2025 IEEE International Conference on Quantum Computing and Engineering (QCE) | 979-8-3315-5736-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/QCE65121.2025.00179

AGENT-Q: Fine-Tuning Large Language Models for Quantum Circuit Generation and Optimization

Linus Jern
Aalto University
linus.jern@aalto.fi

Valter Uotila
Aalto University & University of Helsinki
valter.uotila@aalto.fi

Cong Yu
Aalto University
cong.yu@aalto.fi

Bo Zhao
Aalto University
bo.zhao@aalto.fi

***Abstract*—Large language models (LLMs) have achieved remarkable outcomes in complex problems, including math, coding, and analyzing large amounts of scientific reports. Yet, few works have explored the potential of LLMs in quantum computing. The most challenging problem is to leverage LLMs to automatically generate quantum circuits at a large scale. Fundamentally, the existing pre-trained LLMs lack the knowledge of quantum circuits. In this paper, we address this challenge by fine-tuning LLMs and injecting the domain-specific knowledge of quantum computing.**

We describe AGENT-Q, an LLM fine-tuning system to generate and optimize quantum circuits. In particular, AGENT-Q implements the mechanisms to generate training data sets and constructs an end-to-end pipeline to fine-tune pre-trained LLMs to generate parameterized quantum circuits for various optimization problems. AGENT-Q provides 14,000 quantum circuits covering a large spectrum of the quantum optimization landscape: 12 optimization problem instances and their optimized QAOA, VQE, and adaptive VQE circuits. Based thereon, AGENT-Q fine-tunes LLMs and constructs syntactically correct parametrized quantum circuits in OpenQASM 3.0. We have evaluated the quality of the LLM-generated circuits and parameters by comparing them to the optimized expectation values and distributions. Experimental results show superior performance of AGENT-Q, compared to several state-of-the-art LLMs and better parameters than random. AGENT-Q can be integrated into an agentic workflow, and the generated parametrized circuits with initial parameters can be used as a starting point for further optimization, *e.g.,* as templates in quantum machine learning and as benchmarks for compilers and hardware.

***Index Terms*—large-language models, fine-tuning, quantum circuit generation, optimization, parameter initialization**

I. INTRODUCTION

Large language models (LLMs) have shown increasing capabilities in various tasks. While originally designed for text generation, LLMs have since excelled in code generation [1] [2] [3] [4], music generation [5], and even image and video generation [6] [7]. LLMs will also likely be helpful in supporting quantum algorithm developers and quantum computing end-users in various tasks such as quantum circuit generation, hybrid quantum-classical code generation, and circuit compilation.

Quantum computing requires deep expertise in quantum algorithms and hardware. For instance, in quantum machine learning and optimization problems, it is nontrivial to configure an optimal parametrized circuit that can be used to solve the given problem. Finding a performant initial starting point for the quantum optimization and training routines might be even harder. The convergence of optimization and training depends on the initial selection of circuit parameters. As a result, quantum circuit generation and parameter initialization appear to be promising problems for LLMs.

In this paper, we study how well LLMs can generalize to optimization problems in quantum computing. To this end, we have designed AGENT-Q to fine-tune a pre-trained LLM with specially-crafted data sets that contain optimized QAOA, VQE, and adaptive VQE circuits for common optimization problems. Such optimization problems have been solved, producing circuits with optimized parameters – making it one of the largest quantum circuit data sets of over 14,000 circuits.

AGENT-Q also provides prompts to indicate the optimization problem and the corresponding optimized circuits. After fine-tuning, a user can design a prompt specifying an optimization problem and ask AGENT-Q to produce various circuits with initial parameters to solve the optimization problem. We show that the circuits generated by AGENT-Q's fine-tuned LLM outperform the state-of-the-art LLMs, and the initial parameters are often *closer* to the optimal value than random ones. We evaluate the performance of AGENT-Q's model using three metrics: (i) syntactic correctness, (ii) ability to generate circuits with expectation values close to the optimized targets, and (iii) ability to produce circuits with probability distributions that align with the optimized ones.

AGENT-Q trains an LLM to produce pure quantum circuits instead of creating quantum-classical code. Focusing solely on quantum circuits, we introduced evaluation metrics that depend on solutions to the optimization problems, which makes the evaluation more robust. Current LLMs are very good at Python code but lack knowledge about quantum computing. Thus, it is likely that the models relatively easily learn the hybrid quantum-classical pipelines after they understand quantum computing and classical computing separately.

Moreover, quantum circuit generation might be useful not only as an initial step for quantum optimization pipelines but also for other tasks. For example, circuit compilers, quantum error correction, and mitigation algorithms can be benchmarked with LLM-generated circuits. Additionally, the fact that LLMs could already generate framework-independent quantum circuits (*e.g.*, Qiskit, PennyLane, Cirq) marks an important first step toward more advanced capabilities, where LLMs could help create larger hybrid workloads that seamlessly combine Python and quantum code.

979-8-3315-5736-2/25/$31.00 ©2025 IEEE
DOI 10.1109/QCE65121.2025.00179

We describe AGENT-Q, an end-to-end LLM fine-tuning framework to generate and optimize quantum circuits. AGENT-Q makes the following new technical contributions:

- A comprehensive dataset of over 14,000 quantum circuits, identifying 12 optimization with QAOA, VQE, and adaptive VQE and their optimized circuits, making it one of the largest collections of quantum circuits.
- An end-to-end LLM fine-tuning pipeline that automatically generates syntactically correct quantum circuits.
- We have conducted comprehensive experiments to show AGENT-Q's fine-tuned LLM outperforms the state-of-the-art models and produces initial parameters that are closer to optimal than random. We have open-sourced the code and data in [8], [9].

This article is organized as follows. First, we review the data set generation that consists of 12 optimization problems, which are used to create the training and test circuits with the optimized parameters. Then, we describe the fine-tuning pipeline, which produces a large language model fine-tuned on the previously generated data. Next, we present the evaluation, which covers syntax and performance metrics related to the optimization problems. In the discussion section, we analyze the results and suggest multiple possible next steps.

A. Related work and previous use cases for generated circuits

Only a few previous works have utilized large-language models for quantum code generation. One of these works is [10], which is the basis for Qiskit Code Assistant [11] and evaluated with [12]. These works are strongly built around Qiskit, while we designed our model to work with OpenQASM 3.0 [13], which has become a platform-independent standard supported by Qiskit, Pennylane, and Cirq. Language models have also been used to design quantum experiments [14].

While LLMs have not yet been widely utilized in quantum computing, the standard transformer-based model has been used in various quantum computing applications. For example, the standard GPT model predicted measurement outcomes from a neutral atom quantum computer [15]. The work showed how the standard GPT model has certain limitations when trained to predict measurement outcomes. These findings might be helpful to broaden our understanding of the limitations of the current LLM models.

Nvidia has developed a transformer-based optimization pipeline that generates quantum circuits in the search for ground states of electronic structure Hamiltonians [16]. Since the trainable parameters are in the transformer model, the method aims to circumvent specific problems that the current variational methods have, such as barren plateaus.

KetGPT uses GPT-based models to generate quantum circuits [17], which have been trained on QASMBench circuits [18]. The produced synthetic circuits mimic the structure in the training dataset. The circuits are limited to OpenQASM 2.0 format, without supporting parameters.

II. QUANTUM OPTIMIZATION DATA SET FOR FINE-TUNING

The training dataset is one of the most essential components in fine-tuning large language models. This section describes how we have constructed a large, high-quality, and diverse data set consisting of around 14,000 optimized quantum circuits for multiple key optimization primitives on graphs commonly used in classical and quantum optimization algorithms. As a result, we have not only constructed a comprehensive training data set but also a data set consisting of circuits that can be alternatively used for hardware and circuit compilation benchmarking and other tasks. The circuits are expressed in the most recent OpenQASM 3.0 format, which supports parametrized quantum circuits, and the dataset is available on HuggingFace [9], and the code to generate the dataset is available on GitHub [8].

As pointed out in [19], the key features we want from the dataset are quality, difficulty, and diversity. Considering the dataset we have created, we have aimed to satisfy these key characteristics:

- Quality: The data not only contains circuits for optimization problems but also the optimized parameters. If the circuits are executed with the given parameters, there is a high probability of measuring the bitstring corresponding to the correct solution.
- Difficulty: Creating high-performing quantum circuits with good parameters is known to be a challenging task.
- Diversity: We have included 12 different optimization problems on graphs. Moreover, we have solved the problems with QAOA, VQE, and adaptive VQE.

A. Optimization problems

This subsection briefly reviews the implemented optimization problems and our optimization methods. Since the goal is to generate high-quality parametrized quantum circuits for optimization problems, the training dataset should contain a representative set of such problems. The optimization problems we have considered are standard primitives in various classical optimization algorithms, and many of them appeared in [20]. We have limited our focus on optimization problems on graphs, which form the core of optimization algorithms [21]. The selected problems are listed in Table I, and their implementations are on GitHub [8].

1) Connected components in graphs: Finding connected components in a graph G means finding a partition P of G such that every subgraph in P is connected, meaning that every two nodes in the subgraph are connected with a path. In this implementation, we consider a Quadratic Unconstrained Binary Optimization (QUBO) formulation for the connected components problem. In this formulation, we fix a node in a graph, and the algorithm returns the connected component to which the fixed node belongs.

2) Community detection: In the community detection problem, the goal is to find a partition P of a graph G so that the density of the edges within the partitions in P is higher than the density of edges between them. In this work, we implemented the QUBO formulation for the community

TABLE I

Optimization problems in the training dataset. The complexity refers to the classical decision variant. The binary optimization formulations for problems without reference are developed in this work.

Problem	Formulation	Classical complexity
Connected Components	QUBO	P [22]
Community Detection [23]	QUBO	NP-hard [24]
k-Clique [20]	QUBO	NP-complete [21]
Graph Isomorphism [20]	QUBO	in NP (open) [25], [26]
Graph Coloring [20]	QUBO	NP-complete [21]
Traveling Salesman [20]	QUBO	NP-complete [21]
Weighted Minimal Maximal Matching	QUBO	NP-Hard [20]
Vertex Cover [20]	QUBO	NP-complete [21]
Edge Cover	HUBO	P [27]
Max-Flow [28]	QUBO	P [29]
Min-Cut [28]	QUBO	P [30]
(Hyper)MaxCut [31]	HUBO	MaxCut NP-complete [21]

detection algorithm based on the modularity measure, which describes the quality of the partition into communities [23], [32]. The problem is proved to be NP-hard [24].

3) k-sized clique: The QUBO formulation for finding k-sized clique was developed in [20]. The problem is to find the complete subgraph of size k from a given graph. The decision problem of whether a k-sized clique exists is NP-complete [21].

4) Graph isomorphism: Graph isomorphism is the problem of determining if there exists a bijective mapping $f\colon V_1 \to V_2$ between the vertex sets of graphs G_1 and G_2 such that whenever $(v_1, v_2) \in E_1$ is an edge in graph G_1, then $(f(v_1), f(v_2)) \in E_2$ is an edge in graph E_2. Interestingly, graph isomorphism is known to be in NP, but it is unknown if it is NP-complete [26]. In practice, it is a complex problem. We have implemented the standard QUBO formulation for graph isomorphism [20], but formulations also exist for adiabatic quantum computers [33] and boson samplers [34].

5) Graph coloring: Given n colors and a graph G, the graph coloring problem is to determine if the n colors can be assigned to the vertices of G so that no edge connects two vertices of the same color. The problem is known to be NP-complete [21]. We implemented the QUBO formulation from [20].

6) Traveling salesman: The traveling salesman problem is the optimization problem where, starting from a given node, the goal is to find a path in a weighted graph that visits every node in the graph exactly once. We implement the formulation from [20]. The decision problem is NP-complete [21].

7) Weighted minimal maximal matching: Minimal maximal matching is a special case of matching on graphs. A matching in graph G is a subset of its edges such that no two edges are adjacent to the same vertex. Matching problems generally are not NP-hard [35], [36] without additional constraints requiring minimality over the selected edges [20]. A maximal matching is such a solution that if any edge that is not yet in the matching is included, the subset of edges would not be a matching anymore. In this work, we consider the problem of finding a maximal matching on a weighted graph with the minimum cost [37]. The algorithm returns a perfect matching or a near-perfect matching when they exist, since these matchings are automatically maximal. This problem was formulated in [20], but we consider it as a special instance of the exact set cover, where we identify the edges on the graph with two-element sets. This way, we can use the exact set cover formula in [20], simplifying the formulation.

8) Vertex cover: The vertex cover problem seeks the smallest set of vertices in a graph such that every edge has at least one endpoint in this set. Our QUBO formulation is based on [20], and QAOA was previously benchmarked on this problem [38]. The decision version of the problem is NP-complete [21].

9) Edge cover: The edge cover is similar to the vertex cover problem, but expressed for edges: what is the smallest set of edges such that every vertex in the graph is adjacent to at least one edge in this set? This problem is no longer NP-hard, as we can utilize the maximum matching algorithm to find a matching that can be greedily extended to form an edge cover. We did not find a standard QUBO formulation for this problem. Hence, we present a new higher-order formulation for it, which is inspired by the formulation for the vertex cover problem [20]. We define $|E|$ many binary variables x_e for each edge $e \in E$. If $x_e = 1$, the corresponding edge e belongs to the covering. The first part of the Hamiltonian becomes

$$H_A = A \sum_{v \in V} \sum_{e \in N(v)} (1 - x_e).$$

The Hamiltonian H_A encodes that we must select at least one edge for each vertex. Hamiltonian H_A is a higher-order polynomial because the neighbor set $N(v)$ can generally contain more than 2 elements. Then, we encode the cost with the standard

$$H_B = B \sum_{e \in E} x_e.$$

As in the case of the vertex cover problem, we require $A > B$. The second way to encode the edge cover problem is to use the inequality constraint methods in [20], but these methods require a logarithmic number of slack variables. In this formulation, we do not need the slack variables, which makes it more scalable in terms of required qubits.

10) MaxFlow: A flow network is a directed graph with source and sink nodes so that each edge has a non-negative capacity. The network does not have self-loops. A flow in this graph is a function $f\colon E \to \mathbb{R}$ that assigns a real value $f(u, v)$ for each edge $(u, v) \in E$ representing the amount of flow. A maximum flow problem seeks to find a feasible flow from the source to the sink through the flow network, obtaining the maximum flow rate. The QUBO formulation for the MaxFlow problem was developed in [28].

11) MinCut: In the same work [28], where a QUBO formulation for the MaxFlow problem was introduced, the authors also developed a QUBO formulation for the MinCut problem. MinCut is complementary to MaxCut, as it involves minimizing the cut rather than maximizing it.

12) MaxCut on hypergraphs: While the other problems are well-known optimization problems, MaxCut on hypergraphs

(HyperMaxCut) has not been previously proposed as a quantum optimization problem. We identified that it is well-suited for the data generation task because it has characteristics similar to MaxCut on graphs, which is one of the most studied optimization problems in quantum computing [31], [39]–[45]. The key difference between MaxCut and HyperMaxCut is that HyperMaxCut creates higher-order binary optimization problems, which means that we have terms with more than just two interacting variables. While these problems naturally map to quantum circuits, state-of-the-art classical solvers, such as Gurobi and CPLEX, do not natively support them.

Next, we define MaxCut for hypergraphs [46]. Let $\mathcal{G} = (V, E)$ be an undirected hypergraph, i.e., simply a set of nodes V and a set of edges $E \subset \mathcal{P}(V)$ where $\mathcal{P}(V)$ is the powerset of V. We obtain the standard graph if we restrict $|e| = 2$ for all $e \in E$. We define MaxCut on hypergraphs, called HyperMaxCut, analogously to MaxCut on graphs by seeking a partition z (i.e., two sets A and B) of the vertex set V in such a way that

$$C(z) = \sum_{i=1}^{|E|} C_i(z),$$

is maximized. In this formulation, $C_i(z) = 1$ if the solution z places at least one vertex in the edge e_i to A and the others to the set B (meaning we cut the edge e_i). Otherwise, $C_i(z) = 0$, which also means that every vertex in edge e_i belongs to either A or B. In the case that $|e| = 2$ for all $e \in E$, this formulation reduces to the standard MaxCut on graphs [31].

While the definition is a MaxCut generalization, there are other ways to formulate MaxCut for hypergraphs [46]. We chose this definition due to the partition of two sets (A and B): Easy to encode in binary and spin systems.

Next, we describe the higher-order binary optimization problem formulation for HyperMaxCut. Let us assume that the vertices admit a natural order indexed by $i \in [|V| - 1]$. We create a set of spin variables $Z = \{z_i \in \{-1, 1\} \mid v \in V\}$, which have the interpretation that $z_i = -1$ if $v_i \in A$ and otherwise $v_i \in B$. Thus, there is a clear correspondence between naturally ordered spin-strings, corresponding variables, and partitions of the hypergraph. Moreover, this correspondence is bijective. We use spin variables since the MaxCut problem on graphs notoriously has a simple formulation in terms of these variables in contrast to binary variables [31].

Let e_{ij} be an edge between nodes i and j. Considering MaxCut on graphs, the idea is to express $C_{e_{ij}}(Z)$ in such a way that the Hamiltonian corresponding $C_{e_{ij}}(Z)$ achieves its minimum at eigenstates $|00\rangle$ and $|11\rangle$ because in these cases $z_i = z_j$ which means that there is no cut between vertices i and j. Thus, maximizing such a Hamiltonian will obtain a partitioning Z, which maximizes cuts between the vertices.

For the standard MaxCut on graphs ($|e_i| = 2$ for all i), the corresponding Hamiltonian for each edge is:

$$C_{e_{ij}}(Z) = -z_i z_j.$$

Every edge set contains two vertices producing exactly $|E|$ many quadratic terms: $z_i z_j$ for every $e_{ij} \in E$ between v_i and v_j. Utilizing the standard rewriting technique between spin and binary variables, we obtain the equivalent QUBO:

$$C_{e_{ij}}(X) = -(1 - 2x_i)(1 - 2x_j) = x_i + x_j - 2x_i x_j.$$

The simple evaluation at points 00, 01, 10 and 11, gives us values 0, 1, 1 and 0, respectively. The previous reasoning generalizes to hypergraphs and defines HyperMaxCut.

For a hyperedge $e \in E$, we want to define a formulation for $C_e(Z)$. Again, the function should achieve its minimum at states $|00\dots0\rangle$ and $|11\dots1\rangle$. Let $n := |e|$ and if n is even, let $m = n$ and otherwise $m = n - 1$. Then, the cost Hamiltonian in terms of spin variables is

$$\begin{aligned} C_e(Z) = &\frac{2^{n-2} - 1}{2^{n-2}} I - \frac{1}{2^{n-2}} \sum_{\{i,j\} \subset e} z_i z_j + \dots \\ &- \frac{1}{2^{n-2}} \sum_{M \subset e, |M| = m} \prod_{i \in M} z_i, \end{aligned}$$

where the sum contains every even-length spin variable combination up to $|e|$. Thus, we obtain a higher-order optimization problem, which always contains terms of even order. This formulation can be used to solve the HyperMaxCut problem.

B. Problem generation for optimization

The optimization algorithms previously described cannot be utilized unless we generate optimization problem instances. The problem instances are graphs, and in many cases, they also include additional problem-specific parameters. For example, the k-sized clique problem requires a value for k. Thus, for each graph algorithm, we have constructed a problem instance generator, which constructs graphs to which the algorithms can be applied. For example, for the community detection algorithm, the generator creates graphs with reasonable communities that can be detected. For the graph isomorphism problem, the generator generates a graph and an automorphism, which is then applied to the first graph to obtain a pair of isomorphic graphs. For k-sized cliques, the generator generates a graph that is ensured to contain a k-sized clique. This method is necessarily a reverse-engineered approach to problem generation.

C. Optimization methods

To solve the optimization problems using the quantum formulations for the graph algorithms, we have employed three standard optimization methods from quantum computing: QAOA [31], VQE [47], and adaptive VQE [48]. In this subsection, we briefly describe the methods and the choices that we have made regarding each algorithm. The code is primarily written in Pennylane [49], but a portion of the code relying on OpenQASM 3.0 transformation is based on Qiskit, as QASM is a standard developed by IBM.

1) QAOA: QAOA is one of the most common quantum optimization algorithms. A QAOA circuit consists of cost and mixer layers that are applied repeatedly. The cost layer is constructed based on the Hamiltonian, whose minimum eigenvalue we are solving. The mixer layer we used in this work is the standard x-mixer, i.e., the layer of parametrized R_x rotations. The number of layers varied between 1 and 4.

2) VQE: The VQE optimization routines require ansatzes [47]. We have implemented the well-researched set of ansatzes from [50] and especially focused on the ansatzes having identifiers 1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 18 which refer to the identifiers in [50]. The final full ansatz circuit results from multiple layers of these ansatzes. The number of layers varied between $1-4$. The different ansatz layouts were not mixed, although that might be a feasible method to extend the data set further.

3) Adaptive VQE: The adaptive VQE algorithm adds gates from a fixed pool adaptively depending on the circuit's gradient. Our implementation relies on the Pennylane implementation of the algorithm [51]. The pool of gates for this implementation consists of a single qubit rotation R_x, R_z, and R_y, as well as the two-qubit gates that are the controlled versions of the single qubit rotations. Hence, the pool has six gate types, which can be positioned anywhere in the circuit with an optimized rotation. The application of the adaptive method started from the uniform superposition. An example of an adaptive circuit is visualized in Figure 1.

Stopping criteria. The optimization was interrupted if the most probable solution from the quantum circuit was among the correct solutions from the exact eigensolver. The probabilities were computed analytically without errors. If the optimization did not converge, the case was classified as unsuccessful and not included in the training data.

D. Data characteristics

The attributes we have collected for supervised fine-tuning are as follows.

- Hamiltonian encoding the optimization problem. Since there is no standard code-level notation for Hamiltonians, we decided to use Pennylane since this notation is also human-readable [52].
- Smallest eigenvalue(s) and the first excited state(s) solved exactly with eigensolvers
- Total number of optimization steps to reach a sufficiently high probability to measure the correct solution
- The states with the highest probabilities
- QAOA, VQE, or adaptive VQE circuits with numeric and symbolic parameters
- For adaptive VQE, the circuits that have been created during the optimization process
- Problem-specific details (e.g., problem graph and additional parameters)
- Number of qubits, layers, and ansatz identifiers

The main output from the data generation process is the QAOA, VQE, and adaptive VQE circuits with optimized parameters. To make this system independent of the underlying Python framework, such as Qiskit, Pennylane, or Cirq, we decided to utilize circuits in the most recent OpenQASM 3.0 format [53].

Figure 3 maps the problems with respect to their qubit counts. We can see that the circuits are sufficiently optimized because, in the vast majority of cases, we have a relatively high probability (y-axis) of measuring the correct bit string. The thickness of the bars indicates the number of problems, and the dashed line is the probability of selecting a bitstring uniformly at random. As a summary, Figure 4 displays the counts for different problems, grouped by the methods used.

III. Agent-Q's Fine-Tuning Pipeline

In recent years, transformer models, introduced in the groundbreaking paper [54], have reshaped the landscapes of natural language processing. The precision and flexibility of large language models based on transformer architecture have enabled breakthroughs in many fields, including code generation [1]–[4], music generation [5], and even image and video generation [6], [7]. These models have been shown to be very capable of learning the relations within large amounts of sequential data, essentially learning patterns and logic from all publicly available data. In addition, scaling the model size has been shown to predictably improve the performance of these models via a phenomenon called emergent abilities [55]. This section will discuss the background of fine-tuning large language models and our approach to building a pipeline for generating quantum circuits. We use the open-source state-of-the-art pre-trained model Qwen 2.5 Instruct [56], trained by the Qwen team at Alibaba Cloud, as a base model that we fine-tune to generate quantum circuits.

A. Transformer

Transformers are a class of neural network architectures that have revolutionized sequence modeling and generation by generating tokens without relying on traditional recurrent structures. Instead of relying on the traditional recurrence networks, they use a self-attention mechanism, allowing each element of the input sequence to interact with each other simultaneously, effectively capturing long-range dependencies [54].

At the core of the transformer architecture, there are two components: the encoder and the decoder. The encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representaions $\boldsymbol{z} = (z_1, \ldots, z_n)$. Given the sequence $\boldsymbol{z}$, the decoder sequentially generates an output sequence $(y_1, \ldots, y_m)$. Throughout the generation process, each step is auto-regressive [57], meaning that all previously generated symbols are additional input when generating the next symbol.

To construct deep models, the encoder and decoder are organized into stacks of N identical layers. These stacks form the *Encoder stack* and the *Decoder stack*. Each layer within these stacks is connected via residual connections, which helps preserve the original signal and ensures stable gradients while enabling the increase in depth of the network, followed by layer normalization to improve convergence [58] [59].

Each encoder block consists of two sub-layers. The first layer implements a multi-head self-attention mechanism, and the second is a fully connected feed-forward neural network. In simple terms, the output of each encoder block is given

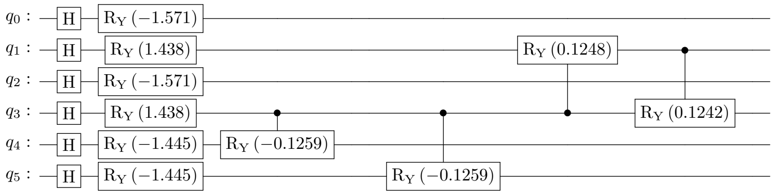


Fig. 1. A short example circuit from the adaptive VQE algorithm

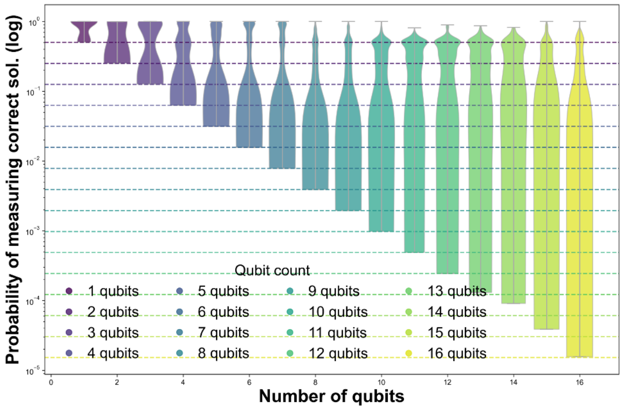


Fig. 2. Distribution of probabilities to measure the correct solution after optimization grouped by the number of qubits. Each dashed line represents the uniform distribution that would be obtained without optimization. Since most circuits are top-of-the-line, we will likely measure the correct solution, indicating that the circuits are of high quality.

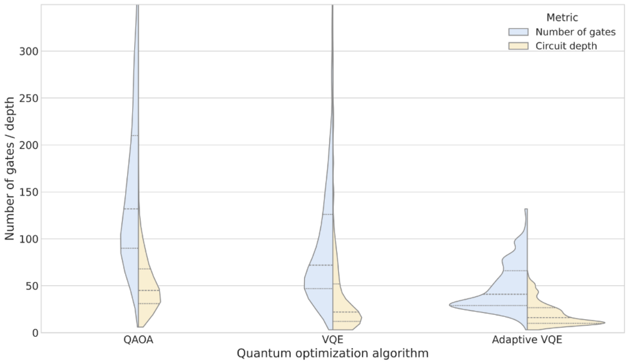


Fig. 3. Distributions for number of gates and circuit depths for the training circuits grouped by the algorithm that has been used to solve the optimization problems

by: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented within the encoder block.

The critical innovation of the transformer architecture is the multi-head attention mechanism. This mechanism effectively enhances the model's ability to capture different aspects of the input by dividing the input into multiple ”heads”. For each head, the mechanism performs the following steps.

Projection: The input sequence is projected linearly to generate *query*, *key* and *value* vectors. Let these vectors be Q, K, and V.

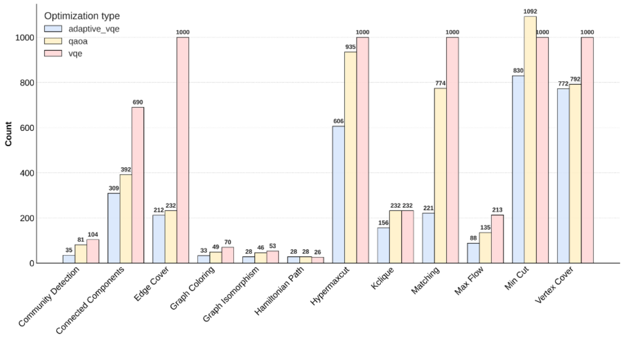


Fig. 4. Number of problems and methods for each optimization problem

Attention Calculations: The calculation for a single head can be written as $\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the dimension of the queries and keys.

Aggregation: The outputs from all heads are concatenated and projected back to the original dimension as

$$\mathrm{MultiHeadAttention(Q, K, V) = Concat(h_1, \ldots, h_p)W^O},$$

where $h_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and p is the amount of heads.

The decoder's architecture is the same as the previously described encoder, but it includes an additional sub-layer that performs multi-head attention over the encoder's output. This extra layer enables the decoder to focus on the most relevant parts of the input sequence when generating the next token.

B. Pre-Training

Large language models are pre-trained on enormous datasets that span trillions of tokens from diverse data sources such as books, academic articles, websites, code repositories, etc. The main objective of pre-training is to guide the model to develop a rich, contextual understanding of language through self-supervised learning.

The training data for pre-training is unlabeled due to the sparse availability of labeled data. There are multiple ways to train models on unlabeled data. Masked Language Modeling, used to train BERT [60], hides parts of the sequence and tasks the model by filling in a masked sequence based on the surrounding unmasked parts. Casual language modeling and next-token prediction, popularized by the GPT models [61], involves training the model to generate the next token in a sequence. The loss of the training sample in these models is calculated as the difference between the generated tokens and the ”true” tokens of the unlabeled training data. Qwen 2.5 Instruct, which is the base model we use, is trained on 18 trillion tokens of diverse training data [56].

Since we utilize the pre-trained model in our work, we will not perform any pre-training steps for quantum circuit generation. However, the benefit we gain from a pre-trained model is that it is already capable of highly complex tasks and few-shot learning [62]. We will further fine-tune this model, which will be discussed in the next section.

C. Supervised Fine-Tuning

While pre-training helps large language models gain a broad understanding of language, code, and reasoning through the enormous amounts of unlabeled data they are trained on, they do not inherently specialize in anything. The pre-trained models necessarily predict the next token in a sequence. Sequence generation is not usually favorable for human communication or more specific use cases like code generation.

This section introduces the concept of fine-tuning the already pre-trained model. Fine-tuning occurs during the subsequent training stage of a large language model, where the model is further trained on a smaller, often task-specific, labeled dataset in order to adapt the generalized knowledge embedded in the model for a more desired output. The notable distinction between fine-tuning and pre-training is that pre-training is predominantly done on unlabeled data, while fine-tuning is done on curated, labeled data that is relevant to the purpose of fine-tuning.

There are several methodologies for fine-tuning large language models. *Supervised Fine-Tuning* (SFT) is typically the first step post pre-training. SFT involves training the model on a dataset of curated input-output pairs, which often are formatted as instructions and their desired responses. This input-output pairing makes the model learn to mimic the style, format, and behavior of these desired responses [63]–[65]. SFT can also be used to introduce new data, essentially making it learn new things in the context of its existing knowledge, to the model, which has been demonstrated well in the context of code generation [1], [3], [66].

D. Supervised Fine-Tuning Pipeline for Circuit Generation

This section presents our fine-tuning pipeline, which makes large language models capable of generating contextually correct quantum circuits. For this purpose, we designed a dedicated SFT pipeline to fine-tune a general-purpose large language model for quantum circuit generation. This pipeline uses a pre-trained foundational model and further trains it on our specifically generated training data from the problems described in section II. Our pipeline uses the open-sourced pre-trained Qwen 2.5 Instruct model [56] as the base model. This model was chosen due to its strong performance in code generation and instruction following tasks. The overall flow of the pipeline is based on a few core elements, visually illustrated in Figure 5.

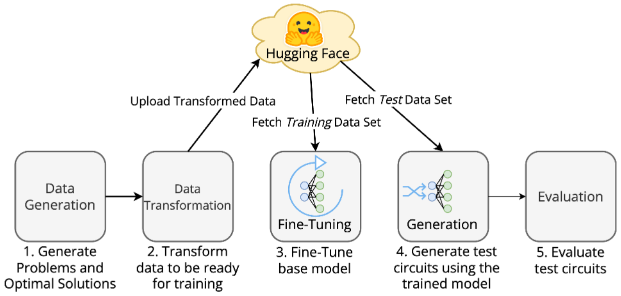

Hugging Face
Upload Transformed Data
Fetch *Training* Data Set
Fetch *Test* Data Set
Data Generation
Data Transformation
Fine-Tuning
Generation
Evaluation
1. Generate Problems and Optimal Solutions
2. Transform data to be ready for training
3. Fine-Tune base model
4. Generate test circuits using the trained model
5. Evaluate test circuits

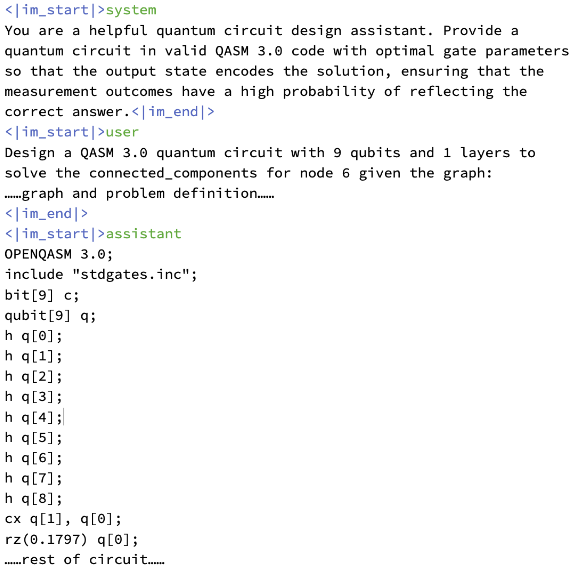
Fig. 5. The end-to-end training pipeline of AGENT-Q

1) Data Generation: The foundation of any training process is a high-quality dataset. We use the dataset generated through the process presented in section II for our specific use case of fine-tuning a model for quantum circuit generation. The optimization of the circuits is computationally heavy, and the data generation was thus conducted in a parallel manner over hundreds of processes. The raw dataset pre-transformation can be found in [9]. One training sample contains all the relevant information necessary to build a training prompt from it and evaluate the generated circuit during evaluation. Our training dataset consists of 13,914 training samples. A train-test split of 96-4 is performed on this dataset. Although the 4% test set is low by traditional ML standards, this split was chosen because it is reserved exclusively for computationally heavy post-training evaluation and is not consulted during the training phase itself.

2) Data Transformation and Formatting: The raw data points must be transformed into a format that is suitable for SFT. This involves constructing a prompt from the relevant attributes that describes both the given problem and the optimized circuit, which serves as the label for the SFT training. In our approach, we adopted a chat-based instruction template that formats the text into a string with special tokens, signaling to the model where the instruction begins and ends. A sample snippet of this string is shown in Figure 6.

Thus, one training sample is made up of one string, structured so that the training framework understands which part of said string is the *prompt*, and which is the *answer* (or label). To further increase the variation in the data, we have five different prompt variants, each permutation having slightly different phrasing and ordering, which are randomly sampled during processing. We have used Huggingface’s *Transformers* APIs [67] to build the training SFT pipeline.

3) Supervised Fine-Tuning: The fine-tuning took 181 minutes using 5 Nvidia H200 GPUs. We use simple fine-tuning hyperparameters, training for 15 epochs with a batch size of 1 per GPU and gradient accumulation steps of 4, resulting in a total of 10,425 optimization steps. We train using BF16 floating-point precision and a learning rate of 2×10^{-5}. The optmizer used is AdamW [68] using $\beta_1 = 0.99$, $\beta_2 = 0.999$



```
<|im_start|>system
You are a helpful quantum circuit design assistant. Provide a
quantum circuit in valid QASM 3.0 code with optimal gate parameters
so that the output state encodes the solution, ensuring that the
measurement outcomes have a high probability of reflecting the
correct answer.<|im_end|>
<|im_start|>user
Design a QASM 3.0 quantum circuit with 9 qubits and 1 layers to
solve the connected_components for node 6 given the graph:
……graph and problem definition……
<|im_end|>
<|im_start|>assistant
OPENQASM 3.0;
include "stdgates.inc";
bit[9] c;
qubit[9] q;
h q[0];
h q[1];
h q[2];
h q[3];
h q[4];
h q[5];
h q[6];
h q[7];
h q[8];
cx q[1], q[0];
rz(0.1797) q[0];
……rest of circuit……
```

Fig. 6. The snippet of one of the instruction prompts used during SFT. The blue strings are the model's special tokens that signal the start and end of sections. The green strings classify the sections.

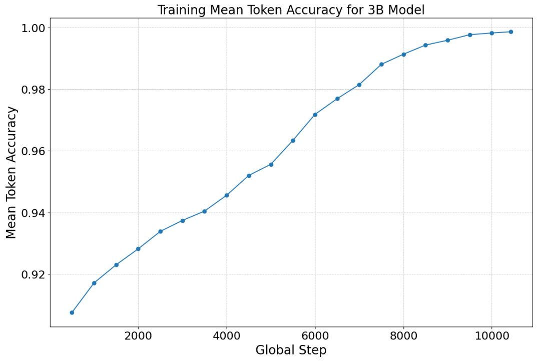


Fig. 7. Training accuracy over global steps during training.

and weight decay of 1×10^{-8}. A plot of the mean token accuracy during training is shown in Figure 7.

4) Test Circuit Generation: Following the SFT training phase, the next step in the pipeline is to evaluate the performance of the model. To evaluate the model's performance, a sample of test circuits needs to be generated based on data that the model has not yet seen. The generated data, discussed in Section III-D1, is divided into a *training* split and a *test* split. The test split consists of 580 unique data points not present in the training data. We sample 200 of these data points and use the fine-tuned model to generate quantum circuits used in evaluation.

IV. Evaluation

We implement a comprehensive evaluation approach to systematically evaluate the performance of the fine-tuned model for quantum circuit generation. We compare our model with other leading open-weight models: Llama 3.2 Instruct 3B [69], DeepSeek R1 Distill 1.5B [70], and Gemma 3 4B [71]. Since we found that the general-purpose large language models struggle to produce any valid quantum circuits, we also include leading code generation models in our comparison: Qwen Coder 3B Instruct [72] and CodeGemma 7B [4]. Furthermore, to fairly assess the capabilities of the previously mentioned models, we also evaluate all models using few-shot learning to give them QASM 3.0 syntax context [62]. Additionally, we compare the probability distributions of the generated circuits with those of the optimized circuits in the training data, as well as the expectation values of both circuits.

TABLE II
Syntactical Correctness

Model	# Correct circuits	Accuracy rate
Agent-Q	**171**	**85.5** %
CodeGemma 7B + Few-Shot	150	75 %
Llama 3.2 3B Instruct + Few-Shot	122	61 %
DeepSeek R1 1.5B Distill 1.5B + Few-Shot	57	28.5 %
Qwen 2.5 3B Instruct + Few-Shot	49	24.5 %
Qwen 2.5 3B Coder Instruct + Few-Shot	43	21.5 %
Qwen 2.5 3B Instruct	0	0 %
Qwen 2.5 3B Coder Instruct	0	0 %
DeepSeek R1 1.5B Distill 1.5B	0	0 %
CodeGemma 7B	0	0 %

Our evaluation is structured around the following metrics:

1) Syntatical Correctness: A fundamental requirement of any model-generating code is the ability to produce syntactically correct code. Over larger outputs, this can be a challenging task for smaller models [73]. In the context of quantum circuit generation, this means ensuring that the generated sequences are valid QASM 3.0 code. We assess the syntactical correctness of each generated circuit by parsing the generated sequences using a QASM 3.0 parser in Qiskit [74]. A quantum circuit is deemed syntactically correct if it parses without errors, indicating that it follows all the grammatical rules of QASM 3.0. The results are displayed in Table II.

2) Expectation Value Analysis: Given a syntactically correct quantum circuit, the natural question is whether the generated circuit represents the problem it was prompted to solve. To answer this, we evaluate the expectation value of the generated quantum circuit with respect to the cost Hamiltonian of the problem. In quantum optimization algorithms, the expectation value of a cost Hamiltonian effectively works as a value of the quality of the solution encoded in the quantum state [31]. To assess the performance of our generated circuits, we use the following values based on expectation values:

a) Generated Expectation Value: Given a syntactically correct circuit, we simulate it using the Qiskit AerSimulator and then compute the expectation value of the problem-specific cost Hamiltonian. Let his value be E_{gen}.

b) Solution Expectation Value: To establish a reference for comparison, we also calculate the expectation value of the optimized circuit from the test data set. Let this value be E_{sol}.

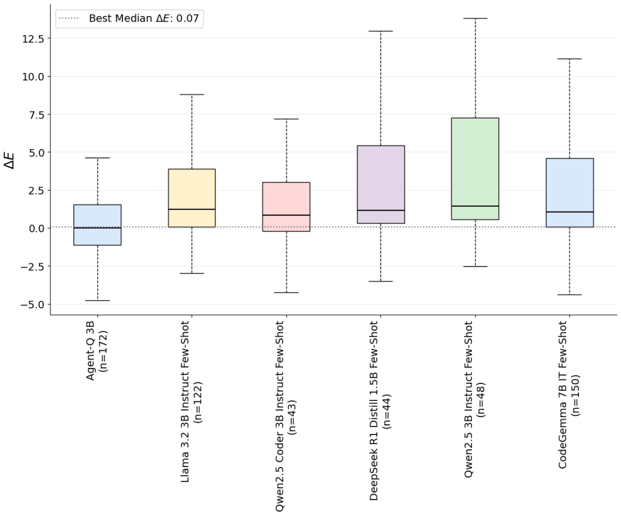


Fig. 8. ΔE for successfully compiled circuits.

c) Expectation Value Difference: To quantify the performance of the generated circuit, we calculate the absolute difference between the generated expectation values and the solution as: $\Delta E = |E_{\text{gen}} - E_{\text{sol}}|$. A smaller ΔE indicates a higher quality circuit.

d) Randomized Expectation Value Baseline: Additionally, to give context to the performance of our generated circuits, a baseline of circuits with random parameters is also introduced. These random circuits have the same structure as the generated circuit, but the parameters are uniformly randomized. By doing this, we can more effectively measure the performance of the LLM-generated parameters. Let the expectation value of this circuit be E_{rand}. We anticipate that well-generated circuit parameters should outperform this randomized baseline, given that the large language model has effectively generalized the problem structure. The Expectation Value Difference ΔE is displayed in Figure 8. The exact values are also shown in Table III.

TABLE III
SUMMARY STATISTICS FOR THE ΔE ACROSS MODELS.

Model Name	Mean ΔE	Median ΔE	Std ΔE
AGENT-Q	**0.53**	**0.07**	8.01
Llama 3.2 3B Instruct Few-Shot	3.72	1.21	8.46
Qwen2.5 3B Coder Few-Shot	5.20	0.85	26.91
DeepSeek R1 Distill 1.5B Few-Shot	5.69	1.16	14.25
Qwen2.5 3B Instruct Few-Shot	6.14	1.44	14.66
CodeGemma 7B IT Few-Shot	6.76	1.04	23.55

3) Relative Entropy of Probability Distributions: Furthermore, we evaluate the performance of the generated circuits by calculating the similarity of the probability distributions of the generated circuits and the optimal solution circuit. The output of a quantum circuit is fundamentally a probability distribution over measurement outcomes. To argue that LLM-generated circuits have favorable characteristics, they should reproduce a probability distribution close to the target distribution of the optimal solution.

To quantify this, we calculate *relative entropy*, also known as the *Kullback-Leibler divergence* (KL)

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right),$$

where P and Q are two probability distributions such that the support of Q is a subset of the support of P. Relative entropy can be viewed as a measure of the distance between two probability distributions. Given a generated circuit's probability P_{gen} and the optimal solution circuit's probability P_{sol}, the relative entropy is $D_{KL}(P_{\text{sol}} \parallel P_{\text{gen}})$. Lower relative entropy indicates higher similarity between the two distributions.

In our evaluation, we calculate the relative entropy between the probability distributions obtained from simulating the generated circuit P_{gen}, the optimal solution circuit P_{sol}, and the baseline circuit with randomized parameters P_{rand} (same as in Section IV-2). The results are displayed in Table IV.

TABLE IV
AVERAGE RELATIVE ENTROPY (D_{KL}) VALUES OF THE GENERATED PARAMETERS COMPARED TO RANDOM PARAMETERS.

Metric	Value
Average $D_{KL}(P_{\text{sol}} \parallel P_{\text{gen}})$	6.781
Average $D_{KL}(P_{\text{sol}} \parallel P_{\text{rand}})$	9.623
Improvement over random	**29.5 %**

V. DISCUSSION

The results show that the circuits' syntax can be learned efficiently, and the few-shot learning made the models comparable at the syntax level. OpenQASM syntax is relatively simple compared to natural language and programming languages. The recently published Google Gemma 7B performed the best with a few-shot tuning. Note that this model has 7B parameters, whereas we used the smallest Qwen model with 3B parameters. These results are consistent with the observation that the competitor models lack the relevant quantum knowledge to perform well.

The fact that we focused on optimization problems provides us with the theoretical foundation to use the expectation values and relative entropy as evaluation metrics. Well-defined optimization problems have a limited set of optimal solutions that can be used in evaluation. Regarding the results from these two metrics, we note that our fine-tuned model outperformed all other state-of-the-art models. More importantly, we also found that the initial parameter values produced by the LLM produce lower expectation values than random guesses, and the corresponding distributions are closer to those measured from the optimized circuits. We view this finding as evidence that these circuits might be more efficient to optimize from the initial point given by the LLM model. This requires further experimental evaluation since optimization is dependent on multiple aspects. Nevertheless, the fine-tuned LLM model learned specific structures from the circuits and their parameters, and it is interesting to study further what these structures are.

We have identified multiple promising points to improve the model. First, the current implementation can be extended with reasoning based on the optimization process that the adaptive VQE method creates. Every step in adaptive VQE creates a circuit containing one more gate with a parameter to minimize the gradient. This leads to a sequence of circuits that could work as training data for reasoning models.

Considering the circuit structure, using LLMs for quantum circuit compilation seems promising. This could be approached by the idea of ”translating circuits” as natural language is translated. A logical circuit would correspond to the text in the source language, and the target text would be the compiled circuit, taking into account the fixed hardware topology.

Secondly, we are working on extending the model with reinforcement learning (RL). Recently, impressive improvements in model performance for complex problems have been presented through the use of RL algorithms for fine-tuning, such as Group Relative Policy Optimization (GRPO) [75]. We have also built an addition to the pipeline that implements GRPO for the quantum circuit generation process. Due to its exploratory nature and the potential to define effective reward functions based on quantum simulation results (e.g., expectation value, circuit depth), GRPO has significant potential in both improving circuit parameters and developing novel circuit architectures. GRPO-based circuits could potentially be tuned to compress circuit size, reducing both the size and depth of the circuits while maintaining expressivity. This will be left for further research.

Furthermore, methods for model explainability are another direction of potential research. Understanding why the model selects specific circuit sequences could reveal insights into its learned heuristics and logic, potentially leading to a deeper understanding of effective circuit construction.

Additionally, one consideration is the model's generalizability to various types of optimization problems. Although the training data consists of a diverse set of 12 optimization problems, it has not yet been established how well the model performs on different optimization problems. The current approach relies on natural language descriptions of the problem. It could be possible to improve the generalization by encoding the problem more directly through its mathematical structure, for instance, by the cost Hamiltonian description. We also believe that the extensive publicly available training dataset of over 14,000 circuits will be useful for various tasks beyond fine-tuning LLM models.

VI. Conclusion and future work

A. Conclusion

In this study, we presented a fine-tuning approach based on Supervised Fine-Tuning (SFT) for large language models specifically tailored for quantum optimization tasks, demonstrating their potential to effectively generate parametrized quantum circuits and suitable initial parameters. Our fine-tuned model significantly outperformed baseline state-of-the-art language models, achieving high syntactical correctness and generating initial parameters closer to optimal values than random initialization. Furthermore, the generated circuits exhibited probability distributions that were considerably closer to optimal solutions as measured by relative entropy.

The significance of this work lies in its practical applications in quantum computing. Our model can assist quantum algorithm developers by providing strong starting points for optimization routines, thus accelerating quantum algorithm development and potentially enhancing the efficiency of quantum computation processes. Furthermore, the produced circuits can serve as benchmarks for quantum compilers and quantum hardware evaluations, marking a step toward more sophisticated hybrid quantum-classical programming frameworks.

B. Limitation

Our current model's generalization capability to entirely unseen quantum optimization problems has not been fully assessed. The diversity and complexity of the optimization problems used for training, although extensive, may still restrict the model's performance on vastly different quantum computational tasks. Additionally, the complexity of quantum computations inherently limits current simulation-based evaluation methods to relatively small quantum systems; we still need to explore more efficient methods for large-scale quantum circuit verifications. Furthermore, because the training and evaluation sets draw on the same problem classes, the strong metrics reported may partially stem from overfitting to recurring structures rather than from task-agnostic learning.

C. Future Work

Future research directions, as discussed in detail in the preceding section, include integrating reinforcement learning approaches such as Group Relative Policy Optimization (GRPO) into the fine-tuning process. These methods could enhance parameter selection and lead to novel circuit structures optimized for quantum hardware constraints. Future research could focus on using explainability tools to better understand model decisions, integrating mathematical concepts like Hamiltonian encodings into training prompts, and expanding into other areas of quantum computing.

VII. Acknowledgement

This work is funded by Business Finland (grant number 169/31/2024), Research Council of Finland (grant number 362729) and the Quantum Doctoral Programme. We also acknowledge the computational resources provided by the Aalto Science-IT project.

References

[1] DeepSeek-AI, Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma, W. Zeng, X. Bi, Z. Gu, H. Xu, D. Dai, K. Dong, L. Zhang, Y. Piao, Z. Gou, Z. Xie, Z. Hao, B. Wang, J. Song, D. Chen, X. Xie, K. Guan, Y. You, A. Liu, Q. Du, W. Gao, X. Lu, Q. Chen, Y. Wang, C. Deng, J. Li, C. Zhao, C. Ruan, F. Luo, and W. Liang, “DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source

Models in Code Intelligence," *arXiv preprint arXiv:2406.11931*, no. arXiv:2406.11931, Jun. 2024.

[2] S. Huang, T. Cheng, J. K. Liu, J. Hao, L. Song, Y. Xu, J. Yang, J. H. Liu, C. Zhang, L. Chai, R. Yuan, Z. Zhang, J. Fu, Q. Liu, G. Zhang, Z. Wang, Y. Qi, Y. Xu, and W. Chu, "OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models," *arXiv preprint arXiv.2411.04905*, no. arXiv:2411.04905, Nov. 2024.

[3] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code Llama: Open Foundation Models for Code," *CoRR*, vol. abs/2308.12950, 2023.

[4] C. Team, H. Zhao, J. Hui, J. Howland, N. Nguyen, S. Zuo, A. Hu, C. A. Choquette-Choo, J. Shen, J. Kelley, K. Bansal, L. Vilnis, M. Wirth, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, Z. Gong, J. Fine, T. Warkentin, A. J. Hartman, B. Ni, K. Korevec, K. Schaefer, and S. Huffman, "Codegemma: Open code models based on gemma," *arXiv preprint arXiv:2406.11409*, 2024. [Online]. Available: https://arxiv.org/abs/2406.11409

[5] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, M. Sharifi, N. Zeghidour, and C. Frank, "Musiclm: Generating music from text," *arXiv preprint arXiv:2301.11325*, 2023. [Online]. Available: https://arxiv.org/abs/2301.11325

[6] J. Y. Koh, D. Fried, and R. Salakhutdinov, "Generating images with multimodal language models," *arXiv preprint arXiv:2305.17216*, 2023. [Online]. Available: https://arxiv.org/abs/2305.17216

[7] Z. Yang, J. Teng, W. Zheng, M. Ding, S. Huang, J. Xu, Y. Yang, W. Hong, X. Zhang, G. Feng, D. Yin, Y. Zhang, W. Wang, Y. Cheng, B. Xu, X. Gu, Y. Dong, and J. Tang, "Cogvideox: Text-to-video diffusion models with an expert transformer," *arXiv preprint arXiv:2408.06072*, 2025. [Online]. Available: https://arxiv.org/abs/2408.06072

[8] L. Jern, "Quantum code generation repository," https://github.com/LinuzJ/quantum-code-generation, 2025, accessed: 2025-07-11.

[9] ——, "Graph data quantum," https://huggingface.co/datasets/linuzj/graph-data-quantum, 2025, accessed: 2025-07-11.

[10] N. Dupuis, L. Buratti, S. Vishwakarma, A. V. Forrat, D. Kremer, I. Faro, R. Puri, and J. Cruz-Benito, "Qiskit code assistant: Training llms for generating quantum computing code," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–4.

[11] IBM Quantum, "Qiskit code assistant," 2025, accessed: 2025-07-11. [Online]. Available: https://quantum.cloud.ibm.com/docs/en/guides/qiskit-code-assistant

[12] S. Vishwakarma, F. Harkins, S. Golecha, V. S. Bajpe, N. Dupuis, L. Buratti, D. Kremer, I. Faro, R. Puri, and J. Cruz-Benito, "Qiskit humaneval: An evaluation benchmark for quantum code generative models," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. IEEE, 2024, pp. 1169–1176.

[13] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. d. Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "OpenQASM 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, pp. 1–50, Sep. 2022, arXiv:2104.14722 [quant-ph]. [Online]. Available: http://arxiv.org/abs/2104.14722

[14] S. Arlt, H. Duan, F. Li, S. M. Xie, Y. Wu, and M. Krenn, "Meta-designing quantum experiments with language models," *arXiv preprint arXiv:2406.02470*, 2024. [Online]. Available: https://arxiv.org/abs/2406.02470

[15] D. Fitzek, Y. H. Teoh, H. P. Fung, G. A. Dagnew, E. Merali, M. S. Moss, B. MacLellan, and R. G. Melko, "Rydberggpt," *arXiv preprint arXiv:2405.21052*, may 2024. [Online]. Available: https://arxiv.org/abs/2405.21052

[16] K. Nakaji, L. B. Kristensen, J. A. Campos-Gonzalez-Angulo, M. G. Vakili, H. Huang, M. Bagherimehrab, C. Gorgulla, F. Wong, A. McCaskey, J.-S. Kim, T. Nguyen, P. Rao, and A. Aspuru-Guzik, "The generative quantum eigensolver (gqe) and its application for ground state search," *arXiv preprint arXiv:2401.09253*, jan 2024. [Online]. Available: https://arxiv.org/abs/2401.09253

[17] B. Apak, M. Bandic, A. Sarkar, and S. Feld, "Ketgpt – dataset augmentation of quantum circuits using transformers," in *Computational Science – ICCS 2024*, L. Franco, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds. Cham: Springer Nature Switzerland, 2024, pp. 235–251.

[18] A. Li, S. Stein, S. Krishnamoorthy, and J. Ang, "Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation," *ACM Transactions on Quantum Computing*, vol. 4, no. 2, Feb. 2023. [Online]. Available: https://doi.org/10.1145/3550488

[19] N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto, "s1: Simple test-time scaling," *arXiv preprint arXiv:2501.19393*, mar 2025. [Online]. Available: https://arxiv.org/abs/2501.19393

[20] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, Feb. 2014. [Online]. Available: https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2014.00005/full

[21] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, p. 85–103. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_9

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[23] C. F. A. Negre, H. Ushijima-Mwesigwa, and S. M. Mniszewski, "Detecting multiple communities using quantum annealing on the d-wave system," *PLOS ONE*, vol. 15, no. 2, pp. 1–14, 02 2020. [Online]. Available: https://doi.org/10.1371/journal.pone.0227538

[24] S. Fortunato and D. Hric, "Community detection in networks: A user guide," *Physics Reports*, vol. 659, p. 1–44, Nov. 2016, arXiv:1608.00163 [physics].

[25] L. Babai, "Graph isomorphism in quasipolynomial time [extended abstract]," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, D. Wichs and Y. Mansour, Eds. ACM, 2016, pp. 684–697. [Online]. Available: https://doi.org/10.1145/2897518.2897542

[26] D. S. Johnson, "The np-completeness column: An ongoing guide," *Journal of Algorithms*, vol. 8, no. 3, p. 438–448, Sep. 1987.

[27] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[28] T. Krauss, J. McCollum, C. Pendery, S. Litwin, and A. J. Michaels, "Solving the max-flow problem on a quantum annealing computer," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–10, 2020.

[29] J. B. Orlin, "Max flows in o(nm) time, or better," in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, D. Boneh, T. Roughgarden, and J. Feigenbaum, Eds. ACM, 2013, pp. 765–774. [Online]. Available: https://doi.org/10.1145/2488608.2488705

[30] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k," *Math. Oper. Res.*, vol. 19, no. 1, pp. 24–37, 1994. [Online]. Available: https://doi.org/10.1287/moor.19.1.24

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

[32] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 066111, Dec. 2004.

[33] F. Gaitan and L. Clark, "Graph isomorphism and adiabatic quantum computing," *Phys. Rev. A*, vol. 89, p. 022342, Feb 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.89.022342

[34] K. Brádler, S. Friedland, J. Izaac, N. Killoran, and D. Su, "Graph isomorphism and gaussian boson sampling," *Special Matrices*, vol. 9, no. 1, pp. 166–196, 2021. [Online]. Available: https://doi.org/10.1515/spma-2020-0132

[35] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of Mathematics*, vol. 17, p. 449–467, Jan. 1965.

[36] ——, "Maximum matching and a polyhedron with 0,1-vertices," *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, vol. 69B, no. 1 and 2, p. 125, Jan. 1965.

[37] W. Cook and A. Rohe, "Computing minimum-weight perfect matchings," *INFORMS Journal on Computing*, vol. 11, no. 2, p. 138–148, May 1999.

[38] J. Cook, S. Eidenbenz, and A. Bärtschi, "The Quantum Alternating Operator Ansatz on Maximum k-Vertex Cover," in *2020 IEEE International Conference on Quantum Computing and Engineering*, 10 2019.

[39] J. Basso, E. Farhi, K. Marwaha, B. Villalonga, and L. Zhou, "The quantum approximate optimization algorithm at high depth for maxcut on large-girth regular graphs and the sherrington-kirkpatrick model," *LIPIcs, Volume 232, TQC 2022*, vol. 232, pp. 7:1–7:21, 2022, arXiv:2110.14206 [quant-ph].

[40] G. G. Guerreschi and A. Y. Matsuura, "Qaoa for max-cut requires hundreds of qubits for quantum speed-up," *Scientific Reports*, vol. 9, no. 1, p. 6903, May 2019.

[41] R. Herrman, L. Treffert, J. Ostrowski, P. C. Lotshaw, T. S. Humble, and G. Siopsis, "Impact of graph structures for qaoa on maxcut," *Quantum Information Processing*, vol. 20, no. 9, p. 289, Sep. 2021.

[42] R. Majumdar, D. Madan, D. Bhoumik, D. Vinayagamurthy, S. Raghunathan, and S. Sur-Kolay, "Optimizing ansatz design in qaoa for max-cut," *arXiv preprint arXiv:2106.02812*, jun 2021. [Online]. Available: https://arxiv.org/abs/2106.02812

[43] D. Rehfeldt, T. Koch, and Y. Shinano, "Faster exact solution of sparse maxcut and qubo problems," *Mathematical Programming Computation*, vol. 15, no. 3, p. 445–470, Sep. 2023.

[44] Z. Wang, S. Hadfield, Z. Jiang, and E. G. Rieffel, "Quantum approximate optimization algorithm for maxcut: A fermionic view," *Physical Review A*, vol. 97, no. 2, p. 022304, Feb. 2018.

[45] Z. Zhou, Y. Du, X. Tian, and D. Tao, "Qaoa-in-qaoa: Solving large-scale maxcut problems on small quantum machines," *Physical Review Applied*, vol. 19, no. 2, p. 024027, Feb. 2023.

[46] D. Conlon, J. Fox, M. Kwan, and B. Sudakov, "Hypergraph cuts above the average," *Israel Journal of Mathematics*, vol. 233, no. 1, p. 67–111, Aug. 2019.

[47] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, p. 4213, Jul. 2014.

[48] H. R. Grimsley, S. E. Economou, E. Barnes, and N. J. Mayhall, "An adaptive variational algorithm for exact molecular simulations on a quantum computer," *Nature Communications*, vol. 10, no. 1, p. 3007, Jul. 2019.

[49] V. Bergholm and et al., "Pennylane: Automatic differentiation of hybrid quantum-classical computations," 2022. [Online]. Available: https://arxiv.org/abs/1811.04968

[50] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.

[51] Xanadu, *PennyLane AdaptiveOptimizer*, 2025, accessed: 2025-03-28. [Online]. Available: https://docs.pennylane.ai/en/stable/code/api/pennylane.AdaptiveOptimizer.html

[52] ——, *PennyLane LinearCombination*, 2025, accessed: 2025-03-28. [Online]. Available: https://docs.pennylane.ai/en/stable/code/api/pennylane.ops.op_math.LinearCombination.html

[53] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. d. Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "Openqasm 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, p. 1–50, Sep. 2022, arXiv:2104.14722 [quant-ph].

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[55] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022. [Online]. Available: https://arxiv.org/abs/2206.07682

[56] Qwen, A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Tang, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2025. [Online]. Available: https://arxiv.org/abs/2412.15115

[57] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2014. [Online]. Available: https://arxiv.org/abs/1308.0850

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[59] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. [Online]. Available: https://arxiv.org/abs/1607.06450

[60] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[61] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," https://www.mikecaptain.com/resources/pdf/GPT-1.pdf, 2018, openAI Technical Report.

[62] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[63] A. Chowdhery, S. Narang, and et al., "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022. [Online]. Available: https://arxiv.org/abs/2204.02311

[64] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[65] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022. [Online]. Available: https://arxiv.org/abs/2203.02155

[66] N. Dupuis, L. Buratti, S. Vishwakarma, A. V. Forrat, D. Kremer, I. Faro, R. Puri, and J. Cruz-Benito, "Qiskit code assistant: Training llms for generating quantum computing code," *arXiv preprint arXiv:2405.19495*, 2024. [Online]. Available: https://arxiv.org/abs/2405.19495

[67] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2020. [Online]. Available: https://arxiv.org/abs/1910.03771

[68] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2019. [Online]. Available: https://arxiv.org/abs/1711.05101

[69] A. Grattafiori, A. Dubey, and et al., "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[70] DeepSeek-AI, D. Guo, and et al., "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[71] G. Team, A. Kamath, and et al., "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025. [Online]. Available: https://arxiv.org/abs/2503.19786

[72] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, K. Dang, Y. Fan, Y. Zhang, A. Yang, R. Men, F. Huang, B. Zheng, Y. Miao, S. Quan, Y. Feng, X. Ren, X. Ren, J. Zhou, and J. Lin, "Qwen2.5-coder technical report," *arXiv preprint arXiv:2409.12186*, 2024. [Online]. Available: https://arxiv.org/abs/2409.12186

[73] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," 2023. [Online]. Available: https://arxiv.org/abs/2307.03172

[74] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," *arXiv preprint arXiv:2405.08810*, 2024. [Online]. Available: https://arxiv.org/abs/2405.08810

[75] Z. DeepSeek AI, Shao and et al., "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models," Apr. 2024.