A PennyLane-Centric Dataset to Enhance LLM-based Quantum Code Generation using RAG

Abdul Basit[1], Nouhaila Innan[1,2], Muhammad Haider Asif[1,2], Minghao Shao[1],
Muhammad Kashif[1,2], Alberto Marchisio[1,2], Muhammad Shafique[1,2]
[1]*eBRAIN Lab, Division of Engineering New York University (NYU) Abu Dhabi*, Abu Dhabi, UAE
[2]*Center for Quantum and Topological Systems (CQTS), NYUAD Research Institute, New York University Abu Dhabi*, UAE
{abdul.basit, nouhaila.innan, ma8183, shao.minghao, muhammadkashif, alberto.marchisio, muhammad.shafique}@nyu.edu

***Abstract*—Large Language Models (LLMs) offer powerful capabilities in code generation, natural language understanding, and domain-specific reasoning. Their application to quantum software development remains limited, in part because of the lack of high-quality datasets both for LLM training and as dependable knowledge sources. To bridge this gap, we introduce *PennyLang*, an off-the-shelf, high-quality dataset of 3,347 PennyLane-specific quantum code samples with contextual descriptions, curated from textbooks, official documentation, and open-source repositories. Our contributions are threefold: (1) the creation and open-source release of *PennyLang*, a purpose-built dataset for quantum programming with PennyLane; (2) a framework for automated quantum code dataset construction that systematizes curation, annotation, and formatting to maximize downstream LLM usability; and (3) a baseline evaluation of the dataset across multiple open-source and commercial models, including ablation studies, all conducted within a retrieval-augmented generation (RAG) pipeline. Using PennyLang with RAG substantially improves performance: for example, Qwen 7B's success rate rises from 8.7% without retrieval to 41.7% with full-context augmentation, and LLaMa 4 improves from 78.8% to 84.8%, while also reducing hallucinations and enhancing quantum code correctness. Moving beyond Qiskit-focused studies, we bring LLM-based tools and reproducible methods to PennyLane for advancing AI-assisted quantum development.**

***Index Terms*—LLM, Quantum Computing, RAG**

I. Introduction

Quantum computing has emerged as a transformative technology with the potential to solve complex problems that are infeasible for classical systems [1]–[3]. Breakthroughs such as Google's demonstration of "quantum supremacy" [4], where a quantum processor performed a calculation in seconds that would take thousands of years on a classical supercomputer, highlight its disruptive capabilities in domains ranging from cryptography and quantum chemistry to machine learning and optimization [5]–[19].

Research Challenges: Despite these advances, programming quantum systems is challenging due to the specialized nature of existing frameworks and benchmarks, hindering widespread adoption [20], [21]. PennyLane is a leading open-source Python framework designed for hybrid quantum-classical computing, allowing seamless integration of quantum circuits with machine learning workflows [22]. However, unlike the Qiskit framework [23], [24], which benefits from dedicated AI-driven code assistants [25], PennyLane lacks equivalent tools to assist developers in writing optimized quantum code.

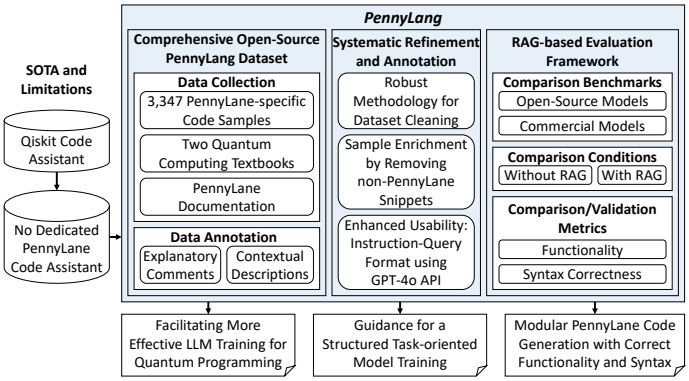

Fig. 1: Overview of novel contributions in this work.

In parallel, the field of Natural Language Processing (NLP) has seen rapid advancements in Large Language Models (LLMs), which now excel at reasoning, contextual understanding, and automated code generation [26]–[28]. These models have shown success in classical programming [29], [30], improving developer productivity and reducing syntax and logic errors. However, their application in quantum programming, especially within the PennyLane ecosystem, remains underexplored. This motivates our work: curating a high-quality, PennyLane-centric dataset and establishing a structured methodology for refinement, annotation, and evaluation, ultimately enabling LLM-based quantum code generation and debugging.

State-of-the-Art and Limitations: LLMs have been extensively benchmarked for classical programming [31], [32], but their performance in quantum domains remains less explored. Early efforts such as Qiskit HumanEval and Qiskit Code Assistant [25] adapt LLMs to quantum programming but focus on Qiskit's hardware-centric paradigm. PennyLane, by contrast, targets variational quantum algorithms and quantum machine learning [33], requiring dedicated LLM support and a specialized training corpus. A major barrier is the scarcity of suitable data: quantum code is scattered across papers, forums, repositories, and documentation [34], and often lacks the contextual annotations necessary for LLMs to link quantum operations with their intended use.

Our Approach and Contributions: In this work, we introduce *PennyLang*, a large-scale, high-quality dataset specif-

TABLE I: Summary of Qiskit and PennyLane Datasets for LLM Training

Dataset Name	Framework	Source	Samples Available	Dataset Type	Format
QISKIT FRAMEWORK DATASETS					
Qiskit Textbook Dataset [35]	Qiskit	Qiskit.org Textbook	**Multiple**	Structured Qiskit Code	Python Scripts
IBM Quantum Challenge Data [36]	Qiskit	IBM Quantum Experience	**Derived**	Instruction-Response Format	JSON/Notebook
Qiskit Code Repository [37]	Qiskit	GitHub (Qiskit AI repos)	**Thousands**	Raw Qiskit Code Samples	Python Scripts
Q-Gen Quantum Circuit [38]	Qiskit	Kaggle (Possible)	**309**	Pretrained Benchmark Data	CSV/JSON
QDataSet [39]	Qiskit	GitHub	**52**	Quantum ML Data	Various
QCircuitNet [40]	Qiskit	Research Paper	**4,800**	Quantum Algorithm Design Benchmark	Python/QASM
PENNYLANE FRAMEWORK DATASETS					
PennyLane Documentation [41]	PennyLane	PennyLane.ai Docs	**Various**	Raw PennyLane Code Samples	Python Scripts
PennyLane GitHub Repos [42]	PennyLane	GitHub Open-Source	**Scattered**	Unstructured Quantum Code	Python Scripts
PennyLang (Ours)	**PennyLane**	**Open-Source Dataset**	**3,347**	**Curated and Structured Dataset**	**JSON file**

ically designed for PennyLane code generation, accompanied by a rigorous evaluation pipeline. We implement a baseline Retrieval-Augmented Generation (RAG) framework to assess the dataset's effectiveness across multiple models. Our key contributions, shown in Fig. 1, consist of:

1) *Comprehensive PennyLane Dataset:* Curated 3,347 PennyLane-specific code samples from GitHub repositories, quantum computing textbooks, and official PennyLane documentation, each enriched with contextual descriptions and annotations to support instruction tuning.
2) *Systematic Data Refinement and Annotation:* Developed a robust pipeline for dataset cleaning, metadata enrichment, and conversion of code snippets into instruction–query pairs using the GPT-4o API, ensuring compatibility with LLM training workflows.
3) *RAG-based Evaluation Framework:* Designed a RAG evaluation framework to assess the dataset's impact on model performance. Framework was evaluated on seven LLMs, including both open-source and commercial models under both retrieval-augmented and non-augmented settings, scoring outputs on functional correctness.

Summary of Key Results: Retrieval augmentation with PennyLang yields substantial gains for smaller open-source models: Qwen 7B's success rate rises from 8.71% to 41.66%, and LLaMa 4 improves from 78.78% to 84.84%. In contrast, stronger baselines (commercial models) do not benefit significantly from naive full-context retrieval. For example, the best performance is achieved by Claude 3.5 Sonnet without retrieval (95.07%), with full retrieval resulting in slight degradation. Detailed analysis is presented in Section V.

II. BACKGROUND AND RELATED WORK

The intersection of LLMs and quantum computing is an emerging research domain. While LLMs have demonstrated impressive capabilities in NLP [43] and *automated code generation* [44], their application in quantum software development remains underexplored. Most existing studies focus on classical programming languages or quantum frameworks such as *Qiskit* [25], with little attention to the PennyLane framework [22], [45], [46], despite its popularity in hybrid quantum-classical workflows. This section presents an in-depth review of related work in three key areas:

1) LLMs for Code Generation in Classical Programming
2) LLMs for Quantum Programming
3) Datasets for Training LLMs in Quantum Code Generation (summarized in Table I)

We then summarize existing contributions and highlight our novel approach in developing a *PennyLane-centric dataset* to advance AI-driven quantum programming, as shown in Table II.

A. LLMs for Code Generation in Classical Programming

Transformer-based language models have revolutionized automated code generation, significantly advancing classical software development. OpenAI's Codex [29], trained on large-scale public code repositories, enabled natural language-to-code translation, setting a new standard for programming assistants. CodeBERT [47] further improved source code understanding through bidirectional transformers, while InCoder [32] introduced multi-turn interactive capabilities for code infilling and synthesis. AlphaCode [28] demonstrated that LLMs could generate solutions for competitive programming problems, reaching near-human performance. To rigorously evaluate these models, benchmarks such as HumanEval [27], MBPP [48], and CodeXGLUE [49] have been proposed, each offering complementary perspectives on correctness, diversity, and execution fidelity. These benchmarks collectively emphasize the importance of well-curated, structured datasets in improving LLM-driven code generation.

B. LLMs for Quantum Programming

In comparison, quantum programming has seen limited adoption of LLMs, with most efforts focused on the Qiskit framework. IBM's Qiskit Code Assistant [25] supports Qiskit programs' debugging and optimizations, while [50] adapted BERT-based models for enhanced quantum code comprehension. CodeLlama-Quantum [30] extended Meta's CodeLlama to address Qiskit-specific use cases. Despite these advances, key limitations persist. The field lacks large, high-quality datasets for training and evaluation, and quantum circuits demand deeper reasoning due to entanglement, measurement, and complex control logic. Furthermore, hybrid programming paradigms like PennyLane interleave classical and quantum instructions, requiring models to understand multi-modal code patterns and contextual dependencies. To address these challenges, we propose *PennyLang*, the first PennyLane-centric dataset, supported by an evaluation pipeline that integrates RAG to improve code relevance and contextual grounding. Our contributions advance AI-assisted quantum programming by enabling LLMs to provide high-quality, domain-specific assistance in PennyLane.

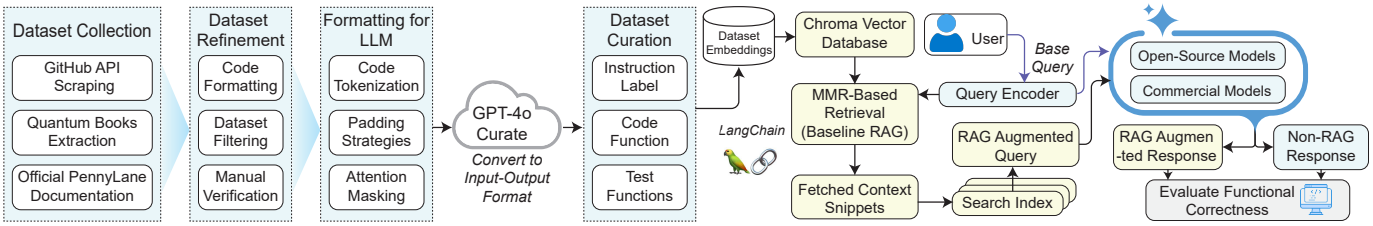

Fig. 2: **Methodology for PennyLang Quantum Code Generation and Evaluation.** We collect and refine quantum code from GitHub, textbooks, and PennyLane documentation, then use GPT-4o to convert cleaned snippets into instruction–query pairs (with code and tests). These examples are embedded in Chroma, and LangChain performs MMR-based retrieval to produce RAG-augmented versus vanilla prompts. Both open-source and commercial models are evaluated on functional correctness to quantify the benefit of the PennyLang dataset.

TABLE II: Comparison: Related Work & Our Contributions

Research Area	Existing Works	Our Contributions
LLMs for Code Generation	Codex [29] CodeBERT [47]	**PennyLane-specific dataset**
LLMs for Quantum Programming	Qiskit Assistant [25] [50]	**Hybrid quantum-classical dataset**
Quantum Code Datasets	QCircuitNet [40] QDataSet [39]	**First PennyLane dataset**

III. PennyLang Framework

A. Methodology for Data Collection and Refinement

Fig. 2 illustrates the methodology employed to construct a high-quality dataset of PennyLane code samples, ensuring that each snippet is genuinely relevant, well-structured, and accurately annotated. Our data collection strategy focused on extracting, filtering, and refining code snippets from multiple sources, including GitHub repositories, quantum computing books, and official PennyLane documentation.

1) Data Collection from GitHub Repositories

To systematically gather PennyLane-specific code, we utilized the GitHub API to search for publicly accessible repositories with permissive open-source licenses (e.g., MIT, Apache 2.0, BSD). The selection criteria required that the term `"PennyLane"` be present in either the repository name, description, or README file. To maintain dataset quality and avoid redundancy, we followed these constraints:

- Only repositories from the **main branch at the latest commit** were considered.
- **Forked repositories** were systematically excluded.
- Only Python (`.py`) files were extracted for further analysis.
- Large, **autogenerated files** (e.g., test logs, package files) were excluded.

Since PennyLane maintains backward compatibility and rarely deprecates features [22], we did not impose a time constraint on data collection, ensuring a broad dataset scope. Additionally, introductory comment blocks containing license information, author details, or unrelated metadata were removed to retain only executable and relevant code. This curation process resulted in a high-fidelity dataset of PennyLane-specific implementations. In total, 1,321 samples were collected from the official PennyLaneAI GitHub repository [42].

2) Extracting Quantum Code from Books

In addition to GitHub, we extracted PennyLane-related code snippets from two quantum computing books, Quantum Machine Learning: An Applied Approach [51] and A Practical Guide to Quantum Machine Learning and Quantum Optimization [52]. These books provide theoretical foundations and applied examples of PennyLane-based quantum machine learning (QML) and variational quantum algorithms (VQAs). The extraction process involved:

1) Identifying **all PennyLane-related code snippets**.
2) **Diversity Assurance**: The dataset includes a broad spectrum of PennyLane use cases, from basic gate operations to advanced quantum optimization problems.
3) Extracting **only the relevant code** while discarding unrelated sections.
4) **Manual verification** with explanations from the books, converting descriptive text into **structured comments**.

Unlike structured repositories, book content lacked a consistent format for where explanations were placed (before or after code). Thus, manual review was necessary to ensure that annotations correctly corresponded to each snippet.

3) Scraping Code from PennyLane Documentation

The official PennyLane documentation [41] is a comprehensive resource covering topics such as quantum circuits, variational algorithms, operators and templates, hybrid quantum–classical optimization, and quantum chemistry applications. To enrich our dataset, we systematically extracted all code snippets from the official website, including those embedded in tutorials and API references. Documentation text accompanying each example was manually converted into inline comments, ensuring every sample is self-contained and context-rich. This approach improves LLM interpretability, enabling models trained on the dataset to produce both functional code and explanatory text.

4) Dataset Summary and Statistics

The final dataset containing 3,347 PennyLane code samples is shown in Table III.

TABLE III: Dataset Composition

Source	Samples Collected
Unofficial GitHub repositories	1,952
GitHub (PennyLaneAI repository)	1,321
Official PennyLane Documentation	53
Quantum Computing Books	21
Total Dataset Size	**3,347**

Each sample varies in token length, depending on the tokenizer used. This dataset is now suitable for enhancing LLMs for PennyLane-based quantum programming.

5) Ensuring Data Integrity and Diversity

To keep dataset integrity, we applied multiple quality control measures:

1) **Duplicate Removal**: Identical code samples from different sources were eliminated.
2) **Code Formatting**: Samples were formatted consistently to follow **PEP 8 style guidelines** [53].
3) **Manual Review**: A human-in-the-loop verification was conducted to assess annotation accuracy.

These efforts ensure that the dataset is rich, well-structured, and optimized for AI model training.

6) Implications for AI-Driven PennyLane Code Generation

The collected dataset significantly enhances AI-driven PennyLane code assistance in multiple ways:

- **Improving Model Performance**: High-quality annotations ensure models understand both syntax and context.
- **Facilitating Explainable AI**: Models trained on PennyLang can generate explanatory comments alongside code.

This dataset serves as a *foundation for future research* in *AI-assisted quantum programming for PennyLane*.

B. Tokenization and Formatting for LLM

This section details the preprocessing pipeline for preparing our dataset for (LLM), focusing on tokenization, padding strategies, and attention masking to optimize input handling within the transformer architecture.

1) Padding Strategies and Tokenizer Variations

Padding strategies differ across tokenizers:

- **Token-Specific Padding**: Some tokenizers have a dedicated padding token, automatically appended to shorter sequences. For example, Hugging Face's Tokenizer uses `"<pad>"` by default [54].
- **Manually Defined Padding**: If a tokenizer lacks an explicit padding token, a placeholder token (e.g., `"0"` or `"<mask>"`) must be manually assigned.
- **Left vs. Right Padding**: Left padding is commonly used in autoregressive models [55], ensuring that padding tokens do not interfere with causal attention.

In our pipeline, we employ left padding as it aligns with the causal decoder-only transformer architectures used in models like GPT-4o Mini, Claude 3.5 Sonnet, Gemini 2.5 Flash, Claude Haiku, and GPT-5 Mini. This ensures that masked self-attention restricts predictions to depend only on preceding tokens, maintaining auto-regressive consistency.

2) Applying Attention Masking

To improve transformer efficiency, an attention mask [56] was applied post-padding. This mask helps the model distinguish between valid input tokens and padding tokens, ensuring that computation is focused on meaningful data. Table IV shows the attention mask matrix applied to a batch of tokenized samples. Here, the `"<pad>"` tokens are assigned 0s, instructing the model to ignore them during processing.

TABLE IV: Tokenized Input and Attention Masking.

Token	Mask	Token	Mask	Token	Mask
`"<pad>"`	0	`"def"`	1	`"circuit"`	1
`"<pad>"`	0	`":"`	1	`"("`	1
`")"`	1	`"return"`	1	`"qml.expval"`	1

- **Attention Mask Structure**: A binary mask is assigned to each token:
  - `1` → Meaningful tokens to be processed by the model.
  - `0` → Padding tokens ignored by the transformer.
- **Impact**: Self-attention [55] ensures that padding tokens do not influence attention computations, thereby reducing unnecessary computations and preventing misleading dependencies.

C. Methodology for Dataset Evaluation

1) Retrieval-Augmented Generation (RAG) Framework

To improve PennyLane code generation by reducing hallucinations and outdated syntax, and to better align outputs with PennyLane best practices, we adopt a retrieval-augmented generation (RAG) [57] approach. We implement the pipeline in LangChain with modular components for document ingestion, vector embedding, retrieval, and prompt construction. The PennyLang dataset, formatted as instruction–response pairs, is embedded using `OpenAIEmbeddings` and stored in a `Chroma` vector database for efficient similarity search. At inference, user queries are embedded, relevant context is retrieved, and the combined prompt is sent to the target LLM. Table V summarizes the key PennyLang RAG framework components, including their role in dataset loading, embedding storage, prompt formatting, retrieval chaining, and interfacing with multiple LLM backends.

TABLE V: PennyLang Components Used in the RAG-based Evaluation Framework

PennyLang RAG Framework Components
`DirectoryLoader` *(Document Loader)* ▷ Loads PennyLane dataset samples as text files from directory
`Chroma` *(Vector Database)* ▷ Stores embeddings for retrieval using similarity search
`OpenAIEmbeddings` *(Embedding Model)* ▷ Converts text into vector embeddings for indexing and retrieval
`ChatPromptTemplate` *(Prompt Formatting)* ▷ Defines structured prompts for AI models to ensure consistency
`create_retrieval_chain` *(Retrieval Pipeline)* ▷ Creates pipeline integrating stored embeddings for improved responses
`ChatOpenAI` *(OpenAI LLM Interface)* ▷ Connects to OpenAI models (GPT-4o Mini) for response generation
`ChatAnthropic` *(Anthropic LLM Interface)* ▷ Connects to Anthropic Claude models (Claude 3.5 Sonnet) for responses

When a user query is submitted (e.g., a quantum programming task or challenge description), we use Maximum Marginal Relevance (MMR) search to retrieve the most relevant examples from the dataset. The retrieved examples are concatenated into a structured prompt and passed to the LLM for code generation. For a fair assessment of RAG's impact, we generate outputs for each query both with and without retrieved context under identical conditions.

IV. Statistical Evaluation of the Dataset

A comprehensive analysis of the dataset was conducted to assess the distribution of instruction lengths, response lengths, response variations across feature groups, and their correlation with quantum functionalities. This section provides insights into the dataset's structure, response variability, and its implications for training LLMs.

A. Response Lengths Across Feature Groups

Fig. 3 provides a box plot representation of response lengths across different feature groups.

- Response lengths remain relatively stable across feature groups, suggesting uniformity in generated outputs.
- Outliers appear in every feature category, indicating occasional long responses, particularly in core operations, measurement, and mathematical functions.
- The "Other" category has the shortest responses, likely due to simpler queries requiring minimal output.

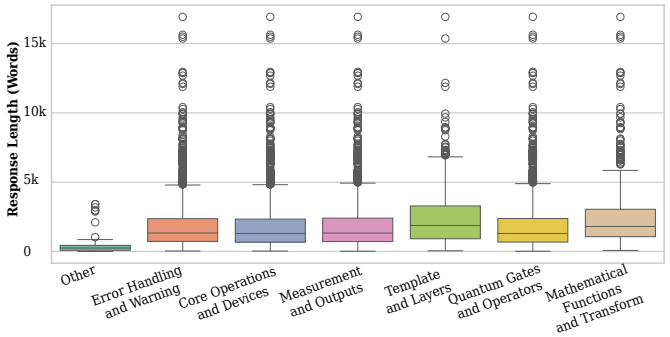

Fig. 3: Box plot of response lengths across different feature groups. Each category represents a specific quantum computing feature set, with response lengths exhibiting variability and outliers, particularly in core operations, measurement, and mathematical functions.

B. Instruction and Response Length Distributions

Fig. 4 illustrates the distribution of instruction and response lengths, highlighting key patterns in the dataset. Instruction lengths, as shown in left plot, follow a normal distribution with a peak around 35 words. The majority of instructions fall within the 30–40 word range, while longer prompts exceeding 60 words are relatively rare. In contrast, response lengths exhibit a right-skewed distribution, as shown in the middle plot. Most responses are under 2500 words, with a distinct peak around 1000 words, though some responses exceed 10,000 words, indicating instances of verbose outputs. The instruction-response correlation, illustrated in the rightmost plot, reveals a positive correlation between instruction and response length. While longer instructions tend to produce longer responses, variability increases significantly beyond 50 words, highlighting the influence of instruction complexity on response verbosity.

C. Feature Group Usage Across Quantum Domains

The relationship between feature groups and quantum computing domains, such as machine learning, hardware, chemistry, and optimization, is illustrated in Fig. 5. Quantum Machine Learning relies predominantly on quantum gates and operators, alongside error handling techniques, reflecting its dependence on structured quantum transformations and noise mitigation. In contrast, Quantum Chemistry and Optimization exhibit relatively lower usage of mathematical functions, suggesting that these domains rely on alternative computational approaches rather than extensive arithmetic operations. Quantum Hardware applications, however, demonstrate a balanced distribution across various feature groups, indicating a broader, more integrated usage of quantum functionalities across different operational layers.

D. Feature Category Association with Quantum Functions

Quantum gates and operators exhibit the strongest associations with multiple features, particularly with commonly used functions such as qml.PauliZ, qml.RX, and qml.RY, which are integral to quantum circuit construction. Core operations and measurement functions also display high correlations, reflecting their frequent application in quantum circuit execution and result interpretation. Meanwhile, error handling and mathematical functions show relatively lower but still significant interactions with select quantum features, indicating their importance in specific computational scenarios rather than across all use cases.

E. Insights and Implications

Findings from Figures 3, 4, and 5 offer several insights:

- **Balanced Instruction Lengths**: The normal distribution of instruction lengths ensures the dataset is well-structured for model training.
- **Response Length Variability**: Most responses are concise, but outliers indicate that certain quantum features demand more detailed explanations.
- **Feature-Specific Trends**: Quantum gates and measurement functions dominate dataset interactions, aligning with core PennyLane applications.
- **Domain-Specific Usage**: Different quantum fields rely on distinct feature sets, highlighting potential customization opportunities for specialized LLM training.

V. Evaluation

To assess the effectiveness of our proposed dataset for quantum code generation, we designed a benchmark suite comprising 264 test cases, each tailored to evaluate various PennyLane functionalities and quantum programming concepts. The test cases (prompts) cover a broad spectrum of tasks, ranging from basic circuit construction to advanced hybrid algorithm design, encompassing state preparation, parameterized gate implementation, differentiation routines, and optimization workflows. Several tasks address realistic applications, including molecular energy estimation via the Variational Quantum Eigensolver (VQE) for systems such as H_2 and LiH, as well as hybrid quantum–classical models trained on toy datasets for classification and regression. An example of an advanced prompt is:

> *"Write a PennyLane program that implements a Variational Quantum Eigensolver to estimate the ground-state energy of the* H_2 *molecule using the* `default.qubit` *device. Define the molecular Hamiltonian, construct a parameterized ansatz circuit, and optimize its parameters using gradient-based optimization until convergence."*

Each test was used as a prompt within an RAG pipeline and evaluated across both open-source and commercial LLMs under different context settings (100%, 75%, 50%, and 0%), where 100% denotes full retrieved context and 0% corresponds to no RAG. This setup was used to evaluate the effect of retrieval on code generation performance. The evaluation employed `pass@1` through `pass@5`, where each model generated five candidate solutions per test. The success rate (%) was computed as $\texttt{pass@k}/264 \times 100$, where `pass@k` denotes the number of test cases, out of 264, for which at least one correct solution was obtained within the first `k` attempts. A generated solution was counted as correct if the notebook

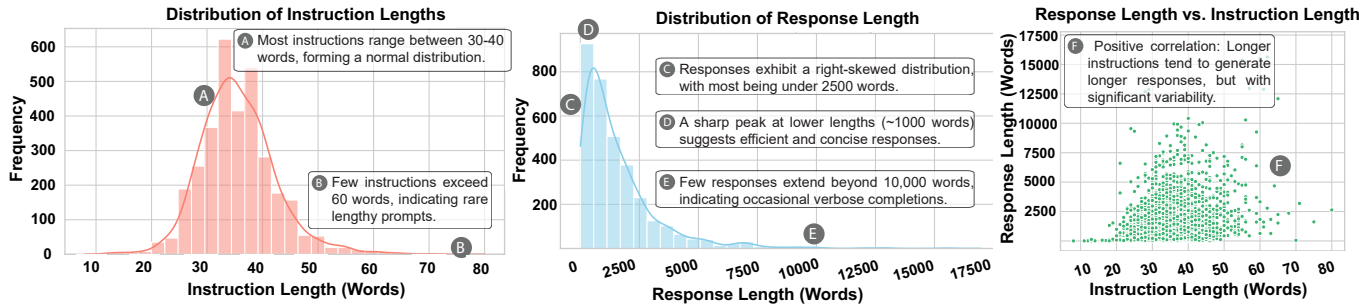

Fig. 4: Distribution of instruction and response lengths with their correlation. The left plot shows the normal distribution of instruction lengths, the middle plot highlights the right-skewed distribution of response lengths, and the right scatter plot illustrates the relationship between instruction and response length.

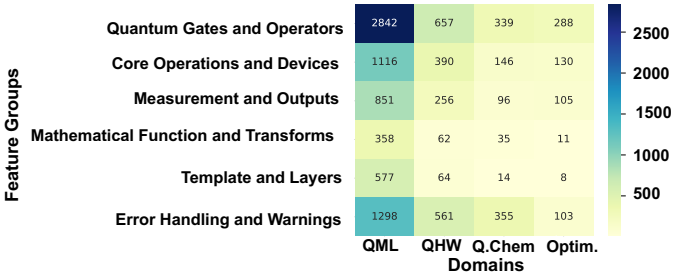

Fig. 5: Feature group usage across domains: Heatmap illustrating how frequently each quantum feature group appears across the different domain categories (QML, Quantum Hardware, Quantum Chemistry and Optimization), with color intensity reflecting count.

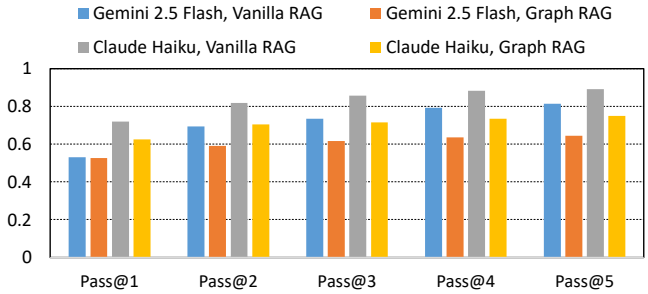

Fig. 6: Ablation Study: Vanilla RAG vs. Graph RAG evaluated on Gemini 2.5 Flash and Claude Haiku.

executed successfully and satisfied the corresponding test-case checks for the given inputs. The evaluation focused on whether the generated code correctly implemented the required task and produced the expected valid output for that test case. For tasks involving learning or optimization, a solution was considered correct if the requested workflow executed properly and returned the required type of result, regardless of the final performance value. In total, 1,320 notebooks were generated and executed per model (264 tests $\times$ 5 completions), providing a rigorous code-level functional evaluation. Table VI lists the evaluated models.

TABLE VI: LLMs Used in the RAG-based Evaluation Framework

Large Language Models in RAG Pipeline
Qwen2.5-7B-Instruct-Turbo *(Alibaba Cloud)* • **Parameters:** 7B | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Generates PennyLane code
LLaMa 4 Maverick *(Meta)* • **Parameters:** 17B | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Code generation
GPT-4o Mini *(OpenAI)* • **Parameters:** Not disclosed | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Evaluates and scores generated responses
Claude 3.5 Sonnet *(Anthropic)* • **Parameters:** Not disclosed | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Alternative model for response generation
Gemini 2.5 Flash *(Google DeepMind)* • **Parameters:** Not disclosed | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Rapid retrieval-augmented generation and code reasoning
Claude Haiku *(Anthropic)* • **Parameters:** Not disclosed | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Lightweight LLM for fast inference and evaluation tasks
GPT-5 Mini *(OpenAI)* • **Parameters:** Not disclosed | **Architecture:** Causal Decoder-Only Transformer • **Usage:** Enhanced multi-turn reasoning and tool-augmented evaluation

a) Analysis: Vanilla RAG vs. GraphRAG.

Figure 6 compares *vanilla* retrieval-augmented generation (RAG) against a GraphRAG variant on our PennyLane benchmark, using two representative commercial models (Gemini 2.5 Flash and Claude Haiku). Across all `pass@k` metrics, vanilla RAG consistently outperforms GraphRAG for both models, indicating that the added graph-structuring step does not translate into better executable-code success in this setting. Overall, these results suggest that, for our dataset and tasks, the primary gains come from retrieving a small set of *directly relevant* PennyLane exemplars rather than from explicitly modeling inter-document relations via a graph.

Since our benchmark emphasizes functional correctness of generated notebooks, where small syntactic or API-level deviations can cause execution failures, the precision of the retrieved snippets is often more important than broader relational coverage. Graph construction can also impose additional routing/selection constraints (e.g., traversing nodes that are only indirectly related), which may reduce the density of immediately usable code patterns in the final prompt compared to standard similarity-based retrieval. Given that vanilla RAG consistently outperforms GraphRAG in this ablation study, we adopt vanilla RAG as the default retrieval strategy for the remainder of the evaluations.

b) Open-Source Models.

Given that the dataset was primarily developed to support open-source development, our core evaluation focused on Qwen 7B and LLaMa 4. As shown in Table VII, these models showed clear improvements when used with our dataset. Notably, RAG with 75% context coverage often outperformed the full-context setting. This suggests that a moderate amount of relevant context may reduce noise and improve grounding, while excessive retrieval might introduce redundancy or irrelevant tokens.

TABLE VII: Evaluation of Qwen 2.5 7B, LLaMa 4 Maverick 17B, GPT-4o Mini, and Claude 3.5 Sonnet on PennyLane. Results for open-source models (Qwen and LLaMa) are highlighted in purple to distinguish their RAG-based performance gains from commercial models.

Metric	Qwen 2.5 7B				LLaMa 4 Maverick				GPT-4o Mini				Claude 3.5 Sonnet			
	Full	75%	50%	0%	Full	75%	50%	0%	Full	75%	50%	0%	Full	75%	50%	0%
Pass@1	**33.0%**	**37.9%**	**33.3%**	7.2%	**67.8%**	**64.8%**	**67.8%**	56.8%	54.5%	59.8%	56.4%	50.8%	70.8%	68.9%	64.4%	73.5%
Pass@2	**37.5%**	**41.3%**	**39.0%**	8.0%	**75.0%**	**76.5%**	**75.4%**	67.4%	64.8%	71.6%	66.3%	66.3%	79.5%	82.6%	76.9%	87.1%
Pass@3	**39.4%**	**42.8%**	**41.3%**	8.0%	**80.3%**	**81.1%**	**79.2%**	72.3%	72.0%	78.4%	73.9%	75.4%	84.5%	87.9%	81.1%	90.2%
Pass@4	**41.3%**	**44.3%**	**41.7%**	8.3%	**81.8%**	**82.2%**	**80.3%**	76.5%	76.5%	82.9%	79.2%	81.4%	87.1%	91.7%	87.1%	93.2%
Pass@5	**41.7%**	**45.1%**	**43.2%**	8.7%	**84.8%**	**84.8%**	**82.2%**	78.8%	80.3%	84.8%	81.8%	85.6%	89.0%	92.4%	87.5%	95.1%

c) Commercial Models.

For comparison, we also tested GPT-4o Mini, Claude 3.5 Sonnet, Gemini 2.5 Flash, Claude Haiku, and GPT-5 Mini. As shown in Tables VII and VIII, these commercial models demonstrated high performance even without RAG. However, the added context from our dataset provided limited benefit. This may be attributed to their broad pretraining, which likely includes similar material (e.g., PennyLane documentation or equivalent quantum code examples), reducing the marginal gain from external retrieval.

TABLE VIII: Evaluation of Gemini 2.5 Flash, Claude Haiku, and GPT-5 Mini on PennyLane under two RAG coverage settings (75% and 0%).

Metric	Gemini 2.5 Flash		Claude Haiku		GPT-5 Mini	
	75%	0%	75%	0%	75%	0%
Pass@1	53.0%	55.3%	72.0%	67.4%	73.9%	74.6%
Pass@2	69.3%	64.8%	81.8%	78.8%	85.6%	85.2%
Pass@3	73.5%	68.9%	85.6%	84.1%	89.8%	88.3%
Pass@4	79.2%	72.0%	88.3%	87.9%	90.5%	89.8%
Pass@5	81.4%	75.0%	89.0%	89.8%	91.3%	92.0%

d) Discussion.

The evaluation reveals important interactions between dataset design, model capacity, and retrieval strategies. Open-source models such as Qwen 7B and LLaMa 4 showed notable improvements when augmented with *PennyLang*, confirming that domain-specific RAG can effectively compensate for limited pretraining in quantum programming. Interestingly, the 75% context setting consistently outperformed the full-context retrieval, indicating that focused and moderately sized context windows improve model grounding by avoiding redundancy and context saturation.

In contrast, commercial models like GPT-4o Mini, Claude 3.5 Sonnet, Gemini 2.5 Flash, Claude Haiku, and GPT-5 Mini achieved high performance across all settings, with only marginal benefits from added retrieval. This behavior is likely due to their extensive pretraining on diverse codebases, which may already include material similar to what our dataset provides. Consequently, the impact of external retrieval is diminished and can even introduce noise if not precisely targeted. These findings emphasize the importance of retrieval quality over quantity in RAG-based code generation. The consistent advantage of partial over full context suggests that carefully curated, relevant examples are more beneficial than exhaustive inclusion.

Finally, by evaluating 1,320 generated notebooks per model, we ensure that our conclusions are grounded in executable and functionally correct outputs, a crucial criterion in the domain of quantum programming. These results underscore the utility of *PennyLang* in enhancing open-source LLM performance while providing insights into optimal RAG configurations for technical code generation tasks.

VI. Conclusion

In this paper, we introduce *PennyLang*, a high-quality instruction–response dataset tailored for PennyLane-based quantum code generation. We present our data collection and preprocessing methodology, drawing from GitHub repositories, textbooks, and official documentation to curate 3,347 diverse and well-structured code samples. To evaluate its practical utility, we develop a RAG evaluation framework and benchmark seven state-of-the-art LLMs (both open-source and commercial models) under both RAG and non-RAG settings. Our findings show that *PennyLang*, when paired with RAG, substantially boosts performance for open-source models, while proprietary models exhibit minimal or negative gains with naive full-context retrieval, likely because their pretraining already covers overlapping PennyLane-related knowledge. This further highlights the importance of calibrated retrieval budgets, as moderate settings (e.g., 75%) often outperform full context by balancing signal and noise. We release both the *PennyLang* dataset and the evaluation framework to foster reproducibility and encourage further research in LLM-assisted quantum development. Future directions include training lightweight instruction-tuned models on this dataset, improving the filtering pipeline for continuously collecting high-quality new code snippets into the database, optimizing retrieval strategies across different model types, and extending to cross-framework generalization between PennyLane, Qiskit, and Cirq.

Data Availability

The dataset proposed in this paper is publicly available on GitHub at https://github.com/eBrain4Everyone/Pennylang.

Acknowledgment

This work was supported in part by the NYUAD Center for Quantum and Topological Systems (CQTS), funded by Tamkeen under the NYUAD Research Institute grant CG008, and Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104. This research was carried out on the High Performance Computing resources at New York University Abu Dhabi.

References

[1] R. P. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics*, vol. 21, no. 6, pp. 467–488, Jun 1982.

[2] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018.

[3] M. Kashif and S. Al-Kuwari, "Demonstrating quantum advantage in hybrid quantum neural networks for model capacity," in *2022 IEEE International Conference on Rebooting Computing (ICRC)*, 2022.

[4] F. Arute, K. Arya, R. Babbush *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct 2019.

[5] A. Montanaro, "Quantum algorithms: an overview," *npj Quantum Information*, vol. 2, no. 1, Jan. 2016.

[6] K. Zaman, A. Marchisio, M. A. Hanif, and M. Shafique, "A survey on quantum machine learning: Current trends, challenges, opportunities, and the road ahead," *arXiv:2310.10315*, 2023.

[7] N. Innan, A. Marchisio, M. Bennai, and M. Shafique, "Qfnn-ffd: Quantum federated neural network for financial fraud detection," in *2025 IEEE International Conference on Quantum Software (QSW)*, 2025.

[8] W. E. Maouaki, N. Innan, A. Marchisio *et al.*, "Quantum clustering for cybersecurity," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Sep. 2024, p. 5–10.

[9] N. Innan, M. A.-Z. Khan, and M. Bennai, "Quantum computing for electronic structure analysis: Ground state energy and molecular properties calculations," *Materials Today Communications*, 2024.

[10] K. Zaman, A. Marchisio, M. Kashif, and M. Shafique, "Po-qa: A framework for portfolio optimization using quantum algorithms," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2024.

[11] N. Innan, A. Sawaika, A. Dhor, S. Dutta, S. Thota, H. Gokal, N. Patel, M. A.-Z. Khan, I. Theodonis, and M. Bennai, "Financial fraud detection using quantum graph neural networks," *Quantum Machine Intelligence*, vol. 6, no. 1, p. 7, 2024.

[12] M. Kashif and S. Al-Kuwari, "The impact of cost function globality and locality in hybrid quantum neural networks on nisq devices," *Machine Learning: Science and Technology*, vol. 4, no. 1, p. 015004, 2023.

[13] P. Pathak, V. Oad, A. Prajapati, and N. Innan, "Resource allocation optimization in 5g networks using variational quantum regressor," in *2024 International Conference on Quantum Communications, Networking, and Computing (QCNC)*. IEEE, 2024, pp. 101–105.

[14] W. El Maouaki, A. Marchisio, T. Said, M. Bennai, and M. Shafique, "Advqunn: A methodology for analyzing the adversarial robustness of quanvolutional neural networks," in *2024 IEEE International Conference on Quantum Software (QSW)*. IEEE, 2024, pp. 175–181.

[15] N. Innan, M. A.-Z. Khan, A. Marchisio, M. Shafique, and M. Bennai, "Fedqnn: Federated learning using quantum neural networks," in *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024.

[16] M. Kashif, A. Marchisio, and M. Shafique, "Computational advantage in hybrid quantum neural networks: Myth or reality?" in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025, pp. 1–7.

[17] A. Marchisio, M. U. Hafeez, N. Innan, M. Kashif, and M. Shafique, "Q-port: Quantum portfolio optimization with resource-efficient encoding and scalability analysis," in *International Conference on Quantum Engineering Sciences and Technologies for Industry and Services*, 2025.

[18] M. Kashif, S. Khalid, N. Innan, A. Marchisio, and M. Shafique, "Evaluating quantum amplitude estimation for pricing multi-asset basket options," in *2025 IEEE International Conference on Quantum Artificial Intelligence (QAI)*. IEEE, 2025, pp. 451–458.

[19] N. Innan, M. Kashif, A. Marchisio, Y.-S. Gan, F. Barbaresco, and M. Shafique, "Quav: Quantum-assisted path planning and optimization for uav navigation with obstacle avoidance," in *2025 IEEE International Conference on Quantum Artificial Intelligence (QAI)*, 2025.

[20] N. Moll *et al.*, "Quantum optimization using variational algorithms on near-term quantum devices," *Quantum Science and Technology*, 2018.

[21] Y. Wang and J. Liu, "A comprehensive review of quantum machine learning: from nisq to fault tolerance," *Reports on Progress in Physics*, vol. 87, no. 11, p. 116402, Oct. 2024.

[22] V. Bergholm *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv:1811.04968*, 2018.

[23] G. Aleksandrowicz, T. Alexander, P. Barkoutsos *et al.*, "Qiskit: an open-source framework for quantum computing," 2019.

[24] M. Kashif and S. Al-Kuwari, "Qiskit as a simulation platform for measurement-based quantum computation," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, 2022.

[25] N. Dupuis *et al.*, "Qiskit code assistant: Training llms for generating quantum computing code," in *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024.

[26] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, 2020.

[27] M. Chen, J. Tworek, H. Jun, Q. Yuan *et al.*, "Evaluating large language models trained on code," *arXiv:2107.03374*, 2021.

[28] Y. Li, D. R. Zhou, N. Kohl, J. Wu *et al.*, "Competition-level code generation with alphacode," *arXiv:2203.07814*, 2022.

[29] OpenAI, "OpenAI Codex: Programming with Natural Language," https://openai.com/index/openai-codex/, 2021.

[30] B. Roziere *et al.*, "Code llama: Open foundation models for code," *arXiv:2308.12950*, 2023.

[31] E. Nijkamp *et al.*, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv:2203.13474*, 2022.

[32] D. Fried *et al.*, "Incoder: A generative model for code infilling and synthesis," *arXiv:2204.05999*, 2022.

[33] M. Schuld and F. Petruccione, *Supervised Learning with Quantum Computers*, 1st ed. Springer, 2018.

[34] X. Ren, T. Zhang, X. Xu, Y.-C. Zheng, and S. Zhang, "Invited: Leveraging machine learning for quantum compilation optimization," in *DAC*, 2024.

[35] Q. Community, "The qiskit textbook," 2023.

[36] IBM Quantum, "Ibm quantum challenge," 2024.

[37] IBM Quantum and Qiskit Developers, "Qiskit open-source repositories," 2023, available at https://github.com/Qiskit.

[38] Y. Mao, "Q-gen quantum circuit dataset," 2023.

[39] E. Perrier, A. Youssry, and C. Ferrie, "Qdataset, quantum datasets for machine learning," *Scientific data*, 2022.

[40] R. Yang, Y. Gu, Z. Wang, Y. Liang, and T. Li, "Qcircuitnet: A large-scale hierarchical dataset for quantum algorithm design," *arXiv:2410.07961*, 2024.

[41] PennyLane Developers, "Official pennylane documentation," 2023, available at https://docs.pennylane.ai/en/stable/.

[42] PennyLane Developer, "Pennylaneai github repository," 2023, available at https://github.com/PennyLaneAI.

[43] M. Shao, A. Basit, R. Karri, and M. Shafique, "Survey of different large language model architectures: Trends, benchmarks, and challenges," *IEEE Access*, vol. 12, p. 188664–188706, 2024.

[44] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *ACM Transactions on Software Engineering and Methodology*, 2026.

[45] A. Basit, M. Shao, M. H. Asif, N. Innan, M. Kashif, A. Marchisio, and M. Shafique, "Pennycoder: Efficient domain-specific llms for pennylane-based quantum code generation," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2025.

[46] A. Basit *et al.*, "Qhackbench: Benchmarking large language models for quantum code generation using pennylane hackathon challenges," in *2025 IEEE International Conference on Quantum Artificial Intelligence (QAI)*, 2025.

[47] Z. Feng *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the association for computational linguistics: EMNLP 2020*, 2020.

[48] J. Austin *et al.*, "Program synthesis with large language models," *arXiv:2108.07732*, 2021.

[49] S. Lu *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv:2102.04664*, 2021.

[50] Q. Li *et al.*, "Adapting pre-trained language models for quantum natural language processing," *arXiv:2302.13812*, 2023.

[51] S. Ganguly, *Quantum machine learning: an applied approach*. Springer, 2021.

[52] E. F. Combarro *et al.*, *A practical guide to quantum machine learning and quantum optimization: Hands-on approach to modern quantum algorithms*. Packt Publishing Ltd, 2023.

[53] G. Van Rossum, B. Warsaw, and N. Coghlan, "Pep 8 – style guide for python code," 2001, available at https://peps.python.org/pep-0008/.

[54] T. Wolf, L. Debut, V. Sanh *et al.*, "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.

[55] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[56] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.

[57] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, 2020.