Graph-Conditioned Meta-Optimizer for QAOA Parameter Generation on Multiple Problem Classes

Kien X. Nguyen*, Ilya Safro*†
*Department of Computer and Information Sciences, University of Delaware, Newark, DE, USA
†Department of Physics and Astronomy, University of Delaware, Newark, DE, USA
{kxnguyen, isafro}@udel.edu

***Abstract*—We study parameter transferability for the Quantum Approximate Optimization Algorithm (QAOA) across multiple combinatorial optimization problem classes from a parameter generation perspective. Specifically, a meta-optimizer is trained on one problem class and deployed on another during test time. Prior work employs a Long Short-Term Memory network to emulate QAOA optimization trajectories, but the learned dynamics usually collapse to near-identical paths, limiting cross-problem transfer efficiency. In this paper, we present a *problem-aware graph-conditioned* meta-optimizer for QAOA that learns to generate parameter trajectories over a fixed horizon, providing strong initializations with only a few steps. The optimizer is conditioned on compact graph embeddings and trained end-to-end using differentiable feedback from the QAOA objective, avoiding the need for ground-truth angles. We evaluate across multiple graph problem classes, including MaxCut, Maximum Independent Set, Maximum Clique, and Minimum Vertex Cover. We report both solution quality and feasibility-aware metrics where constraints apply. Results across a comprehensive empirical study consisting of 64 settings show that the learned optimizer can reduce optimization effort and improve performance over standard initialization, while exhibiting transferable behavior across graph families and problem types.**
Reproducibility: Our code is available at: https://github.com/Nyquixt/Cross-Problem-QAOA-ParamGen.

***Index Terms*—Quantum Approximate Optimization Algorithm, Parameter Generation, Parameter Transferability**

I. Introduction

Can a variational quantum optimizer trained on yesterday's problems produce high-quality parameters for tomorrow's, possibly unexpected, optimization task? In realistic deployments, the eventual problem class may not be known when the optimizer is trained, or a new formulation may arise under time constraints that make full quantum–classical re-optimization impractical. This creates a pressing need for variational parameter-generation methods that can transfer across problem classes and adapt to new instances in only a few steps. Let us focus on the quantum approximate optimization algorithm (QAOA) [1]–[3] that is one of the leading approaches for combinatorial optimization problems, enabling a wide range of applications, such as finance [4], biology [5], and scientific computing [6].

QAOA is a variational hybrid quantum–classical algorithm with a direct application to solving combinatorial optimization problems. It employs a parameterized quantum circuit whose angles are updated iteratively by a classical optimizer. While the framework has advanced rapidly, efficiently tuning such parameters remains challenging. As the circuit depth or the number of qubits grows, the optimization landscape can become exponentially flat, a phenomenon referred to as barren plateaus [7]–[10]. Moreover, scaling QAOA to larger instances typically requires deeper circuits, which further complicates optimization and amplifies the impact of hardware noise on near-term devices [11].

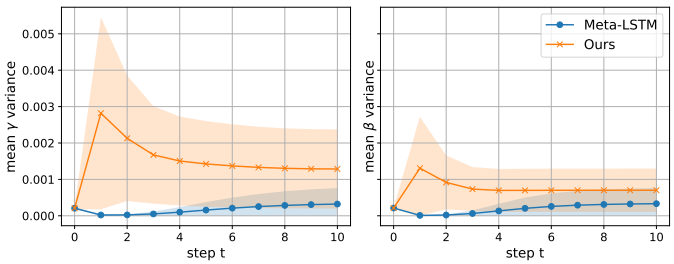

Fig. 1. Diversity of generated parameter trajectories on QAOA circuits for MaxCut with depth $p = 10$. At each rollout step t, we compute the mean squared deviation of the generated angles from the instance-mean trajectory, averaged over angle dimensions $\{\gamma_\ell\}_{\ell=1}^{p}$ (left) and $\{\beta_\ell\}_{\ell=1}^{p}$ (right). Curves show the mean across test instances; shaded bands indicate ± 1 standard deviation across instances. Meta-LSTM exhibits lower diversity (near-identical paths) than our graph-conditioned meta-optimizer.

These challenges have motivated a range of parameter initialization strategies, including linear-ramp schedules [11], [12], multistart optimization [13], GPT-based [14], and parameter transfer [15]–[19]. A complementary approach is *parameter generation*, in which a neural meta-optimizer is trained to propose good initial parameters [20]–[24]. Verdon *et al.* [20] proposed a lightweight Long Short-Term Memory (LSTM) network [25] as a meta-optimizer that produces a parameter trajectory over a fixed horizon, mimicking the iterative updates of a classical optimization routine. Huang *et al.* [23] subsequently refined the training objective and extended the framework to the variational quantum eigensolver for quantum chemistry applications. However, our empirical results indicate that the meta-optimizer often produces *near-identical parameter trajectories*, effectively collapsing to an average "best" update path across the training instances. This lack of expressivity limits the optimizer's ability to adapt and consequently hinders generalization to more complex instances. In this paper, we argue that incorporating instance-dependent conditioning is necessary to steer the optimizer

toward more expressive trajectories and improve robustness across diverse instances (Figure 1). Specifically, we demonstrate that conditioning the meta-optimizer on graph embeddings improves generalization to larger problems.

Furthermore, in many settings, closely related combinatorial formulations differ only in the objective or the addition of constraints. Re-optimizing QAOA parameters for each formulation can be expensive, motivating approaches that reuse learned optimization behavior across problem classes. Cross-problem transfer provides a mechanism to amortize parameter search by leveraging shared structure between objectives [15], [19]. While parameter transferability has been studied primarily within the same problem class, cross-problem transfer in QAOA remains relatively underexplored. Nguyen *et al.* [19] show that cross-problem transfer based solely on structural similarity, e.g. using nearest neighbors on Graph2Vec features [26], can be suboptimal. To this end, a natural question arises: *how can we learn graph embeddings whose features transfer across problem classes?* This motivates incorporating problem-specific information directly into the representation by encoding the objective and constraints, producing richer and more transferable features.

Accordingly, we adopt the UniHetCO framework by Nguyen and Safro [27], which maps multiple problem classes into a unified quadratic programming (QP) formulation. UniHetCO augments the original graph by adding objective-coupling relations, as well as introducing constraint nodes and variable–constraint edges (Figure 3). In other words, UniHetCO not only produces structure-aware but also *problem-aware* embeddings, which accelerate cross-problem transfer. It then learns graph embeddings by training a solver to minimize a loss that combines the objective value with constraint-violation penalties. In our pipeline, we first pre-train the graph embeddings and then condition them on the training of the meta-optimizer for QAOA (Figure 2).

Finally, we comprehensively evaluate our graph-conditioned meta-optimizer on 16 single-problem and 48 cross-problem transfer settings. Experiments span four combinatorial optimization problems, Maximum Cut (MaxCut), Maximum Independent Set (MIS), Maximum Clique (MaxClique), and Minimum Vertex Cover (MVC), and four QAOA circuit depths. Moreover, we demonstrate the expressivity of the approach by visualizing the variance of generated parameter trajectories over 100 random graph instances.

In summary, our contributions are threefold:

1) We condition a learned QAOA meta-optimizer on graph embeddings with structure-aware (Graph2Vec) and problem-aware encodings (UniHetCO).
2) We show that graph conditioning improves generalization to larger instances and enables few-shot transfer across problem classes.
3) We quantify the increased expressivity of the conditioned meta-optimizer by visualizing the variance of generated parameter trajectories over many test instances and demonstrate the effectiveness of the proposed approach.

II. Related Work

A. Learning to Learn Variational Quantum Algorithms

Learning to learn (L2L), or meta-learning, has been a long-standing research field in the machine learning community [28]–[32] and has recently been applied to quantum neural networks and quantum variational algorithms [20]–[24]. In the context of variational quantum algorithms (VQAs), L2L methods typically train a classical model to propose circuit parameters that yield low energy with only a few refinement steps. Rather than relying on hand-designed update rules, the meta-optimizer is trained end-to-end using feedback from the quantum objective, enabling amortized optimization and improved sample efficiency. This general framework has been explored in several settings, including learned parameter initialization, learned update rules, and the transfer of optimization behavior across related instances.

B. Cross-Problem Parameter Transferability

This emerging research direction studies how QAOA parameters optimized for one problem class can be reused to accelerate optimization on another. A common strategy is to leverage parameters learned on a more QAOA-native objective (e.g., MaxCut) and transfer them to a more challenging objective (e.g., constrained MIS), with the goal of reducing optimization effort while maintaining solution quality. Montañez *et al.* [15] present a numerical study of transferring QAOA parameters between different combinatorial optimization problems. While their results suggest the promise of cross-problem transfer, their empirical analysis is limited to a small set of randomly generated instances. To this end, Nguyen *et al.* [19] have adopted a machine learning-based approach to systematically identify good donor candidates on MaxCut and transfer their parameters to the corresponding acceptors on MIS.

III. Background

A. Meta-Optimizer for QAOA Parameter Generation

The "learning to learn" (L2L) framework involves training a neural network to act as a meta-optimizer for QAOA. Concretely, an LSTM [25] is employed to generate a short sequence of QAOA parameters that serve as strong initializations after only a few steps, following prior work on learned QAOA optimizers [20], [23]. At each rollout step, the LSTM maintains an internal memory and receives feedback about the current solution quality. The input at step t consists of (i) the energy value achieved by the previous QAOA parameters and (ii) the previous parameters themselves. Using this information, the LSTM proposes an updated set of QAOA angles for the next step. Repeating this procedure for a fixed horizon produces a parameter trajectory that mimics an optimization process, but is generated by the learned model rather than by a hand-crafted optimizer. The set of parameters generated at the last unrolling step is then used to evaluate the QAOA circuit expectation value.

B. Graph Embedding

Graph embedding refers to a broad class of techniques that map vertices, edges, or entire graphs to low-dimensional vector representations. These embeddings capture key structural and semantic information from the graphs, making them suitable for downstream machine learning tasks. Existing approaches can be divided into node-level embedding [33]–[35] and graph-level embedding [26], [36]–[38]. Node-level embedding methods assign a vector to each vertex while preserving local structure, such as neighborhood relationships and attribute similarity, making them well suited for node-centric tasks like classification, link prediction, and community detection. Graph-level embedding techniques produce a single fixed-size vector for an entire graph that summarizes global organization and semantics, which is more appropriate for graph-level objectives such as classification and regression.

IV. Graph-Conditioned Meta-Optimizer

We extend the L2L framework by training a neural meta-optimizer that produces QAOA parameters conditioned on the problem instance. Given a graph instance, we compute a d-dimensional embedding $\mathbf{g} = \Phi(G) \in \mathbb{R}^d$, where Φ is the graph encoder, and use it to condition an optimizer network $f_{\boldsymbol{\varphi}}$, parameterized by the weights $\boldsymbol{\varphi}$. The optimizer generates a short parameter trajectory $\{\boldsymbol{\theta}_t\}_{t=1}^T$, where $\boldsymbol{\theta}_t = (\boldsymbol{\gamma}_t, \boldsymbol{\beta}_t) \in \mathbb{R}^{2p}$ denotes the QAOA angles, with a depth p (Figure 2).

In particular, we model trajectory generation as a recurrent optimization procedure. Let $E(\boldsymbol{\theta}; G) = \langle \psi(\boldsymbol{\theta}) | H_C(G) | \psi(\boldsymbol{\theta}) \rangle$ be the QAOA energy of graph G. At step t, the meta-optimizer takes in the QAOA parameters and energy from the previous step $t-1$, and outputs the next parameters:

$$(\mathbf{h}_t, \mathbf{s}_t) = f_{\boldsymbol{\varphi}}(\mathbf{z}_t, \mathbf{h}_{t-1}, \mathbf{s}_{t-1}), \quad \boldsymbol{\theta}_t = \mathbf{W}\mathbf{h}_t, \tag{1}$$

where $\mathbf{z}_t = [E(\boldsymbol{\theta}_{t-1}; G), \boldsymbol{\theta}_{t-1}]$ denotes the input at step t, $[\cdot, \cdot]$ denotes concatenation, $\mathbf{h}_t$ and $\mathbf{s}_t$ are the hidden and cell states, respectively. The recurrent module is implemented as an LSTM network [25], which is designed to mitigate vanishing gradients and improve stability when training recurrent models via backpropagation through time. We briefly summarize the LSTM update at step t. Given input $\mathbf{z}_t$, previous hidden state $\mathbf{h}_{t-1}$, and cell state $\mathbf{s}_{t-1}$, the LSTM computes

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ii}\mathbf{z}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}), \tag{2}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{if}\mathbf{z}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1}), \tag{3}$$

$$\tilde{\mathbf{s}}_t = \tanh(\mathbf{W}_{is}\mathbf{z}_t + \mathbf{W}_{hs}\mathbf{h}_{t-1}), \tag{4}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{io}\mathbf{z}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1}), \tag{5}$$

where $\sigma(\cdot)$ denotes the sigmoid function. We omit the bias terms for simplicity of notations. The cell and hidden states are then updated as

$$\mathbf{s}_t = \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{s}}_t, \tag{6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{s}_t), \tag{7}$$

with $\odot$ denoting element-wise multiplication (or Hadamard product). The gating vectors $(\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t)$ regulate information flow through the cell state, enabling more stable credit assignment over long horizons.

The optimizer $f_{\boldsymbol{\varphi}}$ learns to minimize a loss function $\mathcal{L}(f_{\boldsymbol{\varphi}})$ derived from the energy trajectory $E_1 \ldots, E_T$. We adopt the loss function from Huang *et al.* [23] which is a decay weighted sum over the trajectory of the normalized energies:

$$\mathcal{L}(f_\varphi) = \sum_t \omega_t \bar{E}_t, \tag{8}$$

where $\{\omega_t\}_{t=1}^T$ assigns larger weights to later rollout steps to emphasize the quality of the final parameter proposals. As different instances may induce different magnitudes of the QAOA energy, we normalize it by the ℓ_1 norm of the Pauli coefficients to stabilize the gradient updates. Given a cost Hamiltonian with Pauli decomposition

$$H_C = \sum_j \boldsymbol{\alpha}_j P_j, \tag{9}$$

we define the Pauli coefficient norm as

$$\|H_C\|_1 \triangleq \|\boldsymbol{\alpha}\|_1 = \sum_j |\boldsymbol{\alpha}_j|. \tag{10}$$

We then use the normalized energy

$$\bar{E}(\boldsymbol{\theta}; G) \triangleq \frac{E(\boldsymbol{\theta}; G)}{\|H_C(G)\|_1}, \tag{11}$$

which is bounded in $[-1, 1]$ since each Pauli string P_j has eigenvalues ± 1. We update the meta-optimizer parameters $\boldsymbol{\varphi}$ using gradient descent:

$$\boldsymbol{\varphi}' \leftarrow \boldsymbol{\varphi} - \eta \nabla_{\boldsymbol{\varphi}} \mathcal{L}, \tag{12}$$

where η is the learning rate. To compute $\nabla_{\boldsymbol{\varphi}} \mathcal{L}$, we differentiate through the unrolled meta-optimization trajectory. At each rollout step t, the loss depends on the QAOA energy

$$E_t = \langle H_C \rangle_{\boldsymbol{\theta}_t} = \langle \psi(\boldsymbol{\theta}_t) | H_C | \psi(\boldsymbol{\theta}_t) \rangle . \tag{13}$$

The gradient of the energy with respect to a circuit parameter $\boldsymbol{\theta}_{t,k}$ is computed using adjoint differentiation method by Jones and Gacon [39]. Concretely, given the QAOA energy $E_t(\boldsymbol{\theta}_t) = \langle H_C \rangle_{\boldsymbol{\theta}_t}$, the simulator evaluates the gradient $\nabla_{\boldsymbol{\theta}_t} E_t$ via an adjoint-backpropagation procedure, which backpropagates through the circuit using an adjoint state and returns $\partial E_t / \partial \boldsymbol{\theta}_{t,k}$ without requiring parameter shifts [40], [41]. Gradients are then propagated to the meta-optimizer parameters via the chain rule:

$$\nabla_{\boldsymbol{\varphi}} \mathcal{L} = \sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial E_t} \cdot \frac{\partial E_t}{\partial \boldsymbol{\theta}_t} \cdot \frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\varphi}}, \tag{14}$$

where $\frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\varphi}}$ is obtained by backpropagating through the unrolled LSTM that generates $\{\boldsymbol{\theta}_t\}_{t=1}^T$. In mini-batch training, we sample a batch $\mathcal{B}$ of instances and replace $\mathcal{L}$ with the average batch loss,

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{G \in \mathcal{B}} \mathcal{L}(G), \tag{15}$$

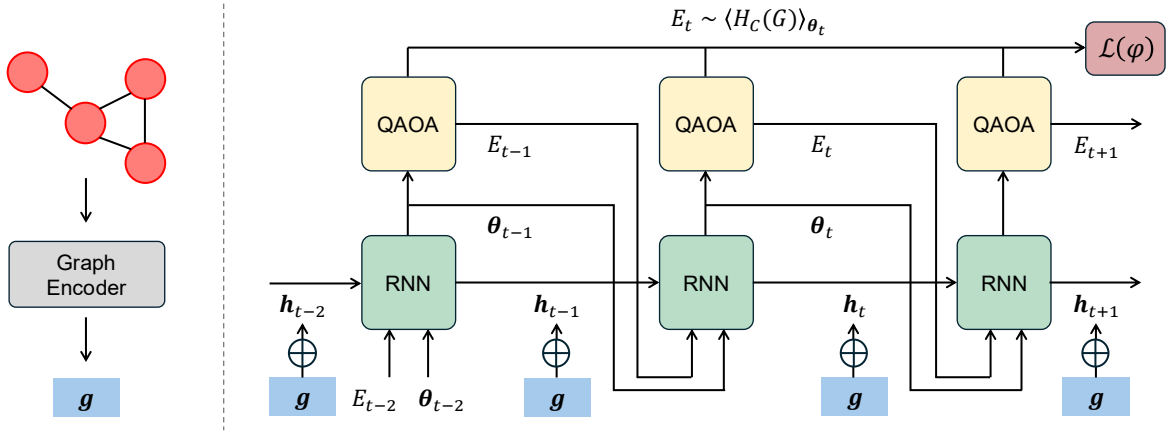

Fig. 2. Overview of the graph-conditioned meta-optimizer training pipeline. **(Left)** The graph instance is first projected to a fixed-dimension vector $\mathbf{g}$ by a graph encoder. **(Right)** A meta-optimizer, realized by a recurrent neural network (RNN), is trained to produce a trajectory of QAOA variational parameters. At each step t, the graph embedding $\mathbf{g}$ is injected to the RNN hidden state $\mathbf{h}_t$, effectively conditioning parameter generation on instance-specific graph features.

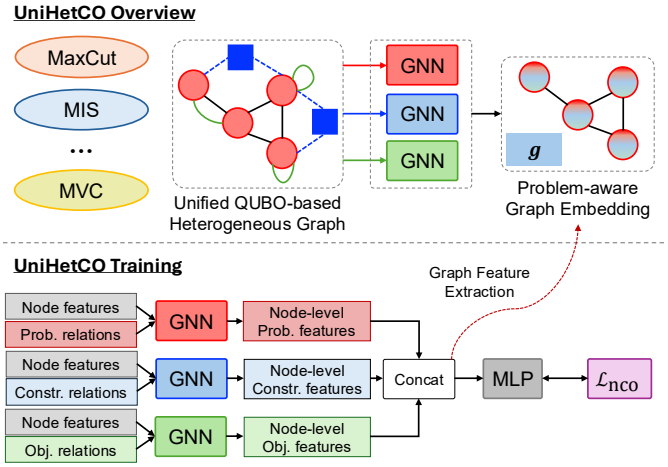

Fig. 3. Overview of the UniHetCO graph encoder. UniHetCO not only encodes the problem instance structure (**red**), but also its objective (**green**) and constraints (**blue**), into a unified graph representation. This property makes feature transfer across different problem classes possible.

so that each update uses $\nabla_{\varphi}\mathcal{L}_{\mathcal{B}}$.

Graph conditioning at each step. To ensure that the instance information remains available throughout the unrolled horizon, we inject the graph embedding at every step by augmenting the recurrent hidden states h_t,

$$\tilde{\mathbf{h}}_t = \mathbf{h}_t + \mathbf{g}, \quad \boldsymbol{\theta}_t = \mathbf{W}\tilde{\mathbf{h}}_t. \tag{16}$$

V. Problem-aware Graph Embedding

Building a graph embedding framework with strong feature transfer is the key to enabling *generalization across larger problem sizes and different problem classes*. To this end, we employ UniHetCO to construct a problem-aware graph representation grounded in the general Quadratic Programming (QP) formulation and learn transferable embeddings via unsupervised neural combinatorial optimization (NCO) [27]. Unsupervised NCO trains a neural solver without labeled optimal solutions by directly minimizing an instance objective on the model's predictions. We refer the reader to Nguyen and Safro [27] for further details on the UniHetCO framework.

Unlike existing graph embedding methods, such as Graph2Vec [26], which focus solely on the structure of the graph instance, UniHetCO builds a representation that casts various node subset-selection problems into a unified learning objective. Given an instance with N decision variables, we first write it in a common QP template with quadratic and linear objective terms and linear constraints:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\mathbf{x}^\top\mathbf{Q}\mathbf{x} + \mathbf{c}^\top\mathbf{x} \quad \text{s.t.}\mathbf{A}\mathbf{x} \le \mathbf{b}, \tag{17}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of decision variables, $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a coefficient matrix, $\mathbf{c} \in \mathbb{R}^n$ denotes linear coefficients, and $\mathbf{A}\mathbf{x} \le \mathbf{b}$ are optional linear inequality constraints with $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. By absorbing $\mathbf{c}$ into the diagonal of $\mathbf{Q}$ and converting the linear constraints into a penalty term, we obtain an equivalent Quadratic Unconstrained Binary Optimization (QUBO) problem of the form:

$$\min_{\mathbf{x}\in\{0,1\}^n} \mathbf{x}^\top\tilde{\mathbf{Q}}\,\mathbf{x} + \lambda\cdot\text{Penalty}(\mathbf{x}), \quad \tilde{\mathbf{Q}} = \mathbf{Q} + \text{diag}(\mathbf{c}). \tag{18}$$

UniHetCO then encodes the resulting objective and constraints *directly into the input graph* so that a single neural model can be trained across problem classes with one unified, label-free objective.

A. Unified Heterogeneous Graph Representation

For each instance, UniHetCO constructs a heterogeneous graph with (i) decision variable nodes $\mathcal{V}_{\text{var}}$, (ii) constraint nodes $\mathcal{V}_{\text{constr}}$, and three relation types:

a) Problem graph: $G_{\text{prob}} = (\mathcal{V}_{\text{var}}, \mathcal{E}_{\text{prob}})$ contains edges capturing the original instance structure.

b) Objective graph: $G_{\text{obj}} = (\mathcal{V}_{\text{var}}, \mathcal{E}_{\text{obj}}, w_{\text{obj}})$ captures the objective-coupling relations, where $\mathcal{E}_{\text{obj}} = \mathcal{E}_{\text{off}} \cup \mathcal{E}_{\text{diag}}$. Off-diagonal edges represent the quadratic terms $\mathcal{E}_{\text{off}} = \{(u,v) : u < v, \mathbf{Q}_{uv} \neq 0\}, w_{uv} = \mathbf{Q}_{uv}$, and self-loops representing

the linear terms in $\mathbf{c}$ that are absorbed into the diagonal of $\mathbf{Q}$, $\mathcal{E}_{\text{diag}} = \{(u,u) : u \in \mathcal{V}_{\text{var}}, \mathbf{Q}_{uu} + \mathbf{c}_u \neq 0\}, w_{uu} = \mathbf{Q}_{uu} + \mathbf{c}_{uu}$.

c) Constraint hypergraph: encodes the variable-constraint relations, where the variables appearing in a constraint form a hyperedge. UniHetCO first standardizes all linear inequality constraints to the "less-than-or-equal" form by multiplying any inequality of the form $\alpha^\top x \geq \beta$ by -1. It then constructs the constraint graph via a bipartite star expansion, connecting $\mathcal{V}_{\text{var}}$ to $\mathcal{V}_{\text{constr}}$, with each constraint node storing its right-hand-side values. This yields $G_{\text{constr}} = (\mathcal{V}_{\text{var}}, \mathcal{V}_{\text{constr}}, \mathcal{E}_{\text{constr}})$.

This representation unifies MaxCut, MIS, MaxClique, MVC, and MDS within the same heterogeneous graph input format $G = (\mathcal{V}_{\text{var}}, \mathcal{V}_{\text{constr}}, \mathcal{E}_{\text{prob}}, \mathcal{E}_{\text{obj}}, \mathcal{E}_{\text{constr}})$.

B. Heterogeneous Graph Neural Network Solver

Given the heterogeneous input, UniHetCO applies relation-specific message passing to compute node-level representations for each relation type. Concretely, for each variable node $v \in \mathcal{V}_{\text{var}}$, a separate message-passing network aggregates information from its neighbors under the problem, objective, and constraint relations, yielding three embeddings $\mathbf{h}_v^{\text{prob}}$, $\mathbf{h}_v^{\text{obj}}$, and $\mathbf{h}_v^{\text{constr}}$. Each embedding captures complementary signals: $\mathbf{h}_v^{\text{prob}}$ summarizes the original instance structure, $\mathbf{h}_v^{\text{obj}}$ encodes quadratic and linear objective couplings, and $\mathbf{h}_v^{\text{constr}}$ propagates constraint-specific context through variable–constraint interactions. The three embeddings are then fused, by concatenation followed by a multi-layer perceptron, to form a unified representation $\mathbf{h}_v$, and a shared prediction head maps $\{\mathbf{h}_v\}_{v=1}^n$ to a relaxed selection vector $\mathbf{x} \in [0,1]^n$, where each entry $\mathbf{x}_v$ represents the predicted likelihood that node v is included in the solution. At inference time, a greedy decoder projects the relaxed output back to a feasible discrete solution.

C. Universal QUBO Training Objective

Training uses a single surrogate objective shared across problem classes, consisting of an objective term derived from the encoded quadratic form and a penalty on constraint violations. The loss function for the objective graph $(\mathcal{V}_{\text{var}}, \mathcal{E}_{\text{obj}})$ is as follows:

$$\mathcal{L}_{\text{obj}}(G) = \mathbf{x}^\top \tilde{\mathbf{Q}} \mathbf{x}, \tag{19}$$

From the constraint graph $(\mathcal{V}_{\text{var}}, \mathcal{V}_{\text{constr}}, \mathcal{E}_{\text{constr}})$, we define a hinge loss over the violation vector $\mathbf{Ax} - \mathbf{b}$:

$$\mathcal{L}_{\text{constr}}(G) = \mathbf{1}^\top \max\big(\mathbf{0}, \mathbf{Ax} - \mathbf{b}\big), \tag{20}$$

where $\mathbf{1} \in \mathbb{R}^m$ is the all-one vector. The model learns to minimize the objective loss while reducing violations of linear constraints, without requiring ground-truth solutions:

$$\mathcal{L}_{\text{NCO}}(G) = \lambda_{\text{obj}} \mathcal{L}_{\text{obj}}(G) + \lambda_{\text{constr}} \mathcal{L}_{\text{constr}}(G), \tag{21}$$

where λ_{obj} and λ_{constr} are the balancing coefficients. Empirically, we set $\lambda_{\text{obj}} = \lambda_{\text{constr}} = 1.0$.

D. Graph Embedding Extraction

After training, we first extract the heterogeneous embeddings of the individual nodes $\mathbf{h}_v = [\mathbf{h}_v^{\text{prob}}, \mathbf{h}_v^{\text{obj}}, \mathbf{h}_v^{\text{constr}}]$ from the heterogeneous GNN Φ. We then obtain a graph-level representation by mean pooling over the node embeddings:

$$\mathbf{g} = \frac{1}{|\mathcal{V}_{\text{var}}|} \sum_v \mathbf{h}_v. \tag{22}$$

VI. Empirical Results

A. Experiment Setup

a) Evaluation Metrics: We evaluate the baselines on several metrics: (i) feasibility rate, (ii) optimal hit rate, and (iii) approximation ratio.

Feasibility Rate. After obtaining the final quantum state $|\psi(\boldsymbol{\theta}^*)\rangle$, we perform 5000 measurement shots and collect the resulting bit strings, denoted as the set $\mathcal{S}$. For constrained problem classes, we compute the feasibility rate as the fraction of sampled bit strings that satisfy the problem constraints.

$$\text{FR} = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \mathbb{I}[\mathbf{x} \in \mathcal{F}], \tag{23}$$

where $\mathcal{F} \subseteq \mathcal{S}$ denotes the feasible set and $\mathbb{I}[\cdot]$ is the indicator function.

Approximation Ratio. The approximation ratio is defined as the ratio of the expectation value of the QAOA circuit to the optimal solution value. Given the optimized variational parameters $\boldsymbol{\theta}^*$, the expectation value is calculated as

$$\text{AR} = \langle H_C(G; \boldsymbol{\theta}^*) \rangle \coloneqq \langle \psi(\boldsymbol{\theta}^*) | \, H_C(G) \, | \psi(\boldsymbol{\theta}^*) \rangle \tag{24}$$

$$= \sum_{\mathbf{x} \in \{0,1\}^n} \mathcal{C}(\mathbf{x}) \text{Pr}(\mathbf{x}), \tag{25}$$

where $\text{Pr}(\mathbf{x})$ is the probability of observing $\mathbf{x}$ when measuring $|\psi(\boldsymbol{\theta}^*)\rangle$. For unconstrained problem classes, we report the expectation value in (24). For constrained problem classes, we estimate an empirical expectation value conditioned on feasibility by restricting to sampled bitstrings in the feasible set $\mathcal{F}$, extending Eq. (25) as:

$$\langle H_C(G; \boldsymbol{\theta}^*) \rangle \coloneqq \sum_{\mathbf{x} \in \mathcal{F}} \mathcal{C}(\mathbf{x}) \ \text{Pr}(\mathbf{x}). \tag{26}$$

Note that for minimization problems, e.g. MVC, we report the $\text{AR} - 1$ metric, effectively the relative gap above the optimum, with lower value corresponding to better performance.

Optimal Hit Rate. The optimal hit rate is defined as the ratio of sampled feasible bit strings that achieve the optimal value.

$$p(\mathbf{x}^*) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \mathbb{I}[\mathcal{C}(\mathbf{x}) = \mathcal{C}^* \wedge \mathbf{x} \in \mathcal{F}] \tag{27}$$

where $\mathcal{C}^*$ denotes the optimal objective value. This metric is particularly appropriate for constrained problems because it evaluates end-to-end solver performance under the same sampling budget for all methods. In contrast, reporting approximation ratios conditioned on feasibility can obscure differences in feasibility rates across methods. Therefore, by directly quantifying the chance of obtaining an optimal feasible

TABLE I

RESULTS OF SINGLE-PROBLEM SETTING ON THE MAXCUT, MIS AND MAXCLIQUE PROBLEM CLASSES. BEST RESULTS ARE HIGHLIGHTED IN BOLD. ASTERISK (*) DENOTES BETTER RESULTS THAN VANILLA QAOA.

Circuit depth	$p=4$				$p=6$				$p=8$				$p=10$			
Metrics	$p(\mathbf{x}^*)$	AR	FR	T	$p(\mathbf{x}^*)$	AR	FR	T	$p(\mathbf{x}^*)$	AR	FR	T	$p(\mathbf{x}^*)$	AR	FR	T
Maximum Cut																
Vanilla QAOA	25.09	93.80	-	231.44	40.33	96.01	-	296.74	53.38	97.22	-	357.60	66.14	98.16	-	405.29
Meta-LSTM	18.99	92.02	-	10.00	37.24	95.01	-	10.00	**42.71**	95.12	-	10.00	55.59	96.68	-	10.00
G2V-Meta-LSTM	17.47	91.80	-	10.00	27.33	93.76	-	10.00	30.52	94.40	-	10.00	45.69	95.77	-	10.00
Uni-Meta-LSTM	**20.95**	**92.63**	-	10.00	**37.27**	**95.05**	-	10.00	41.23	**95.76**	-	10.00	**60.84**	**97.16**	-	10.00
Maximum Independent Set																
Vanilla QAOA	20.32	71.05	79.62	401.03	33.65	80.34	76.88	434.78	44.52	85.07	80.63	467.31	54.22	87.94	83.39	477.91
Meta-LSTM	14.77	63.59	79.03	10.00	32.57	**82.43***	66.60	10.00	39.72	88.13*	65.17	10.00	41.37	84.00	74.09	10.00
G2V-Meta-LSTM	17.78	**70.93**	70.11	10.00	20.21	73.51	66.79	10.00	44.66	82.77	88.60	10.00	42.10	**86.63**	71.37	10.00
Uni-Meta-LSTM	**18.49**	70.49	80.07	10.00	**33.79***	81.09*	78.30	10.00	**50.17***	**89.05***	79.25	10.00	**46.08**	84.31	89.66	10.00
Maximum Clique																
Vanilla QAOA	27.74	80.34	75.79	338.72	41.73	87.05	76.76	401.56	54.81	90.61	82.77	438.06	64.53	92.83	87.06	460.58
Meta-LSTM	11.57	65.45	53.66	10.00	33.78	86.99	61.93	10.00	33.60	87.25	59.54	10.00	39.70	87.96	67.39	10.00
G2V-Meta-LSTM	21.67	76.88	74.29	10.00	38.00	83.77	81.83	10.00	40.56	89.63	67.24	10.00	48.87	88.37	85.86	10.00
Uni-Meta-LSTM	**23.23**	**77.45**	79.60	10.00	**39.78**	**87.46***	73.95	10.00	**54.82***	**90.87***	83.30	10.00	**50.74**	**88.39**	80.66	10.00

TABLE II

RESULTS OF SINGLE-PROBLEM SETTING ON THE MVC PROBLEM CLASS. BEST RESULTS ARE HIGHLIGHTED IN BOLD. ASTERISK (*) DENOTES BETTER RESULTS THAN VANILLA QAOA.

Circuit depth	$p=4$				$p=6$				$p=8$				$p=10$			
Metrics	$p(\mathbf{x}^*)$	$\mathrm{AR}-1$	FR	T	$p(\mathbf{x}^*)$	$\mathrm{AR}-1$	FR	T	$p(\mathbf{x}^*)$	$\mathrm{AR}-1$	FR	T	$p(\mathbf{x}^*)$	$\mathrm{AR}-1$	FR	T
Vanilla QAOA	20.37	12.55	80.55	403.86	33.44	8.23	77.70	440.75	45.79	6.20	82.02	463.46	54.40	5.00	83.14	468.42
Meta-LSTM	14.80	15.59	79.16	10.00	32.69	8.18*	66.70	10.00	39.80	**5.64***	65.20	10.00	41.42	6.66	74.12	10.00
G2V-Meta-LSTM	17.84	14.04	70.10	10.00	20.36	11.69	66.85	10.00	**44.73**	7.65	88.58	10.00	42.21	5.91	71.42	10.00
Uni-Meta-LSTM	**20.03**	**12.70**	80.47	10.00	**34.01***	**8.14***	74.79	10.00	43.17	5.91*	73.35	10.00	**57.32***	**4.56***	86.10	10.00

solution within $|\mathcal{S}|$ shots, the optimal hit rate provides a fair and practically meaningful comparison when all approaches are evaluated on the same QAOA circuit family and the same measurement budget.

b) Dataset: To create the graph dataset, we follow the experiment setting similar to that of [20], [23], sampling 1000 random connected graphs for training and 100 for testing. Graph sizes for training are drawn from $n \in [6, 10]$, and edges are generated with probability k/n, where $k \in [3, n-1]$. For testing, we fix the graph size to $n = 12$ and use the same data generation procedure. All graphs are non-isomorphic.

c) Baselines: We compare our graph-conditioned meta-optimizer against three baselines. (i) **Vanilla QAOA**: for each instance, we optimize QAOA parameters from a random initialization $\boldsymbol{\theta}$ for up to $T = 500$ steps; we treat this per-instance optimization as an empirical upper bound. (ii) **Meta-LSTM**: an unconditioned LSTM meta-optimizer trained to generate QAOA parameter trajectories, following [20], [23]. (iii) **G2V-Meta-LSTM**: the same LSTM meta-optimizer conditioned on the Graph2Vec embedding. We dub our method **Uni-Meta-LSTM**.

d) Implementation Details: We set the rollout horizon for all meta-optimizer baselines to $T = 10$ and $\omega_t = \frac{t}{10}$ for Eq. (8). During training, we use a mini-batch of $\mathcal{B} = 32$ samples per gradient update and train the meta optimizer for 100 epochs. We set the LSTM hidden state and cell state dimension to 48. We select the best checkpoint as the model that achieves the lowest mean energy on the training set and then deploy it on the test set. For each problem class, we train a separate model at each circuit depth $p \in \{4, 6, 8, 10\}$. We implement quantum simulation in PennyLane [42] and use PyTorch [43] to train the neural meta-optimizer. We use the Adam optimizer [44] to train both the Vanilla QAOA and meta-learning-based methods, setting the learning rate to 0.01 for the former and 0.001 for the latter. We conduct all experiments on the same seed using a single 16GB NVIDIA T4 GPU and run quantum simulations using PennyLane's `lightning.qubit` device.

B. Results on Single-Problem Parameter Generation

In this setting, we train and evaluate the neural meta-optimizer within the same problem class. We report the optimal hit rate, approximation ratio, and feasibility rate in Table I and II for MaxCut, MIS, MaxClique, and MVC, respectively. Additionally, we record the average number of optimization steps required for Vanilla QAOA to converge to a tolerance of 10^{-8}. As shown in Table I and II, Uni-Meta-LSTM consistently achieves the best optimal hit rate $p(\mathbf{x}^*)$ in 14 of the 16 settings and the best approximation ratio in 12 of the 16 settings. Furthermore, the meta-optimizer variants outperform Vanilla QAOA in several settings, particularly for constrained problem classes. For example, on MIS at $p = 8$,

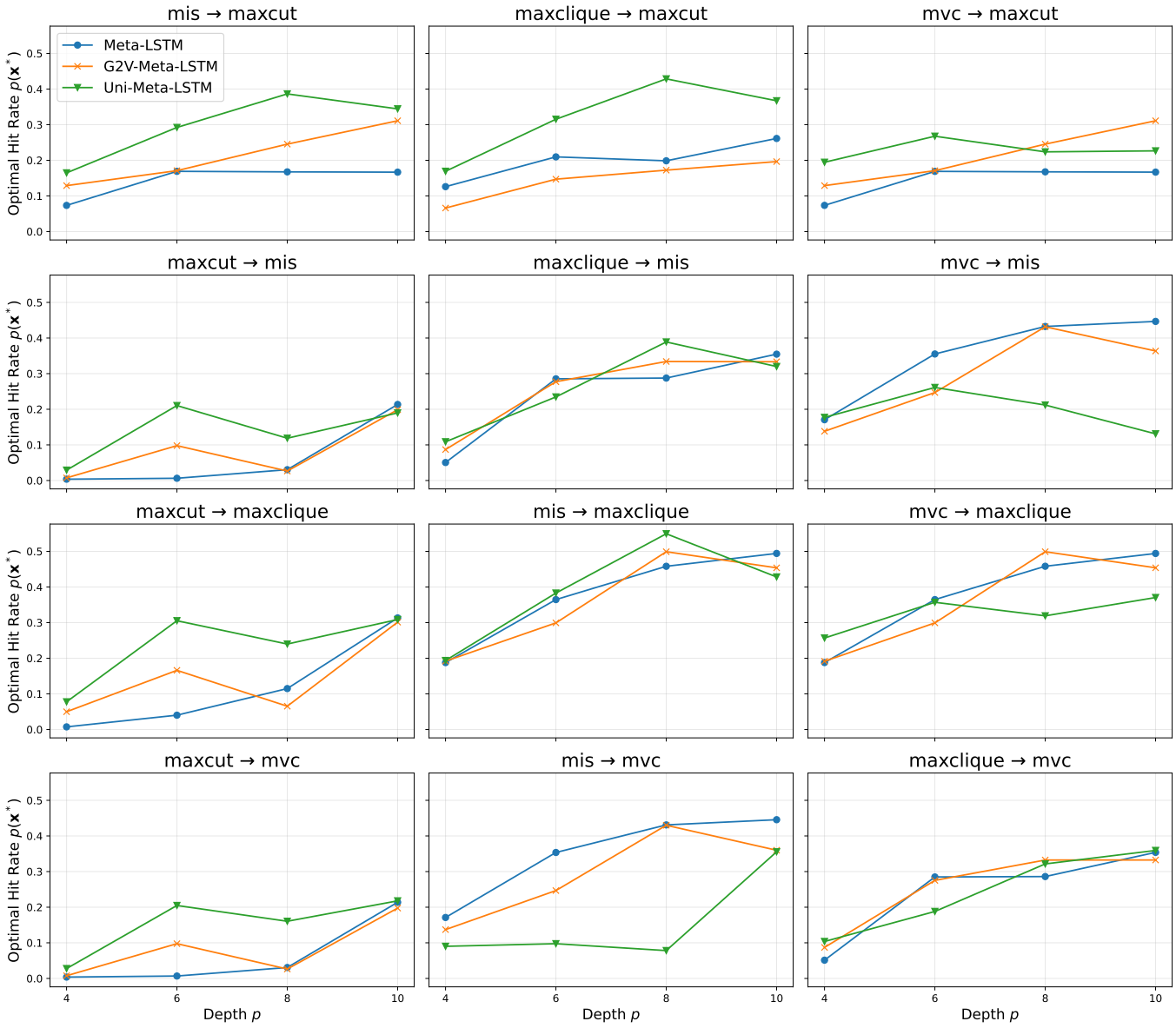

Fig. 4. Optimal hit rate $p(\mathbf{x}^*)$ on cross-problem parameter transfer. We report pairwise problem transfer across four circuit depths $p \in \{4, 6, 8, 10\}$, resulting in 48 transfer settings. The meta-optimizer is trained on one problem class and then fine-tuned on another with 5 gradient steps.

Uni-Meta-LSTM and Meta-LSTM achieve approximation ratios that are 3.98% and 3.06% higher than Vanilla QAOA, respectively, despite running only 10 optimization steps. Additionally, our method sees a 5.65% increase in optimal hit rate for the same setting. Notably, on MaxClique, Uni-Meta-LSTM dominates the other learned baselines across all three metrics and all circuit depths, and it even surpasses Vanilla QAOA in approximation ratio at $p = 6$ and in both $p(\mathbf{x}^*)$ and approximation ratio at $p = 8$. We do, however, observe a slight performance drop from $p = 8$ to $p = 10$ on MIS, MaxClique and MVC, suggesting that additional model capacity or conditioning mechanisms may be needed to fully benefit from deeper circuits. On the other hand, G2V-Meta-LSTM achieves competitive performance in a subset of settings but generally trails Meta-LSTM and Uni-Meta-LSTM. This suggests that purely structure-based graph embeddings may be insufficient for conditioning meta-optimizers; incorporating objective and constraint information appears important for learning transferable, instance-adaptive parameter trajectories.

C. Results on Cross-Problem Parameter Transfer

In this experiment, we train the meta-optimizer on one problem class and directly evaluate it on a different class to assess cross-problem transferability. We perform exhaustive pairwise transfers among the four problem classes across all four circuit depths, yielding $4 \times 3 \times 4 = 48$ transfer settings. For each transfer, we additionally allow a lightweight per-instance adaptation stage in which the meta-optimizer is fine-tuned on

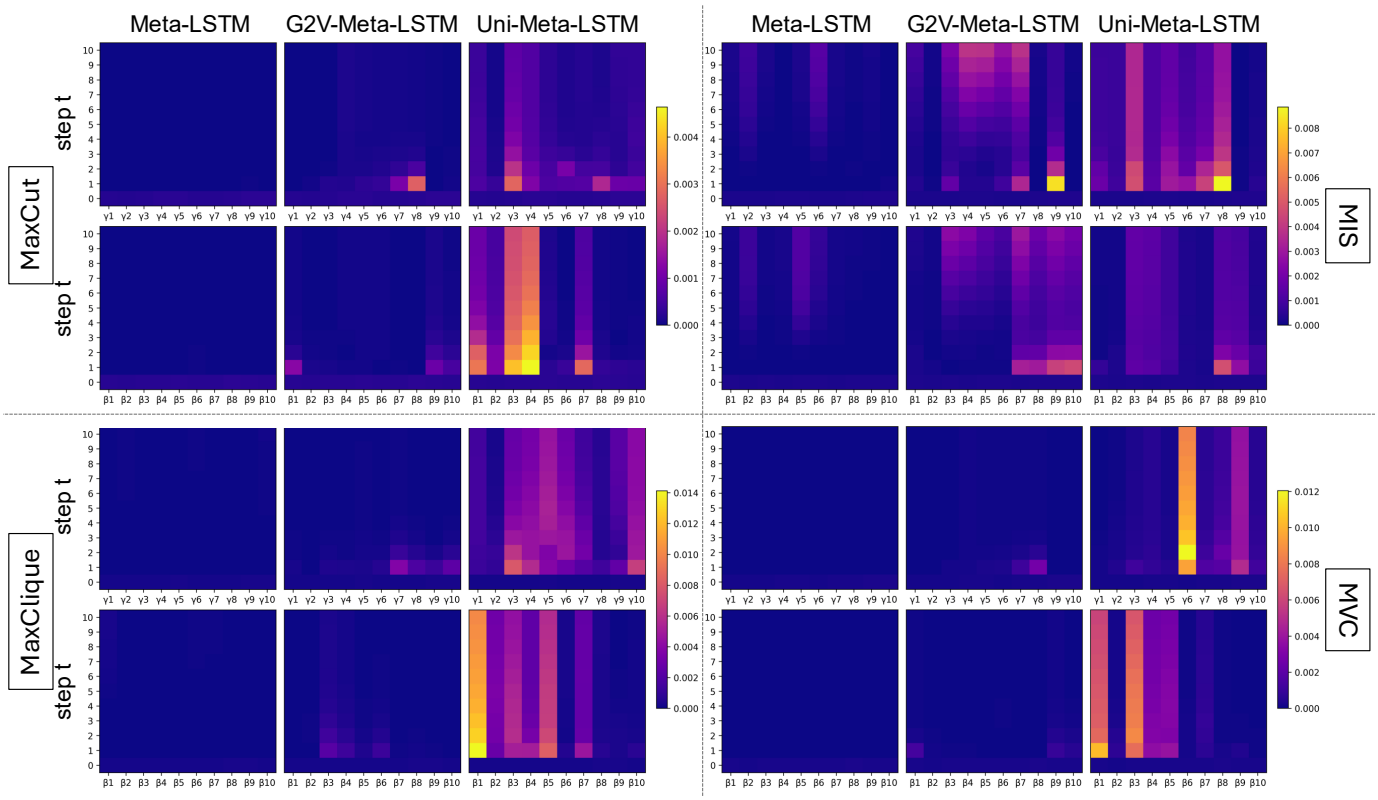

Fig. 5. Visualization of diversity in generated parameter trajectories among three methods, Meta-LSTM, G2V-Meta-LSTM and Uni-Meta-LSTM, on four problem classes (MaxCut, MIS, MaxClique and MVC). The trajectories are plotted on the QAOA circuit depth $p = 10$ and unrolling horizon $T = 10$.

the target instance for 5 gradient steps with a learning rate of 0.001, which is natural in our unsupervised setting since the QAOA objective provides direct training feedback without ground-truth angles.

In Figure 4, we report the optimal hit rate $p(\mathbf{x}^*)$ for the three meta-optimizer variants. Uni-Meta-LSTM achieves better cross-problem performance than Meta-LSTM in 34 of the 48 transfer settings, with the largest gains at depths $p \in \{4, 6, 8\}$ and in transfers from other problems to MaxCut and MaxClique. G2V-Meta-LSTM occasionally improves over Meta-LSTM (e.g., in 21 transfer settings), but it still underperforms Uni-Meta-LSTM overall. A key limitation is that Graph2Vec provides a purely structure-based embedding; therefore, when transferring between problem classes on the same underlying graph, the conditioning signal remains essentially unchanged because the embedding does not encode the target objective or constraints. In contrast, Uni-Meta-LSTM conditions on problem-aware embeddings that incorporate objective and constraint information, enabling the meta-optimizer to adapt its parameter trajectory to the specific problem formulation rather than relying solely on graph structure.

D. Visualization of Parameter Trajectory Expressivity

In this section, we compare the expressivity of our graph-conditioned meta-optimizer framework with the unconditioned counterpart by visualizing the diversity in the generated parameter trajectories. We first collect the trajectories over $N = 100$ test instances in the single-problem setting. For each instance, we run the meta-optimizer for a fixed rollout horizon $T = 10$ and record the QAOA parameters generated at every step, retrieving a tensor of trajectories of dimension $N \times T \times 2p$, where $2p$ corresponds to $(\gamma, \beta)^p$. We then quantify the trajectory diversity by computing the variance of each angle coordinate across instances at each step and aggregate these variances within γ and β separately. Finally, we visualize the per-step variance patterns as heatmaps and compare the unconditioned and conditioned methods under a shared magnitude scale, where higher variance indicates more instance-dependent, thus more expressive, parameter generation.

We observe that the unconditioned Meta-LSTM tends to collapse to near-identical optimization paths, limiting its ability to generalize as problem complexity increases. In contrast, our Uni-Meta-LSTM exhibits substantially higher trajectory diversity than Meta-LSTM, indicating that incorporating instance-specific information yields more adaptive parameter generation and improved performance on more complex instances, as reflected in Tables I–II.

E. Pre-training and Visualization of UniHetCO Embeddings

In this section, we describe the UniHetCO pre-training procedure in detail. We then visualize the resulting problem-aware embeddings using t-distributed stochastic neighbor embedding (t-SNE) [45], showing that instances form separable clusters in the embedding space. Finally, we discuss how the relative

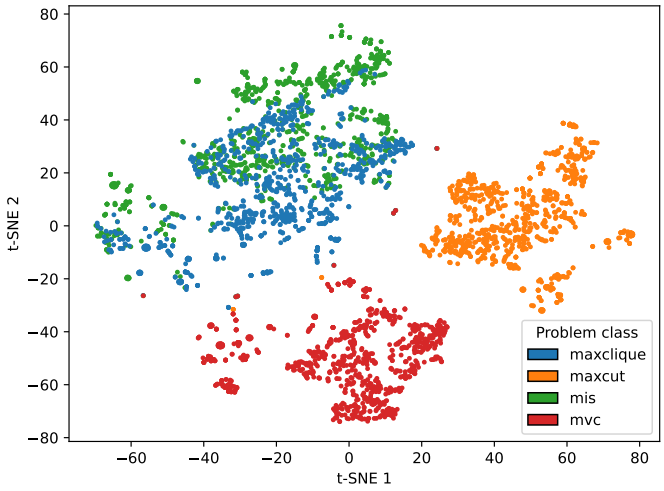

Fig. 6. A demonstration of the UniHetCO embedding space: t-SNE visualization of UniHetCO graph embeddings for 1000 training instances from each of four problem classes (MaxCut, MIS, MaxClique, and MVC), totaling 4000 points. The embeddings form well-separated clusters by problem class, while also reflecting problem similarity; MIS and MaxClique exhibit partially interleaved clusters, consistent with their closely related structure.

geometry between problem classes captures meaningful relationships that can be exploited to condition the meta-optimizer.

Given K problem classes (here, $K = 4$), we cast UniHetCO pre-training as a multi-domain learning problem, where each problem class defines a domain. Let $\mathcal{L}_{\text{NCO},k}$ denote the unsupervised NCO loss for class k. We train a single shared model by minimizing the mean loss across domains:

$$\hat{\mathcal{L}}_{\text{NCO}}(\theta; \mathcal{D}_{\text{train}}) = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}_{\text{NCO},k}(\theta; G), \tag{28}$$

where G denotes an instance sampled from the training distribution of the corresponding domain. In each training iteration, we construct mini-batches by sampling an equal number of instances from each problem class, ensuring balanced exposure to all domains and preventing the training dynamics from being dominated by any single class.

After pre-training, we employ t-SNE [45] to visualize the relationships among the learned problem-aware graph embeddings. We extract the embeddings from the UniHetCO trained GNN Φ on 1000 graph instances across four problem classes, resulting in 4000 points in the Euclidean space.

Why UniHetCO Embeddings Are Informative. As shown in Figure 6, the learned embeddings form well-separated clusters by problem class, indicating that the representation captures problem-specific signals beyond mere graph structure. Moreover, the *relative geometry* between clusters is informative, as MIS and MaxClique exhibit partially interleaved regions, consistent with their closely related subset-selection structure (e.g., a clique in a graph G corresponds to an independent set in its complement $\bar{G}$). This proximity provides a useful conditioning signal for the meta-optimizer, as instances that are embedded closer together are expected to induce more similar QAOA landscapes and parameter-update dynamics, thereby facilitating transfer across related formulations. In contrast, Graph2Vec encodes only structural information and does not incorporate the objective or constraints; consequently, the same underlying graph would receive identical embeddings regardless of the problem class, making it less suitable for guiding cross-problem instance-wise parameter generation.

VII. Conclusion and Future Work

In this paper, we revisited the learning-to-learn paradigm for QAOA parameter generation to address an overlooked shortcoming of existing methods: *near-identical parameter trajectories*. We thus improve instance-wise trajectory diversity by showing that graph embeddings provide an effective conditioning signal for parameter trajectory generation. By injecting problem-aware embeddings that encode both structural and formulation-specific information, our Uni-Meta-LSTM produces more diverse and instance-adaptive parameter trajectories than unconditioned meta-optimizers, leading to improved performance on larger problem sizes and stronger cross-problem transfer. Results suggest incorporating objective- and constraint-aware representations is a promising direction for amortizing QAOA parameter search and improving transferability across related optimization formulations.

Future Work. In our empirical studies, we observe a modest performance degradation when increasing the circuit depth from $p = 8$ to $p = 10$, suggesting that the current conditioning mechanism may not fully exploit the additional expressivity of deeper circuits. Future work could investigate stronger conditioning strategies for long-horizon parameter generation, as well as training objectives that better align intermediate trajectory updates with final performance. Beyond the current setting, an interesting direction is to train a more general meta-optimizer that can generate parameter trajectories across multiple problem classes and circuit depths within a single model, thereby enabling broader transfer and reducing the need to maintain separate models for each formulation.

APPENDIX A
DEFINITION OF COMBINATORIAL PROBLEM CLASSES

We define the objectives and constraints for each problem class, followed by the conversion to the QAOA cost Hamiltonian in PennyLane [42]. The mixing Hamiltonian is constructed as $\sum_{i\in V} X_i$, where X is the Pauli operator X, for all problems. We set the initial state to $|+\rangle^{\otimes n}$.

A. Maximum Cut

The Maximum Cut problem involves finding a cut that splits a graph into two disjoint partitions so that the weighted sum of the edges within the cut is maximized. We focus on unweighted MaxCut, setting all the edge weights to 1.

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{(u,v)\in E} x_u + x_v - 2x_u x_v. \tag{29}$$

To construct the QAOA cost Hamiltonian, we map each binary variable to a qubit via

$$x_i = \frac{1 - Z_i}{2}, \tag{30}$$

where Z_i is the Pauli-Z operator on qubit i. Substituting into (29) yields the cut-size operator

$$C(\mathbf{Z}) = \frac{1}{2} \sum_{(u,v)\in E} \big(\mathbb{I} - Z_u Z_v\big), \tag{31}$$

up to an additive constant. Following PennyLane's convention, we define the QAOA cost Hamiltonian as the negative of the cut-size operator:

$$H_C = -C(\mathbf{Z}) = \frac{1}{2} \sum_{(u,v)\in E} \big(Z_u Z_v - \mathbb{I}\big), \tag{32}$$

so that minimizing $\langle H_C\rangle$ is equivalent to maximizing the MaxCut objective.

B. Maximum Independent Set

The Maximum Independent Set problem seeks the largest subset of vertices such that no two selected vertices share an edge:

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{i\in V} x_i \quad \text{s.t. } x_u + x_v \le 1,\ \forall (u,v)\in E. \tag{33}$$

To obtain an unconstrained formulation compatible with the standard X-mixer, we relax the constraints via a quadratic penalty and maximize

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{i\in V} x_i \ - \ \lambda \sum_{(u,v)\in E} x_u x_v, \tag{34}$$

where $\lambda > 0$ discourages selecting adjacent vertices. Up to an additive constant, this yields a cost Hamiltonian of the form

$$H_C \propto \sum_{(u,v)\in E} \big(Z_u Z_v - Z_u - Z_v\big) + \sum_{i\in V} Z_i. \tag{35}$$

We adopt PennyLane's unconstrained MIS convention:

$$H_C = 3 \sum_{(u,v)\in E} \big(Z_u Z_v - Z_u - Z_v\big) + \sum_{i\in V} Z_i, \tag{36}$$

so that minimizing $\langle H_C\rangle$ promotes large independent sets while imposing a strong penalty on constraint violations.

C. Maximum Clique

The Maximum Clique problem aims to find the largest subset of vertices such that every pair of selected vertices is connected by an edge. This can be enforced by requiring that for any non-edge $(u,v)\notin E$, at most one of u and v can be selected:

$$\max_{\mathbf{x}\in\{0,1\}^n} \sum_{i\in V} x_i \quad \text{s.t. } x_u + x_v \le 1,\ \forall (u,v)\notin E,\ u\neq v. \tag{37}$$

To obtain an unconstrained formulation compatible with the standard X-mixer, we relax the constraints via a quadratic penalty and maximize

$$\max_{\mathbf{x}\in\{0,1\}^N} \sum_{i\in V} x_i - \lambda \sum_{(u,v)\in E(\bar{G})} x_u x_v, \tag{38}$$

where $\lambda > 0$ discourages selecting non-adjacent vertex pairs in G (i.e., edges in the complement graph $\bar{G}$). Following PennyLane's unconstrained Maximum Clique convention, the QAOA cost Hamiltonian is defined as

$$H_C = 3 \sum_{(i,j)\in E(\bar{G})} \big(Z_i Z_j - Z_i - Z_j\big) + \sum_{i\in V(G)} Z_i, \tag{39}$$

so that minimizing $\langle H_C\rangle$ promotes large cliques while imposing a strong penalty on violations encoded by $E(\bar{G})$.

D. Minimum Vertex Cover

The Minimum Vertex Cover problem seeks the smallest subset of vertices that covers all edges:

$$\min_{\mathbf{x}\in\{0,1\}^n} \sum_{i\in V} x_i \quad \text{s.t. } x_u + x_v \ge 1,\ \forall (u,v)\in E. \tag{40}$$

To obtain an unconstrained formulation compatible with the standard X-mixer, we enforce the edge-cover constraints via a penalty and minimize

$$\min_{\mathbf{x}\in\{0,1\}^N} \sum_{i\in V} x_i \ + \ \lambda \sum_{(u,v)\in E} (1 - x_u)(1 - x_v), \tag{41}$$

where $(1-x_u)(1-x_v)$ equals 1 if and only if neither endpoint of (u,v) is selected, and 0 otherwise. Following PennyLane's unconstrained MVC convention, the QAOA cost Hamiltonian is defined as

$$H_C = 3 \sum_{(u,v)\in E} \big(Z_u Z_v + Z_u + Z_v\big) - \sum_{i\in V} Z_i, \tag{42}$$

up to an additive constant, so that minimizing $\langle H_C\rangle$ promotes small vertex covers while penalizing uncovered edges.

ACKNOWLEDGMENT

This work was supported in part by NSF award #2444042.

REFERENCES

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. [Online]. Available: https://arxiv.org/abs/1411.4028

[2] S. Hadfield, Z. Wang, B. O'gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, p. 34, 2019.

[3] R. Herrman, P. C. Lotshaw, J. Ostrowski, T. S. Humble, and G. Siopsis, "Multi-angle quantum approximate optimization algorithm," *Scientific Reports*, vol. 12, no. 1, p. 6781, 2022.

[4] D. Herman, C. Googin, X. Liu, Y. Sun, A. Galda, I. Safro, M. Pistoia, and Y. Alexeev, "Quantum computing for finance," *Nature Reviews Physics*, vol. 5, no. 8, pp. 450–465, 2023.

[5] C. Outeiral, M. Strahm, J. Shi, G. M. Morris, S. C. Benjamin, and C. M. Deane, "The prospects of quantum computing in computational molecular biology," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 11, no. 1, p. e1481, 2021.

[6] L. Lin, "Lecture notes on quantum algorithms for scientific computation," *arXiv preprint arXiv:2201.08309*, 2022.

[7] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature communications*, vol. 9, no. 1, p. 4812, 2018.

[8] E. R. Anschuetz and B. T. Kiani, "Quantum variational algorithms are swamped with traps," *Nature Communications*, vol. 13, no. 1, p. 7760, 2022.

[9] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, "Noise-induced barren plateaus in variational quantum algorithms," *Nature communications*, vol. 12, no. 1, p. 6961, 2021.

[10] A. Kulshrestha and I. Safro, "Beinit: Avoiding barren plateaus in variational quantum algorithms," *arXiv preprint arXiv:2204.13751*, 2022.

[11] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Physical Review X*, vol. 10, no. 2, p. 021067, 2020.

[12] J. Montanez-Barrera and K. Michielsen, "Towards a universal qaoa protocol: Evidence of a scaling advantage in solving some combinatorial optimization problems," *arXiv preprint arXiv:2405.09169*, 2024.

[13] R. Shaydulin, I. Safro, and J. Larson, "Multistart methods for quantum approximate optimization," in *2019 IEEE high performance extreme computing conference (HPEC)*. IEEE, 2019, pp. 1–8.

[14] I. Tyagin, M. H. Farag, K. Sherbert, K. Shirali, Y. Alexeev, and I. Safro, "QAOA-GPT: Efficient generation of adaptive and regular quantum approximate optimization algorithm circuits," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2025, pp. 1505–1515.

[15] J. Montanez-Barrera, D. Willsch, and K. Michielsen, "Transfer learning of optimal qaoa parameters in combinatorial optimization (2024)," *arXiv preprint arXiv:2402.05549*, 2024.

[16] F. G. Brandao, M. Broughton, E. Farhi, S. Gutmann, and H. Neven, "For fixed control parameters the quantum approximate optimization algorithm's objective function value concentrates for typical instances," *arXiv preprint arXiv:1812.04170*, 2018.

[17] A. Galda, X. Liu, D. Lykov, Y. Alexeev, and I. Safro, "Transferability of optimal qaoa parameters between random graphs," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2021, pp. 171–180.

[18] J. Falla, Q. Langfitt, Y. Alexeev, and I. Safro, "Graph representation learning for parameter transferability in quantum approximate optimization algorithm," *Quantum Machine Intelligence*, vol. 6, no. 2, 2024.

[19] K. X. Nguyen, B. Bach, and I. Safro, "Cross-problem parameter transfer in quantum approximate optimization algorithm: A machine learning approach," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2025, pp. 2202–2208.

[20] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, "Learning to learn with quantum neural networks via classical neural networks," *arXiv preprint arXiv:1907.05415*, 2019.

[21] M. Wilson, R. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. G. Rieffel, "Optimizing quantum heuristics with meta-learning," *Quantum Machine Intelligence*, vol. 3, no. 1, p. 13, 2021.

[22] H. Wang, J. Zhao, B. Wang, and L. Tong, "A quantum approximate optimization algorithm with metalearning for maxcut problem and its simulation via tensorflow quantum," *Mathematical Problems in Engineering*, vol. 2021, no. 1, p. 6655455, 2021.

[23] R. Huang, X. Tan, and Q. Xu, "Learning to learn variational quantum algorithm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 11, pp. 8430–8440, 2022.

[24] L. Friedrich and J. Maziero, "Learning to learn with an evolutionary strategy applied to variational quantum algorithms," *Physical Review A*, vol. 111, no. 2, p. 022630, 2025.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *arXiv preprint arXiv:1707.05005*, 2017.

[27] K. X. Nguyen and I. Safro, "Unihetco: A unified heterogeneous representation for multi-problem learning in unsupervised neural combinatorial optimization," 2026. [Online]. Available: https://arxiv.org/abs/2603.11456

[28] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," *Advances in neural information processing systems*, vol. 29, 2016.

[29] T. Munkhdalai and H. Yu, "Meta networks," in *International conference on machine learning*. PMLR, 2017, pp. 2554–2563.

[30] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[31] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals, "Rapid learning or feature reuse? towards understanding the effectiveness of maml," *arXiv preprint arXiv:1909.09157*, 2019.

[32] K. X. Nguyen, F. Qiao, and X. Peng, "Adaptive cascading network for continual test-time adaptation," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 2024, pp. 1763–1773.

[33] J. Sybrandt and I. Safro, "FOBE and HOBE: First-and high-order bipartite embeddings," *ACM KDD 2020 Workshop on Mining and Learning with Graphs, preprint at arXiv:1905.10953*, 2019.

[34] F. Ding, X. Zhang, J. Sybrandt, and I. Safro, "Unsupervised hierarchical graph representation learning by mutual information maximization," *arXiv preprint arXiv:2003.08420*, 2020.

[35] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016.

[36] L. Wang, C. Huang, W. Ma, X. Cao, and S. Vosoughi, "Graph embedding via diffusion-wavelets-based node feature distribution characterization," in *Proceedings of the 30th ACM international conference on information & knowledge management*, 2021, pp. 3478–3482.

[37] C. Cai and Y. Wang, "A simple yet effective baseline for non-attributed graph classification," *arXiv preprint arXiv:1811.03508*, 2018.

[38] A. Galland and M. Lelarge, "Invariant embedding for graph classification," in *ICML 2019 workshop on learning and reasoning with graph-structured data*, 2019.

[39] T. Jones and J. Gacon, "Efficient calculation of gradients in classical simulations of variational quantum algorithms," 2020. [Online]. Available: https://arxiv.org/abs/2009.02823

[40] G. E. Crooks, "Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition," *arXiv preprint arXiv:1905.13311*, 2019.

[41] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Physical Review A*, vol. 99, no. 3, p. 032331, 2019.

[42] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong,

S. Zhang, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," 2022. [Online]. Available: https://arxiv.org/abs/1811.04968

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: https://arxiv.org/abs/1912.01703

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.