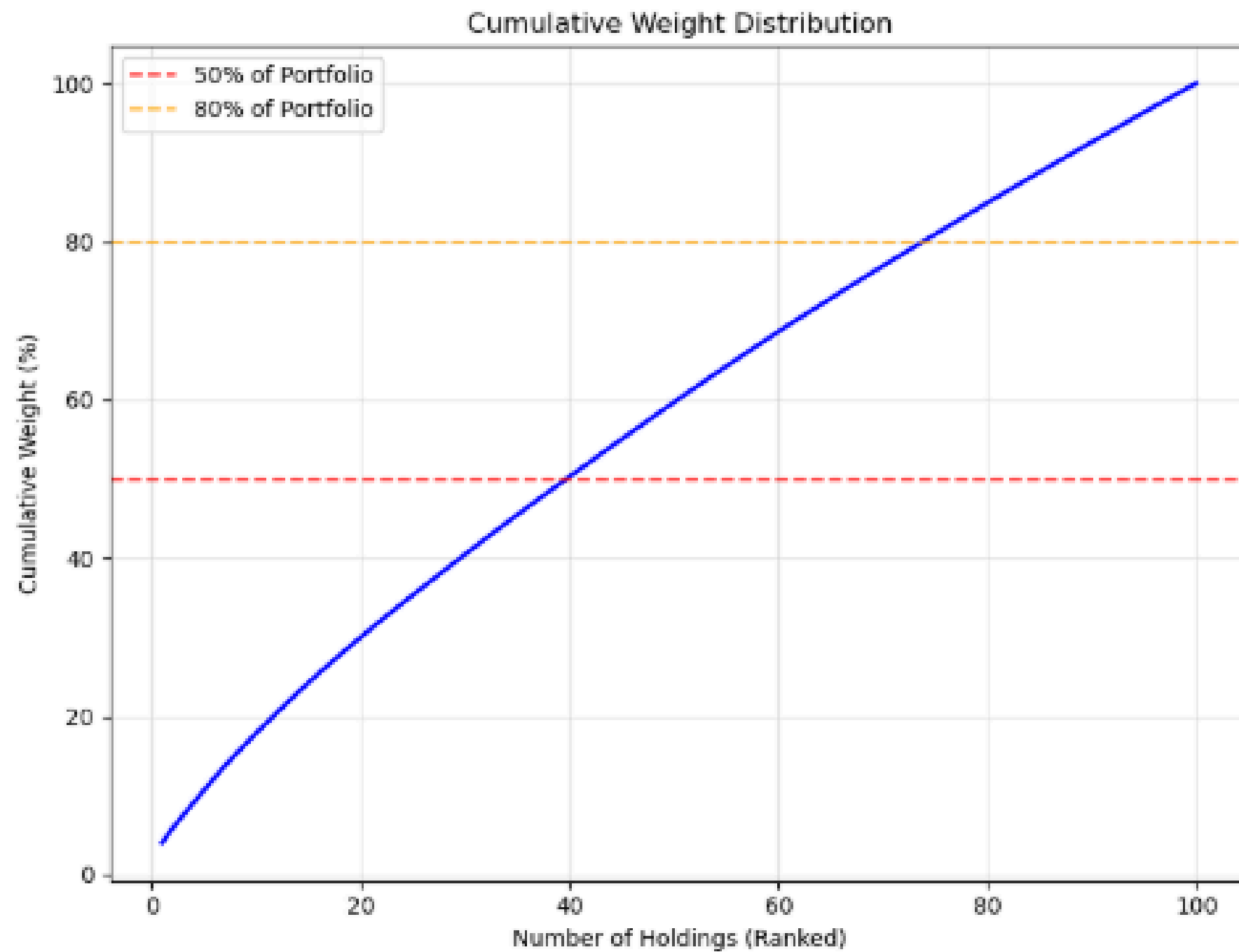
#### characteristic-level bounds:

- Builds a penalty forcing the portfolio's overall exposure, e.g. duration, credit quality, to lie between a lower bound and upper bound ( $b_{lo}$  and  $b_{up}$ )

**You MUST ensure  $Q$  is perfectly symmetric (REQUIRED by most QUBO solvers)**

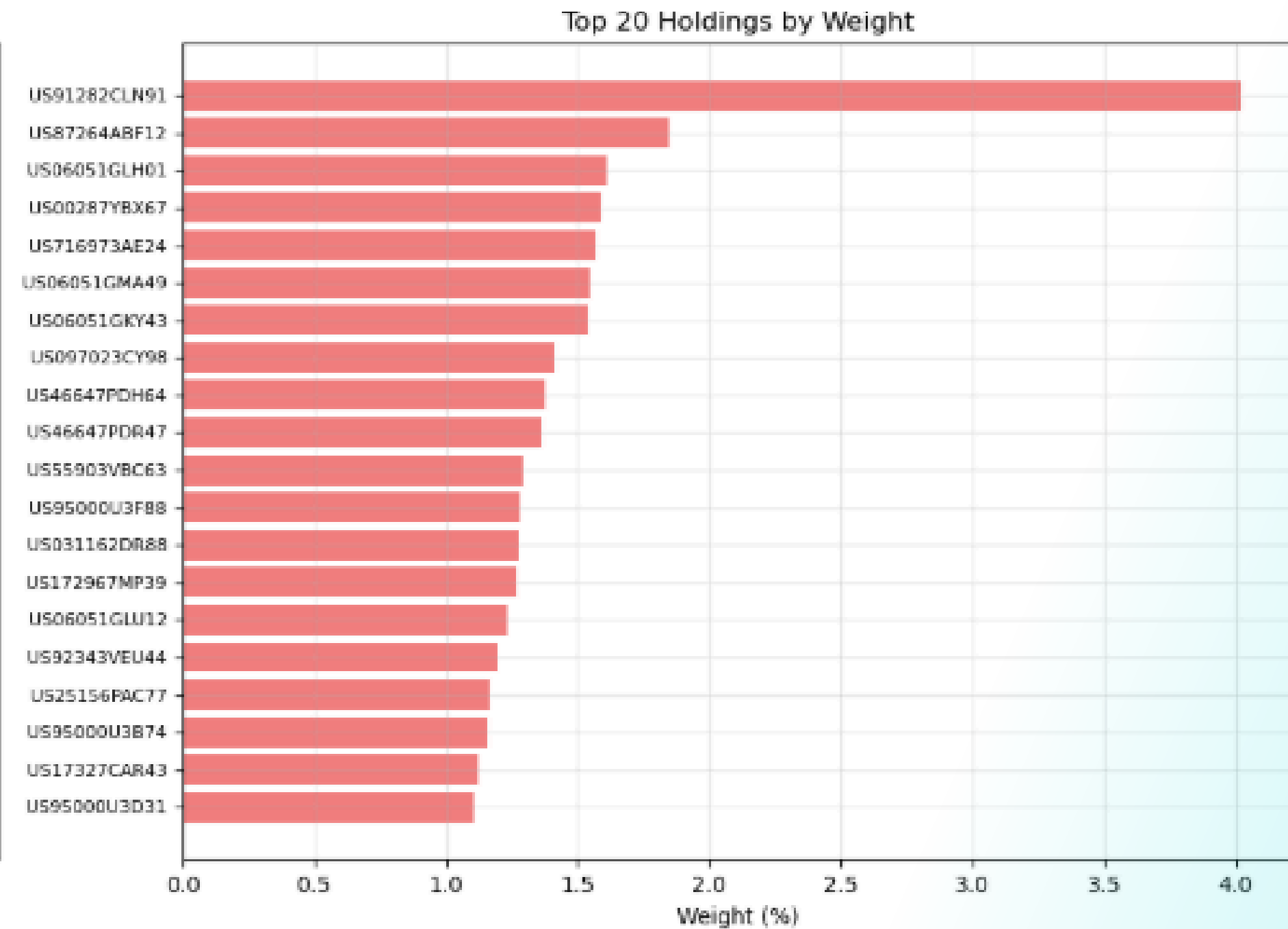
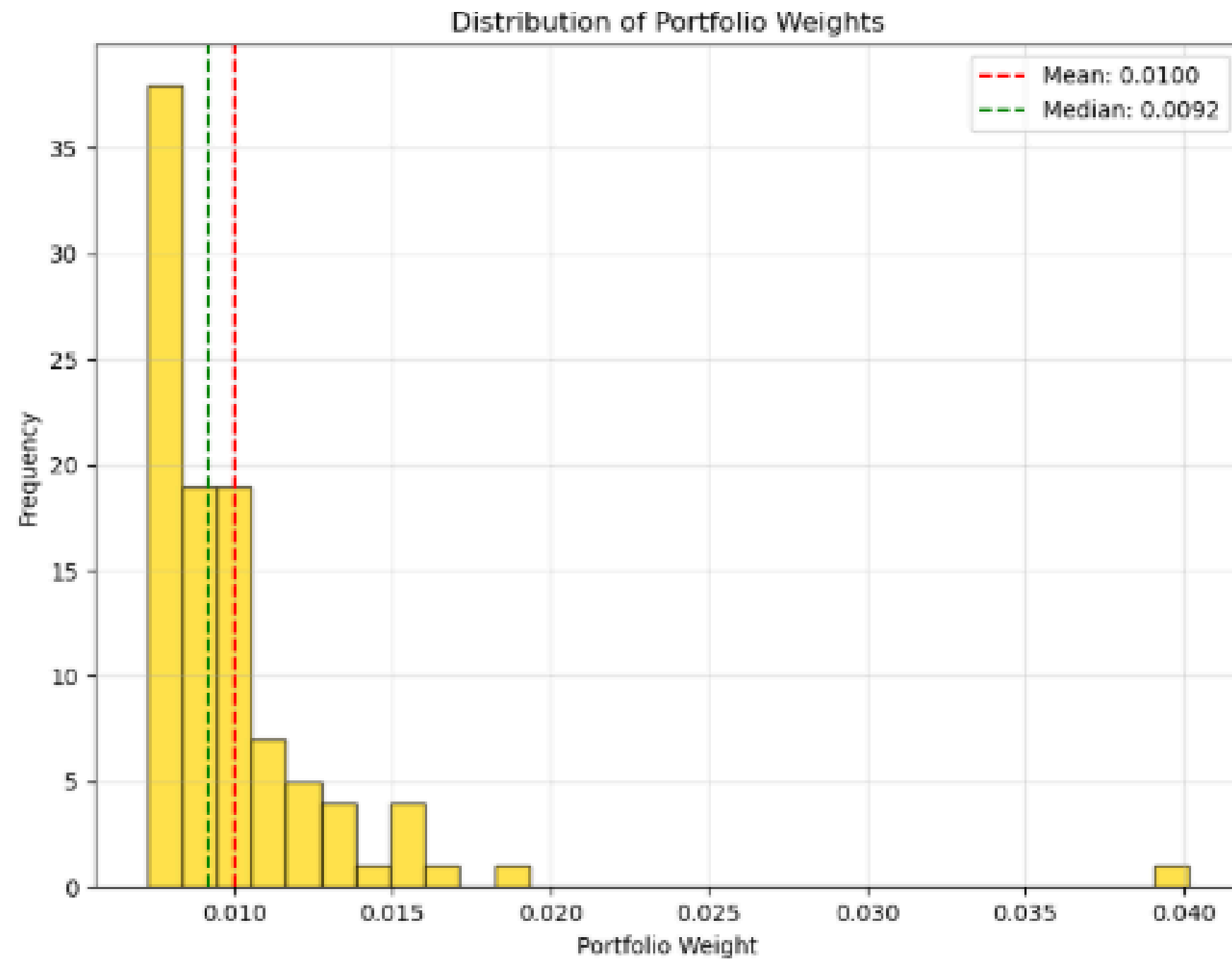


## Overview of the Vanguard Data (Quick Classical Run: No Quantum Computing yet )





## Overview of the Vanguard Data Part II



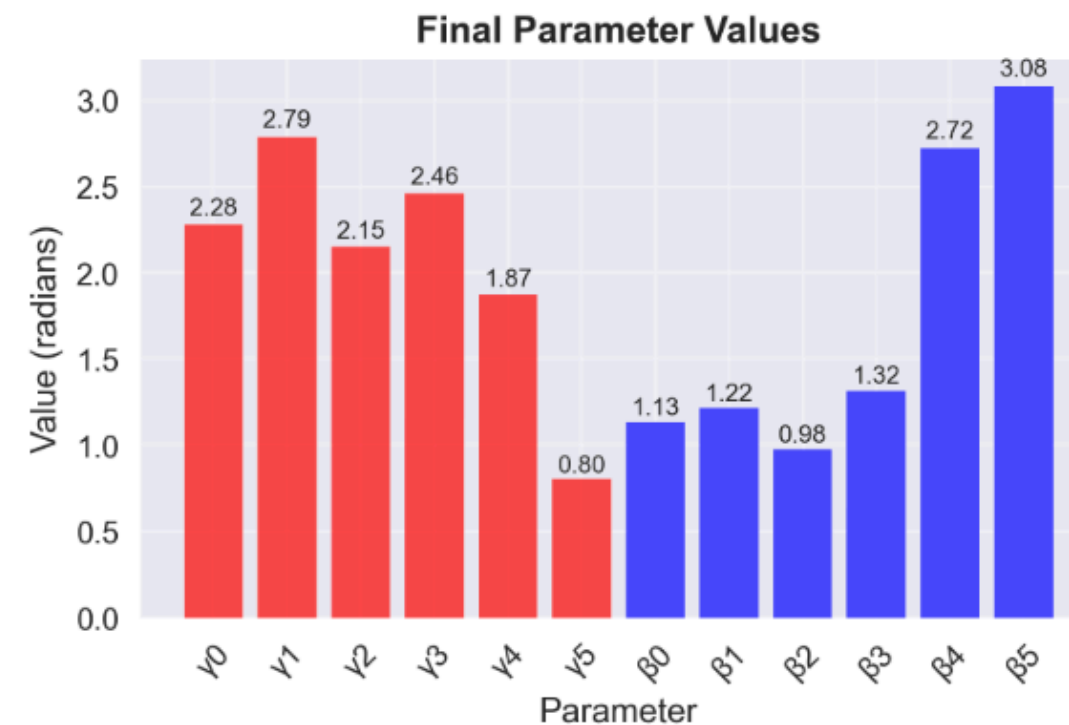
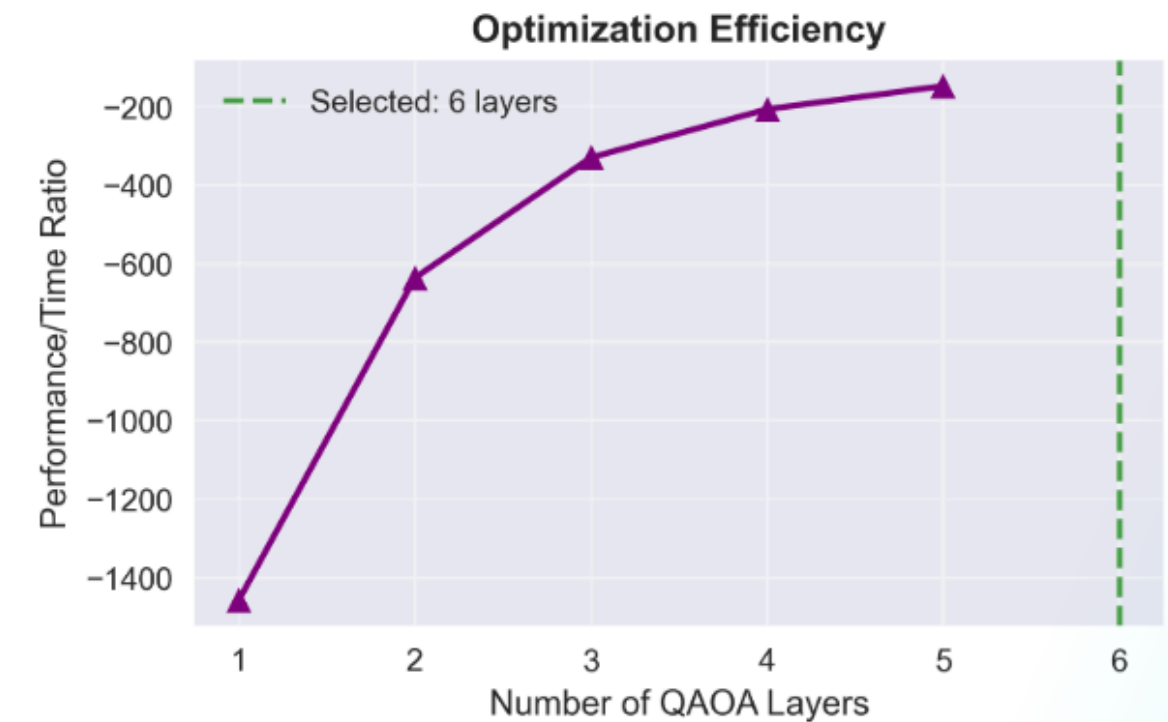
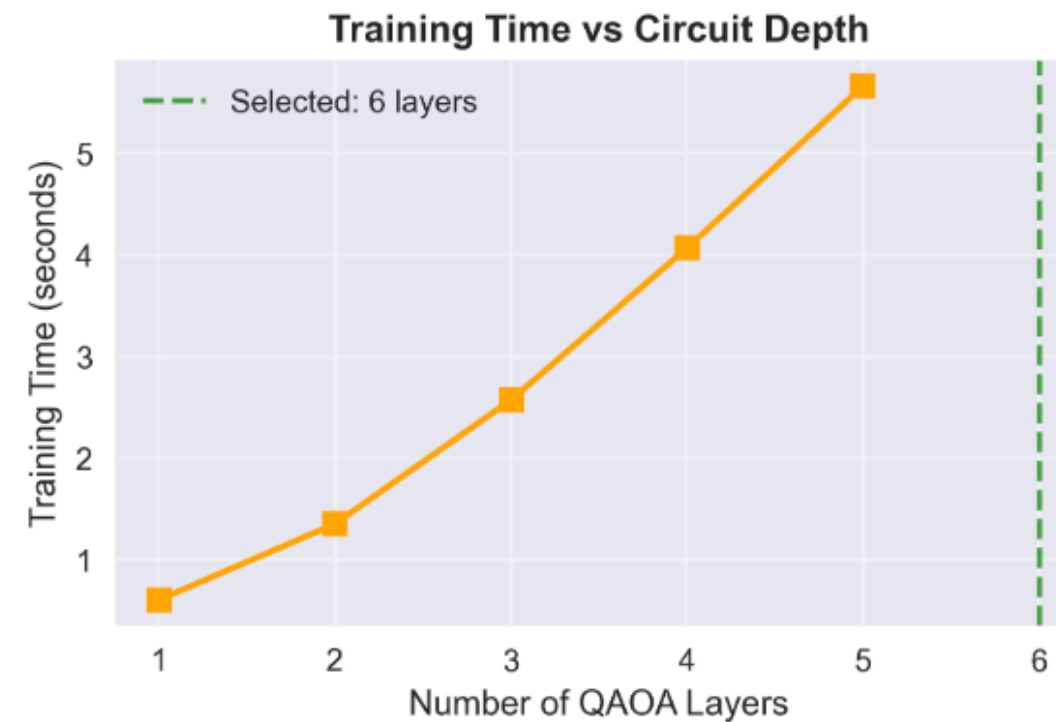




## Step 3: Building the VQE with the optimized QAOA circuit

- **Parameterization:**  
2p parameters  
( $\gamma_1 \dots \gamma_p, \beta_1 \dots \beta_p$ )
- **Layers:** Adaptive p  
= 2-6 based on  
problem size
- **Gate Optimization:**  
Significant gates  
filtering

note visualizations are  
for a run with just 10  
qubits to make them  
easy to follow





## Step 4: Hardware-optimized backend selection

### Memory Analysis:

- State Vector:  $2^n \times 16$  bytes (complex128)
- 24 qubits: ~268 MB
- 31 qubits: ~32 GB (memory limit!)

### Backend Priority:

1. Lightning GPU  
( $n \leq 28$ , <8GB GPU memory)
2. Lightning CPU  
(state\_memory < 70% RAM)
3. Default.qubit (fallback)

You can see that simulating 31 qubits classically is very computationally heavy - however, high fidelity, effective 31 qubits are capable of solving optimization problems extremely well. As you will see in the results, qubit capabilities scale exponentially - just like the problem difficulty.



## Step 5: Optimizing Circuit

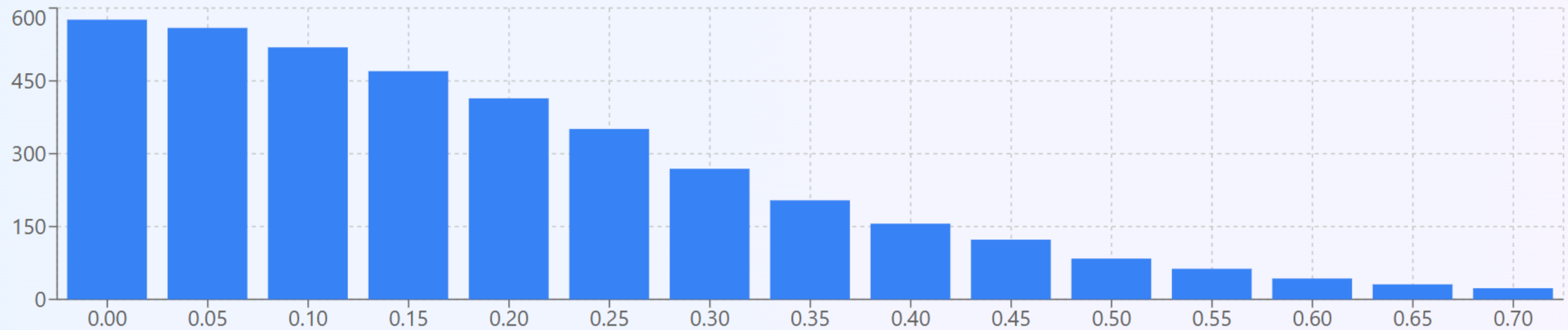
### 1. Threshold Filtering

Significance Threshold: 0.32

**Before filtering:** 576 gate operations (24 qubits)

**After filtering:** 238 operations

**Reduction:** 58.7%





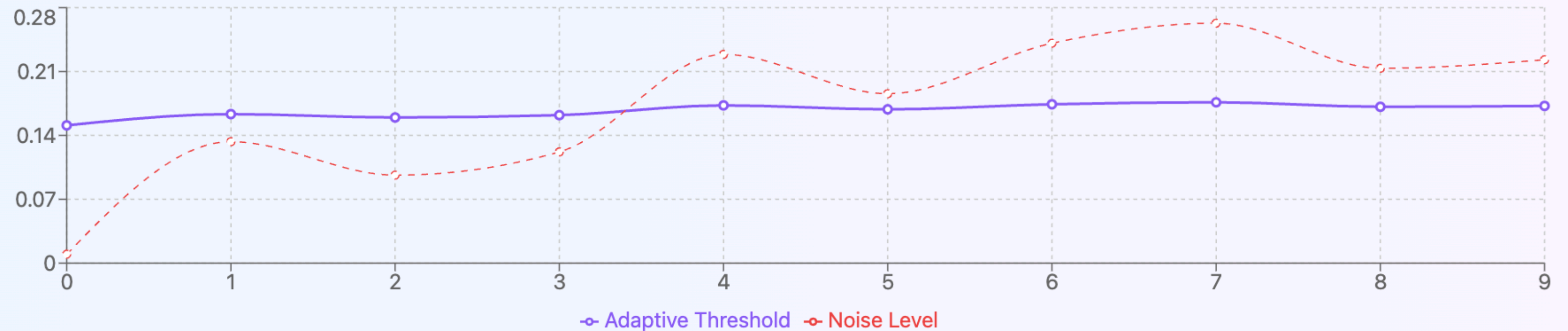
## Step 5: Optimizing Circuit

### 2. Adaptive Threshold Tuning

**Current Threshold:** 0.100

**Purpose:** Helps cancel out noise

**Adaptation:** Responds to circuit performance



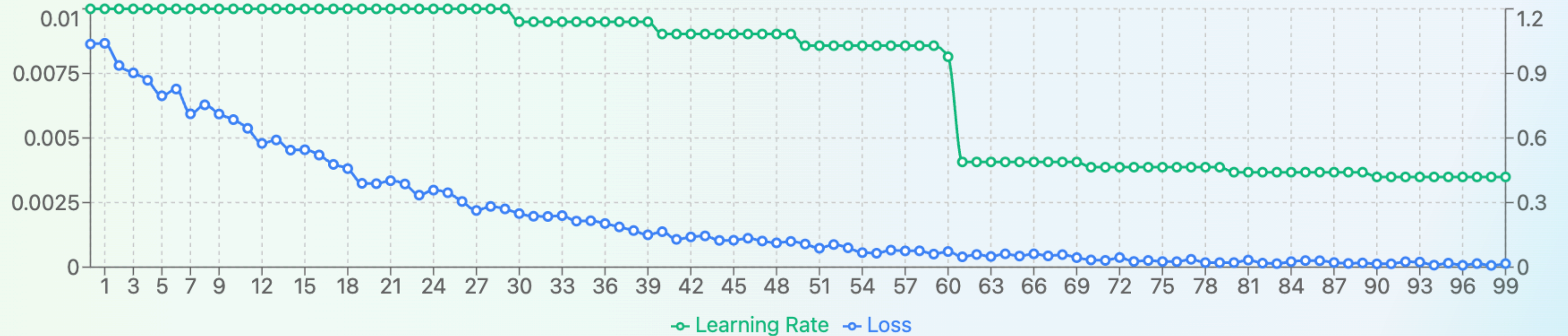




## Step 5: Optimizing Parameter

### 3. Adaptive Learning Rate

**Schedule:** Exponential decay  
**Decay rate:** 0.95 every 10 epochs  
**Additional decay:** 0.5x at epoch 60  
**Purpose:** Fine-tune convergence





## Quick Comparison of Classical vs Quantum Vanguard Run #5

Constraint Analysis of 24 bonds, basket size 8 (2 QAOA layers deep)

- More graphs under vanguard\_quantum\_analysis-4 folder in the GitHub

Basket Size:  $\min(8, \max(5, n // 3))$ , where  $n$  is the actual number of bonds loaded

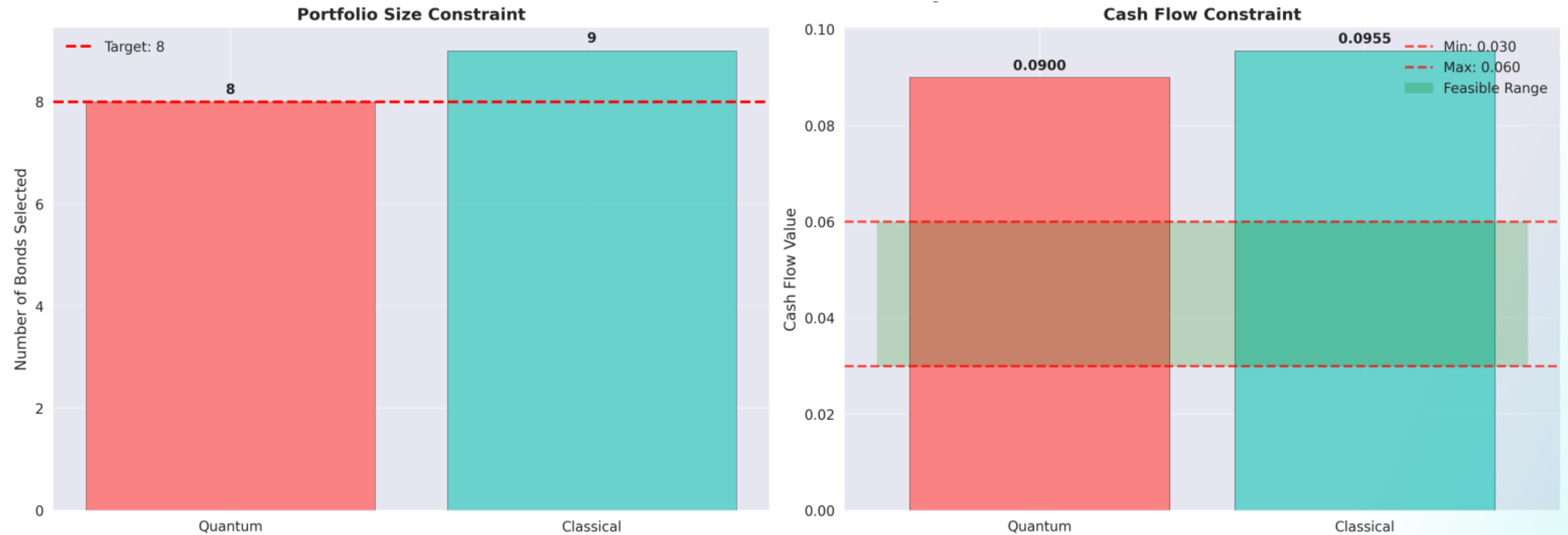
- So if we have 24 bonds  $\rightarrow \min(8, \max(5, 8)) \rightarrow \min(8, 8) \rightarrow 8$  basket size

Solution	Basket Size	Cash Flow	Characteristic	Total Violation	Cost
Quantum	8	0.0900	0.3478	<b>0.0822</b>	-72117
Classical	9/8	0.0955	0.3793	<b>1.0562</b>	-72122

- 24 bonds (qubits) took 10+ hours on a Nvidia 4060 TI GPU! (using Lightning.gpu as well on PennyLane)
- 20 bonds took 30 minutes on the same GPU

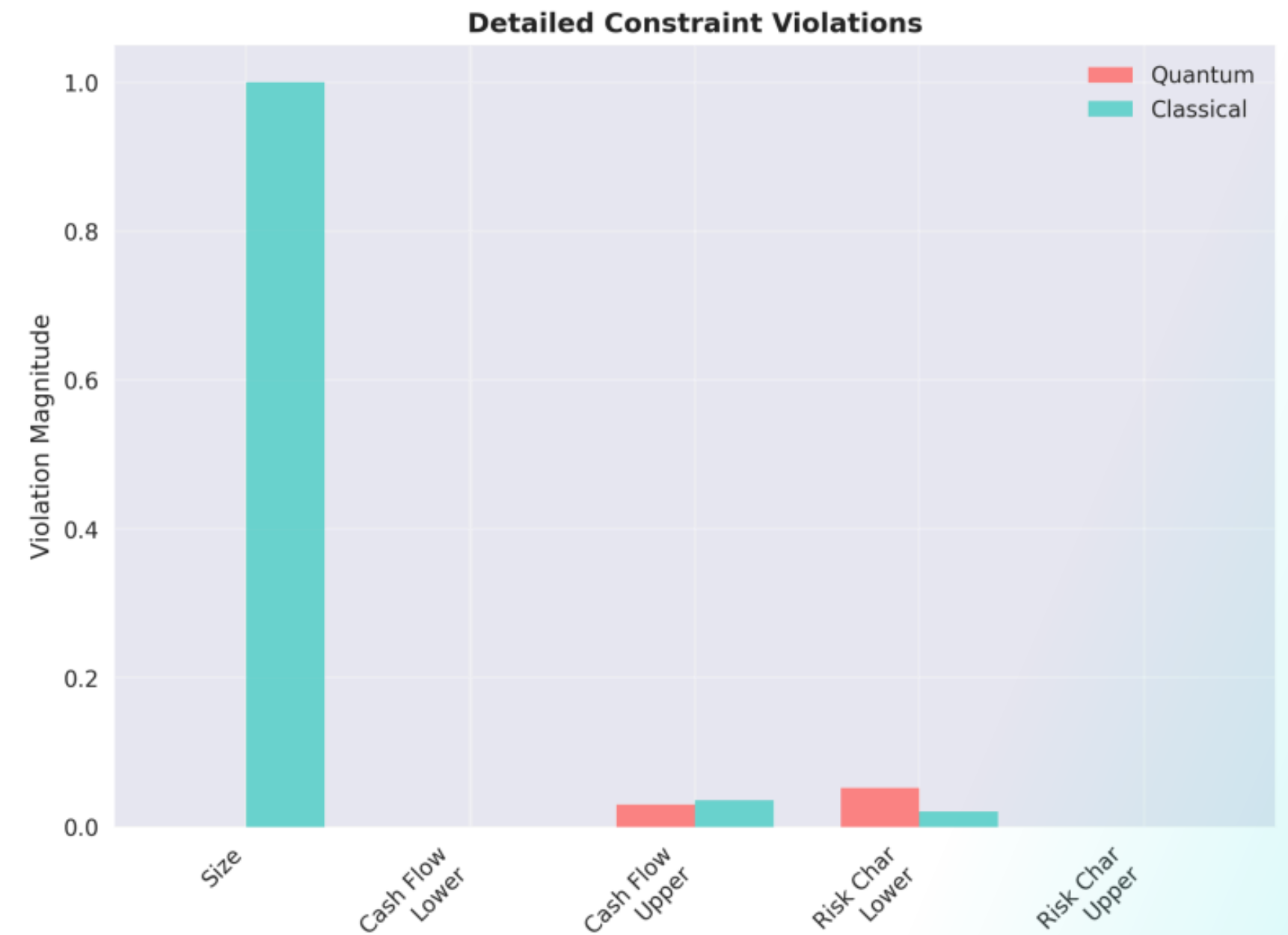
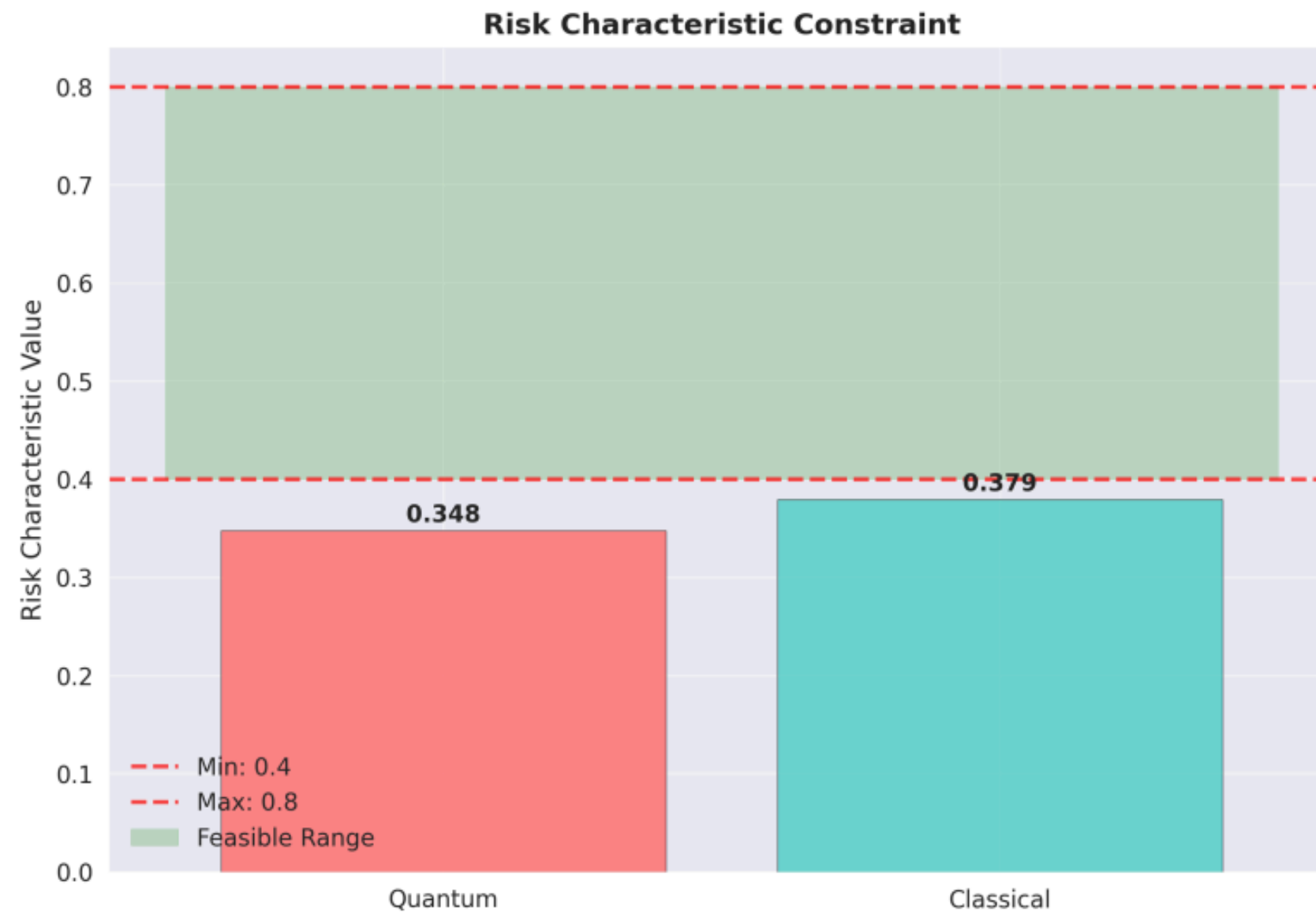


## Quick comparison: visualizations of constraint violation (pt 1 of 2)





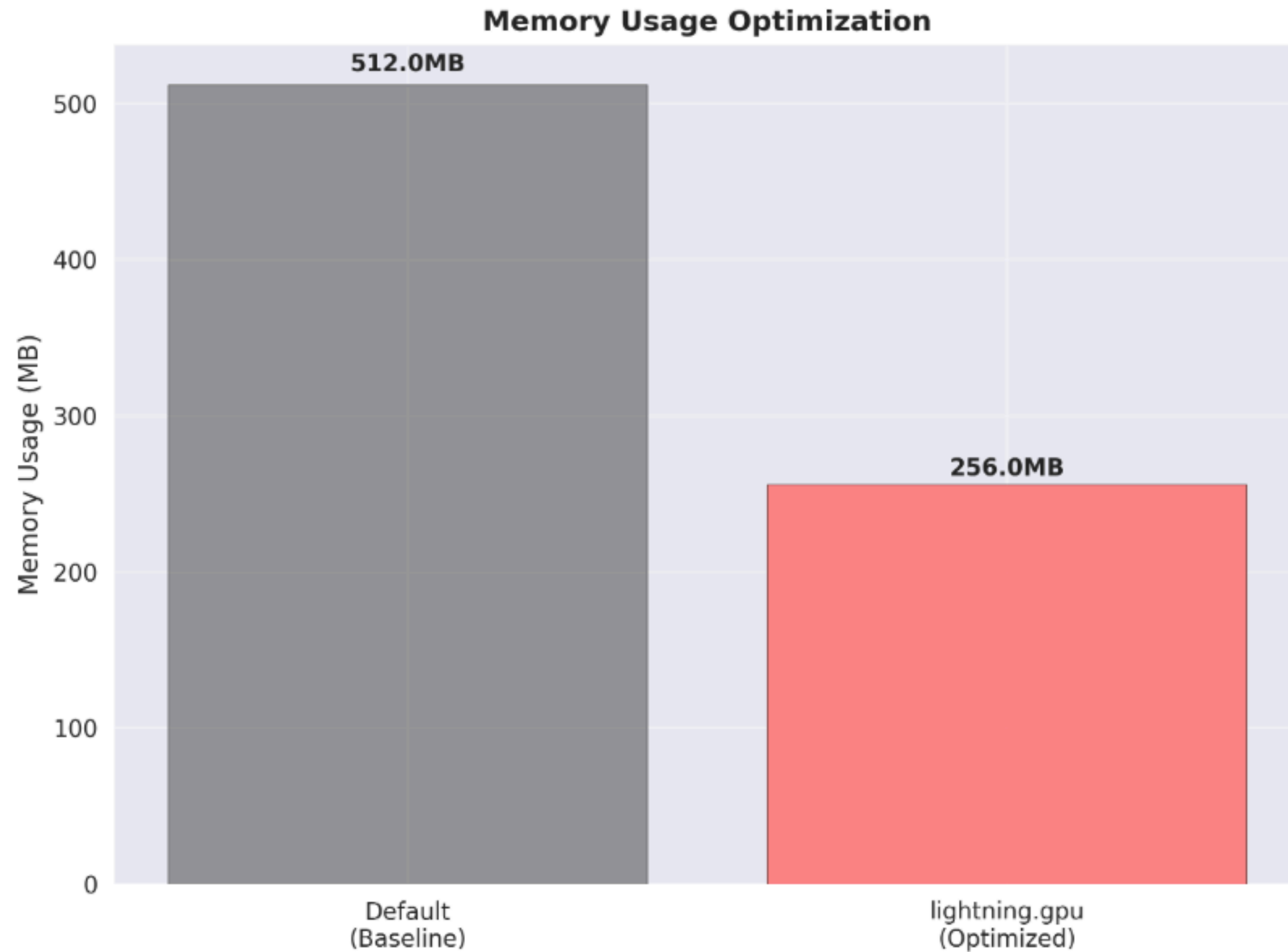
## Quick comparison: visualizations of constraint violation (pt 2 of 2)







## Using Lightning.gpu to speedup computation





## Scalability Analysis

### Current Implementation:

- 16 qubits (20-24 qubits for lightning.GPU): Substantial classical simulation requirements
- 65,536+ possible solutions: Classical enumeration becomes challenging
- Complexity: Exponential classical search space vs polynomial VQE optimization

### Scaling Potential:

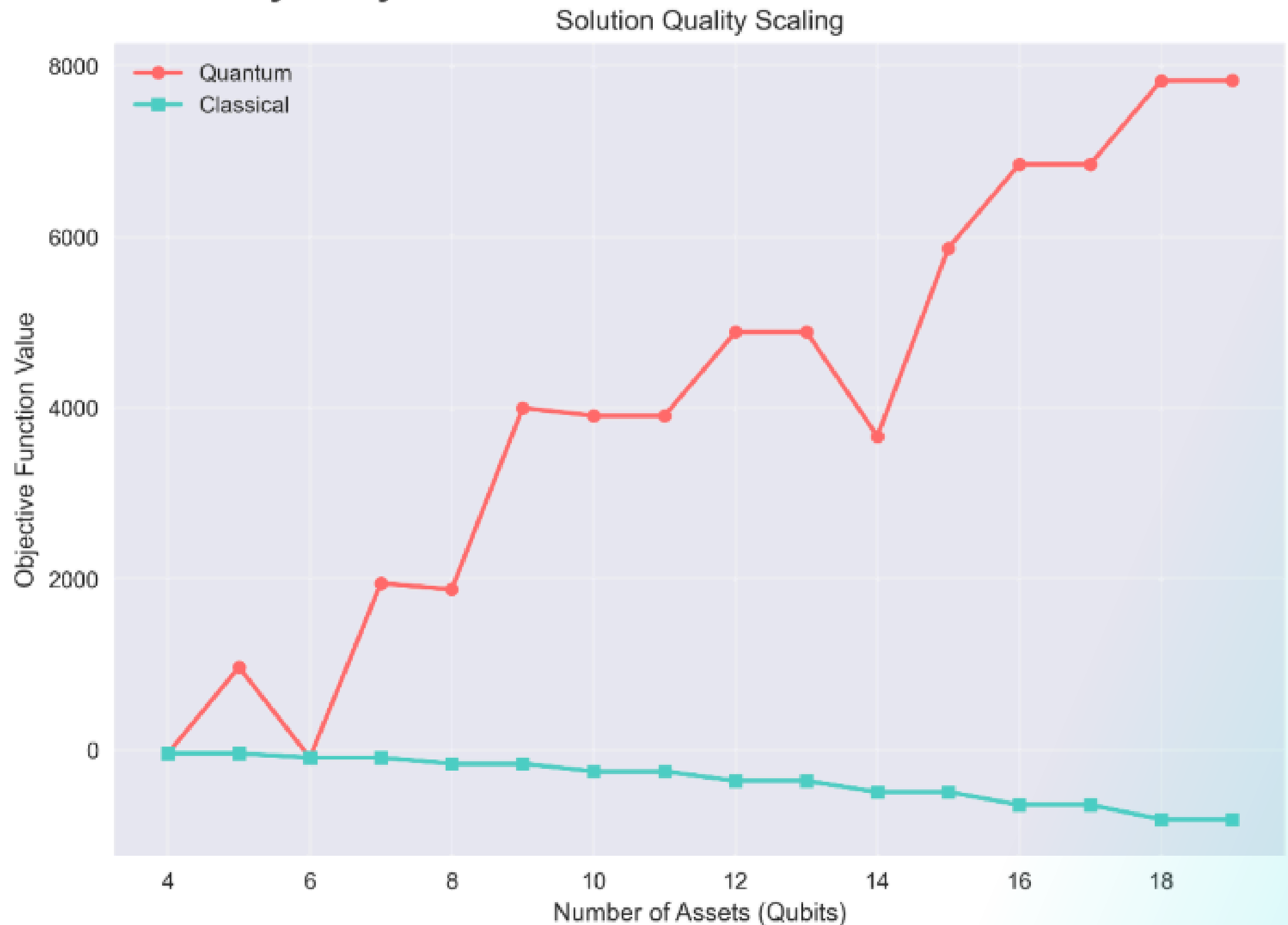
- Real-world portfolios: 50-500+ assets (using vanguard data now!)
- Quantum advantage: Expected at 24+ qubits where classical optimization becomes intractable
- Hybrid approaches: Quantum sampling with classical post-processing proves effective (You can see more of this in the part 3 py file)



## What can we do to further explore the code?

1) Run with greater memory so we can run more qubits (~32). As you can see, there's only so much potential we can capture with a limited number of qubits.

2) Look further into adaptive circuits





Problem

Methodology

Results

Next Steps

# Thank you for your time!