We're Not Shor

Tom Yao, Tudor Rosca, Aradhya Jain, Sebastian DeCosta, Eashan Iyer

February 2026

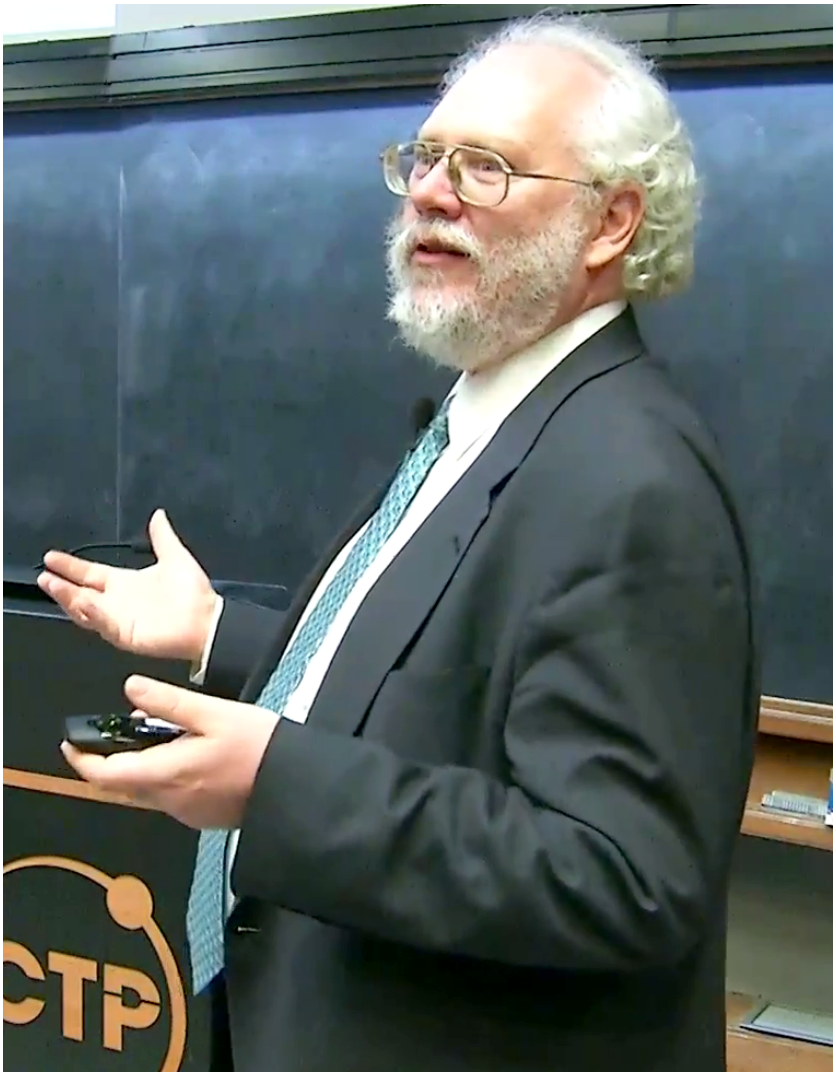

Figure 1: Peter Shor's genuine reaction to our paper

1 The VaR Problem

VaR or value at risk is a quantity that measures the maximum expected loss over a given time period at a certain confidence level. For instance, if your VaR for a one day time frame at a 95% confidence interval is 1,000,000 dollars, then you would lose more than 1,000,000 dollars in a single day due to

random variation only 5% of the time. Given a normal Gaussian distribution

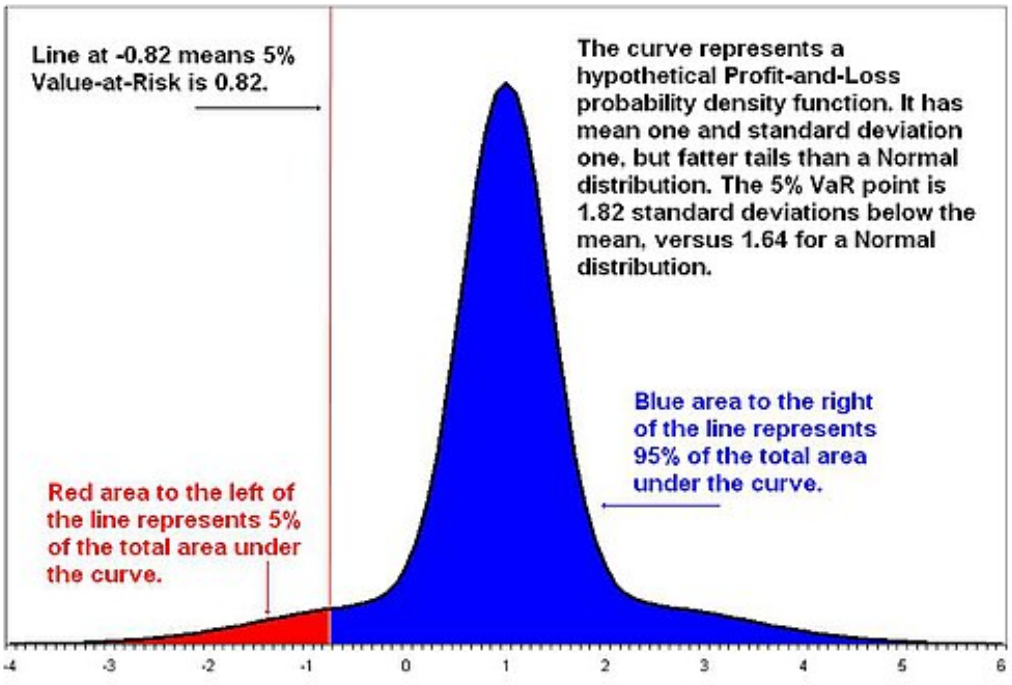

Figure 2: The VaR here is denoted by the vertical red line and is a percentage of the total portfolio value

function G with variations within a day and confidence level c, we can solve for VaR with the following: Assuming daily returns are normally distributed,

$$g(x) = N(\mu, \sigma),$$

the Value at Risk (VaR) at confidence level c is computed using the lower-tail quartile of the distribution.

$$\int_{-\infty}^{z} g(t)\, dt = G(z) = (1 - c)$$

$$VaR = portfolio\_value \cdot (\mu - \sigma \cdot z)$$

Where the distribution function measures the percent change in the portfolio over a single day. And $G(z)$ is the cdf function.

Essentially, we find when the cdf equals 1-c and get the z-score at that point. We then use that z-score calculate the percent change which is equal to $(\mu - \sigma * z)$. Then we multiply the percent change by the portfolio value to get the net loss in portfolio value.
In our code, we accomplished this with built in functions from packages

```
model = GaussianModel(mu=args.mu, sigma=args.sigma)
confidence = args.confidence
portfolio_value = args.portfolio
theoretical_var_gaussian(model, confidence, portfolio_value)
```

For more complicated distributions, the theoretical value is much harder to obtain (the integral/cdf) becomes much harder to solve analytically. We then have two approximation methods that will give us a quick and dirty solution, saving time and resources and improving the flexibility of our calculation: Classical Monte Carlo and Iterative Quantum Amplitude Estimation.

2 Classical Solution (Monte Carlo)

We first generate a random number r from the Gaussian distribution.

$$r \in \mathcal{N}(\mu, \sigma)$$

And then, we check if that number falls into the 5% tail portion which would indicate a "breach" of the VaR. Finally, we take a ratio of all breached sample points versus the total number of sample points to get an approximation for the VaR. This is summarized in the code below.

```
def monte_carlo_var(
    model: GaussianModel,
    confidence: float,
    n_samples: int,
    portfolio_value: float = 1.0,
    rng: np.random.Generator | None = None,
) -> float:
    """Monte Carlo estimate of VaR using Gaussian sampling."""

    if rng is None:
```

```
    rng = np.random.default_rng()
returns = model.sample_returns(rng, n_samples)
losses = -portfolio_value * returns
return float(np.quantile(losses, confidence))
```

Running the simulation with $\mu = 0.0005, \sigma = 0.02, c = 0.95$, we received the following graphs.

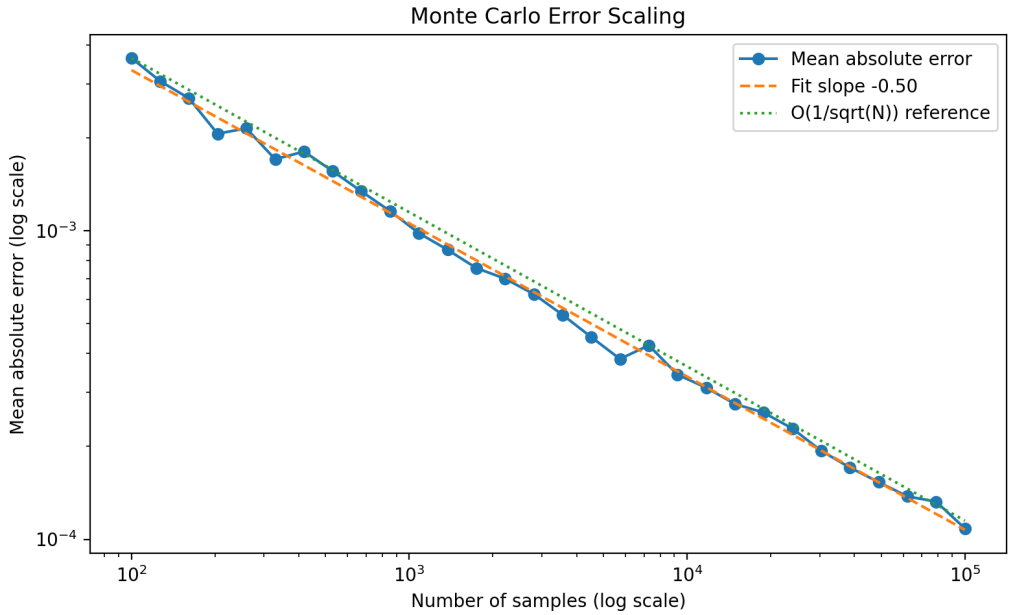

Figure 3: log(Error) = -0.50 log(N)

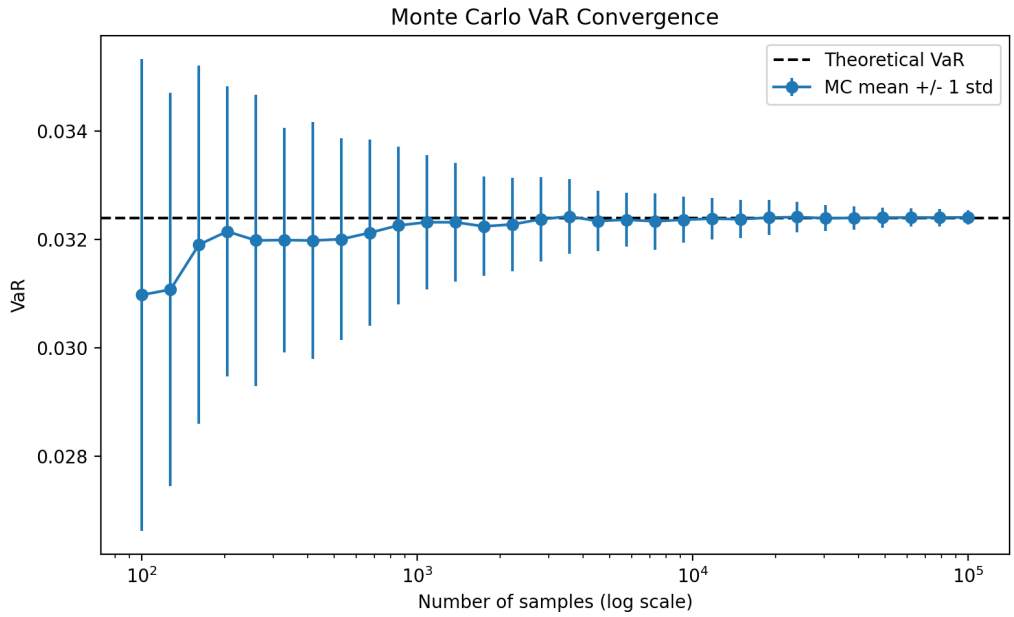

Figure 4: Converges to theoretical for $N \approx 5000$

The Jump-Diffusion Simulation Model

We wanted to consider a more complex model of financial markets to factor in the concept of "jump days" to see if it could be an even better model

The total return R for a given time step is simulated by stacking two independent distributions: a **Normal distribution** for routine noise and a **Poisson-triggered Normal distribution** for shocks (sudden drops and gains).

1. Diffusion Component (Continuous Noise)

The routine market movement follows a Gaussian distribution:

$$r_{diff} \sim N(\mu, \sigma^2)$$

2. Jump Component (Discrete Shocks)

The number of jumps N in a single period follows a Poisson distribution:

$$N \sim \text{Poisson}(\lambda)$$

If $N > 0$, the size of the jumps is the sum of N independent random values selected from a normal distribution:

$$r_{jump} = \sum_{i=1}^{N} J_i, \quad \text{where } J_i \sim N(\mu_j, \sigma_j^2) \tag{1}$$

3. Total Simulated Return

The final return used in the Monte Carlo simulation is the sum of these two processes:

$$R_{total} = r_{diff} + r_{jump}$$

This "stacked" approach allows the model to capture the **Fat Tails** (kurtosis) and **Skewness** observed in real market data, which standard Gaussian models often ignore.

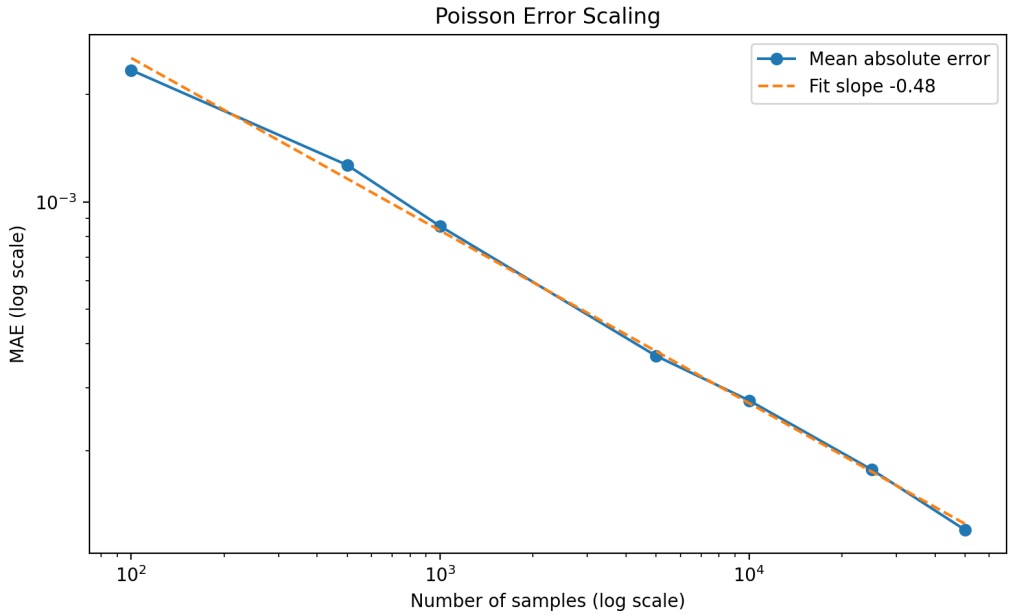

Figure 5: Jump-Diffusion Error Scaling

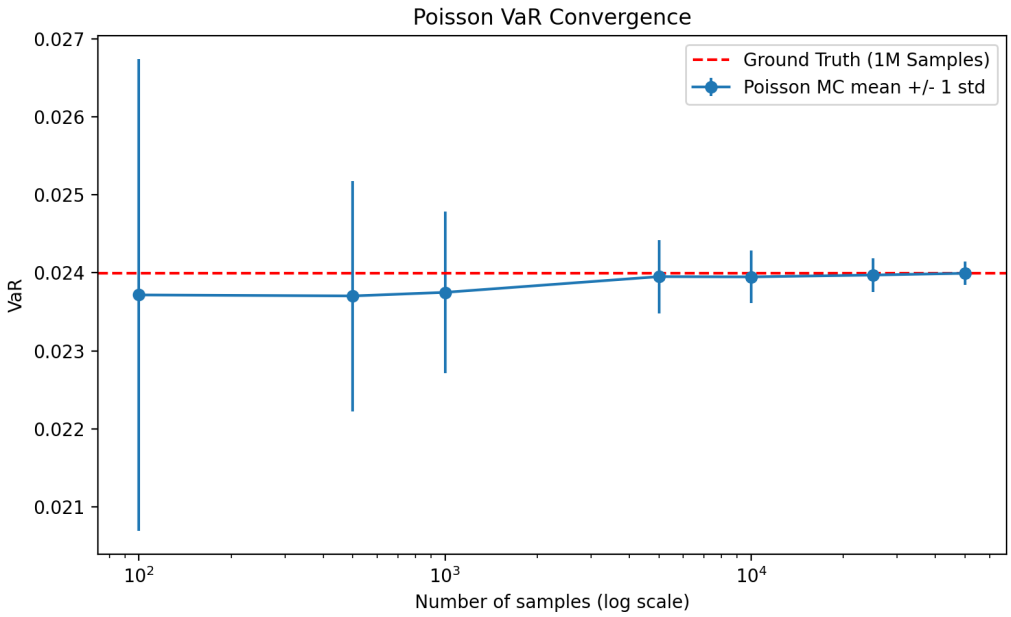

Figure 6: Jump-Diffusion Method Convergence

2.1 Comparing the Models

2.1.1 Data

We tested the accuracy of the Poisson model and the Gaussian model by initializing the models based off the data from 2024 and making a prediction for 2025 which we then checked against the true market behavior in 2025.

Based off 252 trading days in a year and a confidence level of 95%, there should be $252 \cdot 0.05 = 12.6$ days when the loss is more than the VaR. We can compare this to how many days the stock underperformed the calculated VaR from the two methods and calculate an accuracy.

$$accuracy = 1 - \frac{|num\_breaches - 12.6|}{12.6} \tag{2}$$

Table 1: Backtesting Results: Poisson vs. Gaussian VaR (95% Confidence)

Ticker	Method	VaR Prediction	Breaches	Accuracy
SPY	Poisson	1.21%	19	48.6%
	Gaussian	1.25%	19	48.6%
TSLA	Poisson	5.58%	13	96.4%
	Gaussian	6.17%	9	71.7%
NVDA	Poisson	4.91%	10	79.7%
	Gaussian	5.61%	8	63.7%
COIN	Poisson	7.54%	9	71.7%
	Gaussian	8.40%	3	23.9%
MDB	Poisson	4.68%	15	80.5%
	Gaussian	6.16%	7	55.8%
RIVN	Poisson	7.00%	5	39.8%
	Gaussian	8.25%	0	0.0%

2.1.2 Discussion

SPY, S&P 500:

The two methods had the same accuracy because the S&P is not very volatile which the Poisson method is better at modeling.

Other Stocks

The other stocks shown are more volaltile than the S&P and are therefore modeled more accurately by the Poisson method which accounts for market volatility. For example Coinbase's (COIN) price acts as a high-beta leveraged proxy for the cryptocurrency market, making it hyper-sensitive to "jump" events like sudden regulatory shifts, Bitcoin price shocks, and the evolving sentiment of the crypto-fear-and-greed index. Rivian (RIVN) is another good example. As a pre-profit EV manufacturer, its valuation is driven by binary production milestones—such as the critical 2026 R2 launch—and extreme sensitivity to macro

headwinds like interest rate changes and the repeal of federal EV tax credits.

3 Quantum Solution: Iterative Amplitude Estimation (IQAE)

The algorithm used to simulate the Quantum Monte Carlo method is known as **Quantum Amplitude Estimation (QAE)**. This method leverages quantum interference to estimate the value of an unknown parameter a with a significant speedup over classical sampling.

3.1 The Premise of QAE

Consider a quantum state that is initialized in the following form:

$$|\Psi\rangle = \sqrt{1-a}|\psi_0\rangle|0\rangle + \sqrt{a}|\psi_1\rangle|1\rangle$$

In our specific Value at Risk (VaR) implementation, the state preparer constructs a final state of the form:

$$|\Psi_{final}\rangle = \sum_{i>i^*} \sqrt{p_i}|i\rangle|0\rangle + \sum_{i\leq i^*} \sqrt{p_i}|i\rangle|1\rangle$$

QAE utilizes a Grover-like operator to estimate the value of a, which represents the total probability mass in the "loss" territory.

3.2 Quantum Probability Encoding

The reduction of the VaR problem into a quantum circuit follows a two-step process:

1. **State Preparation:** A quantum circuit $\mathcal{A}$ is constructed to prepare a superposition whose amplitudes correspond to the discretized probability distribution (e.g., a discretized Gaussian or Jump-Diffusion). The probability of finding $X = x$ is encoded in the qubit state with the binary value $|x\rangle$.

2. **Comparator Circuit:** A separate circuit checks whether the sampled value $|x\rangle$ lies below the candidate threshold x^*. If $x \leq x^*$, a dedicated "flag" qubit is flipped to $|1\rangle$; otherwise, it remains $|0\rangle$.

After these steps, the probability of measuring the flag qubit in state $|1\rangle$ is exactly the cumulative probability we want to determine. The problem is thus reduced to estimating a single unknown probability a.

3.3 Why IQAE is Used Instead of Simple Sampling

If one were to simply measure the flag qubit N times, the approach would be equivalent to **Classical Monte Carlo sampling**, where the estimation error ϵ shrinks at a rate of $1/\sqrt{N}$.

Iterative Quantum Amplitude Estimation (IQAE) improves upon this via coherent amplitude amplification. It defines a Grover-like operator $\mathcal{Q}$ that acts as a rotation in a two-dimensional subspace spanned by the "success" states (flag = 1) and "failure" states (flag = 0). Applying $\mathcal{Q}$ multiple times amplifies the sensitivity of the quantum state to the underlying probability, allowing for much finer information extraction per execution.

3.4 Mathematical Mechanism of IQAE

Rather than estimating a directly, IQAE estimates an angle θ such that:

$$a = \sin^2(\theta)$$

After applying the Grover operator $\mathcal{Q}$ exactly k times, the probability of measuring the flag qubit as $|1\rangle$ becomes:

$$p_k = \sin^2((2k+1)\theta)$$

By measuring how this probability oscillates across several values of k, the algorithm gathers information about θ. Larger values of k correspond to larger rotations and therefore greater sensitivity, yielding an error scaling of $O(1/\epsilon)$, a quadratic speedup over the classical $O(1/\epsilon^2)$.

3.5 The "Iterative" Refinement

Unlike standard QAE, IQAE does not require Quantum Phase Estimation or the Quantum Fourier Transform, making it more suitable for near-term quantum hardware.

- **Interval Maintenance:** The algorithm maintains a classical confidence interval that is guaranteed to contain the true value of θ.
- **Recursive Shrinking:** For a chosen value of k, measurement statistics restrict $(2k+1)\theta$ to a specific range. This new range is intersected with the current interval to shrink the possible values of θ.
- **Ambiguity Avoidance:** By carefully selecting k to avoid the periodicity of the sine function, the interval for θ shrinks steadily until the desired precision is reached.

3.6 Implementation Details (Classiq IQAE)

We discretize the Gaussian distribution into 2^n grid points spanning $\mu \pm 3\sigma$ and normalize the probabilities. These probabilities are loaded into the amplitude distribution using Classiq's `inplace_prepare_state` on a quantum register. A comparator circuit (payoff) flips a single indicator qubit when the asset index is below a threshold i^*, so the probability of measuring $|1\rangle$ on the indicator equals the tail probability.

IQAE is run with a fixed number of shots through an `ExecutionPreferences` object, which allows us to study shot-noise effects separately from the ϵ-driven query complexity. For the Gaussian experiments we compute a classical CDF and find the VaR index i^* via classical bisection; IQAE then estimates the tail probability at that index. The estimated probability $\hat{\alpha}$ is mapped back to a VaR estimate using the analytic inverse CDF, and error is reported against the analytic Gaussian VaR.

We perform a grid search over ϵ and the IQAE confidence parameter to tune hyperparameters, then reuse the best pair for the shots-scaling experiment. We also run a separate epsilon-scaling experiment with a fixed shot budget to highlight the expected $O(1/\epsilon)$ query scaling. The plots below are placed immediately after the relevant discussion.

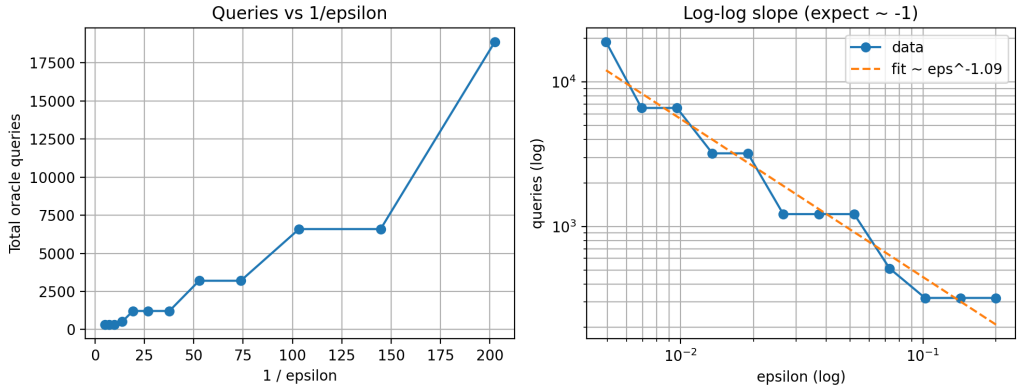

Figure 7: IQAE scaling: total oracle queries vs $1/\epsilon$ (and log-log slope).

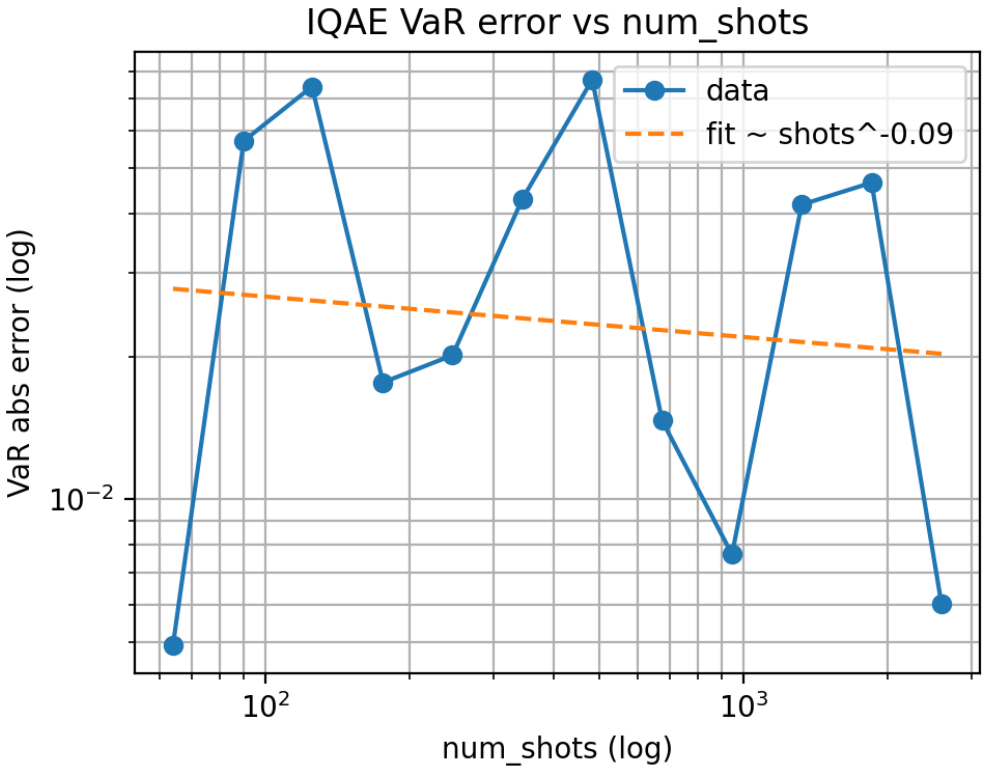

Figure 8: IQAE shot-noise scaling: VaR error vs number of shots (log-log).

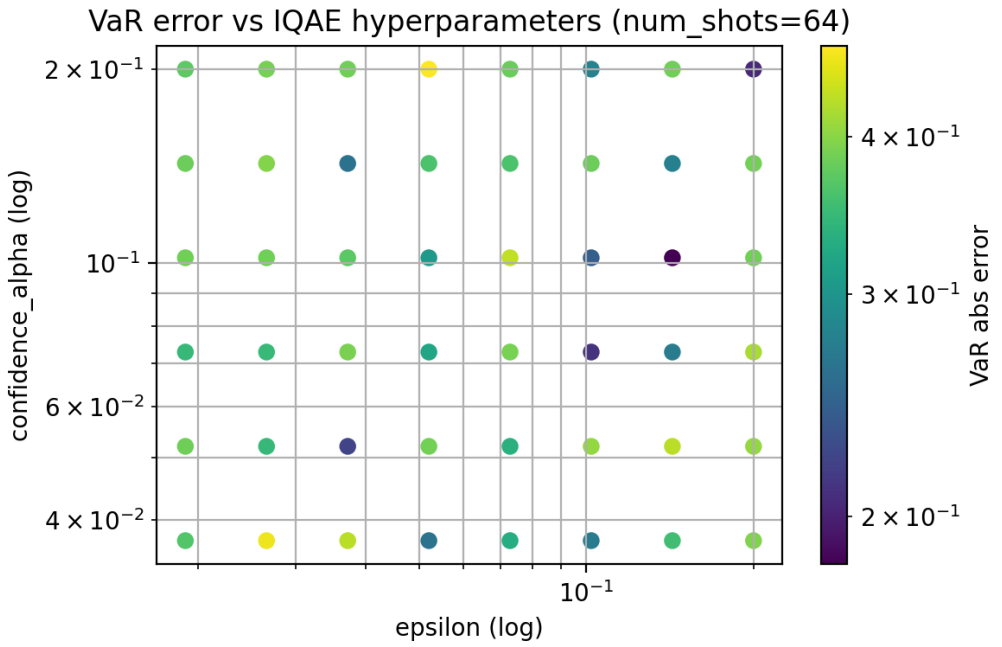

Figure 9: Hyperparameter tuning: grid search over ϵ and confidence parameter.

4 Comparison

4.1 Scaling Contrast: Monte Carlo vs IQAE

Classical Monte Carlo (MC) estimation error scales as $O(1/\sqrt{N})$, so achieving error ϵ typically requires $N = O(1/\epsilon^2)$ samples. Iterative Quantum Amplitude Estimation (IQAE) reduces the *query* complexity to $O(1/\epsilon)$, yielding a quadratic asymptotic advantage in the number of oracle calls. Importantly, this does *not* remove shot noise: when we estimate a probability from repeated measurements, the error still falls like $O(1/\sqrt{\text{shots}})$ for both MC sampling and IQAE runs. The speedup therefore appears in the number of *queries* (Grover iterations / oracle uses)

required to reach a target ϵ, not in the scaling of error with the number of shots. This is why Figures 7 and 8 are separated: the former isolates query scaling, while the latter isolates shot noise.

4.2 Asymptotic Advantage vs Practical Overheads

IQAE's advantage is asymptotic: it appears as $\epsilon \to 0$ when counting oracle queries. In practice (especially on a simulator), the gap is narrowed by:

- **Per-query overhead:** each IQAE query involves deeper circuits, controlled Grover iterations, and more complex state preparation than a single classical sample.
- **Finite shots:** for fixed shot budgets, statistical noise can dominate at small ϵ and mask the quadratic advantage in query complexity.
- **Bisection cost:** VaR estimation requires multiple probability estimates at different thresholds, so total runtime includes the search loop as well as IQAE.

Thus, IQAE's scaling is a statement about *oracle complexity*, not necessarily wall-clock speedup in near-term settings. Our results reflect this: the slopes match theory, but the constant factors and simulator overheads compress the practical gap.

4.3 Modeling Error vs Estimation Error

Both classical and quantum pipelines inherit **modeling/discretization error** from the choice of distribution (Gaussian vs jump-diffusion),

grid resolution, and truncation range. IQAE improves **estimation error** per query, but does not fix model error. In our backtesting experiments, shifts in VaR driven by the jump-diffusion model were often larger than estimator noise, emphasizing that accurate modeling can dominate the overall error budget. A fair comparison therefore separates:

- **Model error:** distributional mismatch + discretization.
- **Estimation error:** sampling / amplitude estimation noise.

4.4 Comparison Plots

Figure 10 summarizes the core scaling difference on log-log axes: MC error drops as $N^{-1/2}$ while IQAE error drops as N^{-1} with respect to total probability queries. Figure 11 reframes the same result as queries versus $1/\epsilon$, showing the quadratic gap between MC and IQAE. Finally, Figure 12 separates shot noise from query complexity: the left panel shows both methods scale as shots$^{-1/2}$, while the right panel shows the IQAE query scaling with $1/\epsilon$.

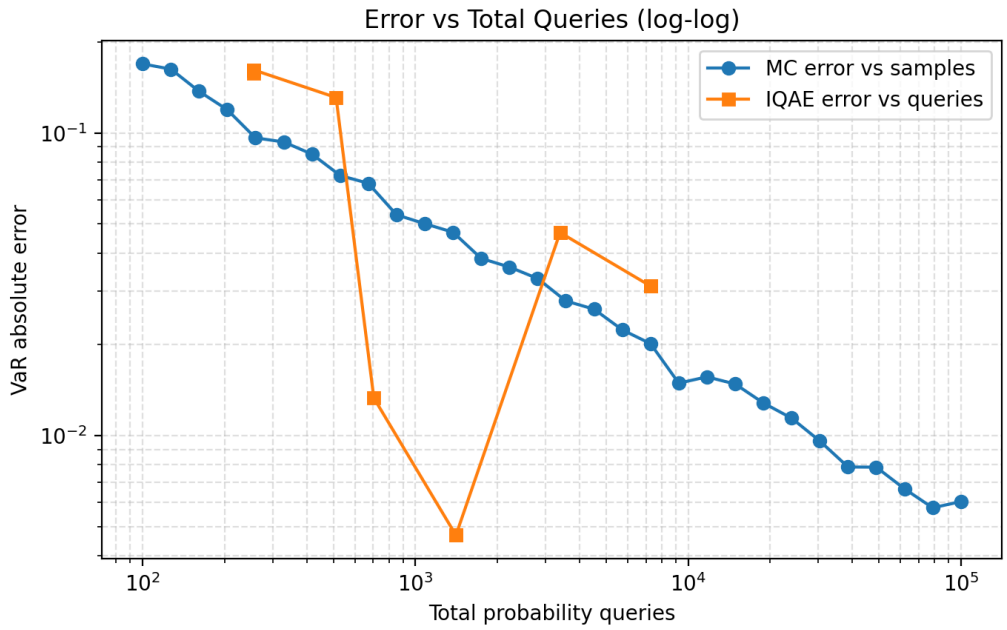

Figure 10: Error vs total queries (MC vs IQAE) on log-log axes.

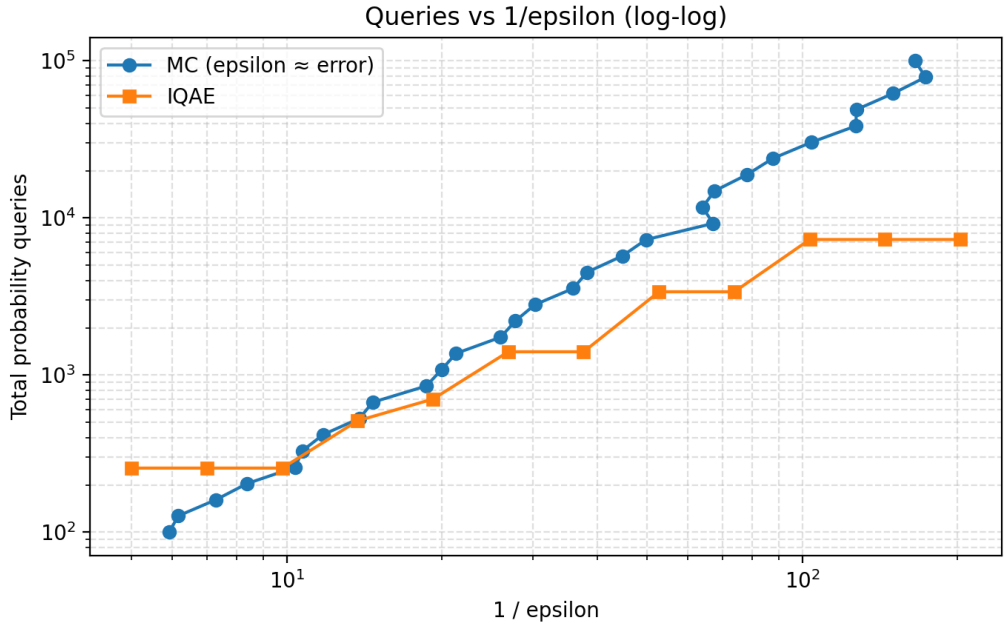

Figure 11: Query scaling vs $1/\epsilon$ (MC vs IQAE).

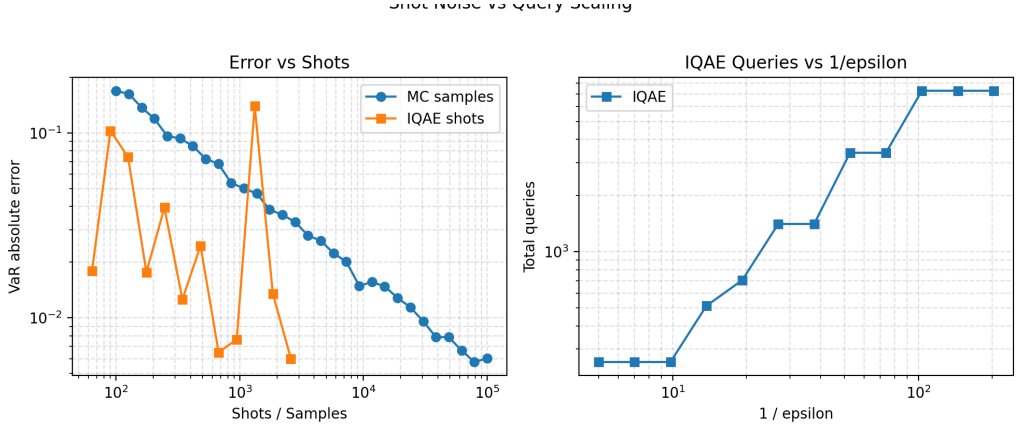

Figure 12: Shot-noise scaling vs query-complexity scaling.

5 Conclusion

Because frankly, we're not Shor.