Bridging the Quantum Coding Gap: Instruction-Tuned LLMs for Qiskit

Sixu Chen
Kent State University
Kent, Ohio, USA
schen53@kent.edu

Yuqi Zhang
Kent State University
Kent, Ohio, USA
yzhan135@kent.edu

Qiang Guan
Kent State University
Kent, Ohio, USA
qguan@kent.edu

ABSTRACT

Large Language Models (LLMs) have advanced code generation ability across many domains, but often struggle with quantum code due to limited domain-specific data and inherent domain complexity. To address this issue, we focus on the Qiskit framework and fine-tune pretrained LLMs using quantum code from GitHub and datasets including OASST1 and COMMITPACKFT. More importantly, we construct instruction-style prompt/completion pairs based on real-world Qiskit code to improve alignment during fine-tuning. Experiments show that our finetuned models significantly improve quantum code generation ability, validating the effectiveness of our approach.

KEYWORDS

Quantum Computing, Large Language Models, Qiskit, Fine-tuning

1 INTRODUCTION AND MOTIVATION

Quantum computing[1] is an emerging and fast-moving paradigm area. One of the most widely used open-framework for quantum computing is Qiskit which enables the construction, simulation, and execution of quantum circuits while providing access to IBM Quantum hardware.

LLMs[2] show exceptional ability to generate human-like text and write codes. However, despite the strong capabilities of LLMs for general code assistance, there remains a notable gap between quantum computing and other common tasks. This is mainly because that the training of LLMs on writing codes heavily rely on datasets of publicly available code, however, quantum computing is comparatively scarce and heterogeneous, thus it only occupied a small portion in the massive public code datasets. This heavily constrains model's ability to learn the domain-specific APIs, execution constraints and gate semantics. As a result, code LLMs acquire limited quantum-specific knowledge and underperform on quantum programming tasks.

In this project, we extend large language models' code-generation capabilities to the quantum computing domain, with a particular focus on Qiskit. Given the substantial learning time and specialized expertise required for quantum computing, an LLM that produces high quality Qiskit-related code can lower barriers to entry and accelerate research and development.

2 METHOD

Our approach builds upon pretrained large language models (LLMs) and applies further fine-tuning for quantum code generation. The overall pipeline is illustrated in Figure 1.

To enhance natural language understanding, we adopt the OASST1 dataset[3] and retain 8,587 high-quality assistant responses after filtering[4]. To capture programming constraints, we leverage COMMITPACKFT[4], a large-scale multilingual code dataset, and extract 5,000 Python samples since Qiskit is Python-based. These steps enable the model to better align with both natural language instructions and Python-specific coding practices relevant to Qiskit.

In order to exposure the model to domain knowledge in quantum computing, we crawled 18.42 MB code which qiskit in the name from GitHub for pretraining. This dataset was used for enrich the model's understanding of quantum programming constructs and real-world usage patterns. After this step, to make the model better generalize and respond to downstream quantum-specific tasks, we construct a new Qiskit code dataset comprising 3,000 instruction-style prompt–completion pairs to perform supervised finetuning. In more detail, We crawl GitHub for Qiskit-related functions and, to focus on standalone routines, exclude methods defined within classes. For each crawled qiskit function, we use Llama-3-8B-Instruct model to generate a corresponding natural-language description of its purpose and then wrap this description as the prompt for the qiskit function. Unlike prior works that rely on synthetic code and auto-generated unit tests— which often fail to reflect true semantic correctness or practical utility— our dataset consists of real-world, human-authored quantum code. This ensures both functional relevance and execution reliability, eliminating the need for synthetic validation via unit tests.

3 EXPERIMENT

We fine-tuned Qwen2.5-Math-7B and AutoCoder_QW_7B, and evaluated them on two execution-based benchmarks: HumanEval (HE) and Quantum HumanEval (QHE)[5]. Results are reported as pass@1 scores in Table 1. All models perform significantly better on HE than QHE, revealing a clear gap between general-purpose and quantum-specific code generation, and emphasizing the need for domain adaptation in quantum programming.

Among them, qiskit-Qwen2.5-Math-7B achieves the highest pass@1 score on QHE, outperforming its base model, Qwen2.5-Math-7B. Both fine-tuned models show improvements over their respective baselines, demonstrating the effectiveness of our proposed method.

Figure 2 shows example outputs from Qwen2.5-Math-7B and qiskit-Qwen2.5-Math-7B, prompted with identical import statements, function headers, and docstrings. Notably, only qiskit-Qwen2.5-Math-7B generates the correct quantum function, which also indicates the effectiveness of our methods.

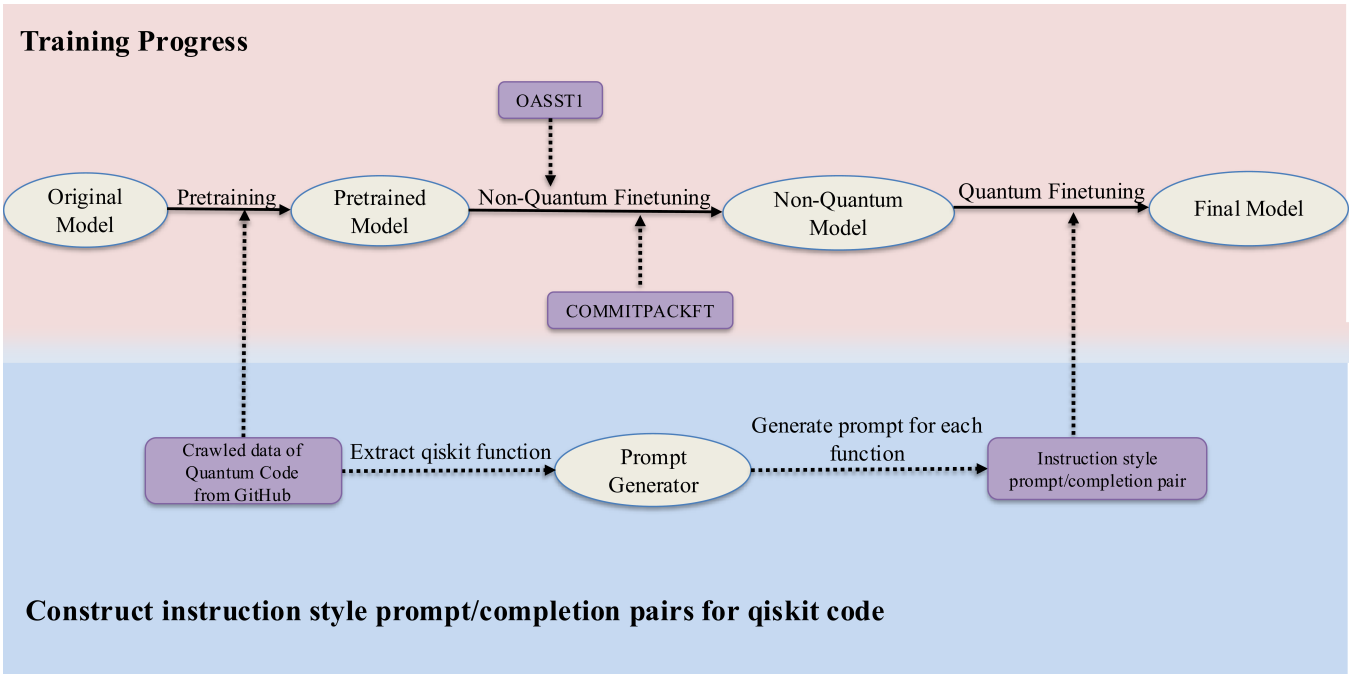

Figure 1: An overview of our training pipeline. (a) We begin with a pretrained large language model (LLM), which is first pretraining using crawled Qiskit-related quantum code from GitHub. The model is then further fine-tuned on two non-quantum datasets, OASST1 and COMMITPACKFT, to enhance general code understanding. Finally, the model undergoes quantum-specific fine-tuning using our constructed instruction-style prompt/completion pairs. (b) To construct these instruction-style pairs, we extract Qiskit functions from the crawled data, use Llama-3-8B-Instruct model to generate corresponding task descriptions, and format them into prompt/completion examples.

```
from qiskit import QuantumCircuit

def create_cz_gate()->QuantumCircuit:
       """ Design a CZ gate using only H and CNOT gates
       and return the quantum circuit.
       """
       qc = QuantumCircuit(2)
       qc.h(1)
       qc.cx(0, 1)
       qc.h(1)
       return qc
```

(a) qiskit-Qwen2.5-Math-7B

```
from qiskit import QuantumCircuit

def create_cz_gate():
       """ Design a CZ gate using only H and CNOT gates
       and return the quantum circuit.
       """
       qc = QuantumCircuit(2)
       # Step 1: Apply a Hadamard gate to the target
       qubit
       qc.h(1)
       # Step 2: Apply a CNOT gate with the control
       qubit as the control and the target qubit as the
       target
       qc.cx(0, 1)
       # Step 3: Apply a Hadamard gate to the target
       qubit again
       qc.h(1)
       # Step 4: Apply a CNOT gate with the control
       qubit as the control and the target qubit as the
       target
       qc.cx(0, 1)
       return qc
```

(b) Qwen2.5-Math-7B

Figure 2: Generated outputs from Qwen2.5-Math-7B and qiskit-Qwen2.5-Math-7B. Both models are prompted with the same import statements, function header, and Python docstring.

Table 1: Evaluation on Qiskit HumanEval (QHE) and HumanEval (HE) via pass@1 score.

Model	HE	QHE
Qwen2.5-Math-7B	0.58	0.15
qiskit-Qwen2.5-Math-7B	0.61	0.23
AutoCoder_QW_7B	0.89	0.11
qiskit-AutoCoder_QW_7B	0.71	0.18

4 CONCLUSION

In this work, we enhance LLMs for quantum code generation with a focus on Qiskit. We pretrain on quantum code from GitHub and finetune using non-quantum datasets. To enable instruction-style tuning, we construct prompt–completion pairs by generating natural language descriptions for real Qiskit functions. Our approach improves the model's ability to generate correct and meaningful quantum code.Our results highlight a performance gap between general and quantum code generation. In future work, we plan to improve the quality of quantum prompt–completion data and extend beyond Qiskit to other quantum frameworks.

REFERENCES

[1] Andrew Steane. Quantum computing. *Reports on Progress in Physics*, 61(2):117, 1998.

[2] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pages 1–13, 2024.

[3] Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681, 2023.

[4] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. In *NeurIPS 2023 workshop on instruction tuning and instruction following*, 2023.

[5] Nicolas Dupuis, Luca Buratti, Sanjay Vishwakarma, Aitana Viudes Forrat, David Kremer, Ismael Faro, Ruchir Puri, and Juan Cruz-Benito. Qiskit code assistant: Training llms for generating quantum computing code. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pages 1–4. IEEE, 2024.