QCircuitBench: A Large-Scale Dataset for Benchmarking Quantum Algorithm Design

Anonymous Author(s)
Affiliation
Address
email

Abstract

Quantum computing is an emerging field recognized for the significant speedup it offers over classical computing through quantum algorithms. However, designing and implementing quantum algorithms pose challenges due to the complex nature of quantum mechanics and the necessity for precise control over quantum states. Despite the significant advancements in AI, there has been a lack of datasets specifically tailored for this purpose. In this work, we introduce QCircuitBench, the first benchmark dataset designed to evaluate AI's capability in designing and implementing quantum algorithms in the form of quantum circuit codes. Unlike using AI for writing traditional codes, this task is fundamentally more complicated due to highly flexible design space. Our key contributions include:

1. A general framework which formulates the key features of quantum algorithm design task for Large Language Models.
2. Implementation for quantum algorithms from basic primitives to advanced applications, spanning 3 task suites, 23 algorithms, and 128,573 data points.
3. Automatic validation and verification functions, allowing for iterative and interactive evaluation without human inspection.
4. Promising potential as a training dataset through primitive fine-tuning results.

We observed several interesting experimental phenomena: fine-tuning does not always outperform few-shot learning, and LLMs tend to exhibit consistent error patterns. In all, QCircuitBench is a comprehensive benchmark for AI-driven quantum algorithm design, while it also reveals limitations of LLMs in this domain.

1 Introduction

Quantum computing is an emerging field in recent decades because algorithms on quantum computers may solve problems significantly faster than their classical counterparts. From the perspective of theoretical computer science, the design of quantum algorithms have been investigated in various research directions - see the survey [Dalzell et al., 2023] and the quantum algorithm zoo [Jordan, 2025]. However, the design of quantum algorithms on quantum computers has been completed manually by researchers. This process is notably challenging due to highly flexible design space and extreme demands for a comprehensive understanding of mathematical tools and quantum properties.

For these reasons, quantum computing is often considered to have high professional barriers. As the discipline evolves, we aim to explore more possibilities for algorithm design and implementation in the quantum setting. This is aligned with recent advances among AI for Science, including AlphaFold [Jumper et al., 2021], AlphaGeometry [Trinh et al., 2024], etc. Recently, large language models (LLMs) have also become widely applicable among AI for science approaches [Yang et al., 2024b, Zhang et al., 2024, Yu et al., 2024]. LLMs represent the best practice of sequential modeling

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

methods at current stage. They have an edge over other models in possessing abundant pre-training knowledge and providing human-friendly interfaces which support human-machine collaboration. Therefore, we gear LLMs for quantum algorithm design.

As far as we know, there has not been any dataset for AI in quantum algorithm design. Existing work combining quantum computing and AI mostly targets at exploiting quantum computing for AI; there are some papers applying AI for quantum computing, but they either consider niche problems [Nakayama et al., 2023, Schatzki et al., 2021] or limited functions [Tang et al., 2023, Fürrutter et al., 2024], not quantum algorithm datasets of general interest (see Section 2). However, unlike classical code generation where abundant data exist, the most challenging aspect for quantum algorithm design is the lack of sufficient data, and hence the difficulty of generalization in training AI models. Therefore, datasets for quantum algorithm design are solicited.

Descriptions of quantum algorithms in natural language could be verbose and vague. Mathematical formulas, while precise and succinct, are difficult to verify automatically. To accommodate with LLMs, we make a change of perspective by formulating quantum algorithms as programming languages. This allows for precise representation of a quantum algorithm, enables automatic verification procedure, and bridges the gap between theoretical design and circuit implementations. Furthermore, meaningful quantum algorithms which can be efficiently implemented have no more than polynomially many gates [Poulin et al., 2011], and thus such formulations have the theoretical benefits allowing for scalable representations.

Key Contributions. We propose QCircuitBench, the first comprehensive, structured dataset for quantum algorithm design. Technically, it has the following key contributions:

- It formulates the task for Large Language Models (LLMs) with a carefully designed framework encompassing the key features of quantum algorithm design, including problem description, quantum circuit codes, classical post-processing, and verification functions. It maintains the black-box nature of oracles and characterizes query complexity properly.
- It implements a wide range of quantum algorithms, covering 3 task suites, 23 algorithms, and 128,573 data points. The dataset spans from basic primitives and textbook-level algorithms to advanced applications such as Generalized Simon's Problem, demonstrating compatibility with complex algorithms and easy extensibility.
- It has automatic validation and verification functions, enabling iterative, human-free evaluation and supporting interactive reasoning to enhance performance.
- It showcases the potential as a training dataset through primitive fine-tuning results. As we expand the dataset to include more algorithms and explore novel fine-tuning methods, it will hopefully contribute to interactive quantum algorithm design and implementation significantly.

2 Related Work

Quantum Machine Learning. To the best of our knowledge, QCircuitBench is the first dataset tailored specifically for quantum algorithm design. Previous efforts combining quantum computing with AI primarily fall under the category of Quantum Machine Learning (QML), which aims at leveraging the unique properties of quantum systems to enhance machine learning algorithms and achieve improvements over their classical counterparts [Schuld et al., 2015, Biamonte et al., 2017, Ciliberto et al., 2018]. Corresponding datasets often focus on encoding classical data into quantum states. For instance, MNISQ [Placidi et al., 2023] is a dataset of quantum circuits representing the original MNIST dataset [LeCun et al., 1998] generated by the AQCE algorithm [Shirakawa et al., 2021]. Another category of datasets focuses on collecting quantum data to demonstrate quantum advantages since classical machine learning methods can fail to characterize particular patterns of quantum data. Nakayama et al. [2023] created a VQE-generated quantum circuit dataset for classification of variational ansatzes. NTangled [Schatzki et al., 2021] further investigated different types of entanglement and composed quantum states with various multipartite entanglement for classification. While these datasets successfully demonstrate quantum supremacy, the practical applications of the problem addressed are unclear.

AI for Quantum Computing. This research direction explores the possibility of leveraging AI to facilitate the advancement of quantum computing. QDataSet [Perrier et al., 2022] collects data from

simulations of one- and two-qubit systems and targets training classical machine learning algorithms for quantum control, quantum tomography, and noise mitigation. LLM4QPE [Tang et al., 2023] is a large language model style paradigm for predicting quantum system properties with pre-training and fine-tuning workflows. While the paradigm is interesting, the empirical experiments are limited to two downstream tasks: quantum phase classification and correlation prediction. Fürrutter et al. [2024] studied the application of diffusion models [Sohl-Dickstein et al., 2015, Rombach et al., 2022] to quantum circuit synthesis [Saeedi and Markov, 2013, J. et al., 2022]. Scalability issues must be addressed to achieve practical and meaningful unitary compilation through this methodology.

Quantum Circuit Benchmarks. The aforementioned works represent meaningful explorations at the intersection of AI and quantum computing. However, none of them considers the task which interests the quantum computing community (from the theoretical side) the most: quantum algorithm design. Our work aims to take the first step in bridging this gap. It is worth noting that several quantum algorithm circuit benchmarks already exist, such as QASMBench [Li et al., 2023], MQTBench [Quetschlich et al., 2023], and VeriQBench [Chen et al., 2022]. However, these benchmarks are designed to evaluate the performance of NISQ (Noisy Intermediate-Scale Quantum) [Preskill, 2018] machines or quantum software tools, rather than for training and evaluating AI models. For instance, QASMBench includes a diverse variety of quantum circuits based on OpenQASM representation [Cross et al., 2022], covering quantum circuits with qubit sizes ranging from 2 to 127. However, it fails as a dataset for AI in that it includes only a few entries for each algorithm and ignores the post-processing procedure and construction of different oracles, which are crucial to quantum algorithm design. Similar limitations apply to MQTBench and VeriQBench.

3 QCircuitBench Dataset

3.1 Task Suite

For the general purpose of quantum algorithm design, we consider three categories of tasks: oracle construction, algorithm design, and random circuit synthesis. These tasks are crucial for devising and implementing quantum algorithms, with oracle construction serving as the premise for algorithm design, and random circuits serving as a main demonstration for quantum supremacy. These task suites encompass 23 algorithms and a total of 128,573 data points with the following distribution:

3.1.1 Task I: Oracle Construction

This task suite contains 32,249 data points in total, focused on two types of oracle constructions.

To study a Boolean function $f\colon \{0,1\}^n \to \{0,1\}^m$, we need to gain its access. In quantum computing, the function f is encoded as an oracle U_f such that for any $x \in \{0,1\}^n$, $z \in \{0,1\}^m$, $U_f|x\rangle|z\rangle = |x\rangle|z \oplus f(x)\rangle$, where $\oplus$ is the plus modulo 2. The construction of U_f using quantum gates is deeply rooted in reversible quantum logic synthesis, which remains a challenge for complex Boolean functions. In this dataset, we mainly focus on the construction of textbook-level oracles: Bernstein-Vazirani Problem [Bernstein and Vazirani, 1993], Deutsch-Jozsa Problem [Deutsch and Jozsa, 1992], Simon's Problem [Simon, 1997], and Grover's algorithm for unstructured search [Grover, 1996] (including constructions of both the oracle and the diffusion operator).

There is another category of more flexible oracle construction tasks which we refer to as "Problem Encoding". For example, one can apply Grover's oracle to solving constraint problems such as SAT and triangle finding [Ambainis, 2004]. Formulating problem encoding tasks for LLMs slightly differs from quantum logic synthesis, and we refer the readers to Appendix A.2 for more detailed discussion.

3.1.2 Task II: Quantum Algorithm Design

In this category, we cover a wide range of quantum algorithms with varying complexity, from fundamental primitives to *advanced applications*, covering 5,464 data points:

- Textbook-level algorithms: These range from the Bernstein-Vazirani problem [Bernstein and Vazirani, 1993], Deutsch-Jozsa problem [Deutsch and Jozsa, 1992], Simon's problem [Simon, 1997], Grover's algorithm [Grover, 1996], phase estimation [Kitaev, 1995], quantum Fourier transform [Coppersmith, 2002], GHZ state preparation [Greenberger et al., 2007], W state

preparation [Dür et al., 2000], random number generator [Herrero-Collantes and Garcia-Escartin, 2017], swap test [Barenco et al., 1997, Buhrman et al., 2001] to Shor's algorithm [Shor, 1999] for factorization, one of the most famous quantum algorithms with superpolynomial speedup.

- Generalized Simon's Problem [Ye et al., 2021]: This is a more advanced version of the standard Simon's problem and an active area of research in recent years [Ye et al., 2021, Wu et al., 2022]. The setting is formally stated as follows: given an (unknown) function $f\colon \mathbb{Z}_p^n \to X$ where X is a finite set and a k is a positive integer satisfying $k < n$, it is guaranteed that there exists a subgroup $S \leq \mathbb{Z}_p^n$ of rank k such that for any $x, y \in \mathbb{Z}_p^n$, $f(x) = f(y)$ iff $x - y \in S$. The goal is to find S. Intuitively, the generalized Simon's problem extends the standard Simon's problem from binary to p-ary bases and from a single secret string to a subgroup of rank k.
- Variational quantum algorithms (VQAs): Beyond universal quantum algorithms, VQAs including VQE [Peruzzo et al., 2014] for finding the ground-state energy of a given Hamiltonian and QAOA [Farhi et al., 2014] for solving the maximum cut problem for a given graph are potentially implementable on near-term quantum computers. Unlike traditional quantum algorithms with fixed quantum circuits, VQAs rely on iterative optimization of parameterized quantum circuits, introducing unique challenges as models must generate not only quantum circuits but also suitable parameter initialization and optimization methods.
- Quantum information protocols: Additionally, we also include quantum information protocols such as quantum teleportation [Bennett et al., 1993] and quantum key distribution [Bennett and Brassard, 2014], which have wide applications in quantum communications, quantum cryptography, etc. See Appendix B for further details.

3.1.3 Task III: Random Circuit Synthesis

The third task we consider is random circuit synthesis, containing 90,860 data points. On the one hand, random circuit sampling is the first algorithm for showing quantum supremacy by Google [Arute et al., 2019], and is still widely applied to demonstrate the power of quantum algorithms in recent research [Wu et al., 2021, Bluvstein et al., 2024, DeCross et al., 2024]. In this suite, circuits are randomly sampled from a Clifford gate set {H, S, CNOT} and a universal set {H, S, T, CNOT}, and the task is to generate circuits reproducing the specified quantum state.

3.2 Dataset Structure

The overall structure of QCircuitBench is illustrated as follows (more details are given in Appendix A):

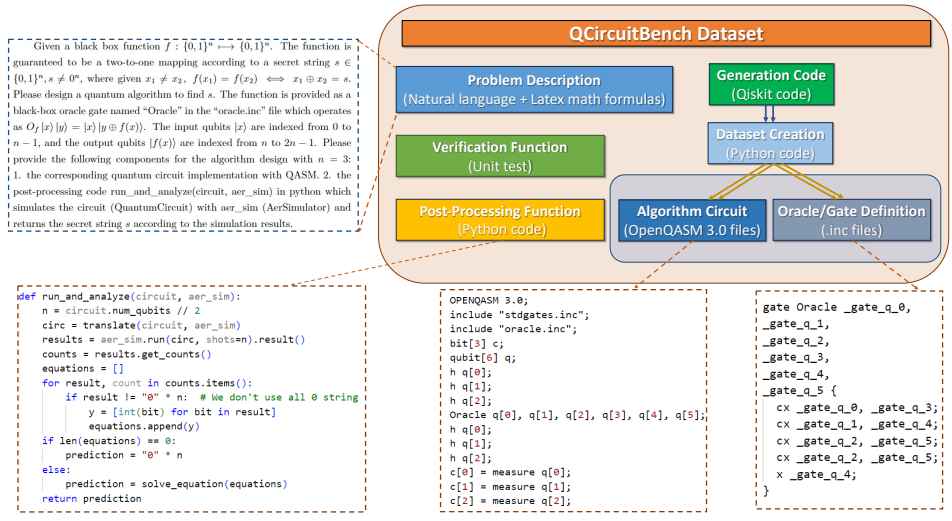

Figure 1: Structure of QCircuitBench. The components of QCircuitBench are presented in the frame on the top-right. As a showcase, this figure presents the components for Simon's problem [Simon, 1997], including its problem description in natural language, post-processing function in python code, circuit in a .qasm file, and oracle definition in a .inc file.

Design Principles. Different tasks encounter different challenges. Here we highlight the following construction principles, which are specially designed to adapt to these tasks:

- **Paradox of Oracle Implementation:** Quantum algorithms often treat the oracle U_f as a black-box, aiming to deduce properties of function $f(x)$ without directly accessing its implementation. However, quantum circuits for real-world platforms need an explicit gate definition to compile and run successfully. To address this challenge, which is often overlooked in tutorials and benchmarks, we provide the oracle as a black-box gate with its explicit definition in a separate "oracle.inc" library. This complies with OpenQASM syntax while ensures the oracle's functionality is accessible to the model without exposing its internal structure.
- **Classical Processing Specification:** A quantum algorithm involves not only quantum circuits but also the classical processing steps to interpret measurement results. For example, in Simon's algorithm, the model must solve linear equations $s \cdot y_i = 0$ from measured y_i. In addition to quantum circuits, we require the model to specify the classical processing function and define the shot count to characterize query complexity, crucial for the theoretical analysis of the algorithm.
- **Custom Quantum Gates:** Some composite gates, not part of the standard QASM library, are essential for advanced algorithms. To avoid model distractions, we provide these custom gates, such as multi-controlled X gates (45,060 lines for 14 qubits), in a "customgates.inc" file. These gates are defined hierarchically, allowing the model to use them without the burden of generating complex gate structures.
- **Automated Verification Function:** To ensure model outputs are syntactically correct and functionally valid, we implement automatic verification tools that check QASM syntax and circuit functionality. Instead of performing exhaustive Logic Equivalence Checking (LEC), we use extensive test cases to validate the correctness of the generated circuits, enabling efficient model evaluation without human intervention.

Based on theses principles, we proposed the framework of QCircuitBench. Below is a more detailed explanation for the 7 components of the dataset:

1. **Problem Description:** carefully hand-crafted prompts stating the oracle to be constructed or the target problem to be solved in natural language and latex math formulas. If the problem involves the usage of a quantum oracle or composite gates beyond the standard gate library, the interfaces of the oracle / gate will also be included (input qubits, output qubits, function mechanism).
2. **Generation Code:** one general Qiskit [Javadi-Abhari et al., 2024] code to create quantum circuits for oracles or algorithms of different settings, such as distinct secret strings or various qubit numbers. We choose Qiskit as the main experiment platform because it is a general quantum programming software widely used for the complete workflow from creating quantum circuits to transpiling, simulation, and execution on real hardware.
3. **Algorithm Circuit:** a .qasm file storing the quantum circuit for each specific setting. We choose OpenQASM 3.0 [Cross et al., 2022] as the format to store the quantum circuits, because Qiskit, as a python library, can only create quantum circuits at runtime instead of explicitly saving the circuits at gate level.[1]
4. **Post-Processing Function:** this is for Algorithm Design task only, see Section 3.1.2. The function takes a complete quantum circuit as input, uses the Qiskit AerSimulator to execute the circuit, and returns the final answer to the original problem according to the simulation results. For state preparation problems such as creating a GHZ state of n qubits, this function returns the qubit indices of the generated state.
5. **Oracle / Gate Definition:** a .inc file to provide definitions of composite gates or oracles. For oracle construction tasks, this only includes the definition of composite gates required to build the oracle. For algorithm design tasks, we also provide the gate definition of the oracle in this file, which successfully delivers the oracle in a black-box way.
6. **Verification Function:** a function to evaluate whether the implemented oracle / algorithm achieves the desired purpose with grammar validation and test cases verification. If there exist

[1]Although currently the Qiskit APIs for importing and dumping OpenQASM 3.0 files are still in experimental stage, we choose to adopt version 3.0 over 2.0 in that it supports saving parameterized circuits, which allows for extending the framework to variational quantum algorithms [Cerezo et al., 2021].

grammar errors, the function returns -1 and provides a detailed error message, which can be used as the feedback for LLMs to improve through interactive reasoning. If the program can execute successfully, the function returns a score between $[0, 1]$ indicating the success rate on test cases.[2]

7. **Dataset Creation Script:** the script to create the dataset from scratch in the format suitable for benchmarking / fine-tuning LLMs. It contains the following functions: 1. generate primitive QASM circuits. 2. extract gate definitions and add include instructions to create an algorithm circuit as the direct output. 3. validate and verify the correctness of the data points in the dataset. 4. concatenate the circuit with problem description as a json file for the benchmark pipeline.

This structure of QCircuitBench provides a general framework to formulate quantum algorithm design for large language models, with an easy extension to more advanced quantum algorithms.

4 Experiments

4.1 Benchmarking LLMs on QCircuitBench

We benchmark the quantum algorithm design capabilities of leading closed-source and open-source large language models using QCircuitBench. The workflow of our benchmark is illustrated in Figure 2. The total computation cost is approximately equivalent to two days on an A100 GPU.

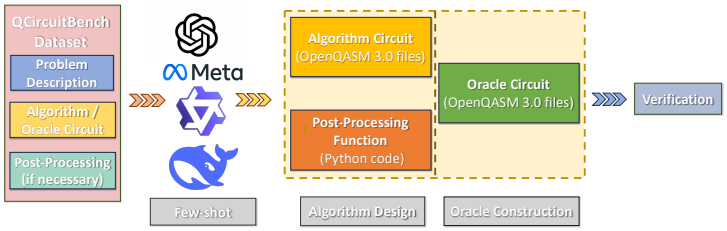

Figure 2: Flowchart of benchmarking QCircuitBench.

Models. Recently, the GPT series models have become the benchmark for generative models due to their exceptional performance. Specifically, we include two models from OpenAI, GPT-3.5-turbo [Brown et al., 2020] and GPT-4 [OpenAI et al., 2024], in our benchmark. Additionally, the LLAMA series models [Touvron et al., 2023a,b] are widely recognized as leading open-source models, and we have selected LLAMA-3-8B for our study. For a comprehensive evaluation, we also benchmark Qwen 2.5 [Yang et al., 2024a] and DeepSeek-R1 [Guo et al., 2025].

On these models, we employ a few-shot learning framework, a prompting technique that has shown considerable success in generative AI [Xie et al., 2021]. In this approach, we utilize either 1, 3, or 5 examples, followed by a problem description. To ensure we do not train and test on the same quantum algorithm, we implement k-fold validation among all algorithms.

Evaluation Metrics. We use three evaluation metrics (see Appendix C.1 for more details):

1. BLEU Score: This metric measures how closely the generated code matches the reference code, with a higher BLEU score indicating more similarity. Formally, the BLEU score is defined as:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right),$$

where BP is the acronym for brevity penalty, w_n is the weight for the n-gram precision (typically $\frac{1}{N}$ for uniform weights), p_n is the precision for n-grams. BP is calculated as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c \leq r \end{cases},$$

[2]The verification function explicitly integrates the oracle / gate definition library with output algorithm circuit since Qiskit importer for OpenQASM 3.0 does not support non-standard gate libraries currently.

Table 1: Benchmarking algorithm design in verification function scores.

Model	Shot	Bernstein Vazirani	Deutsch Jozsa	Grover	Phase Estimation	QFT	Simon	GHZ	Random Number Generator	Swap Test	W State	Generalized Simon (multi-str)	Generalized Simon (ternary)	VQE	QAOA	Shor	Avg
gpt4o	1	0.8461 (±0.1538)	0.8307 (±0.1533)	0.7644 (±0.1618)	0.6638 (±0.1141)	-1.0000 (±0.0000)	-0.2015 (±0.1268)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	0.5292 (±0.1745)	-0.0900 (±0.2279)	-1.0000 (±0.0000)	-0.4152 (±0.1857)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.7452
gpt4o	few	0.9592 (±0.0408)	0.9692 (±0.0208)	0.9165 (±0.0766)	0.4400 (±0.1784)	-1.0000 (±0.0000)	0.0600 (±0.0364)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	0.8090 (±0.0292)	0.4022 (±0.0776)	-1.0000 (±0.0000)	-0.3222 (±0.1698)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.8188
Llama3	1	-0.7538 (±0.1727)	-0.1077 (±0.2214)	-0.6154 (±0.2130)	-0.9231 (±0.0769)	-1.0000 (±0.0000)	-0.9231 (±0.0769)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.6933 (±0.2031)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.8583
Llama3	few	-0.5962 (±0.2184)	0.3962 (±0.2308)	-0.4605 (±0.2155)	-0.6615 (±0.1492)	-1.0000 (±0.0000)	-0.5846 (±0.1543)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.4572 (±0.2143)	-0.6122 (±0.2599)	-0.8889 (±0.1111)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.7893
gpt3.5	1	0.0769 (±0.2392)	0.2731 (±0.1794)	-0.0779 (±0.2099)	-0.1277 (±0.2722)	-1.0000 (±0.0000)	-0.2181 (±0.1238)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	0.1563 (±0.2399)	-0.6922 (±0.2111)	-1.0000 (±0.0000)	-0.2211 (±0.1472)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.6670
gpt3.5	few	0.2292 (±0.1534)	0.2038 (±0.1708)	0.0010 (±0.2508)	-0.5577 (±0.2331)	-1.0000 (±0.0000)	-0.1296 (±0.1078)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.3742 (±0.2332)	-0.8778 (±0.1222)	-1.0000 (±0.0000)	-0.3267 (±0.1684)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.7514
Qwen 2.5	1	-0.8923 (±0.1077)	-0.6285 (±0.1967)	-0.9231 (±0.0769)	-0.8462 (±0.1042)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.6270 (±0.1982)	-0.8444 (±0.1556)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.9115
Qwen 2.5	few	-0.4123 (±0.2268)	-0.0746 (±0.2211)	-0.9230 (±0.0769)	-0.8307 (±0.1151)	-1.0000 (±0.0000)	-0.6838 (±0.1371)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.3552 (±0.2403)	-1.0000 (±0.0000)	-0.8888 (±0.1111)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.7978
DeepSeek-R1	1	-0.8462 (±0.1538)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.8392 (±0.1090)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.6000 (±0.2651)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.9490
DeepSeek-R1	few	-0.5385 (±0.2152)	-0.2892 (±0.2607)	-1.0000 (±0.0000)	-0.7712 (±0.1618)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.6182 (±0.0000)	-0.8833 (±0.2031)	-1.0000 (±0.1167)	-1.0000 (±0.0000)	-0.5000 (±0.4999)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.8286

where c is the length of the generated text and r is the length of the reference text. Furthermore, n-gram precision p_n is calculated as:

$$p_n = \frac{\sum_{C \in \text{Candidates}} \sum_{n\text{gram} \in C} \min(\text{Count}(n\text{gram in candidate}), \text{Count}(n\text{gram in references}))}{\sum_{C \in \text{Candidates}} \sum_{n\text{gram} \in C} \text{Count}(n\text{gram in candidate})}.$$

2. Verification function: This function checks the syntax validation and the result correctness of the code produced by the language model. To be specific, we evaluate the result using three criteria:
   (a) **QASM Syntax Verification**: We first check the syntax of the QASM code provided by the model. The syntax verification function $V_{\text{QASM}}(q)$ is set to be 1 if the QASM syntax is correct, and 0 otherwise.
   (b) **Python Syntax Verification**: Similarly, the syntax of the post-processing Python code (which includes the run_and_analyze function), denoted $V_{\text{code}}(c)$, is set to be 1 if the Python syntax is correct, and 0 otherwise.
   (c) **Execution and Evaluation**: If at least one syntax check passes, we proceed to evaluating the functional correctness. For each test case t, we run the quantum circuit simulation for a number of shots M, and compare the result with the ground truth. The success rate is calculated as:
   $$\text{acc} = \frac{\sum_{t=1}^{T} \sum_{m=1}^{M} \mathbb{I}[\text{result = ground-truth}]}{T \times M}.$$
   The final verification score is a triplet $(V_{\text{QASM}}(q), V_{\text{code}}(c), \text{acc})$. In addition, all the verification functions were executed by classical simulations in our experiments, but the APIs we implemented are compatible with IBM hardware and can be easily adapted to quantum computers.
3. Byte Perplexity: This metric evaluates the model's ability to predict the next byte in a sequence. Formally, the Perplexity score is defined as:
   $$\text{PPL}(x) = 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 p(x_i|x_{<i})},$$
   where $p(x_i|x_{<i})$ is the probability of the i-th byte x_i given the preceding bytes $x_{<i}$ and N is the length of the byte sequence. Lower byte perplexity indicates better performance by reflecting the model's predictive accuracy.

The results for BLEU scores are shown in Figure 3. The verification scores of algorithm design tasks are shown in Table 1. We include the results of Byte Perplexity and the verification scores of oracle construction tasks in Appendix C.2.

We observe the following phenomena from the results:

- Most models achieve better scores in the few-shot setting than the 1-shot setting. This indicates their capability to learn effectively from contextual examples. Specifically, the score of tasks such as Deutsch-Jozsa were notably increased by 0.7108 after few-shot learning in the DeepSeek-R1 model. However, all models struggle with more complicated algorithms such as Grover, phase estimation, and quantum Fourier transform, with a score increase of 0.2616 by the Llama3 model on phase estimation. This highlights the differences in task difficulty.

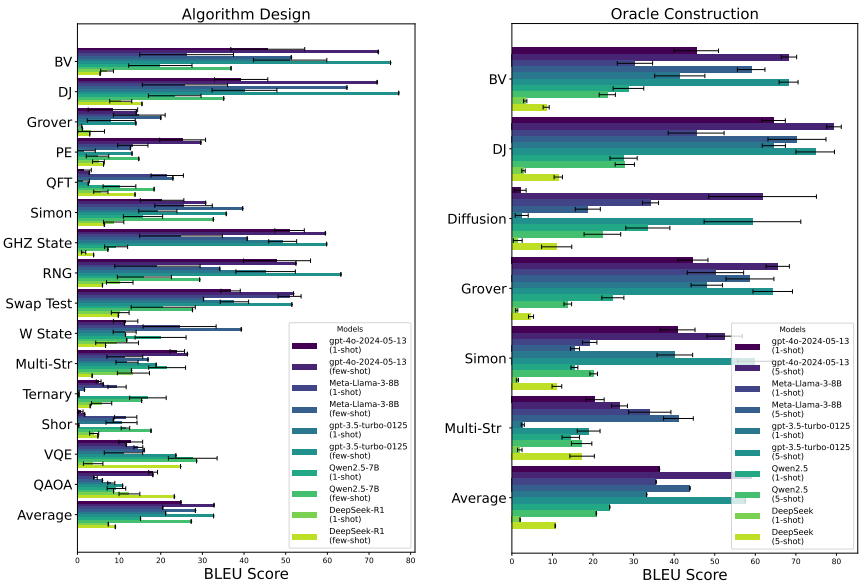

Figure 3: Benchmarking algorithm design and oracle construction tasks in BLEU scores.

- There exist challenges for near-term quantum algorithms. In particular, for the VQE and QAOA tasks, the models often fail to construct right parameterized circuits or apply optimization strategies correctly, with a score of at most -0.5000 by the DeepSeek-R1 model. This reflects the limitation of LLMs in handling hybrid quantum-classical workflows.
- GPT-4o and GPT-3.5 consistently excel in long-context comprehension, significantly outperforming other models across tasks, which highlights their superior in-context learning capabilities. In contrast, DeepSeek-R1 underperforms due to its long-chain reasoning style, which often exceeds the context length before producing a complete and verifiable solution.
- Although BLEU scores generally align with verification results, some discrepancies arise, such as the swap test showing relatively high BLEU scores but incorrect algorithm generation by most models. This observation emphasizes the need for complementary evaluation metrics such as our verification function.

Types of Errors Made by LLMs. In Appendix C.3, we include several case studies to illustrate and analyze various types of errors made by LLMs. In particular, they can be summarized as follows:

- Improvision error: GPT-4o tends to use advanced OpenQASM 3.0 features unsupported by Qiskit yet and novel namespace which might result in global conflicts in one-shot setting. This tendency to improvise by drawing on pre-trained knowledge rather than closely following the syntax of the example leads to avoidable "errors" and low verification scores. This issue is significantly alleviated in the 5-shot setting, highlighting GPT-4o's strong in-context learning ability. A detailed case study is given in Appendix C.3.1.
- Counting error: LLMs often fail to correctly identify the positions of ones in a binary string when constructing oracles for problems such as Bernstein-Vazirani. For instance, given the secret string $s = 000101$, GPT-4o misplaces the control qubits for CX gates, despite being explicitly reminded of the correct rule and asked to list the indices with value 1. This misidentification, likely due to tokenization issues, highlights a fundamental limitation of LLMs in performing basic indexing tasks. A detailed case study is given in Appendix C.3.2.
- Data contamination: We observe a performance separation between writing general Qiskit codes and explicit gate-level circuits in QASM. Since Qiskit provides detailed tutorial with general codes for several algorithms, LLMs may rely on memorization and retrieval rather than genuine algorithm design. Our dataset, based on QASM files created from scratch, may help circumvent this issue and serve as a stable and fair method for benchmarking AI syntax learning. A detailed case study is given in Appendix C.3.3.

4.2 Fine-tuning on QCircuitBench

Although QCircuitBench is targeted as a benchmark dataset at current stage, we consider fine-tuning / training from scratch based on our dataset as an interesting and important research direction. The unique nature of quantum data requires novel fine-tuning methods and model architecture designs, which could serve as a standalone topic. For a primitive demonstration, we present fine-tuning results on data from the oracle construction task here.

Following Dettmers et al. [2024], we quantize the model to 8-bits and then train it with LoRA [Hu et al., 2022]. In our experiments, we use fp16 computational datatype. We set LoRA $r = 16, \alpha = 32$ and add LoRA modules on all the query and value layers. We also use AdamW [Loshchilov and Hutter, 2019] and LoRA dropout of 0.05. The results are shown as follows:

Table 2: Fine-tuning oracle construction scores.

Score	Model	Setting	Bernstein-Vazirani	Deutsch-Jozsa	Grover	Simon	Clifford	Universal	Avg
BLEU	gpt4o	few-shot(5)	95.6388 (±0.3062)	91.0564 (±0.6650)	92.0620 (±0.6288)	80.3390 (±2.0900)	39.5469 (±3.6983)	33.3673 (±3.1007)	72.0017
	Llama3	few-shot(5)	53.5574 (±5.2499)	69.8996 (±5.7812)	61.3102 (±5.4671)	26.3083 (±2.0048)	13.0729 (±0.9907)	13.4185 (±1.2299)	39.5945
	Llama3	finetune	76.0480 (±7.9255)	71.8378 (±2.4179)	67.7892 (±7.8900)	43.8469 (±3.2998)	10.8978 (±0.6169)	7.1854 (±0.5009)	46.2675
Verification	gpt4o	few-shot(5)	0.0000 (±0.0246)	0.4300 (±0.0590)	0.0000 (±0.1005)	-0.0200 (±0.0141)	-0.0333 (±0.0401)	-0.1023 (±0.0443)	0.0457
	Llama3	few-shot(5)	-0.2700 (±0.0468)	0.0900 (±0.0668)	-0.5200 (±0.0858)	-0.6600 (±0.0476)	-0.7303 (±0.0473)	-0.5056 (±0.0549)	-0.4327
	Llama3	finetune	-0.1300 (±0.0485)	-0.2000 (±0.0402)	-0.3300 (±0.0900)	-0.7400 (±0.0441)	-0.8741 (±0.0343)	-0.9342 (±0.0262)	-0.5347
PPL	Llama3	few-shot(5)	1.1967 (±0.0028)	1.1174 (±0.0015)	1.1527 (±0.0021)	1.1119 (±0.0017)	1.4486 (±0.0054)	1.4975 (±0.0051)	1.2541
	Llama3	finetune	1.0004 (±0.0002)	1.1090 (±0.0014)	1.0010 (±0.0006)	1.1072 (±0.0011)	1.2944 (±0.0053)	1.3299 (±0.0055)	1.1403

We observe that the Llama3 model demonstrates the most notable improvement on Grover's algorithm after fine-tuning, with the verification score increased by 0.3077. Case studies on the Bernstein-Vazirani oracle reveal that, before fine-tuning, the model would indiscriminately apply CX gates to all qubits. After fine-tuning, it begins to selectively apply CX gates to qubits corresponding to '1's in the secret string. While some counting errors persist, the model occasionally identifies all correct positions, demonstrating a marked improvement. This suggests that fine-tuning enables the model to internalize structural patterns in oracle construction, leading to improved performance across tasks.

Regarding the interesting performance decrease on Clifford and universal random circuits, we conducted additional experiments on temperature and refer to Appendix C.2 for more details.

5 Conclusions and Future Work

In this paper, we propose QCircuitBench, the first comprehensive, structured universal quantum algorithm dataset and quantum circuit generation benchmark for AI models. This framework formulates quantum algorithm design from the programming language perspective and includes detailed descriptions and implementation of most established and important quantum algorithms / primitives, allowing for automatic verification methodologies. Benchmarking of QCircuitBench on up-to-date LLMs is systematically conducted. Fine-tuning results also showcase the potential of QCircuitBench as a training dataset, and implementation of the Generalized Simon's Problem mentioned in Section 3.1.2 showcases the compatibility of our framework with more complex algorithms. In addition, our framework is designed to scale with increasing qubit numbers and support complex quantum algorithms as long as they are efficiently implementable with polynomial gates.

Our work leaves several open questions for future investigation:

- QCircuitBench is a benchmarking dataset for LLMs. It is of general interest to extend benchmarking to training, which will help LLMs better maneuver quantum algorithm design. We have implemented advanced algorithms such as the Generalized Simon's Problem, but this in general needs implementations of more advanced algorithms to make it impactful.
- Since quantum algorithms have fundamental difference from classical algorithms, novel fine-tuning methods to attempt quantum algorithm design and quantum circuit implementation, or even developments of new quantum algorithms by LLMs are solicited.

References

A. Ambainis. Quantum search algorithms. *ACM SIGACT News*, 35(2):22–35, 2004. arXiv:quant-ph/0504012

F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574 (7779):505–510, 2019. doi: 10.1038/s41586-019-1666-5. URL https://doi.org/10.1038/s41586-019-1666-5.

A. Barenco, A. Berthiaume, D. Deutsch, A. Ekert, R. Jozsa, and C. Macchiavello. Stabilization of quantum computations by symmetrization. *SIAM Journal on Computing*, 26(5):1541–1557, 1997. arXiv:quant-ph/9604028

C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore*, pages 175–179, 1984.

C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, 2014.

C. H. Bennett and S. J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys. Rev. Lett.*, 69:2881–2884, Nov 1992. doi: 10.1103/PhysRevLett.69.2881. URL https://link.aps.org/doi/10.1103/PhysRevLett.69.2881.

C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, pages 11–20, 1993.

J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017. arXiv:1611.09347

D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter, J. P. Bonilla Ataides, N. Maskara, I. Cong, X. Gao, P. Sales Rodriguez, T. Karolyshyn, G. Semeghini, M. J. Gullans, M. Greiner, V. Vuletić, and M. D. Lukin. Logical quantum processor based on reconfigurable atom arrays. *Nature*, 626(7997):58–65, 2024.

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, 2020. arXiv:2005.14165

H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Phys. Rev. Lett.*, 87: 167902, Sep 2001.

M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9): 625–644, 2021. arXiv:2012.09265

K. Chen, W. Fang, J. Guan, X. Hong, M. Huang, J. Liu, Q. Wang, and M. Ying. VeriQBench: A benchmark for multiple types of quantum circuits, 2022. arXiv:2206.10880

C. Ciliberto, M. Herbster, A. D. Ialongo, M. Pontil, A. Rocchetto, S. Severini, and L. Wossnig. Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170551, 2018. arXiv:1707.08561

D. Coppersmith. An approximate fourier transform useful in quantum factoring, 2002. arXiv:quant-ph/0201067

A. Cross, A. Javadi-Abhari, T. Alexander, N. De 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*, 3(3):1–50, 2022. arXiv:2104.14722

A. M. Dalzell, S. McArdle, M. Berta, P. Bienias, C.-F. Chen, A. Gilyén, C. T. Hann, M. J. Kastoryano, E. T. Khabiboulline, A. Kubica, G. Salton, S. Wang, and F. G. Brandao. Quantum algorithms: A survey of applications and end-to-end complexities, 2023. arXiv:2310.03011

M. DeCross, R. Haghshenas, M. Liu, E. Rinaldi, J. Gray, Y. Alexeev, C. H. Baldwin, J. P. Bartolotta, M. Bohn, E. Chertkov, J. Cline, J. Colina, D. DelVento, J. M. Dreiling, C. Foltz, J. P. Gaebler, T. M. Gatterman, C. N. Gilbreth, J. Giles, D. Gresh, A. Hall, A. Hankin, A. Hansen, N. Hewitt, I. Hoffman, C. Holliman, R. B. Hutson, T. Jacobs, J. Johansen, P. J. Lee, E. Lehman, D. Lucchetti, D. Lykov, I. S. Madjarov, B. Mathewson, K. Mayer, M. Mills, P. Niroula, J. M. Pino, C. Roman, M. Schecter, P. E. Siegfried, B. G. Tiemann, C. Volin, J. Walker, R. Shaydulin, M. Pistoia, S. A. Moses, D. Hayes, B. Neyenhuis, R. P. Stutz, and M. Foss-Feig. The computational power of random quantum circuits in arbitrary geometries, 2024. arXiv:2406.02501

T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

W. Dür, G. Vidal, and J. I. Cirac. Three qubits can be entangled in two inequivalent ways. *Phys. Rev. A*, 62:062314, Nov 2000.

E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm, 2014. arXiv:1411.4028

F. Fürrutter, G. Muñoz-Gil, and H. J. Briegel. Quantum circuit synthesis with diffusion models. *Nature Machine Intelligence*, pages 1–10, 2024. arXiv:2311.02041

D. M. Greenberger, M. A. Horne, and A. Zeilinger. Going beyond bell's theorem, 2007. arXiv:0712.0921

L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996. arXiv:quant-ph/9605043

D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu,

Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025. arXiv:2501.12948

M. Herrero-Collantes and J. C. Garcia-Escartin. Quantum random number generators. *Rev. Mod. Phys.*, 89:015004, Feb 2017.

E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. arXiv:2106.09685

A. J., A. A. Adedoyin, J. J. Ambrosiano, P. M. Anisimov, W. R. Casper, G. Chennupati, C. J. Coffrin, H. N. Djidjev, D. O. Gunter, S. Karra, N. W. Lemons, S. Lin, A. Malyzhenkov, D. D. L. Mascarenas, S. M. Mniszewski, B. T. Nadiga, D. O'Malley, D. A. Oyen, S. D. Pakin, L. Prasad, R. M. Roberts, P. R. Romero, N. Santhi, N. Sinitsyn, P. J. Swart, J. G. Wendelberger, B. Yoon, R. J. Zamora, W. Zhu, S. J. Eidenbenz, A. Bärtschi, P. J. Coles, M. D. Vuffray, and A. Y. Lokhov. Quantum algorithm implementations for beginners. *ACM Transactions on Quantum Computing*, 3(4), 7 2022. doi: 10.1145/3517340. arXiv:1804.03719

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, 2024. arXiv:2405.08810

S. P. Jordan. Quantum algorithm zoo. https://quantumalgorithmzoo.org, 2025. Accessed: 2025-05-15.

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

A. Y. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995. arXiv:quant-ph/9511026

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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*, 4(2):1–26, 2023. arXiv:2005.13018

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

A. Nakayama, K. Mitarai, L. Placidi, T. Sugimoto, and K. Fujii. VQE-generated quantum circuit dataset for machine learning, 2023. arXiv:2302.09751

M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2000.

OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo,

C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. GPT-4 technical report, 2024. URL `https://openai.com`. arXiv:`2303.08774`

E. Perrier, A. Youssry, and C. Ferrie. QDataSet, quantum datasets for machine learning. *Scientific Data*, 9(1):582, 2022. arXiv:`2108.06661`

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*, 5(1):4213, 2014. arXiv:`1304.3061`

L. Placidi, R. Hataya, T. Mori, K. Aoyama, H. Morisaki, K. Mitarai, and K. Fujii. MNISQ: A large-scale quantum circuit dataset for machine learning on/for quantum computers in the NISQ era, 2023. arXiv:`2306.16627`

D. Poulin, A. Qarry, R. Somma, and F. Verstraete. Quantum simulation of time-dependent hamiltonians and the convenient illusion of hilbert space. *Phys. Rev. Lett.*, 106:170501, Apr 2011. doi: 10.1103/PhysRevLett.106.170501. URL `https://link.aps.org/doi/10.1103/PhysRevLett.106.170501`.

J. Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, 2018. arXiv:`1801.00862`

N. Quetschlich, L. Burgholzer, and R. Wille. MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing. *Quantum*, 2023.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. arXiv:`2112.10752`

M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)*, 45(2):1–34, 2013. arXiv:`1110.2574`

L. Schatzki, A. Arrasmith, P. J. Coles, and M. Cerezo. Entangled datasets for quantum machine learning, 2021. arXiv:`2109.03400`

M. Schuld, I. Sinayskiy, and F. Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, 2015. arXiv:`1409.3097`

T. Shirakawa, H. Ueda, and S. Yunoki. Automatic quantum circuit encoding of a given arbitrary quantum state, 2021. arXiv:`2112.14524`

P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999. arXiv:quant-ph/9508027

D. R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine learning*, pages 2256–2265. PMLR, 2015. arXiv:1503.03585

Y. Tang, H. Xiong, N. Yang, T. Xiao, and J. Yan. Q-TAPE: A task-agnostic pre-trained approach for quantum properties estimation. In *The Twelfth International Conference on Learning Representations*, 2023.

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, 2023a. arXiv:2302.13971

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. Singh Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b. arXiv:2307.09288

T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong. Solving Olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, M. Gong, C. Guo, C. Guo, S. Guo, L. Han, L. Hong, H.-L. Huang, Y.-H. Huo, L. Li, N. Li, S. Li, Y. Li, F. Liang, C. Lin, J. Lin, H. Qian, D. Qiao, H. Rong, H. Su, L. Sun, L. Wang, S. Wang, D. Wu, Y. Xu, K. Yan, W. Yang, Y. Yang, Y. Ye, J. Yin, C. Ying, J. Yu, C. Zha, C. Zhang, H. Zhang, K. Zhang, Y. Zhang, H. Zhao, Y. Zhao, L. Zhou, Q. Zhu, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan. Strong quantum computational advantage using a superconducting quantum processor. *Physical review letters*, 127(18):180501, 2021.

Z. Wu, D. Qiu, J. Tan, H. Li, and G. Cai. Quantum and classical query complexities for generalized simon's problem. *Theoretical Computer Science*, 924:171–186, 2022.

S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2021. arXiv:2111.02080

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. 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, 2024a. arXiv:2412.15115

K. Yang, A. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar. LeanDojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024b. arXiv:2306.15626

Z. Ye, Y. Huang, L. Li, and Y. Wang. Query complexity of generalized simon's problem. *Information and Computation*, 281:104790, 2021.

B. Yu, F. N. Baker, Z. Chen, X. Ning, and H. Sun. LlaSMol: Advancing large language models for chemistry with a large-scale, comprehensive, high-quality instruction tuning dataset, 2024. arXiv:2402.09391

Z. Zhang, Y. Zhang, H. Yao, J. Luo, R. Zhao, B. Huang, J. Zhao, Y. Liao, K. Li, L. Zhao, et al. Xiwu: A basis flexible and learnable LLM for high energy physics, 2024. arXiv:2404.08001

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims accurately reflect the contributions and scope of the QCircuit-Bench dataset.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 5

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: We do not have theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See Section 4 and supplemental material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: See supplementary material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: See Section 4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: See Section 4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Section 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Quantum computing is still a nascent technology at the moment. Therefore, our work does not have negative societal impacts from our perspective. In the future, we welieve that our dataset can be beneficial for quantum algorithm design and the field of quantum computing as a whole.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our dataset contains purely quantum circuits and does not pose such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cited Qiskit [Javadi-Abhari et al., 2024], OpenQASM [Cross et al., 2022], and QASMBench [Li et al., 2023] in our paper. The links of the aforementioned assets are given in reference.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: See supplementary material.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are used for benchmarking. See Section 4.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Details of QCircuitBench

The QCircuitBench Dataset, along with its Croissant metadata, is available on Harvard Dataverse at the following link: https://doi.org/10.7910/DVN/ZC4PNI

QCircuitBench has the following directory structure:

```
QCircuitBench
├── Oracle Construction ..... All data for the oracle construction task
│   ├── Quantum Logic Synthesis ..... Textbook-level and advanced oracles
│   └── Problem Encoding ..... Oracles encoding application scenarios
├── Algorithm Design ..... All data for the quantum algorithm design task
│   ├── Quantum Computing ..... Universal quantum computing algorithms
│   └── Quantum Information ..... Quantum information tasks and protocols
└── Random Circuits ..... All data for the random circuit synthesis task
    ├── Clifford ..... Random circuits with the Clifford gate set
    └── Universal ..... Random circuits with the universal gate set
```

In each subdirectory, there is a folder for each specific algorithm. For instance, the folder structure for Simon's algorithm is as follows:

```
Algorithm Design
└── Quantum Computing
    └── simon ..... All data for the Simon's Problem
        ├── simon-dataset.py ..... Dataset creation script
        ├── simon-generation.py ..... Qiskit generation code
        ├── simon-post-processing.py ..... Post-processing function
        ├── simon-utils.py ..... Utility functions for verification
        ├── simon-verification.py ..... Verification function
        ├── simon-description.txt ..... Problem description
        ├── simon-verification.txt ..... Verification results of the data points
        ├── full circuit ..... Raw data of quantum circuits
        │   ├── simon-n2
        │   │   └── simon-n2-s11-k11.qasm ..... Full circuit for a concrete setting
        │   ├── simon-n3
        │   │   ├── simon-n3-s011-k001.qasm
        │   │   └── simon-n3-s011-k101.qasm
        │   └── ...
        ├── test oracle ..... Extracted oracle definitions
        │   ├── n2
        │   │   └── trial1
        │   │       ├── oracle.inc ..... Oracle definition as a .inc file
        │   │       └── oracle-info.txt ..... Oracle information (such as key strings)
        │   ├── n3
        │   │   ├── trial1
        │   │   │   ├── oracle.inc
        │   │   │   └── oracle-info.txt
        │   │   └── trial2
        │   │       ├── oracle.inc
        │   │       └── oracle-info.txt
        │   └── ...
        ├── simon-n2.qasm ..... Algorithm circuit for model output
        ├── simon-n3.qasm
        └── ...
```

We expect to extend QCircuitBench under this general structure.

A.1 Format

In this subsection, we provide concrete examples to illustrate the different components of QCircuitBench. We use the case of Simon's Problem throughout the demonstration to achieve better consistency. For further details, please check the code repository.

1. **Problem Description:** this is the carefully hand-crafted description of the task in natural language and latex math formulas. The description is provided as one template for each algorithm, and the concrete settings (such as the qubit number) are replaced when creating the data points in json. The file is named as "{algorithm_name}_description.txt".

Problem Description Template for Simon's Problem

Given a black box function $f : \{0,1\}^n \longmapsto \{0,1\}^n$. The function is guaranteed to be a two-to-one mapping according to a secret string $s \in \{0,1\}^n, s \neq 0^n$, where given $x_1 \neq x_2$, $f(x_1) = f(x_2) \iff x_1 \oplus x_2 = s$. Please design a quantum algorithm to find s. The function is provided as a black-box oracle gate named "Oracle" in the "oracle.inc" file which operates as $O_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle$. The input qubits $|x\rangle$ are indexed from 0 to $n-1$, and the output qubits $|f(x)\rangle$ are indexed from n to $2n-1$. Please provide the following components for the algorithm design with n ={qubit number}: 1. the corresponding quantum circuit implementation with {QASM / Qiskit}. 2. the post-processing code run_and_analyze(circuit, aer_sim) in python which simulates the circuit (QuantumCircuit) with aer_sim (AerSimulator) and returns the secret string s according to the simulation results.

2. **Generation Code:** one general Qiskit code to create quantum circuits of different settings. Note that the oracle for the problem is provided as a black-box gate "oracle" here. This code is used to generate the raw data, but can also be used as a testing benchmark for writing Qiskit codes. The file is named as "{algorithm_name}_generation.py".

```python
from Qiskit import QuantumCircuit


def simon_algorithm(n, oracle):
    """Generates a Simon algorithm circuit.

    Parameters:
    - n (int): number of qubits
    - s (str): the secret string of length n

    Returns:
    - QuantumCircuit: the Simon algorithm circuit
    """
    # Create a quantum circuit on 2n qubits
    simon_circuit = QuantumCircuit(2 * n, n)

    # Initialize the first register to the |+> state
    simon_circuit.h(range(n))

    # Append the Simon's oracle
    simon_circuit.append(oracle, range(2 * n))

    # Apply a H-gate to the first register
    simon_circuit.h(range(n))

    # Measure the first register
    simon_circuit.measure(range(n), range(n))

    return simon_circuit
```

Listing 1: Qiskit generation code for Simon's algorithm.

3. **Algorithm Circuit:** the OpenQASM 3.0 format file storing the quantum circuit in gate level for each specific setting. Note that the explicit construction of "Oracle" is provided separately in "oracle.inc" file, which guarantees the usage of oracle in a black-box way. This filed is named as "{algorithm_name}_n{qubit_number}.qasm".

```
OPENQASM 3.0;
include "stdgates.inc";
```

```
include "oracle.inc";
bit[3] c;
qubit[6] q;
h q[0];
h q[1];
h q[2];
Oracle q[0], q[1], q[2], q[3], q[4], q[5];
h q[0];
h q[1];
h q[2];
c[0] = measure q[0];
c[1] = measure q[1];
c[2] = measure q[2];
```

Listing 2: OpenQASM 3.0 Code for Simon's algorithm with $n = 3$.

4. **Post-Processing Function:** this function simulates the quantum circuit and derives the final answer to the problem. The file is named as "{algorithm_name}_post_processing.py".

```
from sympy import Matrix
import numpy as np
from Qiskit import transpile


def mod2(x):
    return x.as_numer_denom()[0] % 2


def solve_equation(string_list):
    """
    A^T | I
    after the row echelon reduction, we can get the basis of the
        ↪ nullspace of A in I
    since we just need the string in binary form, so we can just
        ↪ use the basis
    if row == n-1 --> only one
    if row < n-1 --> get the first one (maybe correct or wrong)
    """
    M = Matrix(string_list).T

    # Augmented  : M | I
    M_I = Matrix(np.hstack([M, np.eye(M.shape[0], dtype=int)]))

    # RREF row echelon form , indices of the pivot columns
    # If x % 2 = 0, it will not be chosen as pivot (modulo 2)
    M_I_rref = M_I.rref(iszerofunc=lambda x: x % 2 == 0)

    # Modulo 2
    M_I_final = M_I_rref[0].applyfunc(mod2)

    # Non-Trivial solution
    if all(value == 0 for value in M_I_final[-1, : M.shape[1]]):
        result_s = "".join(str(c) for c in M_I_final[-1, M.shape[1]
            ↪  :])

    # Trivial solution
    else:
        result_s = "0" * M.shape[0]

    return result_s


def run_and_analyze(circuit, aer_sim):
    n = circuit.num_qubits // 2
    circ = transpile(circuit, aer_sim)
```

```
    results = aer_sim.run(circ, shots=n).result()
    counts = results.get_counts()
    equations = []
    for result, count in counts.items():
        if result != "0" * n:  # We don't use all 0 string
            y = [int(bit) for bit in result]
            equations.append(y)
    if len(equations) == 0:
        prediction = "0" * n
    else:
        prediction = solve_equation(equations)
    return prediction
```

Listing 3: Post-processing code for Simon's algorithm.

5. **Oracle / Gate Definition:** this .inc file provides the definitions of composite gates or oracles. The file is named "customgates.inc" for oracle construction tasks and "oracle.inc" for algorithm design tasks.

```
gate Oracle _gate_q_0, _gate_q_1, _gate_q_2, _gate_q_3, _gate_q_4,
    ↪ _gate_q_5 {
  cx _gate_q_0, _gate_q_3;
  cx _gate_q_1, _gate_q_4;
  cx _gate_q_2, _gate_q_5;
  cx _gate_q_2, _gate_q_5;
  x _gate_q_3;
}
```

Listing 4: One test case oracle for Simon's algorithm with $n = 3$.

For algorithm design tasks, this .inc file is accompanied with an "oracle_info.txt" file to describe the encoded information of the oracle. This helps the verification function to check the correctness of the derived answer by the model. The above test case is equipped with the following information text:

oracle_info.txt for Simon's Problem with qubit number 3 and test case 2.

Secret string: 100
Key string: 001

6. **Verification Function:** the function to evaluate the output with grammar validation and test cases verification. The file is named as "{algorithm_name}_verification.py".

```
from simon_utils import *


def check_model(qasm_string, code_string, n):
    """Check the Simon model."""
    # Verify the syntax of the QASM code with the first test case
        ↪ oracle
    t = 1
    with open(f"test_oracle/n{n}/trial{t}/oracle.inc", "r") as file
        ↪ :
        oracle_def = file.read()
    full_qasm = plug_in_oracle(qasm_string, oracle_def)
    circuit = verify_qasm_syntax(full_qasm)
    if circuit is None:
        return -1
    try:
        exec(code_string, globals())
        aer_sim = AerSimulator()
        total_success = 0
        total_fail = 0
        t_range = min(10, 4 ** (n - 2))
```

```
        shots = 10
        for t in range(1, 1 + t_range):
            print(f"    Running Test Case {t}")
            with open(f"test_oracle/n{n}/trial{t}/oracle.inc", "r")
                ↪  as file:
                oracle_def = file.read()
            full_qasm = plug_in_oracle(qasm_string, oracle_def)
            circuit = loads(full_qasm)
            with open(f"test_oracle/n{n}/trial{t}/oracle_info.txt",
                ↪  "r") as file:
                content = file.read()
            match = re.search(r"Secret string: ([01]+)", content)
            if match:
                secret_string = match.group(1)
            else:
                raise ValueError("Secret string not found in the
                    ↪ file.")

            cnt_success = 0
            cnt_fail = 0
            for shot in range(shots):
                prediction = run_and_analyze(circuit.copy(),
                    ↪ aer_sim)
                if not isinstance(prediction, str):
                    raise TypeError("Predicted secret string should
                        ↪  be a string.")
                if prediction == secret_string:
                    cnt_success += 1
                else:
                    cnt_fail += 1
            print(f"        Success: {cnt_success}/{shots}, Fail: {
                ↪ cnt_fail}/{shots}")
            total_success += cnt_success
            total_fail += cnt_fail
        print(f"Total Success: {total_success}; Total Fail: {
            ↪ total_fail}")
        return total_success / (total_fail + total_success)

    except Exception as e:
        print(f"Error: {e}")
        return -1
```

Listing 5: Verification function for Simon's algorithm.

This verification function is accompanied with an "{algorithm_name}_utils.py" file to provide necessary utility functions.

```
from Qiskit.qasm3 import loads
from Qiskit_aer import AerSimulator
import re


def print_and_save(message, text):
    print(message)
    text.append(message)


def plug_in_oracle(qasm_code, oracle_def):
    """Plug-in the oracle definition into the QASM code."""
    oracle_pos = qasm_code.find('include "oracle.inc";')
    if oracle_pos == -1:
        raise ValueError("Oracle include statement not found in the
            ↪  file")
    full_qasm = (
        qasm_code[:oracle_pos]
```

```
        + oracle_def
        + qasm_code[oracle_pos + len('include "oracle.inc";') :]
    )
    return full_qasm


def verify_qasm_syntax(output):
    """Verify the syntax of the output and return the corresponding
        ↪ QuantumCircuit (if it is valid)."""
    assert isinstance(output, str)
    try:
        # Parse the OpenQASM 3.0 code
        circuit = loads(output)
        print(
            "    The OpenQASM 3.0 code is valid and has been
                ↪ successfully loaded as a QuantumCircuit."
        )
        return circuit
    except Exception as e:
        print(f"    Error: The OpenQASM 3.0 code is not valid.
            ↪ Details: {e}")
        return None
```

Listing 6: Utility functions for verification of Simon's algorithm.

7. **Dataset Creation Script:** this script involves all the code necessary to create the data points from scratch. The file is named as "{algorithm_name}_dataset.py". The main function looks like this:

```
    def main():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-f",
        "--func",
        choices=["qasm", "json", "gate", "check"],
        help="The function to call: generate qasm circuit, json
            ↪ dataset or extract gate definition.",
    )
    args = parser.parse_args()
    if args.func == "qasm":
        generate_circuit_qasm()
    elif args.func == "json":
        generate_dataset_json()
    elif args.func == "gate":
        extract_gate_definition()
    elif args.func == "check":
        check_dataset()
```

Listing 7: Main function of the dataset script for Simon's algorithm.

Here the "generate_circuit_qasm()" function generates the raw data of quantum circuits in OpenQASM 3.0 format where the algorithm circuit and the oracle definition are blended, then "extract_gate_definition()" function extracts the definition of oracles and formulates the algorithm circuits into the format suitable for model output. The "check_dataset()" function is used to check the correctness of the created data points and "generate_dataset_json()" function to combine the data into json format for easy integration with the benchmarking pipeline.

A.2 Discussion of more tasks

Problem Encoding. In Section 3.1.1, we mentioned another category of oracle construction tasks referred to as "Problem Encoding", which involves applying quantum algorithms, such as Grover's algorithm, to solve practical problems such as SAT and triangle finding. The crux of this process is encoding the problem constraints into Grover's oracle, thereby making this a type of oracle construction task. Unlike quantum logic synthesis, which encodes an explicit function $f(x)$ as a unitary operator U_f, this task involves converting the constraints of a particular problem into the

required oracle form. We provide implementations of several concrete problems in this directory as demonstrations and will include more applications in future work.

Quantum Information Protocols. In Section 3.1.2, we have also implemented three important quantum information protocols: Quantum Teleportation, Superdense Coding, and Quantum Key Distribution (BB84). A brief introduction to these protocols can be found in Appendix B. We did not include the experiments for these protocols as they involve communication between two parties, which is challenging to characterize with a single OpenQASM 3.0 file. We recommend revising the post-processing function as a general classical function to schedule the communication and processing between different parties specifically for these protocols. The fundamental quantum circuits and processing codes are provided in the repository.

A.3 Datasheet

Here we present a datasheet for the documentation of QCircuitBench.

Motivation

- *For what purpose was the dataset created?* It was created as a benchmark for the capability of designing and implementing quantum algorithms for LLMs.
- *Who created the dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?* The authors of this paper.
- *Who funded the creation of the dataset?* We will reveal the funding resources in the Acknowledgement section of the final version.

Composition

- *What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)?* The dataset comprises problem description, generation code, algorithm circuit, post-processing function, oracle / gate definition, verification function, and dataset creation script for various quantum algorithms.
- *How many instances are there in total (of each type, if appropriate)?* The dataset has 3 task suites, 23 algorithms, and 128,573 data points. There are additional quantum information protocols and problem encoding tasks not included for experiments.
- *Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set?* The dataset contains instances with restricted qubit numbers due to the current scale of real quantum hardware.
- *What data does each instance consist of?* Qiskit codes, OpenQASM 3.0 codes, python scripts, and necessary text information.
- *Are relationships between individual instances made explicit?* Yes, the way to create different instances are clearly described in Appendix A.1.
- *Are there recommended data splits?* Yes, we recommend splitting the data according to different algorithms in algorithm design task.
- *Are there any errors, sources of noise, or redundancies in the dataset?* There might be some small issues due to the dumping process of Qiskit and programming mistakes (if any).
- *Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)?* The dataset is self-contained.
- *Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals' non-public communications)?* No.
- *Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety?* No.

Collection Process

- *How was the data associated with each instance acquired?* The data is created by first composing Qiskit codes for each algorithm and then converting to OpenQASM 3.0 files using Qiskit.qasm3.dump function, with additional processing procedure.
- *What mechanisms or procedures were used to collect the data (e.g., hardware apparatuses or sensors, manual human curation, software programs, software APIs)?* Manual human programming and Qiskit APIs.
- *Who was involved in the data collection process (e.g., students, crowd workers, contractors), and how were they compensated (e.g., how much were crowd workers paid)?* Nobody other than the authors of the paper.
- *Over what timeframe was the data collected?* The submitted version of the dataset was created in May 2025.

Uses

- *Has the dataset been used for any tasks already?* It has been used in this paper to benchmark LLM's ability for quantum algorithm design.
- *Is there a repository that links to any or all papers or systems that use the dataset?* The only paper which uses the dataset for now is this paper.

Distribution

- *Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created?* Yes, the dataset will be made publicly available on the Internet after the review process.
- *How will the dataset be distributed (e.g., tarball on website, API, GitHub)?* It will be distributed on the GitHub platform.
- *Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)?* The dataset is distributed under CC BY 4.0.
- *Have any third parties imposed IP-based or other restrictions on the data associated with the instances?* No.
- *Do any export controls or other regulatory restrictions apply to the dataset or to individual instances?* No.

Maintenance

- *Who will be supporting/hosting/maintaining the dataset?* The authors of this paper.
- *How can the owner/curator/manager of the dataset be contacted (e.g., email address)?* The email for contact will be provided after the review process.
- *Is there an erratum?* Not at this time.
- *Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)?* Yes, it will be continually updated.
- *If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?* Yes, they can do so with the GitHub platform.

A.4 Copyright and Licensing Terms

This work is distributed under a CC BY 4.0 license. The implementation of the code references open-source projects such as Qiskit, QuantumKatas, Cirq, and NWQBench. We bear responsibility in case of violation of rights.

B Preliminaries for Quantum Computing and Quantum Information

In this section, we will introduce necessary backgrounds for quantum computing related to this paper. A more detailed introduction to quantum computing can be found in the standard textbook by Nielsen and Chuang [2000].

Quantum States. In classical computing, the basic unit is a bit. In quantum computing, the basic unit is a *qubit*. Mathematically, n ($n \in \mathbb{N}$) qubits forms an N-dimensional Hilbert space for $N = 2^n$. An n-qubit *quantum state* $|\phi\rangle$ can be written as

$$|\phi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle, \text{ where } \sum_{i=0}^{N-1} |\alpha_i|^2 = 1. \tag{1}$$

Here $|\cdot\rangle$ represents a column vector, also known as a ket state. The tensor product of two quantum states $|\phi_1\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle$ and $|\phi_2\rangle = \sum_{j=0}^{M-1} \beta_j |j\rangle$ with $M = 2^m$, $m \in \mathbb{N}$ is defined as

$$|\phi_1\rangle \otimes |\phi_2\rangle = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \alpha_i \beta_j |i, j\rangle, \tag{2}$$

where $|i, j\rangle$ is an $(n+m)$-qubit state with first n qubits being the state $|i\rangle$ and the last m qubits being the state $|j\rangle$. When there is no ambiguity, $|\phi_1\rangle \otimes |\phi_2\rangle$ can be abbreviated as $|\phi_1\rangle|\phi_2\rangle$.

Quantum Oracles. To study a Boolean function $f\colon \{0,1\}^n \to \{0,1\}^m$, we need to gain its access. Classically, a standard setting is to being able to *query* the function, in the sense that if we input an $x \in \{0,1\}^n$, we will get the output $f(x) \in \{0,1\}^m$. In quantum computing, the counterpart is a quantum query, which is instantiated by a *quantum oracle*. Specifically, the function f is encoded as an oracle U_f such that for any $x \in \{0,1\}^n$, $z \in \{0,1\}^m$,

$$U_f |x\rangle |z\rangle = |x\rangle |z \oplus f(x)\rangle, \tag{3}$$

where $\oplus$ is the plus modulo 2. Note that a quantum query to the oracle is stronger than a classical query in the sense that the quantum query can be applied to a state in *superposition*: For an input state $\sum_i c_i |x_i\rangle |z_i\rangle$ with $\sum_i |c_i|^2 = 1$, the output state is $\sum_i c_i |x_i\rangle |z_i \oplus f(x_i)\rangle$; measuring this state gives x_i and $z_i \oplus f(x_i)$ with probability $|c_i|^2$. A classical query for x can be regarded as the special setting with $c_1 = 1$, $x_1 = x$, $z_1 = 0^m$, and $c_i = 0$ for all other i.

Quantum Gates. Similar to classical computing that can stem from logic synthesis with AND, OR, and NOT, quantum computing is also composed of basic quantum gates. For instance, the Hadamard H is the matrix $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, satisfying $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. In general, an n-qubit quantum gate is a unitary matrix from $\mathbb{C}^{2^n \times 2^n}$.

Quantum Circuit Diagram. A quantum algorithm is composed of a series of quantum gates. By default, a quantum algorithm starts from the all-0 state $|0^n\rangle$. A quantum algorithm can be illustrated by its quantum gate diagram, drawn from left to right. The initial all-0 state is placed at the left side of the diagram. After that, whenever we apply a quantum gate, it is placed on the corresponding qubits, from left to right. At the end of the quantum gates, we need to measure and read the outputs, and these measurements are placed at the right side of the diagram. See Figure 4 for the quantum gate diagram of Simon's algorithm [Simon, 1997].

Superdense Coding. Superdense coding [Bennett and Wiesner, 1992] is a quantum communication protocol that allows Alice to transmit two classical bits of information to Bob by sending only one qubit, given that they share a pair of entangled qubits. The protocol can be divided into five steps:

1. **Preparation:** Charlie prepares a maximally entangled Bell state, such as $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.
2. **Sharing:** Charlie sends the qubit 1 to Alice and the qubit 2 to Bob. Alice and Bob can be separated by an arbitrary distance.

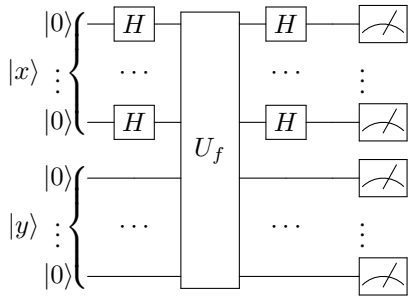

Figure 4: Quantum gate diagram of Simon's algorithm.

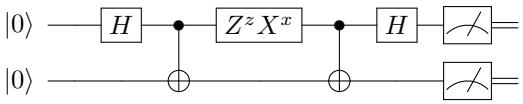

Figure 5: Quantum circuit diagram for superdense coding.

3. **Encoding:** Depending on the two classical bits $zx \in \{00, 01, 10, 11\}$ that Alice wants to send, she applies the corresponding quantum gate operation to her qubit, transforming the Bell state $|\beta_{00}\rangle$ into one of the four Bell states:

$$|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \text{ if } zx = 00$$
$$|\beta_{01}\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \text{ if } zx = 01$$
$$|\beta_{10}\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \text{ if } zx = 10$$
$$|\beta_{11}\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \text{ if } zx = 11$$

   Alice achieves these transformations by applying the operation $Z^z X^x$ to her qubit, where Z is the phase-flip gate, X is the bit-flip gate. Specifically:

   - If $zx = 00$, Alice applies $Z^0 X^0 = I$ (identity gate).
   - If $zx = 01$, Alice applies $Z^0 X^1 = X$ (bit-flip gate).
   - If $zx = 10$, Alice applies $Z^1 X^0 = Z$ (phase-flip gate).
   - If $zx = 11$, Alice applies $Z^1 X^1 = ZX = iY$ gate.

4. **Sending:** Alice sends her qubit to Bob through a quantum channel.

5. **Decoding:** Bob applies a CNOT gate followed by a Hadamard gate to the two qubits, transforming the entangled state into the corresponding computational basis state $|zx\rangle$. By measuring the qubits, Bob obtains the two classical bits zx sent by Alice.

Superdense coding exploits the properties of quantum entanglement to transmit two classical bits of information using only one qubit. The quantum circuit diagram for superdense coding is shown in Figure 5.

Quantum Teleportation. Quantum teleportation [Bennett et al., 1993] is a technique for transferring quantum information from a sender (Alice) to a receiver (Bob) using shared entanglement and classical communication. The protocol can be described as follows:

1. **Preparation:** Telamon prepares a maximally entangled Bell state, such as $|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

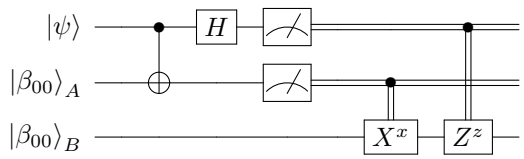

Figure 6: Quantum circuit diagram for quantum teleportation.

2. **Sharing:** Alice has qubit 1 in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, which she wants to teleport to Bob. Telamon shares qubit 2 with Alice and qubit 3 with Bob, creating the shared entangled state $|\beta_{00}\rangle_{23}$.
3. **Encoding:** Alice wants to teleport an unknown quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to Bob. She applies a CNOT gate to qubits 1 and 2, with qubit 1 as the control and qubit 2 as the target. Then, she applies a Hadamard gate to qubit 1. The resulting state of the three-qubit system is:

$$|\Psi\rangle = \frac{1}{2}[|\beta_{00}\rangle(\alpha|0\rangle + \beta|1\rangle) + |\beta_{01}\rangle(\alpha|1\rangle + \beta|0\rangle) + |\beta_{10}\rangle(\alpha|0\rangle - \beta|1\rangle) + |\beta_{11}\rangle(\alpha|1\rangle - \beta|0\rangle)].$$

4. **Measurement:** Alice measures qubits 1 and 2 in the Bell basis and obtains one of four possible outcomes: $|\beta_{00}\rangle$, $|\beta_{01}\rangle$, $|\beta_{10}\rangle$, or $|\beta_{11}\rangle$. This measurement collapses the three-qubit state into one of the following:

$$|\beta_{00}\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$$
$$|\beta_{01}\rangle \otimes (\alpha|1\rangle + \beta|0\rangle)$$
$$|\beta_{10}\rangle \otimes (\alpha|0\rangle - \beta|1\rangle)$$
$$|\beta_{11}\rangle \otimes (\alpha|1\rangle - \beta|0\rangle)$$

5. **Classical Communication:** Alice sends the result of her measurement (two classical bits) to Bob via a classical channel.
6. **Reconstruction:** Depending on the classical information received from Alice, Bob applies the operation $Z^z X^x$ to qubit 3, where z and x correspond to the two classical bits sent by Alice:
   - If Alice measured $|\beta_{00}\rangle$, she sends $zx = 00$, and Bob applies $Z^0X^0 = I$ (identity operation).
   - If Alice measured $|\beta_{01}\rangle$, she sends $zx = 01$, and Bob applies $Z^0X^1 = X$ (bit-flip).
   - If Alice measured $|\beta_{10}\rangle$, she sends $zx = 10$, and Bob applies $Z^1X^0 = Z$ (phase-flip).
   - If Alice measured $|\beta_{11}\rangle$, she sends $zx = 11$, and Bob applies $Z^1X^1 = ZX = iY$ (bit-flip and phase-flip).

   After applying the appropriate operation, Bob's qubit 3 will be in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, which is the original state that Alice wanted to teleport.

The quantum circuit diagram for quantum teleportation is shown in Figure 6.

Quantum Key Distribution. Quantum key distribution (QKD) [Bennett and Brassard, 1984] is a secure communication protocol that allows two parties, Alice and Bob, to produce a shared random secret key, which can then be used to encrypt and decrypt messages. The security of QKD is based on the fundamental principles of quantum mechanics that measuring a qubit can change its state. One of the most well-known QKD protocols is the BB84 protocol, which works as follows:

1. Alice randomly generates a bit string and chooses a random basis (X or Z) for each bit. She then encodes the bits into qubits using the chosen bases and sends them to Bob through a quantum channel.
2. Bob measures the received qubits in randomly chosen bases (X or Z) and records the results.

3. Alice and Bob communicate over a public classical channel to compare their basis choices. They keep only the bits for which their basis choices coincide and discard the rest.
4. Alice and Bob randomly select a subset of the remaining bits and compare their values. If the error rate is below a certain threshold, they conclude that no eavesdropping has occurred, and the remaining bits can be used as a secret key. If the error rate is too high, they abort the protocol, as it indicates the presence of an eavesdropper (Eve).

The security of the BB84 protocol relies on the fact that any attempt by Eve to measure the qubits during transmission will introduce detectable errors, alerting Alice and Bob to the presence of an eavesdropper.

C Additional Experiment Results

In this section, we include detailed analysis of the experiments and additional experiment results. In Section C.1, we introduce the metrics: BLEU score, verification score, and byte perplexity, and provide a detailed analysis for the experiments on BLEU and verification score. In Section C.2, we include additional experimental results. In Section C.3, we present concrete cases of typical patterns observed in model outputs.

C.1 Metrics

BLEU Score. Bilingual Evaluation Understudy (BLEU) score is a metric used to evaluate the quality of machine-translated text compared to human-translated text. It measures how close the machine translation is to one or more reference translations. The BLEU score evaluates the quality of text generated by comparing it with one or more reference texts. It does this by calculating the n-gram precision, which means it looks at the overlap of n-grams (contiguous sequences of n words) between the generated text and the reference text. Originally the BLEU score ranges from 0 to 1, where 1 indicates a perfect match with the reference translations. Here rescaling the score makes it ranges from 0 to 100.

The BLEU score, originally designed for machine translation, can also be effectively used for evaluating algorithm generation tasks. Just as BLEU measures the similarity between machine-translated text and human reference translations, it can measure the similarity between a generated algorithm and a gold-standard algorithm. This involves comparing sequences of tokens to assess how closely the generated output matches the reference solution. In the context of algorithm generation, n-grams can represent sequences of tokens or operations in the code. BLEU score captures the precision of these n-grams, ensuring that the generated code aligns closely with the expected sequences found in the reference implementation.

The formula for BLEU score is given by:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right).$$

where BP is the acronym for brevity penalty, w_n is the weight for the n-gram precision (typically $\frac{1}{N}$ for uniform weights), p_n is the precision for n-grams. BP is calculated as:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c \leq r \end{cases}.$$

where c is the length of the generated text and r is the length of the reference text. Furthermore, n-gram precision p_n is calculated as:

$$p_n = \frac{\sum_{C \in \text{Candidates}} \sum_{n-\text{gram} \in C} \min(\text{Count}(n-\text{gram in candidate}), \text{Count}(n-\text{gram in references}))}{\sum_{C \in \text{Candidates}} \sum_{n-\text{gram} \in C} \text{Count}(n-\text{gram in candidate})}.$$

This formulation ensures that the BLEU score takes into account both the precision of the generated n-grams and the overall length of the translation, providing a balanced evaluation metric.

In our experiments, the BLEU scores for various quantum algorithm design tasks are illustrated in Figure 3(a). This figure not only displays the average performance of each model but also highlights the differences in performance across individual quantum algorithm tasks. The first notable observation is that the figure clearly demonstrates the varying levels of difficulty among quantum algorithms. For example, models achieve higher BLEU scores on tasks such as Bernstein-Vazirani and Deutsch-Jozsa, whereas they perform significantly worse on tasks like Grover, phase estimation, and quantum Fourier transform. This indicates that the former tasks are considerably easier than the latter ones. Another significant observation is that most models score higher in a five-shot prompt compared to a one-shot prompt, which confirms the large language models' ability to improve performance through contextual learning.

Similar patterns are observed in oracle construction tasks, as illustrated in Figure 3(b). The figure highlights that the Diffusion Operator task is notably more challenging than the Grover oracle construction task. Interestingly, we found that adding more in-context examples actually reduced the performance of the Qwen 2.5 and DeepSeek-R1 models. This decline in performance could be attributed to the significant differences between each oracle construction task, which may be too out-of-distribution. Consequently, the additional examples might cause the models to overfit to the specific examples provided in the context, rather than generalizing well across different tasks.

Detailed Analysis of Verification Score. In addition to evaluating the BLEU score, we conducted an experiment to measure the correctness of the machine-generated algorithms, and the results are shown in Table 1. By running a verification function, we discovered that phase estimation and the swap test are significantly more challenging than other problems, leading most models to score -1 (indicating they cannot even generate the correct syntax). Notably, the BLEU score for the swap test is above average compared to other algorithms, yet almost none of the models produced a correct algorithm. This discrepancy highlights a critical limitation of using BLEU as a metric for algorithm evaluation. BLEU measures average similarity, but even a single mistake in an algorithm can render it entirely incorrect, thus failing to capture the true accuracy and functionality of the generated algorithms. Another important finding is that in a five-shot setting, GPT-4 and GPT-3.5 surpass all other models by a large margin. This demonstrates their exceptional capabilities, particularly in long-context comprehension and in-context learning. These models not only excel in understanding and generating text based on minimal examples but also maintain high performance over extended sequences, highlighting their advanced architecture and training methodologies.

As variational algorithms with parametric quantum circuits, VQE and QAOA require specifically designed metrics. For VQE, we compare the energy obtained from the machine-generated ansatz with the ground truth and compute a correctness score as follows:

$$1 - \frac{|E_{\text{LLM}} - E_{\text{expected}}|}{|E_{\text{expected}}|} \tag{4}$$

This ratio-based metric is used because VQE optimizes in a continuous space, where solutions are approximations rather than exact values. In contrast, for QAOA, when applied to solving the MaxCut problem in a discrete space, is directly evaluated against the ground truth partition, with 0 or 1 correctness score.

Since quantum algorithms based on parametric quantum circuits often share a common structure in their variational optimization process, a refined verification function is required to evaluate both the quantum ansatz and its associated classical optimization step. Therefore, the final verification score is decomposed into two components: one for the quantum circuit (QASM) generation and another for the optimization code (Python implementation). The evaluation criteria are as follows:

- **QASM Syntax Check:**
  - If the machine-generated QASM contains syntax errors, the score is -1.
- **Optimization Code Check:**
  - If the QASM is valid but the Python code has syntax errors, the QASM output is evaluated using a ground truth implementation of the optimization code.
  - If the result matches the ground truth, the score is $0.5 \times correctness\_score$.
  - If the result is incorrect, the score is 0.
  - If further syntax errors occur during evaluation, the score remains -1.

We found that GPT-4 outperforms other models in ansatz design. However, most models frequently fail to generate correct optimization code, often encountering syntax errors. A detailed analysis suggests that a major source of these errors is inconsistency in qiskit versions, leading to incorrect function calls and deprecated API usage.

The verification results of the oracle construction task, as shown in Table 3, confirm our previous conclusions. In the five-shot setting, GPT-4 and GPT-3.5 consistently outperform all other models. Additionally, this table highlights the inconsistency between BLEU scores and verification scores. For instance, while the Diffusion Operator task achieves the lowest BLEU score, it is the Grover oracle construction that receives the lowest verification score. This discrepancy suggests that BLEU scores may not fully capture the performance of models in certain complex tasks, and it is necessary to include verification score as a comprehensive evaluation.

Byte Perplexity. Perplexity is a measure of how well a probability distribution or a probabilistic model predicts a sample. In the context of language models, it quantifies the uncertainty of the model when it comes to predicting the next element in a sequence. Byte perplexity specifically deals with sequences of bytes, which are the raw binary data units used in computer systems. For our purposes, we consider byte perplexity under UTF-8 encoding, a widely used character encoding standard that represents each character as one or more bytes.

For a given language model, let $p(x_i|x_{<i})$ be the probability of the i-th byte x_i given the preceding bytes $x_{<i}$. If we have a sequence of bytes $x = (x_1, x_2, \ldots, x_N)$, the perplexity $\mathrm{PPL}(x)$ of the model on this sequence is defined as:

$$\mathrm{PPL}(x) = 2^{-\frac{1}{N}\sum_{i=1}^{N} \log_2 p(x_i|x_{<i})}.$$

A notable feature of byte perplexity is that, it does not rely on any specific tokenizer, making it versatile for comparing different models. Therefore, byte perplexity can be used to measure the performance in quantum algorithm generation tasks. In such tasks, a lower byte perplexity indicates a better-performing model, as it means the model is more confident in its predictions of the next byte in the sequence.

C.2 Additional Experimental Results

Byte perplexity (PPL) scores. The Byte Perplexity results, shown in Figure 7, provide valuable insights into the performance of our model. Evaluated in a zero-shot setting, byte perplexity trends closely mirror those observed with BLEU scores. This alignment suggests that our model's predictive capabilities are consistent across Perplexity and BLEU evaluation metrics. Specifically, in the context of quantum algorithm design tasks, the results indicate that the Bernstein-Vazirani and Deutsch-Jozsa algorithms are relatively straightforward for the model, whereas the Simon algorithm presents greater difficulty. This differentiation highlights the varying levels of complexity inherent in these quantum algorithms.

Oracle construction. The details are presented in Table 3. GPT-4o and GPT-3.5 consistently outperform other models, with both showing substantial improvements under few-shot prompting. GPT-4o achieves the highest overall performance, raising its average score from -0.3912 (1-shot) to -0.1245 (5-shot). GPT-3.5 follows closely, improving from -0.5910 to -0.2474. They are the only models capable of generating partially correct solutions with positive scores for challenging tasks such as the diffusion operator, showcasing strong in-context learning and generalization capabilities of the GPT series. In contrast, Qwen 2.5 struggles to generalize, with only marginal improvement from -0.6216 to -0.5258 and persistent failures on advanced tasks like Grover and Generalized-Simon. DeepSeek-R1 performs the worst overall, with highly negative scores in both settings. Its long-chain reasoning often leads to outputs that exceed the maximum context length, resulting in truncated or invalid circuits, highlighting the inefficiency of reasoning models for oracle construction tasks.

Temperature. Regarding the counter-intuitive phenomenon where the performance on Clifford and universal random circuits decreases after fine-tuning, we conducted additional experiments and fine-tuned the model on 4,800 samples specifically for the Clifford task. Upon closer inspection, we observed that the model more frequently generated outputs with infinite loops and increased

Table 3: Benchmarking oracle construction in verification function scores.

Model	Shot	Bernstein-Vazirani	Deutsch-Jozsa	Diffusion-Operator	Grover	Simon	Clifford	Universal	Generalized-Simon (multi-str)	Avg
gpt4o	1	0.3600 (±0.0659)	0.1600 (±0.0801)	-1.0000 (±0.0000)	-0.9540 (±0.0323)	-0.4348 (±0.0542)	-0.4348 (±0.0224)	-0.1144 (±0.0341)	-0.7188 (±0.0808)	-0.3912
gpt4o	5	0.5400 (±0.0521)	0.3700 (±0.0677)	0.0769 (±0.2878)	-0.9770 (±0.0230)	-0.1739 (±0.0453)	0.1052 (±0.0210)	-0.1111 (±0.0361)	-0.6250 (±0.0870)	-0.1245
Llama3	1	-0.7600 (±0.0571)	-0.7000 (±0.0595)	-0.8462 (±0.1538)	-0.9770 (±0.0230)	-0.8261 (±0.0397)	-0.1862 (±0.0349)	-0.1424 (±0.0338)	-0.8438 (±0.0652)	-0.6602
Llama3	5	0.0300 (±0.0771)	-0.3400 (±0.0807)	-1.0000 (±0.0000)	-0.9310 (±0.0394)	-0.3587 (±0.0503)	-0.1348 (±0.0325)	-0.1572 (±0.0329)	-0.0313 (±0.0951)	-0.3654
gpt3.5	1	-0.1000 (±0.0859)	-0.1100 (±0.0764)	-1.0000 (±0.0000)	-0.9540 (±0.0323)	-0.4130 (±0.0561)	0.0650 (±0.0178)	0.0538 (±0.0190)	-0.9688 (±0.0313)	-0.5910
gpt3.5	5	0.2700 (±0.0723)	0.1300 (±0.0734)	0.2308 (±0.2809)	-0.7701 (±0.0688)	-0.3043 (±0.0482)	0.0816 (±0.0163)	0.0723 (±0.0159)	-0.5625 (±0.0891)	-0.2474
Qwen 2.5	1	-0.5100 (±0.0689)	-0.5500 (±0.0687)	-0.8462 (±0.1538)	-0.8391 (±0.0587)	-0.9891 (±0.0109)	-0.1065 (±0.0345)	-0.1318 (±0.0337)	-1.0000 (±0.0000)	-0.6216
Qwen 2.5	5	-0.0900 (±0.0889)	-0.4000 (±0.0739)	-0.3846 (±0.2665)	-0.8391 (±0.0587)	-0.8913 (±0.0326)	-0.2895 (±0.0442)	-0.3434 (±0.0415)	-0.9688 (±0.0313)	-0.5258
DeepSeek-R1	1	-0.8900 (±0.0424)	-0.9800 (±0.0141)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-1.0000 (±0.0000)	-0.8885 (±0.0000)	-0.8644 (±0.0232)	-1.0000 (±0.02589)	-0.9529
DeepSeek-R1	5	-0.6700 (±0.0697)	-0.7900 (±0.0518)	-0.8462 (±0.1538)	-0.9310 (±0.0394)	-0.9565 (±0.0214)	-0.4355 (±0.0701)	-0.5496 (±0.0375)	-0.8125 (±0.0369)	-0.7489

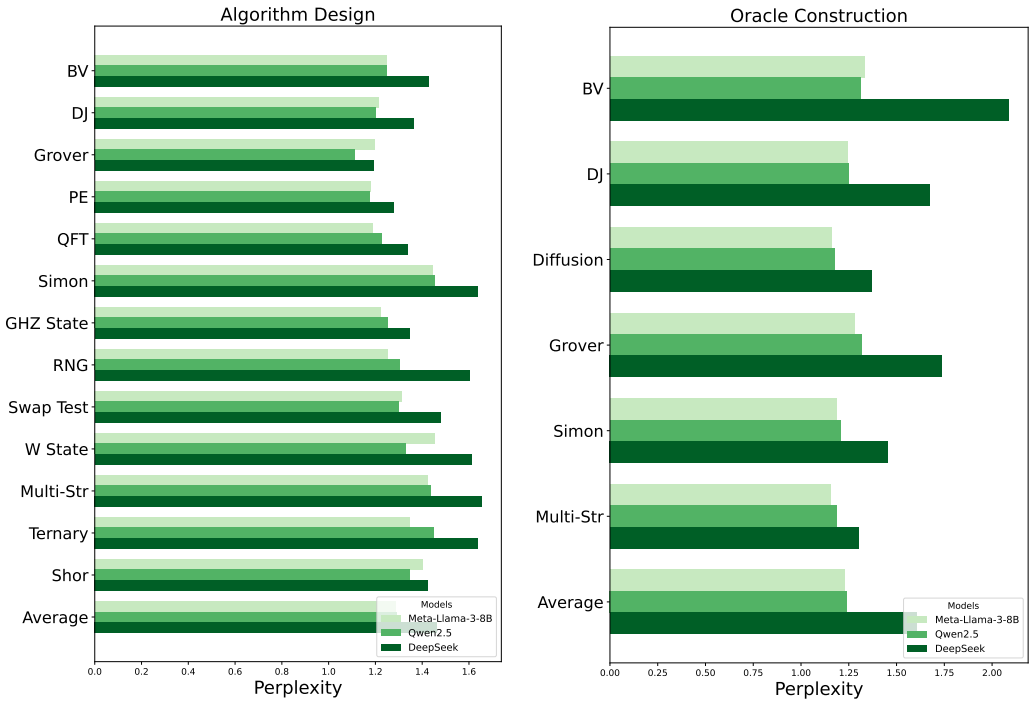

Figure 7: Benchmarking algorithm design and oracle construction in perplexity scores.

monotony, often producing repetitive gate patterns and repeatedly cycling over the same qubit after fine-tuning. We further conducted experiments with different "temperature" parameters, which control the randomness of predictions. Formally, let $T > 0$ be the temperature, z_i be the raw score for token i, the probability for token i is computed as $p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$. Typically, lower temperatures make the model more conservative, while higher temperatures flatten the distribution, increasing the likelihood of generating originally less probable sequences. The results are shown in Table 4:

Table 4: Clifford Model Fine-Tuning Results Across Different Temperature Settings.

Model	Setting	Temperature	BLEU	Verification
Llama3	few-shots (5)	0	13.3796 (±0.9508)	-0.6582 (±0.0360)
		0.2	12.5688 (±0.8276)	-0.6526 (±0.0372)
		1	53.0431 (±3.8422)	-0.1914 (±0.0361)
Llama3	finetune	0	7.6261 (±0.3433)	-0.8895 (±0.0247)
		0.2	13.8714 (±0.6536)	-0.7873 (±0.0306)
		1	32.5241 (±2.0548)	-0.2072 (±0.0358)

One possible explanation for this counter-intuitive result lies in the challenge of encoding quantum state vectors within a language model. In the problem description, the target quantum state is represented by a complex vector with four decimal places of precision, where the dimension scales as with the number of qubits . It is a well-known fact that LLMs generally struggle with very long floating-point numbers, which might contribute to the observed performance decline.

Another potential reason could be overfitting during fine-tuning, particularly for tasks that require high output diversity. The varying degrees of intrinsic difficulty and the amount of relevant pre-training knowledge across different tasks likely played a role. Oracle constructions are relatively simple for the model to learn. For example, in the Bernstein-Vazirani algorithm, the model only needs to apply a CNOT gate at positions corresponding to '1' bits. In contrast, the random circuits in the Clifford and Universal tasks involve more general and complex quantum state transformations, making them significantly more challenging. These tasks are also less common during pre-training, which could have hindered the model's ability to generalize without overfitting. This challenge is one of the reasons we initially considered a few-shot learning approach to be suitable.

While these are plausible hypotheses, we acknowledge that further investigation is required to draw definitive conclusions. We consider this an intriguing topic that warrants additional research.

C.3 Case Studies

After carefully examining the model's output, we observed several interesting patterns. We present a series of case studies to illustrate these observations and provide possible explanations.

C.3.1 Improvision Error

Low Score for GPT-4o in One-Shot Setting. At first glance, it is surprising that GPT-4o performs poorly on many quantum algorithms in the algorithm design task in the one-shot setting compared to Llama3-8B. Given that Llama3-8B has a relatively smaller parameter scale, the results should have been the other way around. A closer examination of the model's output reveals the potential reason: while Llama3-8B closely mimics the input examples, GPT-4o tends to improvise, resulting in outputs that are not well captured by the current syntax support. Here are several concrete examples.

This is the OpenQASM 3.0 code output for the W state with $n = 7$. In this code, GPT-4o uses the advanced "for" loop syntax newly introduced in OpenQASM 3.0 to create the circuit. Although the code fails to produce the W state, it is syntactically correct. However, the Qiskit.qasm3 import module, which converts OpenQASM 3.0 files to QuantumCircuit objects and is used in our verification function to check the correctness of the syntax of output OpenQASM codes, is still in the experimental stage and does not support many of OpenQASM 3.0's advanced features. As a result, GPT-4o's use of these features causes the code to fail syntax validation, resulting in a score of -1.

```
OPENQASM 3.0;
include "stdgates.inc";
qubit[7] q;
h q[0];
for i in[1:6] {
    cx q[i-1], q[i];
}
```

Listing 8: OpenQASM 3.0 Code output by GPT-4o for W state with $n = 7$.

Here is another example where GPT-4o decides to assign novel names to its qubit registers, leading to a conflict in the symbol table in Scope.GLOBAL. If we substitute all the registers x, y, and s with new names, the code can pass syntax validation successfully and is close to the correct solution.

```
OPENQASM 3.0;
include "stdgates.inc";
include "oracle.inc";
bit[9] s;
qubit[10] x;
qubit[11] y;
h x[0];
h x[1];
```

```
h x[2];
h x[3];
h x[4];
h x[5];
h x[6];
h x[7];
h x[8];
Oracle x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], y;
h x[0];
h x[1];
h x[2];
h x[3];
h x[4];
h x[5];
h x[6];
h x[7];
h x[8];
measure x[0] -> s[0];
measure x[1] -> s[1];
measure x[2] -> s[2];
measure x[3] -> s[3];
measure x[4] -> s[4];
measure x[5] -> s[5];
measure x[6] -> s[6];
measure x[7] -> s[7];
measure x[8] -> s[8];
```

Listing 9: OpenQASM 3.0 Code output by GPT-4o for Bernstein-Vazirani Problem with $n = 9$.

Since Llama3-8B tends to follow the provided example more closely, it avoids such kind of errors. These factors may help explain why GPT-4o performs poorly in the one-shot setting compared to Llama3-8B.

High Score for GPT-4o in Five-Shot Setting. In contrast to its relatively low score in the one-shot setting, GPT-4o achieves the best overall performance in the five-shot setting for both oracle construction and algorithm design tasks.

Unlike in the one-shot setting, where GPT-4o frequently uses advanced OpenQASM 3.0 features, it learns from the examples in context and uses simpler syntax in the five-shot setting. For the W state task, here is an output in the five-shot setting:

```
OPENQASM 3.0;
include "stdgates.inc";
qubit[3] q;
h q[0];
cx q[0], q[1];
h q[1];
cx q[1], q[2];
h q[2];
```

Listing 10: OpenQASM 3.0 Code output by GPT-4o for W state with $n = 3$.

This output avoids the use of "for" loops and successfully passes the syntax validation test, although it still cannot generate the W state correctly.

In addition to adapting to plain syntax through in-context learning, GPT-4o achieves outstanding performance on more complicated tasks such as phase estimation. Here is the model output of GPT-4o on the phase estimation task with qubit number $n = 2$.

```
OPENQASM 3.0;
include "stdgates.inc";
include "oracle.inc";
bit[2] c;
qubit[2] q;
```

```
Psi q[0];
h q[1];
CU_0 q[0], q[1];
h q[1];
c[0] = measure q[1];
c[1] = measure q[0];
```

Listing 11: OpenQASM 3.0 Code output by GPT-4o for Phase Estimation with $n = 2$.

```
from Qiskit import transpile


def run_and_analyze(circuit, aer_sim):
    """Run the circuit and analyze the result."""
    circ = transpile(circuit, aer_sim)
    result = aer_sim.run(circ, shots=1).result()
    counts = result.get_counts()
    phase_str = list(counts.keys())[0]
    phase = int(phase_str, 2) / 2**2
    return phase
```

Listing 12: Post-processing code output by GPT-4o for Phase Estimation with $n = 2$.

This suite of OpenQASM 3.0 circuits and post-processing functions successfully outputs the phase within the required precision for the test case, resulting in an impressive verification score of 1.0. Despite the small number of qubits and differences from the reference implementation, the accuracy achieved is noteworthy.

These phenomena reflect that GPT-4o has impressive in-context learning abilities and overall better capabilities in designing and implementing quantum algorithms.

C.3.2 Counting Error

If we ask the GPT4o model to implement the code of Bernstein-Vazirani Problem for a given secret string, for example $s = 000101$, the model fails to apply the CX gate on the correct qubits.

```
OPENQASM 3.0;
include "stdgates.inc";

gate Oracle _gate_q_0, _gate_q_1, _gate_q_2, _gate_q_3, _gate_q_4,
    ↪ _gate_q_5, _gate_q_6 {
  // Apply CX gates according to the secret string "000101"
  cx _gate_q_2, _gate_q_6;
  cx _gate_q_5, _gate_q_6;
}

bit[6] c;
qubit[7] q;

// Apply Hadamard gates to the first 6 qubits
h q[0];
h q[1];
h q[2];
h q[3];
h q[4];
h q[5];

// Prepare the ancilla qubit in the |1> state and apply a Hadamard
    ↪ gate
x q[6];
h q[6];

// Call the oracle
Oracle q[0], q[1], q[2], q[3], q[4], q[5], q[6];
```

```
// Apply Hadamard gates to the first 6 qubits again
h q[0];
h q[1];
h q[2];
h q[3];
h q[4];
h q[5];

// Measure the first 6 qubits
c[0] = measure q[0];
c[1] = measure q[1];
c[2] = measure q[2];
c[3] = measure q[3];
c[4] = measure q[4];
c[5] = measure q[5];
```

Listing 13: OpenQASM code for Bernstein-Vazirani Problem with s = 000101 in few-shot setting.

We reminded the model of the mechanism with the following prompt:

Reminder Prompt

Remember that in the construction of oracles for Bernstein-Vazirani Problem, you should apply cx gates where the control qubits are where the string bit equals 1, and the target qubit is the ancillary qubit.

There was no improvement. We further asked the model about the indices of the bits in the string which equal 1. It provided the following answer:

GPT4o's answer for counting indices

In the secret string s="000101", the bits that equal 1 are found at the following indices (0-based):
Index 2
Index 5
These indices correspond to the positions where the string has a '1'.

This reveals the incapability for GPT models to count the bit of 1 and apply corresponding gates to construct the oracle.

C.3.3 Data Contamination

In our experiments, we observed an interesting phenomenon of performance discrepancy between implementing quantum algorithms with Qiskit codes for the general setting and with OpenQASM codes for a concrete case.

We provided the following prompt:

Prompt for Bernstein-Vazirani Problem with Qiskit

Please write the code with Qiskit implementing Bernstein-Vazirani Algorithm with qubit number n = 5.

The model answered with the following text and codes:

Text Answer by GPT4o

Here's a Python code using Qiskit to implement the Bernstein-Vazirani algorithm with $n = 5$ qubits. The secret string used in this example is $s = 10101$ (you can change this to any string of your choice).

```
from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram

# Secret string s for the Bernstein-Vazirani problem
s = '10101'

# Number of qubits
n = len(s)

# Create a Quantum Circuit with n+1 qubits (one ancillary qubit)
bv_circuit = QuantumCircuit(n+1, n)

# Apply Hadamard gates to all qubits except the ancillary qubit
bv_circuit.h(range(n))

# Apply X and Hadamard gate to the ancillary qubit
bv_circuit.x(n)
bv_circuit.h(n)

# Construct the Oracle
for i, bit in enumerate(s):
    if bit == '1':
        bv_circuit.cx(i, n)

# Apply Hadamard gates to the first n qubits
bv_circuit.h(range(n))
# Measure the first n qubits
bv_circuit.measure(range(n), range(n))
# Visualize the circuit
bv_circuit.draw('mpl')

# Run the circuit using Qiskit Aer simulator
simulator = Aer.get_backend('qasm_simulator')
t_qc = transpile(bv_circuit, simulator)
qobj = assemble(t_qc)
result = simulator.run(qobj).result()

# Get the result counts and plot the histogram
counts = result.get_counts()
plot_histogram(counts)
```

Listing 14: Qiskit code for Bernstein-Vazirani Problem generated by GPT4o in the zero-shot setting.

This code, although adopts an older version of Qiskit syntax, can be compiled and run successfully with the correct answer. Similar performances can be observed with OpenQASM codes:

```
// OpenQASM 2.0
OPENQASM 2.0;
include "qelib1.inc";

// Declare 5 qubits for input and 1 ancilla qubit
qreg q[6];
creg c[5]; // classical register for measuring the result

// Step 1: Apply Hadamard gates to the input qubits
h q[0];
h q[1];
h q[2];
```

```
h q[3];
h q[4];

// Apply Hadamard to the ancillary qubit and set it to |1>
x q[5];
h q[5];

// Step 2: Oracle implementation (CNOT gates based on secret string s
    ↪ = 10110)
cx q[0], q[5]; // Secret string bit 1
// No cx for q[1] because the secret string bit is 0
cx q[2], q[5]; // Secret string bit 1
cx q[3], q[5]; // Secret string bit 1
// No cx for q[4] because the secret string bit is 0

// Step 3: Apply Hadamard gates to the input qubits again
h q[0];
h q[1];
h q[2];
h q[3];
h q[4];

// Step 4: Measure the qubits to get the result
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
measure q[4] -> c[4];
```

Listing 15: OpenQASM code for Bernstein-Vazirani Problem generated by GPT4o.

This phenomenon contrasts with the results in Appendix C.3.2. Considering that Qiskit has implemented tutorial codes for several representative quantum algorithms, the seemingly outstanding performance of quantum algorithm code generation might be a deceptive result caused by data contamination. Therefore, only a carefully designed quantum algorithm dataset can avoid the effects of data contamination, allowing for an effective evaluation of the model's genuine capability in quantum algorithm design and implementation. This dataset is also meaningful for testing general AI code generation and syntax learning, where no existing AI dataset could substitute us.