Vanguard Report

Ran Xue, Zhuo Cao and Zhongyi Jiang

August 2025

1 Introduction

The portfolio optimization task can be formulated as a quadratic constrained binary optimization problem. The decision variable y_c represents whether the corresponding bond c is included in the portfolio, with $y_c = 1$ indicating inclusion and $y_c = 0$ indicating absence. Therefore, given n bonds, a portfolio can be represented as a binary string $y_1y_2...y_n$. The goal of portfolio optimization is to find the optimal choice of $y_1y_2...y_n$ within given constraints so that the objective function is minimized.

The objective function takes a quadratic form:

$$ob(y_1, ..., y_n) = \Sigma_{i\leq j} q_{ij} y_i y_j \tag{1}$$

Note that we have omitted linear terms because all linear terms can be converted to quadratic form given that $y^2 = y, \forall y \in \{0, 1\}$

Using the vector representation $\mathbf{y} = (y_1, ..., y_n)^T$, the objective function can be formulated as:

$$ob(\mathbf{y}) = \mathbf{y}^T Q \mathbf{y} \tag{2}$$

with

$$(Q)_{ij} = \begin{cases} q_{ij} & \text{if } i \leq j \\ 0 & \text{if } i > j \end{cases} \tag{3}$$

Constraints such as maximum number of bonds and residual cash flow, etc. can be formulated in a linear form:

$$b_{lower} < \Sigma_i a_i y_i < b_{upper} \tag{4}$$

Without loss of generality, we convert all constraints into the single-side form:

$$\Sigma_i a_{ki} y_i > b_k \tag{5}$$

Suppose there are m constrains, they can then be expressed as:

$$\Sigma_i a_{ki} y_i > b_k \text{ , for } k = 1, ..., m \tag{6}$$

In vector form:

$$\min(A\mathbf{y} - \mathbf{b}) > 0 \tag{7}$$

where min is defined as the smallest element of the vector.

With the definitions above, the portfolio problem is thus formulated as:

$$\begin{aligned} &\min ob(\mathbf{y}) \\ &\quad \text{with } ob(\mathbf{y}) = \mathbf{y}^T Q \mathbf{y} \\ &\qquad \min(A\mathbf{y} - \mathbf{b}) > 0 \end{aligned} \tag{8}$$

2 Task formulation

To solve the problem in Eq. (8) using quantum computing, we first convert it into an unconstrained problem, which can be more conveniently handled on quantum computers. A common method of converting constrained optimization to unconstrained optimization is to encode constraints as penalty terms in the objective function. Mathematically, one can choose a penalty constant $p > 0$, and reformulate Eq. (8) as an unconstrained optimization task:

$$\begin{aligned} &\min ob(\mathbf{y}) \\ &\quad \text{with } ob(\mathbf{y}) = \mathbf{y}^T Q \mathbf{y} + p(\min\{0, \min(A\mathbf{y} - \mathbf{b})\})^2 \end{aligned} \tag{9}$$

We then translate Eq. (9) to one that can be solved on a quantum computer via VQE-like and sample-based algorithms. This can be done by replacing the variable $y_i \in \{0,1\}$ by $z_i \in \{-1,1\}$ using mapping : $y_i = \frac{1-z_i}{2}$. Under this mapping, each bitstring $z_1 z_2 ... z_n$ measured from the quantum computer corresponds to a portfolio configuration $y_1 y_2 ... y_n$. Assuming there are in total n bonds, one can then use n qubits to generate samples of bitstrings $z_1 z_2 ... z_n$ and choose from them the optimal solution of Eq. (9).

3 Our quantum algorithm

We use a sample-based quantum algorithm to solve Eq. (9). Our algorithm consists of two sampling steps. A generative model is used to generate quantum circuit samples. Bitstrings are then sampled from those quantum circuits. More specifically, a GPT model is used for generating quantum circuits. We use the same implementation as in ref. [1] and the Pennylane demo [2] for the GPT model.

It has been proposed as a more scalable method than the conventional VQE approach to search for the ground state. Instead of iteratively optimizing the variational parameters by the classical optimizer, the quantum circuit is generated by the pre-trained generative model i.e. the GPT-QE. Our algorithm is different from GPT-QE mainly in two aspects: gate pool and loss function.

3.1 QAOA gate pool

To target the optimization task, we have chosen to include QAOA-like gate layers in our gate pool. QAOA method is a common technique to search for the optimal combination of bitstrings. In a gate-based quantum circuit, it is equivalent to apply a time evolution to qubits under the Hamiltonian, which is decomposed into the following form,

$$U(\boldsymbol{\gamma},\ \boldsymbol{\beta}) \ = \ e^{-i\beta_n H_M} e^{-i\gamma_n H_C} \ ... \ e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C}, \tag{10}$$

where the n refers to the repetition of QAOA layer, and the parameters β and γ are variational parameters for the mixer and cost layer, respectively. The cost Hamiltonian H_C takes an Ising form that is given by,

$$H_C = \sum_{i<j} q_{ij} z_i z_j + \sum_{i=j} q_{ii} z_i \tag{11}$$

where q_{ij} are identical to the coefficients q_{ij} in the unconstrained objective function Eq. 1. Note that the diagonal terms q_{ii} appear to be coefficients of linear terms given that $y_i^2 = y_i$, $\forall y \in \{0,1\}$.

In general, the mixer Hamiltonian H_M takes the form of Pauli-X operator given by, $\sum_i X_i$, where the summation runs over all qubits. More specifically, the mixer Hamiltonian is composed of Pauli X terms with random coefficients in our implementation,

$$H_M^{(k)} = \Sigma_i c_i^{(k)} X_i \tag{12}$$

The QAOA gate pool, therefore, is defined as:

$$\{\ e^{-i\beta_k H_M^{(k)}}, e^{-i\gamma_k H_C}\}_{k=1}^{L} \tag{13}$$

The generated quantum circuit samples has the same Hadmard layer $H^{\otimes n}$as initialization. The initial state is then $H^{\otimes n}|0\rangle^{\otimes n} = |+\rangle^{\otimes n}$. After initialization, gates are generated from the GPT model and applied to the initial state. In the end, bitstrings are sampled from the generated quantum state.

3.2 Loss functions

We have implemented CVaR loss function for GPT offline training. Suppose K samples are measured from the quantum circuits, the $\alpha-$tail CVaR loss is calculated as the average of the lowest αK samples:

$$\text{CVaR}_\alpha(E) = \frac{1}{\alpha K} \Sigma_{i=1}^{\alpha K} E_i \tag{14}$$

This CVaR function is used to evaluate loss during GPT training.

The algorithm structure is shown in FIG. 1.The QAOA training dataset consists of 1024 QAOA circuits built from the gate pool and their corresponding resulting states (i.e. probability distribution over bitstrings $x_1...x_{20}$).

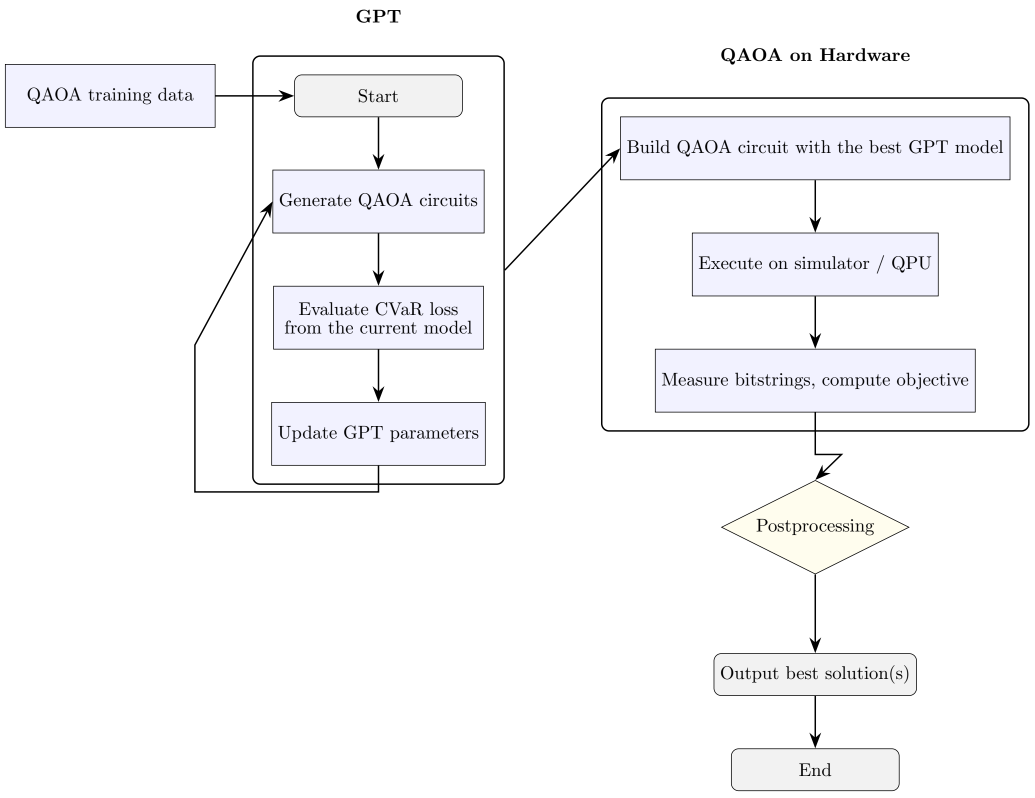
GPT
QAOA training data
Start
Generate QAOA circuits
Evaluate CVaR loss from the current model
Update GPT parameters
QAOA on Hardware
Build QAOA circuit with the best GPT model
Execute on simulator / QPU
Measure bitstrings, compute objective
Postprocessing
Output best solution(s)
End

Figure 1: Flowchart for GPT-assisted QAOA

.

4 Experiments and results

4.1 Training

According to the provided flowchart in Fig. 1, we train the GPT model as follows: In total, 1024 QAOA circuits are sampled from the gate pool which contains 100 layers of QAOA module (i.e. one mixer layer and one cost layer). We randomly sample QAOA the mixer and cost layers from the gate pool. The preliminary data has a sequence length of 4, thus a training size of (1024, 4) with 100 shots per circuit to get the probability distribution of bitstrings. We set 10 as the maximum iteration of the training process (see Fig. 2).

We evaluate the cost every 5 epochs as shown in Fig. 3. The cost is calculated by the Eq. 9. Here the true ground (see the dashed line) represents the baseline of this minimization task ($E_g =$ 58508.9930). The minimum cost of evaluated iteration (i.e. the lower bound of ribbon plots) is presented as well, providing the lowest cost of sampled bitstrings. The estimated cost by the QAOA circuits approaching ground state within 10 epochs. In addition, the minimum cost of all evaluation iterations hit the ground state cost. It means the optimal combintation is successfully sampled by the generated quantum circuit, providing the feasibility of the GPT-QAOA model for this binary combinational task.

4.2 Sampling

The sample-based GPT-QAOA algorithm is implemented with Pennylane and PyTorch. We present the results using a truncated dataset of 20 bonds in total, which are encoded into 20 qubits. When offline training is completed, the GPT-QAOA model is used to generate samples. Afterwards, a Hamming-distance post-processing is done. For each sampled result $x_1...x_{20}$, we first estimate its

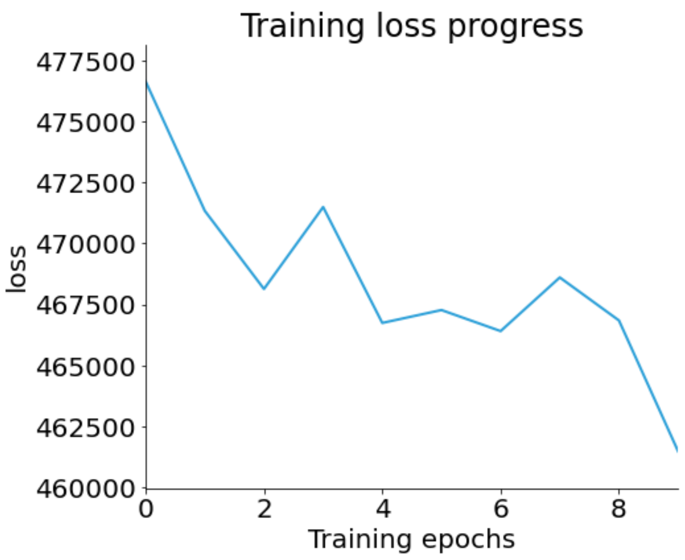

Figure 2: The loss evaluation during training process.

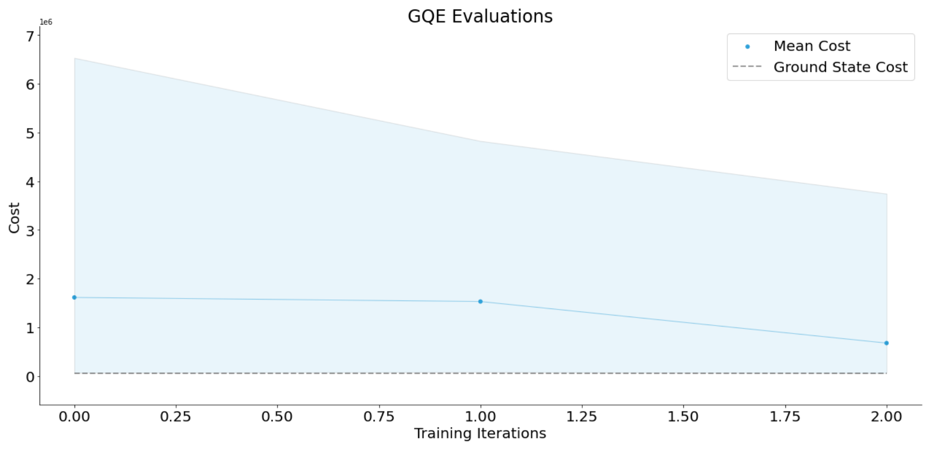

Figure 3: The evaluated cost for every 5 training epochs.

probability by the counts. We then look for the bitstring that gives the lowest objective function within Hamming distance 1 (i.e. flipping one bit) from the current bitstring $x_1...x_{20}$. This gives the probability of finding the optimal bitstring after a local search of flipping one bit. In Fig. 4, we plot both the raw probability distribution from the best model and the probability after a distance-1 local search.

As shown in Fig. 4, even without post-processing, the probability distribution is biased toward the global minimum (left side of the x-axis). Following a distance-1 local search, the success probability is further enhanced. To more clearly demonstrate this improvement, we plot the cumulative distribution function(CDF) after performing a distance-1 local search in Fig. 5. The post-processing is done by searching for the minimum objective function within Hamming distance 1 from each sampled bitstring.

4.3 benchmark with other methods

We compare our results to the classical approaches provided by Scipy.optimize.minimize. Given the objective function (see Eq. 9), we use it as the objective function for the classical optimization for a fair comparison.

Many methods are available for this combinational optimization task, however variables in our case are binary. We limit the boundary of variables to be $[0, 1]$. Outcomes of all available methods are listed in the Tab. 1, where we mainly focus on the number of iterations (N_{itr}) and its hamming distance (d_H) to the reference value (E_g) for benchmark.

Using the similar criteria as for the quantum approach, we evaluate the hamming distance to

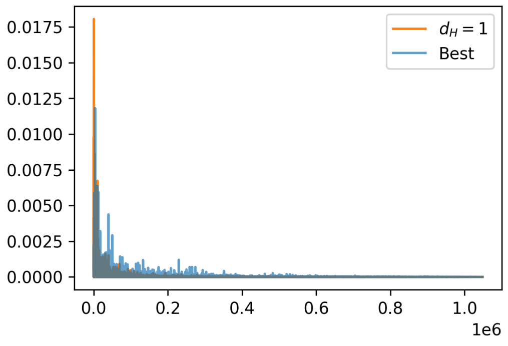

Figure 4: Hamming PDF result

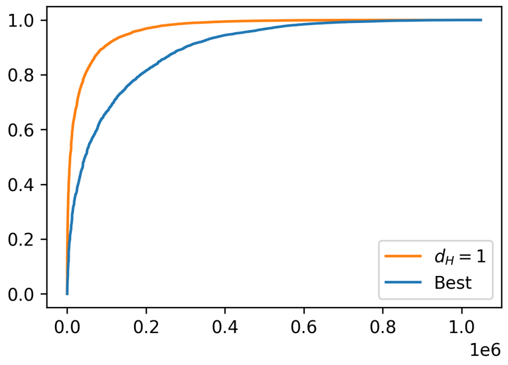

Figure 5: Hamming CDF results

the reference value for each classical method as well. Except for the L-BFGS-B method, TNC and Powell methods find the optimal solution successfully (i.e. $ob(y) = E_g$) with N_{itr} that is significantly higher than the GPT-QAOA approach. The trust-constr method did not converge until exceeding the maximum iterations and the SLSQP did not able to find any combinations with d_H as good as our quantum approach.

Method	**Success**	**ob(y)**	N_{itr}	d_H
L-BFGS-B	True	58508.9930	42	0
TNC	True	58508.9930	1890	0
Powell	True	58546.6448	802	0
trust-constr	False	58540.2213	21000	1
SLSQP	True	1722864.1585	21	8

Table 1: Benchmark results for constrained minimization using different optimization methods

References

1. Nakaji, K. *et al.* The generative quantum eigensolver (GQE) and its application for ground state search. arXiv:2401.09253 [quant-ph]. `http://arxiv.org/abs/2401.09253` (2024) (cit. on p. 2).
2. Bunao, J. & Niu, Z. Generative quantum eigensolver training using PennyLane data. en. *PennyLane Demos.* `https://pennylane.ai/qml/demos/gqe_training` (2024) (cit. on p. 2).