Enhancing LLM-based Quantum Code Generation with Multi-Agent Optimization and Quantum Error Correction

Charlie Campbell
Department of Computing
Imperial College London
London, UK
charlie.campbell22@imperial.ac.uk

Hao (Mark) Chen
Department of Computing
Imperial College London
London, UK
hao.chen20@imperial.ac.uk

Wayne Luk
Department of Computing
Imperial College London
London, UK
w.luk@imperial.ac.uk

Hongxiang Fan
Department of Computing
Imperial College London
London, UK
hongxiang.fan@imperial.ac.uk

Abstract—**Multi-agent frameworks with Large Language Models (LLMs) have become promising tools for generating general-purpose programming languages using test-driven development, allowing developers to create more accurate and robust code. However, their potential has not been fully unleashed for domain-specific programming languages, where specific domain exhibits unique optimization opportunities for customized improvement. In this paper, we take the first step in exploring multi-agent code generation for quantum programs. By identifying the unique optimizations in quantum designs such as quantum error correction, we introduce a novel multi-agent framework tailored to generating accurate, fault-tolerant quantum code. Each agent in the framework focuses on distinct optimizations, iteratively refining the code using a semantic analyzer with multi-pass inference, alongside an error correction code decoder. We also examine the effectiveness of traditional techniques, like Chain-of-Thought (CoT) and Retrieval-Augmented Generation (RAG) in the context of quantum programming, uncovering observations that are different from general-purpose code generation. To evaluate our approach, we develop a test suite to measure the impact each optimization has on the accuracy of the generated code. Our findings indicate that techniques such as structured CoT significantly improve the generation of quantum algorithms by up to 50%. In contrast, we have also found that certain techniques such as RAG show limited improvement, yielding an accuracy increase of only 4%. Moreover, we demonstrate examples of AI-assisted quantum error prediction and correction, demonstrating the effectiveness of our multi-agent framework in reducing the quantum errors of generated quantum programs. We aim to open-source our framework, model, and dataset after the paper acceptance.**

Index Terms—**Machine Learning, Quantum Code Generation, Quantum Computing, Multi-agent Large Language Models**

I. Introduction

Recent advances in Large Language Models (LLMs) have revolutionized a wide range of AI applications, including natural language understanding, machine translation, and automated content generation [1]. Building on the exceptional natural language processing capabilities of LLMs, recent research has extended into the exploration of multi-agent frameworks, where each agent is driven by an LLM to facilitate complex interactions in complicated environments. These multi-agent frameworks form the foundation for embodied AI systems [2], paving the way for a future where AI plays a critical role in assisting and supporting daily human activities.

A key application of LLM is in automatic software development, where AI can assist developers in generating increasingly high-quality code [3]. Various LLMs tailored for code generation, such as StarCoder [4] or CodeLLama [5], have been introduced to produce high-quality code and test suites. Furthermore, recent research has explored multi-agent frameworks, such as AgentCoder [6], to enhance collaborative code generation. However, most of these efforts have focused on general-purpose programming languages like C++ and Python, with relatively less emphasis on domain-specific programming languages.

As a pioneering force in quantum computing, IBM has advanced the application of LLMs for quantum program generation, demonstrating the potential of LLMs in producing domain-specific programs [7]. To facilitate the development of LLM in quantum computing, they introduce training datasets, a customized LLM and testbench for quantum program generation. While this work lays a solid foundation for leveraging LLMs in quantum computing, several key challenges remain in optimizing LLM-assisted quantum code generation:

- The rapid pace of quantum computing advancements requires frequent updates to code libraries, resulting in a lack of high-quality, up-to-date training data. Models trained on data that are only a few months old can quickly become obsolete due to the release of new libraries or algorithms.
- Quantum programs generated by LLMs, with only standard supervised fine-tuning, tend to be error-prone. It is essential to explore the role of advanced prompt engineering in improving the reliability of LLM-assisted quantum code generation.
- Prior approaches have applied traditional code generation techniques used for general-purpose programming languages to quantum computing without accounting for domain-specific optimizations, such as Quantum Error Correction (QEC). This results in suboptimal performance. For instance, to meet the stringent standards of quantum code generation, LLMs must be capable of producing fault-tolerant quantum code using QEC techniques.

To address the aforementioned challenges, this paper proposes a novel multi-agent quantum code generation framework that integrates iterative multi-pass optimization and automatic quantum error correction. Inspired by OpenAI's *GPT-o1* [1] which emphasizes inference time optimization over training time effort, we demonstrate that multiple inference-time optimization techniques can effectively mitigate the issue introduced by limited training data. To enhance the efficiency of quantum code generation, we explore the impact of advanced prompt engineering techniques such as Chain of Thought (CoT) [8] and Retrieval-Augmented Generation (RAG) [9], providing different insights for optimizing LLM-assisted quantum code generation. Furthermore, by leveraging the domain-specific optimization potential in quantum programming, our framework incorporates a quantum error predictor to facilitate automatic quantum error correction, ensuring robust and fault-tolerant quantum code generation. Overall, this work makes the following contributions:

- We propose a novel multi-agent framework comprising three key agents: a code generation agent, a semantic analyzer, and a quantum error predictor. The framework employs a multi-pass inference strategy for iterative optimization (Section III).
- We investigate the effectiveness of different advanced prompt engineering techniques, presenting detailed experimental results that provide insights into optimizing LLM-assisted quantum code generation (Section IV).
- Our framework integrates automatic quantum error correction by leveraging Quantum Error Correction (QEC) decoders, significantly reducing noise in quantum environments (Section V-B).

II. Background and Related Work

A. LLMs and Multi-Agent Frameworks

Large Language Models (LLMs) are built upon the transformer model [10]. The most powerful and recent models, such as GPT-4 [11], are built upon the decoder-only transformer architecture, in which the output sequence is based on the previous tokens, rather than having separate encoders and decoders. Such models are used for a variety of Natural Language Processing (NLP) tasks, proving to be versatile tools that have the potential to have wide-ranging effects.

One developing area of LLM research is multi-agent frameworks. In this, multiple LLM agents are combined together to allow interaction between different LLMs, which could result in a more accurate response or allow users to use multiple input formats. [6] [12]. This has many potential uses, such as safeguarding text generation or a hierarchical chat.

B. Chain-of-Thought and Retrieval-Augmented Generation

Inference-time optimisation techniques have been shown to improve the quality of output generated by LLMs [13]. These techniques often either augment the prompt with additional information or get the model to think in different ways to better their understanding. Two promising methods are Retrieval-Augmented Generation (RAG) [9] and Chain-of-Thought prompting [8].

[1] https://openai.com/index/introducing-openai-o1-preview/

RAG involves collecting data or documents relevant to the generation task and building a vectorstore database. We then retrieve the appropriate chunks of data based on a ranking algorithm and augment the prompt with this additional data [9]. This gives the model more specific context from which to infer the answer, improving generation accuracy. Chain-of-Thought (CoT) prompting [8] encourages models to use logical reasoning when generating answers. Zero-shot CoT [14], in which the model is explicitly told to think "step by step", encourages the usage of logical reasoning. Another technique is manual CoT [8] in which a sample question and answer pair is provided to demonstrate the type of reasoning the model should use. Overall, both RAG and CoT have been shown to improve the accuracy of models at the inference stage by augmenting the context or eliciting the logical reasoning ability of the LLM.

C. Quantum Computing

Quantum computing is a rapidly growing field that shows profound potential. It uses quantum phenomena such as entanglement and superposition to allow information to be stored in an exponentially more dense manner using qubits. Whilst this information cannot be accessed in a one-to-one manner, it does allow access to unique phenomena such as interference and true parallelism, creating new algorithmic possibilities. The ability to exploit these phenomena has allowed quantum algorithms to be made for many problems that are NP or NP-complete in the classical environment. One example is Shor's algorithm for finding prime factors [15], which is a problem that is NP for classical computers. This algorithm uses superposition and interference to manipulate qubits in such a way as to find the factors in polynomial time. There are many more examples of algorithms in cryptography [16] and beyond, making quantum computing a field that has exciting possibilities.

One unique challenge posed by developing quantum algorithms is the presence of quantum noise. This is caused by factors such as thermal fluctuations and can result in qubit measurements being incorrect, meaning that experimental results often differ from theoretical results. Much work has been done in the field of quantum error correction (QEC), with many QEC codes being developed, such as the Steane code [17]. One common subset of QEC codes are surface codes [18], which encode one logical qubit onto a lattice of physical qubits. The nature of these codes means that they are topology-dependent, making it difficult to apply QEC codes in a practical development environment.

D. Related Work

LLM development is undergoing rapid improvement in the field of code generation [3]. Models have shown promising ability to understand both the syntax and semantics of a language, with popular models such as CodeLlama [5] being able to produce high quality code. These models achieve consistently high scores on coding benchmarks like HumanEval [19], which is developed to evaluate models' ability to produce code to the same standard as a human software engineer. Multi-agent frameworks have also been utilized to create a fully-fledged

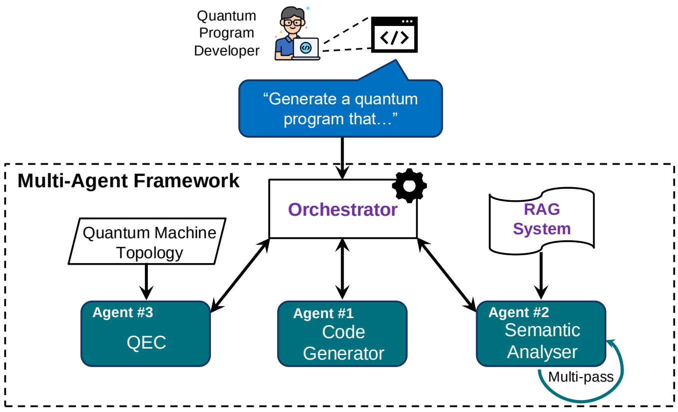

Fig. 1: Multi-Agent Framework for Quantum Code Generation

code development environment, with a framework that can produce code and generate an appropriate test suite [6].

Writing code for quantum computers poses significant challenges, as quantum algorithms must be reversible and error tolerant, which demands developers to approach problems in a different way. This creates a unique opportunity for LLMs to aid developers in code writing. Despite the accessibility of the Qiskit library [7], there is a steep learning curve for beginners to develop novel quantum algorithms. LLMs have the potential to enable developers to create accurate, fault-tolerant code without being experts in the field. However, traditional code models cannot be applied directly to the task due to the unique requirements of quantum algorithms. Quantum noise is a challenge only present in quantum systems and no model has been developed yet to take this into account during code generation. Given the proved ability of LLMs to learn the structure and semantics of classical languages, it is worth exploring their potential to generate high-quality quantum code.

III. Multi-Agent Framework and Training

A. Framework Overview

An overview of the proposed multi-agent framework is depicted in Figure 1. The framework mainly comprises three agents, each responsible for a specific task: code generation, semantic analysis, and quantum error correction.

- **Code Generation Agent**: We adopt Starcoder [4] as the base model for quantum code generation. To enhance the quality of the generated quantum program, we applied the supervised fine-tuning on Starcoder using a custom dataset containing the latest open-source Qiskit [7] code. Additionally, we introduced several optimization techniques aimed to further improve the accuracy of the code generation agent, which will be detailed in Section IV.
- **Semantic Analysis Agent**: To enhance the quantum program generated by the initial model with better semantic accuracy, we design a dedicated agent for semantic analysis. This agent can help the code generation process to mitigate the lack of knowledge available in online sources about quantum algorithms, allowing users to incrementally improve the generated code to meet their requirements.
- **QEC Decoder Generation Agent**: Given the error-prone nature of quantum computing, primarily caused by quantum noise, it is essential to incorporate Quantum Error Correction (QEC) into our code generation process to enhance the reliability and robustness of the generated quantum programs. To achieve this, we adopt a QEC code decoder to predict and reduce quantum error, allowing our framework to generate fault-tolerant code.

By designing the multi-agent framework shown in Figure 1, we are able to address the three main challenges in optimizing quantum code generation. It allows us to not only generate syntactically and semantically valid quantum code, but also ensure that we can run the generated code in a stable environment, by the support of AI-generated Error Correction Codes (ECCs).

B. Training Dataset and Test Suite

We collected our training data by scraping Github repositories with an open-source license. Due to the fast-paced nature of quantum computing, we filtered the repositories to those updated after February 2024. We found that even filtering by a date this recent still resulted in out-of-date code, including code from official Qiskit Community repositories [20]. We further filtered the retrieved Python and Jupyter Notebook files based on whether they contained a Qiskit import statement. The notebooks were then split into code and markdown tiles based on sentinel tokens [7]. After filtering, our total number of tokens was 3M. Whilst this is a small dataset, it did not affect our approach to obtaining meaningful results, as we seek to compare the effectiveness of both fine-tuning techniques and a multi-agent structure. By demonstrating that these optimization techniques and our framework provide a meaningful improvement on the generated quantum code, we can justify working on creating a more accurate code generation agent by creating a larger dataset with higher data quality.

In order to perform our experiments, we created a set of prompt-answer pairs to be passed into our multi-agent system. These prompts cover a wide range of code-generation topics.

- **Basic Code Generation**: This test covers basic code generation for the Qiskit library, allowing us to test the basic syntactic requirements of our model. The tests include basic circuit generation and simulation on quantum devices, ensuring the model can generate and run code on real-world devices.
- **Quantum Circuits and Algorithms**: It covers more advanced quantum circuits and algorithms, with a focus on well-known algorithms, such as Shor's [15] and Grover's [21] algorithms. This is intended to test advanced syntactic code generation, as well as allow for small semantic testing.
- **Advanced Concepts**: This part covers more advanced topics specific to quantum computing. For example, this section contains prompts covering quantum teleportation [22], the quantum walk algorithm [23] and quantum annealing [24]. We expect the model to have little to no knowledge of these algorithms from the base training, so this section can be used to carry out advanced semantic testing.

IV. Optimization

A. Iterative Multi-Pass Optimization

Due to the limited availability of high-quality training data for quantum programming, the quantum programs generated by the fine-tuned LLM model remain error-prone. Inspired by recent advances in OpenAI's GPT-o1, we adopt an inference-time optimization approach to generate high-quality quantum programs. To this end, we implement an iterative multi-pass quantum code generation strategy. Specifically, we created a prompt template for multi-pass inference that includes the original prompt, as well as the generated code and error trace. This new prompt is then passed into the model in the hope of fixing the error generated on the previous pass.

By using a multi-pass structure and iterating upon the previous code, we are able to solve errors as they occur. Since we can pass the incorrect code back into the model multiple times, we can also account for multiple errors, allowing the model to focus on fixing a small, singular error, rather than regenerating the entire program.

B. Quantum Error Correction Enhancement

The quantum programs generated after passing through the code generation agents and semantic analyzer do not contain any form of quantum error correction. As a result, running these codes on a physical quantum computer would contain high levels of noise, which reduces the reliability of the generated quantum programs. Therefore, it is vital to incorporate the mechanism of error correction to mitigate the potential of quantum error.

To do this, we include a third agent in the framework, which generates a decoder for a Quantum Error Correction (QEC) surface code [18], [25]. As shown in Figure 2, this agent uses the topology of the quantum device and the number of logical qubits provided to generate a decoder that allows a surface error correction code to be used when running the algorithm. This is applied after the code has been generated and does not alter its semantics, but does add additional ancilla qubits in order to generate the appropriate surface code lattice. It maps the logical qubits required onto a subset of physical qubits, which allows us to take advantage of the topology of the device to implement a surface code. By applying this code, we extend the average qubit lifetime, which means we are less likely to see qubits affected by quantum noise. This in turn reduces the amount of error present in the results, improving their accuracy.

By including this model, we are able to improve our chances of producing fault-tolerant quantum code. By using the surface code, we increase the average qubit lifetime, making the environment less error-prone. The main drawback of using a surface code is that they are topology-specific. This results in the model needing to be re-trained every time you want to run the code on a different device, which is extremely inefficient, so finding a way to make a QEC surface code that is topology-agnostic is vital to improving the framework's efficiency.

C. Prompt Engineering Optimisations

We adopted several techniques to mitigate the unique challenges posed by generating quantum code. The first was Retrieval-Augmented Generation (RAG) [9], to provide the model with knowledge of both the structure of quantum algorithms and the structure of the Qiskit library. We then implemented Chain of Thought (CoT) [8] and Structure Chain of Thought (SCoT) [26] prompting, as these have been shown to improve the accuracy of code generation. Our goal for these techniques was to improve the semantic accuracy of the code generated as we found that the lack of algorithmic knowledge available often led to syntactically correct but nonsensical code being generated. The final technique we implemented was multi-pass inference, which was intended to improve the syntactic accuracy of the code.

For our CoT and SCoT prompts, we manually created the first 5 prompts from our testing set using the same techniques demonstrated in previous code generation work [26]. For each subsequent prompt, we manually created a version in both CoT and SCoT and generated the same version using the GPT-4o model [11]. Figure 2 shows how generated CoT and SCoT prompts compare the original prompt that was provided to the model.

We collated two different RAG datasets. The first scraped the official Qiskit documentation repository on Github, taking the documentation for the latest Qiskit version, including Qiskit-adjacent libraries such as *qiskit-ibm-runtime*. The goal of this dataset is to improve the model's understanding of the library structure, reducing the amount of import errors due to deprecated features. The second consisted of a collection of guides and tutorials explaining the ideas behind and structures of a collection of quantum algorithms. The goal of this dataset is to improve the model's knowledge of algorithms, improving its semantic accuracy. We used the langchain [27] and ragatouille [28] libraries to create the augmented prompts.

‘

V. Experiments

A. Experimental Setup

We chose the StarCoder models [4] for fine-tuning since they were pre-trained on a wide corpus of languages and library files, making it suitable for adapting to new programming languages [29]. Furthermore, StarCoder models have demonstrated strong performance across a variety of coding tasks, showing their versatility and suitability for learning in new coding environments [3]. In our experiments, we selected StarCoder-3B as the main model for evaluation.

For fine-tuning, we used the *transformers* library [30] with LoRA adapter [31]. To facilitate the use of our custom dataset, we converted it into multiple chunked sets, with randomly applied Fill-in-the-Middle (FIM) transformations [32]. The chunk size was calculated upon training based on the amount of data provided. We found that the optimal FIM rate (the percentage of chunks to which a FIM transformation is applied) was 0.1. Training was conducted for 1400 steps with a batch size of 4. The learning rate was linearly increased from 0 to 3×10^{-4} over the first 100 warm-up steps and subsequently decayed using a cosine scheduler.

When testing our methods using our testing suite, each optimization method was evaluated independently. The CoT

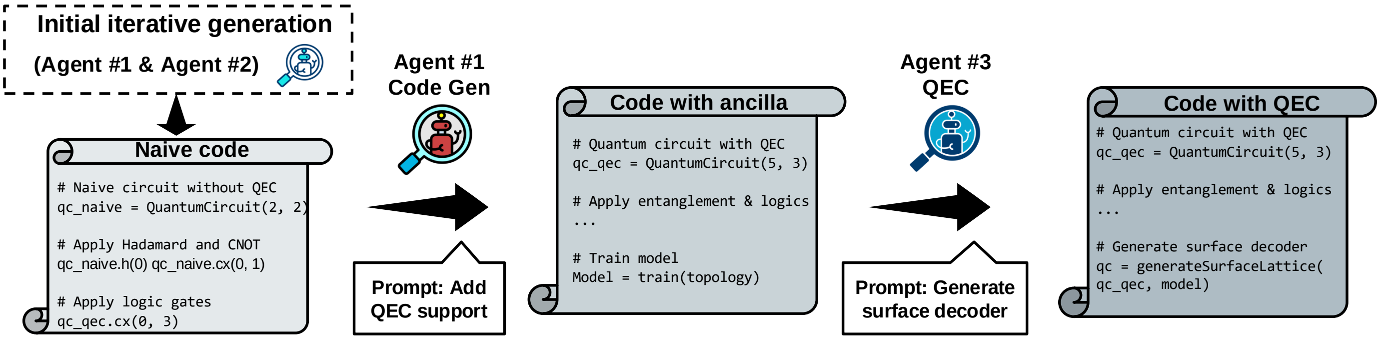

Fig. 2: Evolution of code during QEC generation

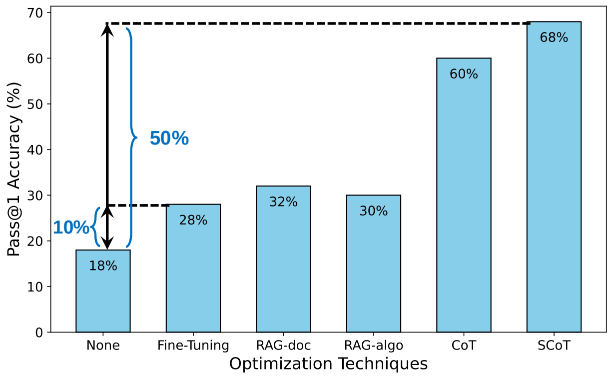

Fig. 3: The percentage of results that were semantically and syntactically valid for each technique.

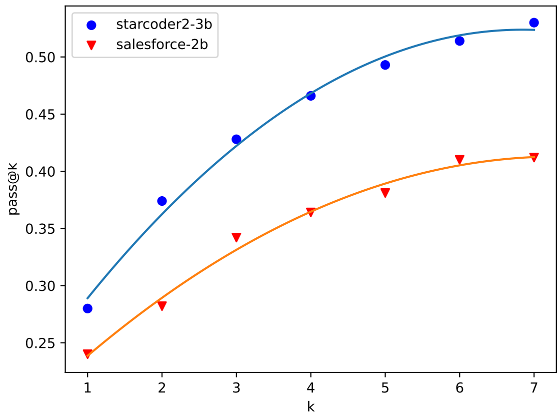

Fig. 4: pass@k values for fine-tuned models

and SCoT prompts were passed directly into the model, upon which it generated the code. To further evaluate the accuracy of the fine-tuned model with no optimizations, we used the *transformers* library [30] to create a sample of results for each prompt and applied the pass@k metric [19] to obtain our results. Additionally, it was also our intention to compare our model with the recent IBM Qiskit Assistant [7] model. However, their model is paywall-locked, so it was not possible to get a comparison, despite multiple attempts of contacting the authors.

B. Overall Accuracy Results

Using our test suite, we were able to evaluate the accuracy of applying our multi-agent framework with the optimization techniques proposed. The overall results of the experiments are shown in Figure 3. The results provided are the percentage of prompts that are both syntactically and semantically correct. By training on the dataset of Qiskit repositories, we were able to increase the pass@1 metric by 10%, up to 28% overall. Figure 4 presents the accuracy of the model using the pass@k metric for k values ranging from 1 to 7, taken by generating a sample of prompts. The graph indicates that accuracy gains diminish as the sample size increases. Specifically, beyond 5 samples per prompt, we observe less improvement in the pass@k metric, indicating diminishing returns with additional samples. To investigate the effect of different model sizes on the model's pass@k accuracy, we also evaluate Salesforce-2B for comparison. As shown in Figure 4, the accuracy of Salesforce-2B model is consistently worse than that of StarCoder-3B across different k values. Therefore, the rest of the experiments adopt StarCoder-3B for evaluation.

C. Effects of RAG, CoT, and SCoT

As shown in Figure 4, applying RAG [9] optimization had negligible impact on the overall accuracy of the model. Whilst the algorithmic knowledge showed 4% improvement relative to the documentation knowledge, there is limited real-world benefit introduced by either method due to the negligible improvement.

The results for CoT [8] and SCoT [26] show remarkable increases, with CoT leading to an increase of 32% and SCoT an increase of 40% from the fine-tuned model. The majority of this increase came from a better semantic understanding of how the algorithms are structured, suggesting that the initial training dataset did not contain many good examples of such algorithms.

D. Effects of Multi-Pass Inference and QEC

We found that applying multi-pass inference into the semantic analyzer model can improve the accuracy to 34% using triple passes. However, additional inference passes, despite incurring higher computational costs, yielded limited benefit.

The diminishing accuracy improvement is likely due to the nature of the error, which was mostly the misuse of imports or the use of deprecated code. If these issues were resolved, the results did demonstrate evidence that multi-pass inference could be used to improve the model's ability to resolve other syntactic and semantic errors.

Leveraging our multi-agent framework with the support of quantum error prediction and correction, we were able to reduce the amount of quantum error in our experiment results. Figure 5 shows an example of Grover's Algorithm [21] under a quantum noise environment, with and without the use of our framework. Compared with the results without quantum error optimization, the optimized results demonstrate a general increase in the probability of the expected quantum states, as well as a decrease in observed errors across other quantum states throughout all experiments. Although this example with surface code applied is topology-dependent, it is sufficient to demonstrate the effectiveness of our framework in reducing quantum error.

E. Observation and Future Directions

Different from LLM-assisted code generation for general-purpose programming languages, our experimental results reveal that different optimization techniques have vastly distinct effects on quantum code generation. Applying RAG enhancement resulted in only a marginal increase in pass@1 accuracy. This technique aimed to enhance the model's understanding of the library structure and knowledge of algorithms. The limited accuracy increase is likely attributable to the fact that the documentation available for Qiskit is not up to date, preventing the effective mitigation of import errors. In contrast, other techniques such as Chain-of-Thought (CoT) reasoning introduced significant accuracy improvement. This improvement is attributed to the enhanced semantic knowledge provided to the model, allowing it to correctly structure the generated quantum program. This approach likely outperformed RAG, as it allowed us to more directly inform the model's decision-making process, rather than inferring how the algorithms work from the, rather limited, dataset we provided.

One limitation encountered during our experiments was the limited dataset sizes, which can be evidenced by both the poor increase in the fine-tuned models' pass@1 accuracy and the lack of improvement from applying the RAG technique. An important direction of our future work is to collect a larger and higher-quality dataset. This presents challenges due to the lack of data online and the rapidly changing nature of the field. One potential solution is to scrape other sites, such as the official Qiskit website and forums, while supplementing the dataset with hand-built examples. Another promising direction to explore is the generation of synthetic data [33], which could further enhance the dataset and address data scarcity.

Moreover, a key area for future development is creating a topology-agnostic QEC decoder model. Currently, our work includes a model that is topology-specific, requiring retraining each time algorithms are adapted for different quantum computers. Whilst our work has shown that using a model like this can aid in producing fault-tolerant code, having a topology-specific model is not a scalable method. Developing a model capable of generating quantum error correction decoders for arbitrary topologies would significantly enhance the scalability of fault-tolerant quantum code generation.

VI. Conclusion

This paper proposes a novel multi-agent framework to provide an accurate, fault-tolerant quantum computing code generation LLM. We propose a three-agent system, with the first being a code generation based on LLM models. We investigate the effect of various conventional optimizations on our model, including Structured Chain of Thought, which showed an accuracy increase of 40%. The second agent is a semantic analyser, which uses multi-pass inference and RAG to improve the accuracy of the generated code. The results observed during the code generation experiments suggest that a model with this purpose is necessary, as the code generation process lacks significant algorithmic knowledge. These techniques could be improved by collecting a larger dataset for RAG, which could also be combined with the multi-pass inference to improve the model's ability to understand why an error occurred. The third agent is a Quantum Error Correction model that predicts QEC surface code decoders. Our results showed a decrease in the average noise present in an experiment on IBM quantum computers. Overall, our framework makes a promising step forward in creating a tool to generate fault-tolerant quantum code, which would greatly improve the accessibility of a complex field.

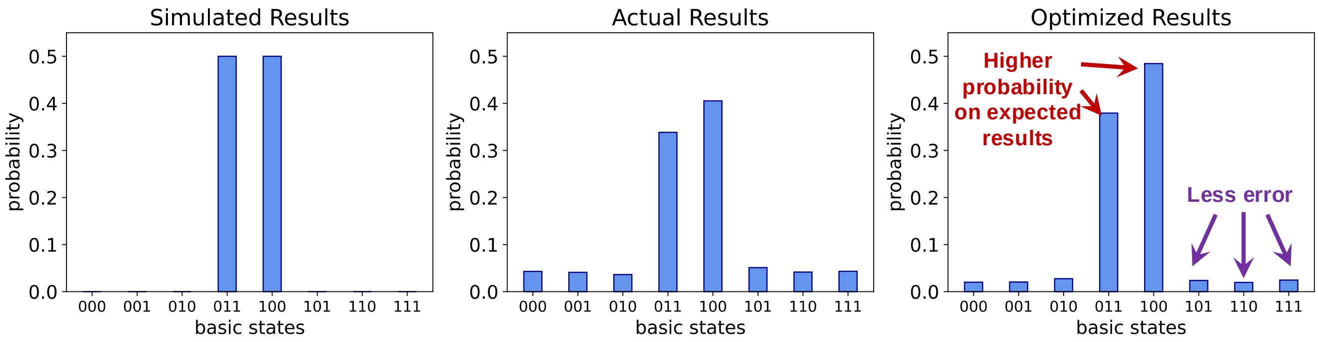

Fig. 5: Results for Grover's Algorithm Experiments

REFERENCES

[1] M. U. Hadi, Q. Al Tashi, A. Shah, R. Qureshi, A. Muneer, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu *et al.*, "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects," *Authorea Preprints*, 2024.

[2] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A survey of embodied ai: From simulators to research tasks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 230–244, 2022.

[3] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[4] A. Lozhkov, R. Li, L. Ben Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang *et al.*, "Starcoder 2 and the stack v2: The next generation," *arXiv preprint arXiv:2402.19173*, 2024.

[5] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[6] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui, "Agentcoder: Multiagent-code generation with iterative testing and optimisation," *arXiv preprint arXiv:2312.13010*, 2023.

[7] N. Dupuis, L. Buratti *et al.*, "Qiskit code assistant: Training LLMs for generating quantum computing code," *arXiv preprint arXiv:2405.19495*, 2024.

[8] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[10] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[11] OpenAI, "Gpt-4 technical report," 2023, accessed: 2023-08-15. [Online]. Available: https://openai.com/research/gpt-4

[12] Y. Talebirad and A. Nadiri, "Multi-agent collaboration: Harnessing the power of intelligent llm agents," *arXiv preprint arXiv:2306.03314*, 2023.

[13] Z. Zhou, X. Ning, K. Hong, T. Fu, J. Xu, S. Li, Y. Lou, L. Wang, Z. Yuan, X. Li *et al.*, "A survey on efficient inference for large language models," *arXiv preprint arXiv:2404.14294*, 2024.

[14] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[15] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[16] E. Gerjuoy, "Shor's factoring algorithm and modern cryptography. an illustration of the capabilities inherent in quantum computers," *American journal of physics*, vol. 73, no. 6, pp. 521–540, 2005.

[17] A. M. Steane, "Simple quantum error-correcting codes," *Physical Review A*, vol. 54, no. 6, p. 4741, 1996.

[18] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 86, no. 3, p. 032324, 2012.

[19] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[20] Q. Community, "Qiskit machine learning," https://github.com/qiskit-community/qiskit-machine-learning, 2023, accessed: 2024-07-27.

[21] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[22] D. Bouwmeester, J.-W. Pan, K. Mattle, M. Eibl, H. Weinfurter, and A. Zeilinger, "Experimental quantum teleportation," *Nature*, vol. 390, no. 6660, pp. 575–579, 1997.

[23] M. Santha, "Quantum walk based search algorithms," in *International Conference on Theory and Applications of Models of Computation*. Springer, 2008, pp. 31–46.

[24] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll, "Quantum annealing: A new method for minimizing multidimensional functions," *Chemical physics letters*, vol. 219, no. 5-6, pp. 343–348, 1994.

[25] R. Sweke, M. S. Kesselring, E. P. van Nieuwenburg, and J. Eisert, "Reinforcement learning decoders for fault-tolerant quantum computation," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025005, 2021.

[26] J. Li, G. Li, Y. Li, and Z. Jin, "Structured chain-of-thought prompting for code generation," *ACM Transactions on Software Engineering and Methodology*, 2023.

[27] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain

[28] B. Clavie, "Ragatouille: State-of-the-art document retrieval," https://github.com/bclavie/RAGatouille, 2024, accessed: 2024-08-21.

[29] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[31] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[32] M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen, "Efficient training of language models to fill in the middle," *arXiv preprint arXiv:2207.14255*, 2022.

[33] Z. Yang, N. Band, S. Li, E. Candès, and T. Hashimoto, "Synthetic continued pretraining," *arXiv preprint arXiv:2409.07431*, 2024.