

# When Quantum Meets Classical: Characterizing Hybrid Quantum-Classical Issues Discussed in Developer Forums

Jake Zappin

*William & Mary*

Williamsburg, Virginia, USA

azappin@wm.edu

Trevor Stalnaker

*William & Mary*

Williamsburg, Virginia, USA

twstalnaker@wm.edu

Oscar Chaparro

*William & Mary*

Williamsburg, Virginia, USA

oscarch@wm.edu

Denys Poshyvanyk

*William & Mary*

Williamsburg, Virginia, USA

denys@cs.wm.edu

**Abstract**—Recent advances in quantum computing have sparked excitement that this new computing paradigm could solve previously intractable problems. However, due to the faulty nature of current quantum hardware and quantum-intrinsic noise, the full potential of quantum computing is still years away. Hybrid quantum-classical computing has emerged as a possible compromise that achieves the best of both worlds. In this paper, we look at hybrid quantum-classical computing from a software engineering perspective and present the first empirical study focused on characterizing and evaluating recurrent issues faced by developers of hybrid quantum-classical applications. The study comprised a thorough analysis of 531 real-world issues faced by developers – including software faults, hardware failures, quantum library errors, and developer mistakes – documented in discussion threads from forums dedicated to quantum computing. By qualitatively analyzing such forum threads, we derive a comprehensive taxonomy of recurring issues in hybrid quantum-classical applications that can be used by both application and platform developers to improve the reliability of hybrid applications. The study considered how these recurring issues manifest and their causes, determining that hybrid applications are crash-dominant (74% of studied issues) and that errors were predominantly introduced by application developers (70% of issues). We conclude by identifying recurring obstacles for developers of hybrid applications and actionable recommendations to overcome them.

## I. INTRODUCTION

In the last decade, quantum computing (QC) has transitioned from a theoretical concept to a practical endeavor [1]. This has been primarily fueled by the limitations of classical computing hardware [2] and the emergence of accessible QC platforms like IBM’s Qiskit [3], Google’s Cirq [4], Microsoft’s Q# [5] and Xanadu’s PennyLane [6]. QC is not merely an incremental step in computing power, but an entirely new kind of computing paradigm that has the potential to fundamentally change the way we represent information and perform computations [7]. Indeed, some computations that take years on modern classical computers may be solved in minutes by quantum computers due to the leap in computational power [8].

However, QC is still in its infancy. For instance, although significant progress has been made, it is unlikely that fault-tolerant quantum computers will be developed in the near-term [9]. As a result, many developers and researchers have turned to *hybrid quantum-classical* (HQC) computing as a

bridge to solve complex problems that may be out of reach for classical computers alone [10]. This approach leverages the respective capabilities of quantum and classical systems. It enables quantum devices to efficiently handle specific computational tasks within an algorithm that classical computers find intractable, while classical systems manage the remaining tasks of the process. This convergence of technologies can enhance an algorithm’s overall performance over a purely classical algorithm designed for the same task, particularly for applications like machine learning [11], even on today’s Noisy Intermediate-Scale Quantum (NISQ)-era computers [12, 13].

The promise of these performance gains has recently prompted developers and researchers to think about QC from a software engineering (SE) perspective [14–17]. This includes the consideration of essential SE tasks, such as debugging and testing, in the context of QC [18, 19]. However, developing quantum applications can pose unique challenges due to the complexity of quantum mechanics, the introduction of quantum principles like superposition and entanglement [20] and the relative infancy of QC platforms. Quantum developers therefore face SE issues and challenges that are new and unique, and have not yet been fully studied.

Hence, it is crucial to develop methodologies and taxonomies for identifying, classifying, and fixing issues faced by developers working on systems incorporating quantum algorithms. To date, though, attention has been mainly focused on identifying bugs in quantum platforms and libraries [19, 21–24] as opposed to exploring recurring issues, including recurrent bug patterns, encountered by developers of quantum algorithms and applications, the end-users of those platforms/libraries. More importantly, there have been no studies specifically looking at recurring issues in HQC systems.

HQC algorithms and applications, therefore, represent a particularly essential ground for research in SE due to their potential to overcome performance challenges on purely classical hardware. Studying the recurring issues encountered by developers within these hybrid systems is critical for several reasons:

- 1) *Prevalence of HQC systems.* Due to their practicality, HQC systems make up most of the current quantum applications being developed (variational quantum eigensolvers, quan-

tum approximate optimization algorithms, hybrid quantum-classical machine learning models, *etc.* [25]). This makes them the most relevant for immediate industry and research utilization [10]; in fact, HQC algorithms and applications are already impacting industries such as finance, material science, logistics, and healthcare [13, 26–28].

- 2) *A novel programming paradigm.* QC represents a new programming paradigm and requires a fundamentally different set of skills and problem-solving techniques. Also, the interaction between classical and quantum parts introduces unique challenges that are absent in purely classical systems — challenges that must be understood and mitigated to realize the potential of quantum-enhanced computing [10].
- 3) *Inform research and development.* The study of recurring HQC system issues encountered by developers will not only prepare us for the arrival of more advanced quantum systems, but also inform the development of current quantum algorithms, libraries, and platforms. These insights provide immediate benefits to a broad range of stakeholders (developers, researchers, industry, government, *etc.*) who are at the forefront of quantum development [29, 30].

Accordingly, we present the first empirical study to characterize and evaluate recurring issues faced by developers in HQC algorithms, applications, and systems (hereafter referred to collectively as *applications*). We collected and inspected 531 real-world issues encountered by developers and researchers reported on two different sources: the Xanadu Discussion Forums (XDF) [31] (now known as the PennyLane Discussion Forums) and the Quantum Computing Stack Exchange (QCSE) message board [32]. By employing a rigorous methodology to collect and analyze recurring issues in hybrid systems, we aim to answer five research questions, which includes developing a comprehensive taxonomy of recurring issues encountered by developers in HQC applications.

The findings of this study, presented Sec. V, offer valuable insights into the current issues and challenges facing HQC developers, as they aim to guide researchers and developers in formulating effective strategies for mitigating them. To facilitate more research in this domain, we make publicly available our HQC issue dataset, annotations, and results [33].

In summary, this paper makes the following contributions:

- A novel comprehensive study of recurring real-world issues faced by developers of HQC algorithms and applications;
- An in-depth review of 531 real-world issues in HQC implementations, documented in 447 discussion threads;
- A thorough taxonomy of recurring HQC issues intended to assist both platform and application developers in troubleshooting issues and in prioritizing the development of testing and debugging tools;
- A comprehensive analysis of the taxonomy, including quantitative and qualitative analyses of the manifestations and causes of the issues to assist developers in implementing HQC applications and in improving the usability and robustness of quantum platforms and libraries;
- A labeled dataset of real-world recurring issues faced by developers that can enable future research [33].

## II. BACKGROUND

QC represents a significant paradigm shift in CS and, more specifically, in SE [34]. It is a multidisciplinary area of computing that harnesses the principles of quantum mechanics to process information in ways fundamentally different from classical computing [16, 17]). This section briefly introduces QC, its challenges, and the concept of HQC computing.

### A. The Qubit, Superposition, and Entanglement

Qubits are the fundamental units of QC, differing from classical bits by utilizing quantum superposition and entanglement [35]. Superposition permits a qubit to represent multiple states concurrently. This permits quantum computers to process numerous calculations simultaneously offering a potential exponential speed-up for certain computations over classical computers [36]. The phenomenon of entanglement creates a dependency between the states of qubits, such that an action on one qubit can affect another instantaneously, a phenomenon Einstein called “spooky action at a distance” [37]. These quantum characteristics are what give quantum computers the ability to perform complex operations in tandem and potentially solve problems that are currently unsolvable by classical systems [36, 38, 39].

The utilization of superposition and entanglement grants QC the capacity to outperform classical systems in specific areas. Such capabilities enable the crafting of quantum algorithms that substantially lower time complexity for complex problems and address tasks deemed intractable for classical computers due to computational limitations [38, 39]. A notable example is prime number factorization, essential for modern encryption, where QC, through Shor’s algorithm, achieves polynomial-time factoring, a significant reduction from classical methods’ exponential time [40]. As QC technology continues to advance, it is anticipated that we will not only develop more efficient quantum algorithms to replace many classical algorithms, but also solve problems currently beyond the reach of classical computing [41].

### B. QC Limitations and the NISQ-Era

Despite its potential, quantum computing faces developmental hurdles due to qubit instability, caused by environmental interference and other intrinsic factors leading to errors and decoherence [42]. These issues necessitate error-correcting qubits, adding complexity and the risk of additional errors [43]. Achieving fault tolerance is anticipated to require a quantum processor with millions of qubits [44], a stark contrast to the Noisy Intermediate-Scale Quantum (NISQ) devices of today which have up to one thousand qubits [45, 46].

This current NISQ-era is characterized by quantum computers with 10 to 1,000+ noisy qubits and no error correction [47]. These systems use probabilistic methods, running quantum programs multiple times to obtain likely outcomes [48]. Despite limitations, NISQ-era quantum computers offer advancements, exemplified by algorithms like the Variational Quantum Eigensolver (VQE), which perform beyond classical capabilities with respect to specific tasks [47]. NISQ-era machines are consequently instrumental in advancing (HQC) computing

architectures, as they are being employed to augment and enhance today’s classical computing systems.

### C. Bridging the Gap: HQC Computing

Hybrid quantum-classical (HQC) systems are an intermediate step towards future fault-tolerant quantum computers, merging quantum computational power with classical reliability and accessibility [12]. These systems utilize quantum computers for complex tasks, such as prime factorization, optimization and other intractable problems for current classical computers, while relying on classical computing for standard operations and processing, thereby avoiding the current quantum limitations like qubit instability and error correction challenges [12, 49].

The synergy in HQC architectures enhances performance beyond what quantum or classical systems can achieve alone [10, 49]. This approach fosters scalable, real-world quantum applications. Notably, HQC algorithms like the Variational Quantum Eigensolver (VQE) excel in quantum chemistry, providing high-precision estimates unattainable by classical systems [49, 50], and are adaptable across various disciplines. The Quantum Approximate Optimization Algorithm (QAOA) is another HQC algorithm that addresses combinatorial problems by combining quantum and classical techniques for simultaneous solution exploration and error correction [49, 51]. HQC is also advancing machine learning (ML), an area where the limitations of classical computers are particularly being felt [52, 53].

Consequently, HQC computing is not just a workaround for the current limitations of quantum technology. Rather, it is a strategic approach that combines the best of both the quantum and classical worlds to increase computing power and efficiency. Given that HQC algorithms and applications will become more and more important in the coming years, it is critical to study them from a SE perspective. This is particularly so given that HQC introduces new challenges like requiring seamless integration of quantum and classical components, including managing data exchange, embedding, encoding/decoding, and synchronization. Additionally, HQC involves balancing quantum hardware limitations with classical capabilities for resource management and performance optimization. These complexities demand specialized expertise and tailored approaches, making HQC programming a distinct paradigm in its own right requiring dedicated research.

## III. RELATED WORK

The introduction and rise of QC has introduced novel challenges from a SE perspective. Recent studies have focused on investigating bug patterns in quantum libraries [24], platforms [19], and languages [22], with a focus on quantum-specific bugs and automated detection [54]. Luo *et al.* identified 80% of bugs as quantum-specific in various quantum programming languages [22], while Paltenghi and Pradel found 39.9% of platform bugs were unique to quantum platforms [19].

Zhao *et al.*’s work includes the development of the Bugs4Q suite [23] and tests for certain recurring bugs in IBM Qiskit platform, despite its limitations due to recent deprecations in

Qiskit [55]. Zhao *et al.* has more recently studied bugs in quantum ML frameworks developing a taxonomy of recurring bug patterns [24]. Similarly, Nayak *et al.*’s Q-PAC framework marks progress in bug-fix pattern detection in quantum code [54].

Aoun *et al.* [20] performed an analysis of quantum GitHub repositories identifying prevalent bug types in quantum software projects, which included simulators, frameworks (*e.g.*, Qiskit, Cirq), algorithms, compilers, tools, and experimental projects. The study revealed quantum systems to be generally buggier than classical ones, with program anomalies, configuration issues, and data structure flaws as common problems.

Our study complements and extends prior research into quantum bugs in several ways. First, while our study encompasses bug identification, the scope of our study is broader. Specifically, it attempts to understand and categorize recurring issues and challenges faced by quantum developers, particularly given QC and HQC are in their infancy. By doing so, we hope to inform the development of methodologies and the evolution of quantum libraries and frameworks still in their infancy. Second, we examine the relatively unexplored domain of issues (including recurring bugs) in HQC systems. We focus on hybrid algorithms and applications as a whole rather than solely on quantum components. Past studies have separated classical and quantum bugs without considering these hybrid applications or the interaction between the two subsystems. By focusing on hybrid applications, we identified a new type of issue: *cross-domain issues*.

Third, our research methodology also differs by sourcing data from XDF and QCSE, providing community-driven insights into the obstacles faced by developers in this emerging field [31, 32]. As Tab. I illustrates, prior research has focused on mining data primarily from GitHub repositories of quantum platforms, frameworks, and libraries. Our research is fundamentally different in that we used quantum-specific forums, which have been left unexplored. We believe these forums are a far richer resource for mining bugs and issues in quantum software, offering more specialized and relevant insights into the challenges unique to developers of quantum applications.

Lastly, and most importantly, our study focuses on recurring issues in HQC *applications* and *algorithms*. Prior research has predominantly focused on identifying bugs in quantum platforms and libraries from the perspective of platform developers (see Tab. I). These developers are concerned with the infrastructure and tools that enable application development, such as Qiskit, Cirq, and Q#. In contrast, application developers, often found on the XDF and QCSE forums, focus on building quantum applications and algorithms to solve particular problems, like VQE and QAOA, using these platforms and libraries. By examining applications, we provide insights relevant to end-users of these platforms, offering information to improve both application and platform development and maintenance.

## IV. STUDY DESIGN

This study aims to characterize recurring issues encountered by developers of HQC applications, their manifestations, and their causes, as well as to identify the major challenges that

TABLE I: Comparison with prior related work

Paper	Data Sources	Software Studied
This Paper	Xanadu Discussion Forums (XDF) and QC Stack Exchange (QCSE)	Quantum Applications and Algorithms (as opposed to platforms and frameworks)
[19]	GitHub Repositories ( <i>i.e.</i> , Issues and Pull Requests)	Quantum Platforms (Qiskit, Cirq, Q#, etc.)
[20]	GitHub Repositories ( <i>i.e.</i> , API Events)	Quantum Simulators, Platforms, Compilers, Tools, and Libraries
[22]	(1) GitHub Repositories, (2) Stack Overflow Posts; and (3) Stack Exchange Posts	Quantum Platforms (Qiskit, Cirq, Q# and ProjectQ)
[24]	GitHub Repositories ( <i>i.e.</i> , Issue Reports)	Quantum ML Platforms (Qiskit, Q#, Cirq, Torch Quantum, PennyLane)
[23]	GitHub Repositories	Quantum Platforms (Qiskit)
[54]	(1) Prior Research, (2) Stack Exchange Posts, and (3) GitHub Repositories	Quantum Platforms (Qiskit)

practitioners face when developing HQC applications. Our intention is to offer actionable insights to practitioners that allow them to develop better and more streamlined HQC applications. To this end, we reviewed the most comprehensive quantum-specific sources available: XDF and QCSE, where developers, students, and researchers (*developers* from hereon) discuss issues encountered while writing HQC applications.

This study answers the following research questions (RQs):

**RQ1:** *What are the recurring issues encountered by developers of HQC applications?* This RQ aims at identifying recurring issues that arise when developing HQC algorithms and applications, providing a foundation for developing strategies to detect and prevent these problems. The data collected in answering this RQ was used to synthesize the taxonomy in Fig. 1.

**RQ2:** *How frequently are issues encountered in HQC applications quantum-specific, classical, or cross-domain?* This RQ aims to categorize issues based on their origin and the interaction between quantum and classical components. We hope this offers insights into the unique operational challenges of HQC systems and where issues typically manifest, which can guide more targeted resolution efforts and enhance overall application efficiency and reliability.

**RQ3:** *How do issues in HQC applications manifest?* HQC applications can be inherently more complex than purely classical ones due to the intertwining of non-intuitive quantum behaviors with traditional classical computing logic. This RQ delves into ways HQC issues appear and manifest to developers, considering the unique challenges posed by the integration of quantum and classical paradigms.

**RQ4:** *What are the root causes of issues encountered in HQC applications?* This RQ examines whether recurring issues stem from the complexities of quantum programming, platform updates, documentation gaps, or other factors.

**RQ5:** *What are the predominant challenges and issues faced by developers when developing HQC applications?* There are unique obstacles presented by QC, the rapidly evolving quantum ecosystem of platforms and libraries, and the inherent complexity of merging with classical computing systems. This RQ attempts to identify the predominant challenges developers encounter while writing HQC applications and offers suggestions on how these challenges may be mitigated.

## A. Data Sources

We rely on two primary sources to answer our RQs. We chose these sources for their focus, depth of analysis and information quality when discussing and resolving HQC issues.

1) *Xanadu Discussion Forums (XDF):* Our study primarily analyzed the XDF, an active discussion board hosted by Xanadu, a leading photonic QC firm [31, 56]. The board, featuring contributions from both community members and Xanadu staff, focuses on in-depth troubleshooting in quantum ML and HQC algorithm development, using tools like the PennyLane library [6]. The technical discussions on XDF, especially those regarding bugs and other software issues, offer rich data for our analysis, a resource previously untapped by prior research focusing solely on quantum bugs.

2) *Quantum Computing Stack Exchange (QCSE):* The QCSE forum, akin to Stack Overflow for QC, serves as our study's secondary data source [32]. It features specialized discussions on QC challenges, including issues related to HQC systems and applications. Its focused content, excluding basic programming queries, provided us with relevant data on complex HQC issues that prior studies focused on bug identification have overlooked.

3) *Other Potential Data Sources:* Although we considered their inclusion, our study excluded GitHub repositories of HQC applications as a data source due to prevalent data limitations. We identified only 540 HQC application GitHub repositories (via GitHub keyword searching and inspection of GitHub's dependency graph), with over half of them being low-impact/educational projects or documentation examples. A significant portion (91%) displayed minimal activity, evidenced by scant commits or issues, and lacked updates within the last year (as of Oct. 2023). Furthermore, rudimentary issue tracking and non-descriptive commit messages obstructed the extraction of informative bug-related information.

## B. Data Collection

1) *Search Terms:* We began the data collection process by deriving a set of search terms that would help us gather as many hybrid-specific threads as possible. The search terms were created and refined by two authors who both have at least two years of prior experience studying quantum computing and writing quantum applications in an academic setting. In developing the list of search terms, the authors did preliminary searches on the XDF and QCSE message boards for hybrid application-related posts to look for common terms that should be included as part of the list. Likewise, they looked at code for several well-known HQC algorithms (*e.g.*, VQE and QAOA) to further refine the search term list and ensure appropriate coverage. Details of these search terms can be found in Tab. II, with the complete list accessible in our replication package [33].

2) *Raw Data Collection:* The search term list guided our data collection from the XDF and QCSE using custom scripts written by the authors to automate the search for and collection of relevant HQC issue discussions and posts, which helped streamline the process amidst the forums' vast data. These scripts curated URLs from the results, excluding duplicates,

TABLE II: Keywords used to identify relevant threads

Type	Rationale	Examples
General	Capture spectrum of issues	“bug”, “invalid”
QC-specific	Linked to QC processes	“embed”, “measure”
PennyLane Template	Prominence of PennyLane in HQC	“AmplitudeEmbedding”
Hybrid Algorithm	Issues related to HQC algorithms	“VQE”, “QAOA”

TABLE III: Statistics about the discussion thread mining

Forum	Mined Threads	Relevant Threads	Coded Threads
Xanadu	2,394	940	293
Stack Exchange	5,884	255	154
<b>Total</b>	<b>8,278</b>	<b>1,215</b>	<b>447</b>

during searches conducted in Oct. 2023, with the archived threads detailed in our replication package [33]. Notably, the range of relevant posts collected included posts from 2018 to late 2023, providing several years of data.

Subsequent to URL curation, each thread was manually reviewed and scrutinized for relevance by one of the authors, prioritizing inclusivity to capture as many potential HQC issue discussions as possible. At this stage, due to the impracticality resulting from the volume and complexity of the data (particularly from XDF; see Tab. IV), one author reviewed each thread to sift out false positives. Specifically, non-HQC or non-bug threads were omitted. The manual review figures and exclusion of false positives are summarized in Tab. III, resulting in 940 relevant threads from XDF and 255 from QCSE retained for analysis.

### C. Coding of HQC Discussion Threads

In our coding framework, each identified issue in a thread was categorized into a top-level code. These top-level codes were developed by the authors based on their research, experience and from prior research of classical and quantum issue and bug categories [20, 57]. The categories were refined iteratively for alignment with the data collected from the threads. Sub-categories were delineated via *open-coding* [58]:

- *Independent Coding:* Two authors coded each identified issue in a thread. This coding involved assigning a top-level code to each issue and labeling a sub-category that was open-coded by each author allowing for a more nuanced and detailed classification of each issue. Sub-categories were later refined and, if necessary, condensed, merged and re-labeled by the two authors during regular reconciliation meetings. This evolved into a shared code catalog. This method, documented in our code catalog (see our replication package [33]), also involved extraction of relevant text from each thread, detailed issue descriptions, and if available, the case of the issue and how the issue was resolved and manifested.

TABLE IV: Statistics about the coded discussion threads

Forum	Users Involved	Posts / Comments	Duration (in days)	Char Count	Word Count
XDF	Min.	1	0	534	77
	Avg.	3	9.14	68.52	9,619.78
	Max.	16	86	1,239.53	1,243.3
QCSE	Min.	1	0	272	40
	Avg.	3.11	4.97	95.61	4,406.82
	Max.	13	20	1,311.54	529.16

- *Verification and Content Relevance:* The two coders ensured threads were relevant to HQC and contained discussions of relevant issues and challenges faced by HQC developers.

- *Reconciliation Meetings:* Authors resolved coding discrepancies through in-depth reconciliation meetings and discussions that generally lasted in the order of hours. Disagreements were rigorously discussed and resolved without the need to bring in a third author, though the option was available. This step ensured consistency and reliability in the coding process mitigating potential bias. Since many of the threads contained multiple issues, the authors attempted to resolve discrepancies in coding through discussion, prioritizing coding consistency over inter-rate agreement metrics.

Our coding approach for threads was conducted in two distinct phases: a pilot coding of 20 XDF discussion threads followed by the coding of a representative sample of both XDF and QCSE threads (at 95% confidence interval and 5% error margin for each data source). In total, we coded 293 threads from XDF, including the pilot issues and a sample of 273, alongside 154 threads from QCSE.

From this analysis, we documented 377 issues on XDF and 154 issues on QCSE. Multiplicity of issues within single threads necessitated individual coding. Some threads were excluded from the count, such as those categorized as “Comprehension or Technical Questions.” These exclusions were adjudicated during reconciliation meetings that differentiated threads aimed at understanding QC or HQC nuances from those reporting explicit issues. Overall, 531 real-world issues were coded, with 483 of these possessing identified causes within the domain of HQC algorithms and applications.

### D. Analysis of Codes

After completing the coding process, the authors conducted an analysis to address RQ1-RQ5, which included:

- *Developing a Taxonomy:* Addressing **RQ1**, this involved formulating a structured classification for common issues in HQC applications. Originating from the coding phase, this taxonomy was refined through author discussions, resulting in a detailed four-level hierarchy presented in Fig. 1.
- *Metrics Calculation and Analysis:* For **RQ2** to **RQ4**, the authors computed metrics from the coded data, to extract statistics with respect to domain classification, manifestations, and causes. Further, the taxonomy is color-coded to highlight distinctions between quantum-specific, classical, and cross-domain issues, with metrics offering a detailed overview of each issue’s frequencies and characteristics.
- *Identifying Recurring Challenges:* To answer **RQ5**, we analyzed recurring and predominant challenges faced by developers in the discussion threads, leading to the identification of common developer obstacles and the formulation of actionable recommendations.

## V. RESULTS AND ANALYSIS

### A. RQ1: Recurrent Issues in HQC Applications

Our study is primarily aimed at identifying the spectrum of issues that HQC developers grapple with when building hybrid

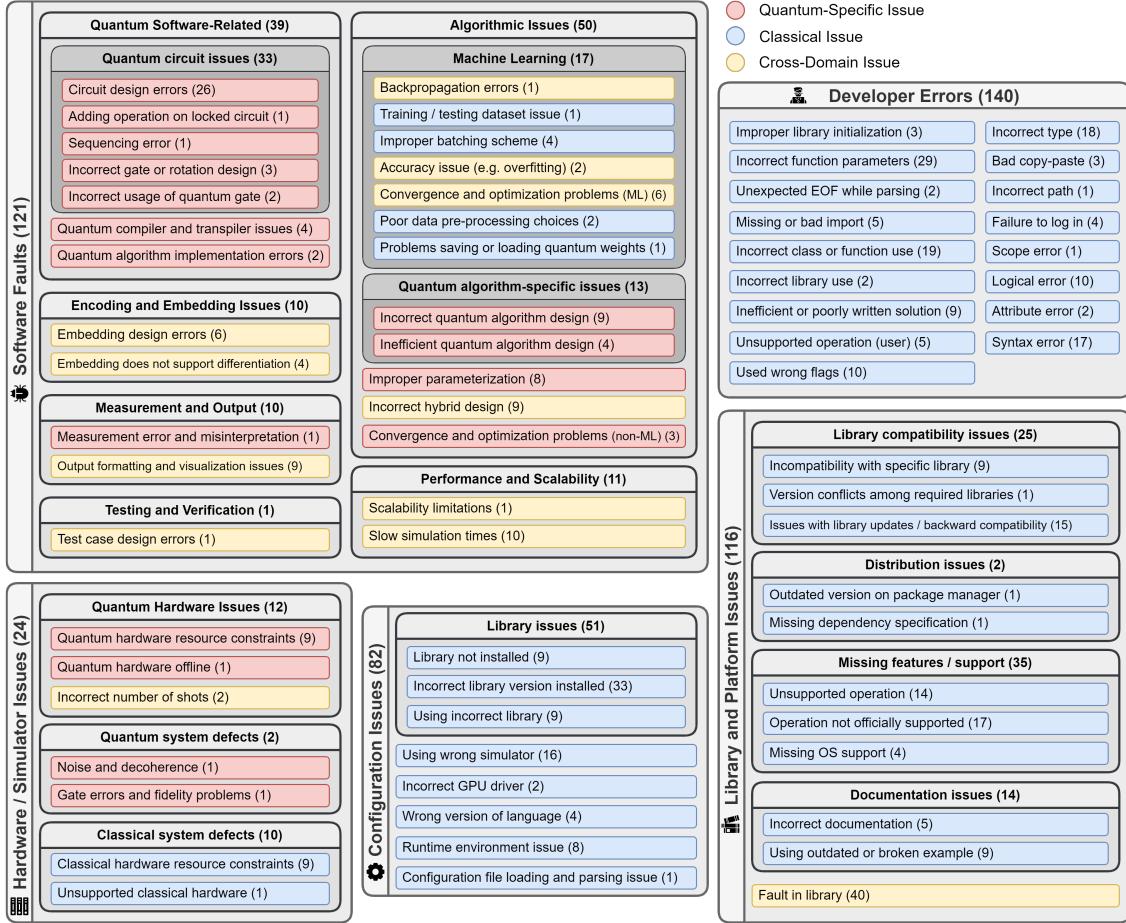


Fig. 1: Taxonomy of issues encountered by developers of HQC applications

applications, extending beyond the realm of conventional ‘bugs’ that prior research has focused on. As we sifted through the forum discussions, it became evident that developers were wrestling with challenges more complex than code faults, but rather broader issues (related to hardware/simulators, library usage and configuration, *etc.*) that significantly hampered HQC application development. Many of these difficulties are symptomatic of the infancy of HQC computing and the brisk evolution of its platforms and practices in QC. Our taxonomy thus captures a wider array of obstacles, including recurrent bug patterns, extracted from the discourse in issue threads.

After categorizing 531 real-world issues from the XDF and QCSE, we analyzed the results and developed a taxonomy as seen in Fig. 1. The taxonomy is organized hierarchically, beginning with five high-level classifications that represent the broad areas where issues arise in HQC applications:

**Software Faults (121)** are issues traditionally referred to as “bugs”. They are faults inherent to the software of HQC applications, where the code or its design is flawed, leading to failures or incorrect program execution. For instance, a software fault includes a circuit design error where the developer uses incorrect gate operations or mismanages qubits. [59].

**Hardware/Simulator Issues (24)** concern problems related

to the physical quantum hardware or quantum computer simulators that HQC applications run on. Issues can range from constraints within the classical simulation environment (*e.g.*, limitations on the number of qubits permitted by a simulator [60]) to quantum hardware resource limitations (*e.g.*, delayed executed due to long queue length on IBM quantum hardware [61]) and even quantum hardware being offline [62].

**Configuration Issues (82)** arise from incorrect or suboptimal setup of the computing environment and/or dependencies used by the developer in building the hybrid application. Unlike the category “Library and Platform Issues” (described below), the problems in this category are generally caused by the developer that uses quantum libraries and can be resolved within the application using such libraries. Examples include having the incorrect version of a library or dependency installed [63], using a wrong simulator [64], and errors in setting up a runtime environment [65].

**Library and Platform Issues (116)** faults and other defects within the libraries and platforms used to develop HQC applications. The issues are generally out of the control of the application developer, who must rely on the platform or library developers to resolve them. Issues in this category include backwards compatibility after library updates [66], operations

that are not officially supported by a library [67], incorrect documentation [68] and bugs in the platforms themselves [69].

**Developer Errors (140)** are mistakes made by application developers, often involving incorrect code syntax [70], incorrect types[71], logic errors [72], or using incorrect function parameters [73]. They are the direct result of human error during the SE process and are generally fixable by correcting the application’s code. Moreover, these issues almost exclusively fall in the realm of classical issues. These types of issues are well-studied in other research, but have been included in the taxonomy for completeness.

The HQC issues categorized according to our taxonomy amount to 483 issues in total, described in 447 discussion threads. In an additional 48 discussion threads, we encountered issues which we could not categorize into the taxonomy due to limited and/or unclear information in the forum thread. Common examples of these threads would be where an original post would go unanswered or where the original poster failed to follow-up to responses in the thread [74, 75].

Each top-level category is further divided into sub-categories, providing more granular insight into specific issues that were observed. To illustrate the frequency of each encountered issue, the taxonomy in Fig. 1 also provides counts for each observed issue. This hierarchical structure enables a systematic examination of recurring issues in HQC application, from general phenomena to specific instances, covering a variety of possible problems a HQC developer could encounter.

Note that our taxonomy includes most bugs and issues outlined in previous works [19, 20, 22, 23], but at different granularities. For instance, Configuration Issues in our taxonomy are broken down into several sub-categories providing more detail as to the issue encountered. This contrasts with prior research that lumped all configuration bugs into a single “misconfiguration” category [19, 22, 24].

Some issue categories, like Circuit Design Errors, are admittedly somewhat broad. The prevalence of these errors aligns with prior studies [19, 20, 22–24]. However, our dataset revealed unique, non-repeating errors (*e.g.*, no measurement, multiple measurements, wrong gate, overlapping gates, wrong qubit order), making finer granularity difficult. This variability in issues encountered underscores the complexities of quantum programming and suggests further research focused specifically on Circuit Design Errors could create more specific subcategories for better classification.

Consequently, there are a handful of bug types, mostly relating to algorithmic and circuit design issues, found in prior works that are not explicitly mentioned in our taxonomy [19, 22]. These bug types are more granular sub-categories of issues, like the aforementioned Circuit Design Errors, found in our taxonomy. A complete chart comparing the issue categories found in taxonomy compared to bugs found in prior works is contained in our replication package [33].

Importantly, though, the taxonomy presented in this paper captures many more bugs and issues that are not contained in prior works due to our focus on HQC applications. These include issues such as software faults related to parameteriza-

tion, convergence and optimization, encoding and decoding, machine learning-specific faults, hardware and simulator issues and, of course, issues specific to HQC application design (see Fig. 1). This is also the case for classical developer errors. Prior works have typically only identified a few of these faults, such syntax errors and type errors [19, 22, 24]. In this study, we were able to capture many more types of developer errors, such as missing or bad imports, used wrong flags, improper library initialization, incorrect function parameters and many others.

Our taxonomy is not just a high-level classification of issues in HQC applications, though. It also serves as a practical tool for debugging and testing. Each category and sub-category offers a structured approach to identifying and addressing specific issues that HQC developers encounter. For instance, developers facing a fault or an error in a library or platform can use the taxonomy’s Library or Platform Issue category to help determine if the problem is due to missing features, poor documentation, version compatibility, or other specific bugs. This detailed guidance helps developers systematically troubleshoot, diagnose, and fix issues, potentially enhancing the efficiency and effectiveness of their debugging processes.

Moreover, the counts in the taxonomy provide insights into the prevalence of different issues in HQC applications. Developers and testers can use this data to prioritize their efforts based on the most common and critical issues. For instance, our taxonomy indicates that “Configuration Issues” are a particularly common occurrence. As a result, developers can focus on improving their setup and configuration processes, while testers can design specific test cases to catch these issues early. These metrics help identify recurring patterns, enabling developers to take proactive measures to prevent common issues in quantum software.

We further discuss a number of issue categories and sub-categories, with examples, in the remaining RQs. Definitions for all categories can be found in our replication package [33].

### B. RQ2: Origins of Issues in HQC Applications

Our taxonomy classifies issues by their origins in HQC systems, where the origins were identified based on whether the fix to the issue was achieved in the quantum and/or classical subsystems of an HQC application. These classifications are delineated with color codes for clarity on their distribution in Fig. 1. We group the identified HQC issues into three categories:

**Quantum-Specific Issues (76)** arise solely within the quantum component of an HQC application, including quantum circuit design [59, 76], gate and rotation operations [77] and quantum algorithm implementation [78].

**Classical Issues (316)** are found in the conventional or classical computing portions of an HQC application [63, 66, 70]. They can also include some machine learning issues as well as classical hardware defects [79].

**Cross-Domain Issues (91)** manifest at the nexus of quantum and classical realms, often being addressed through adjustments in either component. More specifically, these issues are distinguished by their ability to manifest in either the quantum or classical parts of the system and can typically be resolved

TABLE V: Instances of Issue Manifestations by Origin

Manifestation	Quantum-Specific	Classical	Cross-Domain	All Issues
Crash	51 (67.11%)	257 (81.59%)	51 (55.43%)	359 (74.33%)
Incorrect Output	15 (19.74%)	28 (8.89%)	26 (28.26%)	69 (14.29%)
Slow Execution	6 (7.89%)	13 (4.13%)	11 (11.96%)	30 (6.21%)
Unknown	3 (3.95%)	14 (4.44%)	2 (2.17%)	19 (3.93%)
Warning	1 (1.32%)	3 (0.9%)	2 (2.17%)	6 (1.24%)
Total	76 (100%)	315 (100%)	92 (100%)	483 (100%)

through modifications to either subsystem. For example, as shown in Fig. 4, an embedding fault is a cross-domain issue because it can be resolved by changing the quantum embedding circuit(s) or by adapting the classical algorithm and output to conform with the embedding scheme’s constraints. Other issues include incorrect hybrid algorithm design [80] and machine learning-related issues [81]. Notably, this category has not previously been identified in prior quantum bug studies [19, 22].

Tab. V shows the distribution of issues across the three categories. The predominance of classical issues implies the continued relevance and necessity of traditional debugging, error detection, and fault resolution techniques in HQC applications. Moreover, the notable share of cross-domain bugs points to unique challenges that blend quantum and classical computing, as these issues may require a nuanced or more specialized understanding of both quantum and classical computing principles for effective resolution (see Fig. 4). Finally, although quantum-specific issues were less frequent than the other issues, they nonetheless highlight the need for developers to possess special insights into quantum computing principles and employ targeted troubleshooting methods [19].

### C. RQ3: Manifestation of HQC Issues

To answer RQ3, we performed an analysis of how issues encountered in the development of HQC applications manifest themselves in order to understand how to mitigate them. We found that HQC application issues manifested as follows:

**Crash (359):** This is the most direct manifestation of HQC issues, where the application fails to execute to completion, typically resulting in an error message. This abrupt termination could stem from a variety of issues, including quantum circuit and logic errors, classical code exceptions, or incompatibilities between quantum and classical components [71, 81].

**Incorrect Output (69):** At times, the application or algorithm may complete its execution but yield an output that deviates from the expected result [78]. For instance, this discrepancy could be a consequence of logical errors in algorithm implementation, erroneous quantum gate operations, or faulty data processing on the classical side.

**Slow Execution (30):** The performance of HQC applications is often a balancing act between the theoretical speedup offered by QC and the practical limitations of current technology. Slow execution typically arises from inefficient algorithm or circuit designs, limited availability of or long queues for quantum hardware resources, suboptimal simulator performance, or even classical hardware constraints [82].

**Warnings (19):** They serve as a preemptive indicator of potential issues in HQC applications. As shown in Fig. 2, they may inform the user of a suboptimal operation or an

```
% python hybrid_program.py
Warning UserWarning: The device strawberryfields.remote
does not support the PolyXP observable. The analytic
parameter-shift cannot be used for second-order observables;
falling back to finite-differences.

% python hybrid_program.py
WARNING: INSUFFICIENT SUPPORT DETECTED FOR
GPU DEVICE WITH lightning.gpu
!!! DEFAULTING TO CPU DEVICE lightning.qubit
```

Fig. 2: Example PennyLane Warning Messages

Two warning messages encountered by devs in PennyLane that change application behavior regardless of what was specified in the code [84].

automatic fallback to a different simulator or function due to some detected incompatibility or inefficiency [83, 84].

**Unknown (6):** Some analyzed threads did not provide enough information to ascertain how an issue manifests. We labeled these issues as having an unknown manifestation [85].

Notably, we did not encounter issues like corrupted data or hanging applications that are common in classical software [86]. The absence of corrupted data issues likely stems from limited file processing in current QC and HQC applications. While hanging is theoretically possible, it was not observed, possibly due to the stateful nature of quantum components and built-in error handling in platforms like Qiskit and PennyLane, which tend to force crashes.

Tab. V provides an overview of issue manifestations across all identified issues encountered by developers in the threads. These figures contrast starkly with those from classical software studies, where incorrect output predominates [86–88]. The overall higher incidence of crashes in HQC applications likely reflects the relative immaturity and lack of robust error handling in quantum platforms and libraries. The rapidly changing nature of quantum software development tools, coupled with the intricate challenges posed by integrating quantum components, no doubt leads to an increased propensity for crashes. Additionally, the steep learning curve in developing applications leading to algorithmic errors within quantum components may also contribute to the prevalence of crashes.

Quantum-specific issues tended to have more instances of incorrect output and fewer crashes compared to purely classical issues. This may be due to the inherent probabilistic nature of quantum circuit outputs, where even a correctly functioning quantum algorithm might yield varying or unexpected results due to quantum phenomena. This could also result from incorrect quantum logic or poor circuit design, again underscoring the importance of developer education. This is in line with prior work focusing on bugs in quantum platforms [19].

Classical issues appeared to have high instances of crashing. The reasons for this could be manifold. The Library and Platform Issues category from the taxonomy account for 24% of all issues sampled suggesting a landscape where the tools and frameworks like Qiskit and PennyLane are still in a state of flux with updates and changes potentially introducing new points of failure. Furthermore, the quantum learning curve is very steep and can lead to a higher incidence of classical issues as evidenced by the high number of developer errors.

TABLE VI: Instances of Manifestations by Issue Type

Issue Type	Crash	Incorrect Output	Slow Execution	Unknown	Warning
Algorithmic	29	19	1	1	0
Config	68	4	7	1	2
Developer Errors	121	14	3	0	2
Quantum Specific	31	4	0	3	1
Encoding / Embedding	8	1	0	1	0
Library / Platform	85	17	2	10	2
Measurement / Output	3	7	0	0	0
Performance / Scalability	1	0	10	0	0
Hardware / Simulator	13	2	8	0	1
Testing / Verification	0	1	0	0	0
Unknown Cause	34	10	2	1	1
Total	393	79	33	17	9

Cross-domain issues were still crash dominant but had higher instances of incorrect output and slow execution compared to quantum-specific and classical issues. This likely results from our classifications, where slow simulation times may stem from classical hardware issues or quantum algorithm design flaws. Also, cross-domain issues, like embedding issues and hybrid design errors, emerge at the intersection of quantum and classical computing, where mismatches in data formats, incompatible logic flows, or synchronization problems between the two domains can lead to more diverse and complex manifestations of bugs and issues.

When examining issue manifestations in the taxonomy (see Tab. VI), most issue types predominantly cause crashes. However, Algorithmic Issues are more likely to produce higher instances of Incorrect Output, comparatively. This is due to the nature of quantum algorithms, where slight deviations in quantum logic, gate operations, or noise can result in significant discrepancies in output. Unlike crashes, these subtle errors in algorithmic processes may not trigger immediate failures, leading instead to incorrect results that are harder to detect.

Configuration Issues had higher instances of manifesting with slower execution, likely due to suboptimal setup of computing environments or dependencies, which can hinder performance and efficiency. Library and Platform Issues showed notable instances of Incorrect Output and Unknown Manifestations. Incorrect output often stemmed from bugs or undocumented behaviors in libraries, while manifestations with an unknown cause were due to users lacking sufficient data or error messages from platforms for accurate diagnosis. Performance and Scalability Issues, along with Hardware/Simulator Issues, had higher instances of slower execution. This is likely due to the probabilistic nature of quantum hardware, unoptimized simulators, and limited hardware resources.

#### D. RQ4: Causes of HQC Issues

Creating HQC applications introduces unique challenges where the complexity of quantum mechanics meets traditional computing. In exploring the specific root causes of issues encountered by developers in HQC applications, our study identified the following causes of issues:

**Programmer Errors (338):** Discrepancies introduced by developers, ranging from simple syntactical mistakes to deeper conceptual misunderstandings of quantum operations or algorithms, resulting in incorrect or inefficient code.

```
@qml.qnode(dev)
def circuit(params):
    output = []
    for ind in range(8):
        output.append(qml.expval(qml.PauliX(ind)))
        output.append(qml.expval(qml.Identity(ind)))
    - output.append(qml.expval(qml.PauliZ(ind)))
return output
```

Fig. 3: Platform Limitation Example

Attempting multiple measurements on the same wire with non-commutable observables (PauliX / PauliZ) causes errors due to PennyLane restrictions [89]

TABLE VII: Instances of Issue Causes by Origin

	Quantum-Specific	Classical	Cross-Domain	All Issues
Programmer Error	63 (82.89%)	236 (74.92%)	39 (42.39%)	338 (69.98%)
Platform Issue	10 (13.16%)	67 (21.27%)	53 (57.61%)	130 (26.92%)
Platform Limitation	0	3 (0.95%)	0	3 (0.62%)
Hardware Issue	3 (3.95%)	9 (2.86%)	0	12 (2.48%)
Total	76 (100%)	315 (100%)	92 (100%)	483 (100%)

**Platform Issues (130):** Stem from inconsistencies, limitations, or defects within the development environments and libraries themselves used by HQC developers. They manifest in a variety of ways within the QC libraries, such as backward compatibility challenges when library versions change or unexpected behaviors due to poor documentation.

**Platform Limitations (3):** Refer to the inherent constraints set by the development platforms and libraries, which can restrict the capabilities available to developers. Unlike *Platform Issues*, which are unintended faults in the system, platform limitations are known restrictions, such as computational or resource limitations, that developers must navigate within. See Fig. 3 for an example.

**Hardware Issues (12):** Problems that originate from the QC or classical hardware running quantum simulators. These include the physical limitations of quantum devices, such as decoherence, error rates, or connectivity constraints between qubits. While hardware issues are less prevalent in SE stages that primarily utilize simulators, they become significant when applications are deployed on actual quantum hardware where physical phenomena can introduce errors.

Table VII shows statistics for the causes of 483 categorized issues. Programmer Error is the predominant cause across all domains, highlighting the complexity of QC concepts for developers and their application in HQC contexts. This high rate of programmer-induced issues suggests the need for better educational resources, effective debugging tools, and improved guidance on HQC software engineering practices.

The significant portion of platform-induced issues highlights the current state of quantum SE tools, which are still in active development and refinement. As can be seen in Fig. 1, a large number of encountered issues (116) are attributable to Platform and Library Issues. As these platforms mature and stabilize, we might expect a reduction in platform-related issues. In the meantime, platform developers should work more closely with application developers to reduce these issues.

It is notable that hardware-induced issues made up a small portion of our dataset, suggesting that while quantum hardware

is a concern, the current focus of running HQC applications on simulators during development and this NISQ-era of QC guards applications from many quantum hardware-specific issues, particularly those intrinsic to quantum physics.

Tab. VII shows a balanced distribution between platform-induced issues and programmer errors in cross-domain contexts. This reflects the complexity of integrating quantum and classical systems, where issues can often be resolved with either quantum or classical fixes. It appears that both developers and platforms are still refining integration methods as HQC applications become more prevalent.

Additionally, when examining the causes of errors across the taxonomy categories, the results appear to be relatively straightforward. Programmer Errors caused all Developer Errors in the taxonomy and over 90% of Algorithmic, Configuration, Encoding and Embedding, and Quantum Software-Related Issues. Platform-induced errors caused nearly all Library or Platform Issues. Hardware/Simulator Issues were roughly equally caused by Platform Issues and Hardware Issues, reflecting a dual reliance on both hardware and the software platforms managing an HQC application’s operation.

Lastly, Tab. VIII provides some insight on the relationship between manifestations of issues and their causes. As discussed above, crashes are the predominant manifestation, which are primarily attributed to Programmer Errors. This indicates that a substantial proportion of stability problems in HQC applications falls on developers. Additionally, with nearly a quarter of crashes attributable to Platform Issues, quantum computing platforms themselves, such as Qiskit and PennyLane, should continue to focus on refinement and robust error-handling mechanisms to enhance their. This is particularly true in light of the small percentage of hardware issues causing crashes.

Similarly, Programmer Error accounted for nearly two-thirds of issues resulting in Incorrect Output. Meanwhile, Platform Issues accounted for over a quarter. This again highlights the dual need for improved developer training and support as well as efforts by platforms to improve stability.

Issues causing Slow Execution were mainly due to Platform Issues and Programmer Errors, suggesting that developers should continue to try to optimize their quantum algorithms and platforms should focus on enhancing performance. Although less frequent, issues resulting in Warnings highlight the need for better diagnostic and debugging tools, like linters, to preemptively address potential problems before they escalate.

TABLE VIII: Instances of Manifestation Type Based on Cause

	Crash	Incorrect Output	Slow Execution	Warning	Unknown
Platform Issue	82 (20.9%)	21 (26.6%)	18 (54.5%)	2 (28.6%)	7 (36.8%)
Programmer Error	268 (68.2%)	48 (60.8%)	10 (30.3%)	3 (42.9%)	9 (47.4%)
Hardware Issue	8 (3.5%)	0	3 (9.1%)	1 (14.3%)	0
Platform Limitation	1 (2.0%)	0	0	0	2 (10.5%)
Unknown Cause	34 (8.7%)	10 (12.6%)	2 (6.1%)	1 (14.3%)	1 (5.3%)
Total	393	79	33	7	19

#### E. RQ5: Challenges Resolving HQC Bugs

In the course of our study, we observed several predominant and recurring challenges that developers of HQC applications faced. We further analyze these challenges below:

```

Broken Example
n_qubits = 4
dev = qml.device("default.qubit", wires=n_qubits)
inputs = torch.rand(n_qubits, requires_grad=True) # Input data
weights = torch.rand(3, n_qubits, 3), requires_grad=True)
@qml.qnode(dev, interface='torch')
def qnode(inputs, weights):
    qml.templates.AmplitudeEmbedding(processed_inputs, wires=range(n_qubits), normalize=True)

Quantum Fix
@qml.qnode(dev, interface='torch')
def qnode(inputs, weights):
    - qml.templates.AmplitudeEmbedding(processed_inputs, wires=range(n_qubits),
    normalize=True)
    + qml.templates.AngleEmbedding(inputs, wires=range(n_qubits), rotation='Y')

Classical Fix
+ preprocessing_layer = nn.Linear(n_qubits, 2**n_qubits)
+ processed_inputs = preprocessing_layer(inputs)
+
@qml.qnode(dev, interface='torch')
- def qnode(inputs, weights):
+ def qnode(processed_inputs, weights):
    qml.templates.AmplitudeEmbedding(processed_inputs, wires=range(n_qubits),
    normalize=True)

```

Fig. 4: Platform Quirk Example with Cross-Domain Fixes  
AmplitudeEmbedding in PennyLane is not differentiable.

```

- import numpy as np
+ from pennylane import numpy as np

dev = qml.device('default.qubit', wires=2)
@qml.qnode(dev)
- def testAmp(data):
+ def testAmp(data=None):
    AmplitudeEmbedding(data, wires=[0,1], pad=True, normalize=True)
    return qml.expval(qml.PauliZ(0))

```

Fig. 5: Platform Quirks Examples  
Redefining testAmp() to take a keyword argument resolves the error since AmplitudeEmbedding only accepts non-differentiable data.

**Integration of New Features:** A notable hurdle, representing 6.4% of encountered issues, was the lack of support for certain quantum operations, demanding workarounds. Open-source platforms like PennyLane responded swiftly, often incorporating these operations in subsequent versions, and sometimes requesting bug reporters to contribute enhancements (see this thread as an example: [89]). Proactive feature expansion by libraries, mirroring the functionality of quantum hardware in simulators, is recommended for better user support.

**Keeping Pace with Platform Evolution:** The rapid evolution of quantum platforms is a double-edged sword. On one hand, it assists developers by increasing functionality; on the other, it can be prone to introducing errors and faults. As the taxonomy shows, platform and library compatibility issues accounted for 5.2% of the issues encountered. We observed 15 instances where platform updates caused backward compatibility issues with other libraries used by developers. Additionally, the deprecation of entire sub-modules like Qiskit Aqua and Ignis [55] has rendered code, documentation, and resources obsolete. This is reflected in the fact that 1.9% of all issues were caused by broken examples. Moreover, 6.8% of developer issues arose from library-related bugs and issues, primarily due to outdated or incorrect library versions or dependency conflicts. Such rapid changes can confuse developers and disrupt the development of HQC applications. These issues can easily be prevented with increased testing by platforms and better communication between platform and application developers.

**Platform Quirks:** Libraries’ peculiarities, such as PennyLane’s custom NumPy library or the requirement that QNode arguments be non-differentiable if passed as keywords, can cause non-intuitive issues (see Fig. 5) and highlight the practical challenges HQC developers face. In many cases, documentation failed to identify these quirks, and developers were unaware of them until they were raised in forums. To mitigate this, we suggest that platform developers provide clear documentation on quirks, compatibility guides, and tools to switch between library versions. Additionally, implementing warning or descriptive runtime error logs can help developers identify and resolve these issues more efficiently.

**The Quantum Learning Gap:** Around 10.4% of issues stemmed from Algorithm Issues in HQC applications. Another 10.1% of issues were Quantum Software-Related or quantum Encoding and Embedding Issues. These issues most likely stemmed from a lack of understanding of quantum mechanics. Yet, a significant portion of other issues encountered, such as those in the Configuration Issues and Developer Errors categories, stem from classical errors rooted in the misunderstanding of the requirements of quantum platforms. This underscores the necessity for developers to have a robust grasp of both quantum and classical computing principles when developing HQC applications.

For application developers, we recommend developing formal debugging and testing procedures tailored to HQC applications to address these issues, particularly algorithmic bugs. Ensuring environment consistency, such as using containerization technologies or virtual environments, can prevent many configuration-related issues. Regularly updating libraries and dependencies helps avoid compatibility problems and benefits from the latest improvements and bug fixes. However, care must be taken to manage library updates to avoid backward compatibility issues, such as by testing updates in a controlled environment before full deployment.

Platforms and frameworks can reduce the learning gap by maintaining up-to-date documentation and providing clear guidance on common pitfalls. Additionally, continuous learning through courses, workshops, and community engagement is essential to keep up with the rapidly evolving field of quantum computing. Engaging with other quantum developers through forums, conferences, and collaborative projects can provide support and facilitate the sharing of best practices.

**Expanding Developer Resources:** While resources such as the PennyLane and Qiskit documentation provide considerable support to developers of HQC applications, there were still 48 issues (9%) that remained unresolved in forums. Some of these resulted from poor issue reporting or a failure to follow up with requested information (which is also a significant issue in purely classical systems [90]). This indicates a need for more extensive education and training in quantum development. Moreover, there is concern that a significant amount of quantum knowledge could become sequestered within proprietary industry and government projects, limiting widespread learning and development.

We recommend quantum platforms implement streamlined

error reporting mechanisms, like the Xanadu Discussion Forums, that allow application developers to quickly and efficiently identify and resolve issues. Also, establishing feature request platforms can foster better communication between platform developers and applications developers, ensuring that the most pressing needs are addressed quickly and efficiently.

In sum, resolving the challenges and issues that arise during the development of HQC applications is multi-faceted. By addressing these issues through improved development practices, comprehensive and current documentation, and robust platform designs that account for backward and cross-library compatibility, developers can build more reliable hybrid applications and engage in more streamlined resolution processes when an issue is encountered.

## VI. THREATS TO VALIDITY

**Internal Validity.** Our study may be influenced by the choice of data sources, XDF and QCSE, possibly omitting other HQC issues. Single-author thread removal and manual coding introduced subjectivity, despite spot-checks and reconciliation efforts to ensure consistency. The manual nature of the coding could inherently allow for human error in issue identification and classification. We attempted to mitigate some of these threats by applying a rigorous coding methodology.

**External Validity.** The generalizability of our findings is restricted by the selected forums, which might not be fully generalizable to all HQC applications. The rapid evolution of QC could limit the long-term relevance of our issue taxonomy.

**Construct Validity.** The nascent quantum software field’s fluid definitions may affect the construct validity. Our taxonomy, reflecting the current state of HQC understanding, might require updates as the industry matures and evolves.

**Replicability.** While detailed methods and documentation aim to enhance replicability, the fast-paced changes in QC and inherent subjectivity in coding could lead to different results in future replications of this study.

## VII. CONCLUSIONS

We examined, analyzed, and categorized 531 real-world issues, discussed in quantum-focused forums, to construct a comprehensive taxonomy of issues encountered by developers working on HQC applications. The taxonomy identifies key areas where developers struggle, organizing them into a clear framework that can assist developers in identifying and addressing common problems more efficiently. We discussed these recurring challenges and provided actionable recommendations to address them. Lastly, we analyzed how HQC bugs manifest, their causes, and which domain they predominated.

Looking ahead, HQC application development is still in its infancy and there is a clear need for SE perspectives. This includes designing and improving tools for debugging, enhancing learning materials for developers, and ensuring that the platforms and libraries used continue to incorporate new features while maintaining compatibility. Addressing these needs will help make working with HQC systems more accessible, reliable, and efficient, paving the way for broader adoption and optimization of HQC systems.

## REFERENCES

- [1] J. Preskill, “Quantum computing 40 years later,” in *Feynman Lectures on Computation*. CRC Press, 2023, pp. 193–244.
- [2] M. Nagy and S. G. Akl, “Quantum computing: Beyond the limits of conventional computation,” *The International Journal of Parallel, Emergent and Distributed Systems*, vol. 22, no. 2, pp. 123–135, 2007.
- [3] G. Aleksandrowicz, T. Alexander, P. Barkoutsos, L. Bello, Y. Ben-Haim, D. Bucher, F. J. Cabrera-Hernández, J. Carballo-Franquis, A. Chen, C.-F. Chen *et al.*, “Qiskit: An open-source framework for quantum computing,” *Accessed on: Mar*, vol. 16, 2019.
- [4] G. A. Q. Team, “Cirq.” 2018. [Online]. Available: <https://github.com/quantumlib/Cirq>
- [5] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, “Q# enabling scalable quantum computing and development with a high-level dsl,” in *Proceedings of the real world domain specific languages workshop 2018*, 2018, pp. 1–10.
- [6] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, “Pennylane: Automatic differentiation of hybrid quantum-classical computations,” *arXiv preprint arXiv:1811.04968*, 2018.
- [7] A. Bhasin and M. Tripathi, “Quantum computing at an inflection point: Are we ready for a new paradigm,” *IEEE Transactions on Engineering Management*, 2021.
- [8] R. Jozsa and N. Linden, “On the role of entanglement in quantum-computational speed-up,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 459, no. 2036, pp. 2011–2032, 2003.
- [9] F. Battistel, C. Chamberland, K. Johar, R. W. Overwater, F. Sebastiani, L. Skoric, Y. Ueno, and M. Usman, “Real-time decoding for fault-tolerant quantum computing: Progress, challenges and outlook,” *arXiv preprint arXiv:2303.00054*, 2023.
- [10] A. Callison and N. Chancellor, “Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond,” *Physical Review A*, vol. 106, no. 1, p. 010101, 2022.
- [11] D. Arthur and P. Date, “Hybrid quantum-classical neural networks,” in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022, pp. 49–55.
- [12] S. Endo, Z. Cai, S. C. Benjamin, and X. Yuan, “Hybrid quantum-classical algorithms and quantum error mitigation,” *Journal of the Physical Society of Japan*, vol. 90, no. 3, p. 032001, 2021.
- [13] E. H. Houssein, Z. Abohashima, M. Elhoseny, and W. M. Mohamed, “Hybrid quantum-classical convolutional neural network model for covid-19 prediction using chest x-ray images,” *Journal of Computational Design and Engineering*, vol. 9, no. 2, pp. 343–363, 2022.
- [14] M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, and J. L. Hevia, “Toward a quantum software engineering,” *IT Professional*, vol. 23, no. 1, pp. 62–66, 2021.
- [15] S. Ali, T. Yue, and R. Abreu, “When software engineering meets quantum computing,” *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, 2022.
- [16] J. Zhao, “Quantum software engineering: Landscapes and horizons,” *arXiv preprint arXiv:2007.07047*, 2020.
- [17] L. S. Barbosa, “Software engineering for ‘quantum advantage’,” in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 427–429.
- [18] E. Mendiluze, S. Ali, P. Arcaini, and T. Yue, “Muskit: A mutation analysis tool for quantum software testing,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1266–1270.
- [19] M. Paltenghi and M. Pradel, “Bugs in quantum computing platforms: an empirical study,” *Proceedings of the ACM on Programming Languages*, vol. 6, no. OOPSLA1, pp. 1–27, 2022.
- [20] H. Li, F. Khomh, L. Tidjon *et al.*, “Bug characteristics in quantum software ecosystem,” *arXiv preprint arXiv:2204.11965*, 2022.
- [21] P. Zhao, J. Zhao, and L. Ma, “Identifying bug patterns in quantum programs,” in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE, 2021, pp. 16–21.
- [22] J. Luo, P. Zhao, Z. Miao, S. Lan, and J. Zhao, “A comprehensive study of bug fixes in quantum programs,” in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 1239–1246.
- [23] P. Zhao, J. Zhao, Z. Miao, and S. Lan, “Bugs4q: A benchmark of real bugs for quantum programs,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1373–1376.
- [24] P. Zhao, X. Wu, J. Luo, Z. Li, and J. Zhao, “An empirical study of bugs in quantum machine learning frameworks,” *arXiv preprint arXiv:2306.06369*, 2023.
- [25] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.
- [26] S. Fernández-Lorenzo, D. Porras, and J. J. García-Ripoll, “Hybrid quantum-classical optimization with cardinality constraints and applications to finance,” *Quantum Science and Technology*, vol. 6, no. 3, p. 034010, 2021.
- [27] Y. Ding, X. Chen, L. Lamata, E. Solano, and M. Sanz, “Implementation of a hybrid classical-quantum annealing algorithm for logistic network design,” *SN Computer Science*, vol. 2, pp. 1–9, 2021.
- [28] A. Rosmanis, “Hybrid quantum-classical search algorithms,” *arXiv preprint arXiv:2202.11443*, 2022.
- [29] C. Walsh, “The past, present, and future of us government investment in quantum information science: <https://www.quantum.gov>,” 2023.
- [30] A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess *et al.*, “Industry quantum computing applications,” *EPJ Quantum Technology*, vol. 8, no. 1, p. 25, 2021.
- [31] X. AI, “Xanadu discussion forums,” 2023. [Online]. Available: <https://discuss.pennylane.ai/>
- [32] Q. C. S. Exchange, “Quantum computing stack exchange forums,” 2023. [Online]. Available: <https://quantumcomputing.stackexchange.com/>
- [33] “Online replication package,” [https://archive.softwareheritage.org/browse/origin/directory/?origin\\_url=https://github.com/jakezappin/HQC\\_Bug\\_Replication](https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/jakezappin/HQC_Bug_Replication), 2024.
- [34] J. L. Hevia, G. Peterssen, C. Ebert, and M. Piattini, “Quantum computing,” *IEEE Software*, vol. 38, no. 5, pp. 7–15, 2021.
- [35] T. Hey, “Quantum computing: an introduction,” *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 105–112, 1999.
- [36] E. Rieffel and W. Polak, “An introduction to quantum computing for non-physicists,” *ACM Computing Surveys (CSUR)*, vol. 32, no. 3, pp. 300–335, 2000.
- [37] A. Einstein, M. Born, and H. Born, *The Born-Einstein Letters: Correspondence Between Albert Einstein and Max and Hedwig Born from 1916-1955, with Commentaries by Max Born*. Macmillan, 1971. [Online]. Available: <https://books.google.com/books?id=HvZAAQAAIAAJ>
- [38] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, “Quantum computational chemistry,” *Reviews of Modern Physics*, vol. 92, no. 1, p. 015003, 2020.
- [39] R. Shaydulin, C. Li, S. Chakrabarti, M. DeCross, D. Herman, N. Kumar, J. Larson, D. Lykov, P. Minssen, Y. Sun *et al.*, “Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem,” *arXiv preprint arXiv:2308.02342*, 2023.
- [40] V. Bhatia and K. Ramkumar, “An efficient quantum computing technique for cracking rsa using shor’s algorithm,” in *2020 IEEE 5th international conference on computing communication and automation (ICCCA)*. IEEE, 2020, pp. 89–94.
- [41] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, “Opportunities and challenges for quantum-assisted machine learning in near-term quantum computers,” *Quantum Science and Technology*, vol. 3, no. 3, p. 030502, 2018.
- [42] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.
- [43] ———, “Fault-tolerant quantum computation,” in *Introduction to quantum computation and information*. World Scientific, 1998, pp. 213–269.
- [44] D. Gottesman, “An introduction to quantum error correction and fault-tolerant quantum computation,” in *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, vol. 68, 2010, pp. 13–58.
- [45] D. Castelvecchi, “Ibm releases first-ever 1,000-qubit chip,” 2023. [Online]. Available: <https://www.nature.com/articles/d41586-023-03854-1>
- [46] IBM, “Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two,” 2022. [Online]. Available: <https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two>

- [47] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, “Noisy intermediate-scale quantum algorithms,” *Reviews of Modern Physics*, vol. 94, no. 1, p. 015004, 2022.
- [48] S. S. Tannu and M. K. Qureshi, “Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.
- [49] F. Phillipson, N. Neumann, and R. Wezeman, “Classification of hybrid quantum-classical computing,” in *International Conference on Computational Science*. Springer, 2023, pp. 18–33.
- [50] A. Chan, Z. Shi, L. Dellantonio, W. Dür, and C. A. Muschik, “Hybrid variational quantum eigensolvers: merging computational models,” *arXiv preprint arXiv:2305.19200*, 2023.
- [51] G. Acampora, A. Chiatto, and A. Vitiello, “Genetic algorithms as classical optimizer for the quantum approximate optimization algorithm,” *Applied Soft Computing*, vol. 142, p. 110296, 2023.
- [52] G. De Luca, “A survey of nisq era hybrid quantum-classical machine learning research,” *Journal of Artificial Intelligence and Technology*, vol. 2, no. 1, pp. 9–15, 2022.
- [53] L. Woessig, “Quantum machine learning for classical data,” *arXiv preprint arXiv:2105.03684*, 2021.
- [54] P. K. Nayak, K. V. Kher, M. B. Chandra, M. Rao, and L. Zhang, “Q-pac: Automated detection of quantum bug-fix patterns,” *arXiv preprint arXiv:2311.17705*, 2023.
- [55] I. Qiskit, “Qiskit algorithms migration guide,” 2021. [Online]. Available: [https://qiskit.org/documentation/stable/0.28/aqua\\_tutorials/Qiskit%20Algorithms%20Migration%20Guide.html](https://qiskit.org/documentation/stable/0.28/aqua_tutorials/Qiskit%20Algorithms%20Migration%20Guide.html)
- [56] X. AI, “Xanadu, linkedin profile,” 2023. [Online]. Available: <https://ca.linkedin.com/company/xanaduai>
- [57] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, “Not all bugs are the same: Understanding, characterizing, and classifying bug types,” *Journal of Systems and Software*, vol. 152, pp. 165–181, 2019.
- [58] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.
- [59] “Optimization of function,” 4 2022. [Online]. Available: <https://discuss.pennylane.ai/t/optimization-of-function/1829>
- [60] “Failing to create a circuit for a large molecule,” 7 2023. [Online]. Available: <https://discuss.pennylane.ai/t/failing-to-create-a-circuit-for-a-large-molecule/3183>
- [61] “Pennylane-qiskit connection error,” 9 2020. [Online]. Available: <https://discuss.pennylane.ai/t/pennylane-qiskit-connection-error/571>
- [62] “The requested hardware device is offline for maintenance,” 2 2021. [Online]. Available: <https://discuss.pennylane.ai/t/the-requested-hardware-device-is-offline-for-maintenance/798>
- [63] “Classical algorithm to find the ground state of a hamiltonian,” 12 2022. [Online]. Available: <https://discuss.pennylane.ai/t/getting-x8-photon-count-data-to-agree-with-numerical-prediction-for-beamsplitter-phase-sweep/1530/>
- [64] “Noisy channels with default.qubit,” 3 2021. [Online]. Available: <https://discuss.pennylane.ai/t/noisy-channels-with-default-qubit/896>
- [65] “Issue in running the qiskit tutorial for qsvm,” 4 2021. [Online]. Available: <https://quantumcomputing.stackexchange.com/questions/16800/issue-in-running-the-qiskit-tutorial-for-qsvm>
- [66] “Problems with forest plugin,” 9 2020. [Online]. Available: <https://discuss.pennylane.ai/t/problems-with-forest-plugin/558>
- [67] “Quantum natural gradient optimizer with user-defined gates,” 4 2021. [Online]. Available: <https://discuss.pennylane.ai/t/quantum-natural-gradient-optimizer-with-user-defined-gates/1006>
- [68] “Pennylane for quantum machine learning,” 1 2021. [Online]. Available: <https://discuss.pennylane.ai/t/pennylane-for-quantum-machine-learning/766>
- [69] “Different results with vqe on versions 0.26.0 and 0.27.0,” 12 2022. [Online]. Available: <https://discuss.pennylane.ai/t/different-results-with-vqe-on-versions-0-26-0-and-0-27-0/2332>
- [70] “Bit flip, separable state and several question about cirq,” 10 2020. [Online]. Available: <https://quantumcomputing.stackexchange.com/questions/14288/bit-flip-separable-state-and-several-question-about-cirq>
- [71] “Error while converting qnode output to ndarray,” 4 2021. [Online]. Available: <https://discuss.pennylane.ai/t/error-while-converting-qnode-output-to-ndarray/971>
- [72] “Api error when trying to access ibm quantum computer using ibmq api token,” 2 2021. [Online]. Available: <https://quantumcomputing.stackexchange.com/questions/15910/api-error-when-trying-to-access-ibm-quantum-computer-using-ibmq-api-token>
- [73] “How can i use density as output and build my qnn model,” 6 2023. [Online]. Available: <https://discuss.pennylane.ai/t/how-can-i-use-density-as-output-and-build-my-qnn-model/3038>
- [74] “qiskit: Traveling salesman problem using qaoa fails for more than 3 cities,” 10 2021. [Online]. Available: <https://quantumcomputing.stackexchange.com/questions/15163/qiskit-traveling-salesman-problem-using-qaoa-fails-for-more-than-3-cities>
- [75] “The infeasibility of the ‘qiskit.aer’ training cost function?” 3 2023. [Online]. Available: <https://discuss.pennylane.ai/t/the-infeasibility-of-the-qiskit-aer-training-cost-function/2769>
- [76] “Making a one layer neural network,” 8 2020. [Online]. Available: <https://discuss.pennylane.ai/t/making-a-one-layer-neural-network/504>
- [77] “How to implement su(2) rotation with qiskit?” 5 2022. [Online]. Available: <https://quantumcomputing.stackexchange.com/questions/26548/how-to-implement-su2-rotation-with-qiskit/26572>
- [78] “Cv convolutional neural network,” 5 2020. [Online]. Available: <https://discuss.pennylane.ai/t/cv-convolutional-neural-network/408>
- [79] “Problem on running quantumgan,” 10 2023. [Online]. Available: <https://discuss.pennylane.ai/t/problem-on-running-quantumgan/3569>
- [80] “Is there a way to parallelize the same circuit for multiple input data?” 2 2023. [Online]. Available: <https://discuss.pennylane.ai/t/is-there-a-way-to-parallelize-the-same-circuit-for-multiple-input-data/2516>
- [81] “Variational classifiers and qnqoptimizer,” 8 2020. [Online]. Available: <https://discuss.pennylane.ai/t/variational-classifiers-and-qnqoptimizer/524>
- [82] “Running 30 qubit highly entangled circuit,” 11 2021. [Online]. Available: <https://discuss.pennylane.ai/t/running-30-qubit-highly-entangled-circuit/1474>
- [83] “Does pennylane-sf plugin sammulate borealis,” 7 2022. [Online]. Available: <https://discuss.pennylane.ai/t/does-pennylane-sf-plugin-sammulate-borealis/2053>
- [84] “Pennylane 0.19 and lightning qbit,” 11 2021. [Online]. Available: <https://discuss.pennylane.ai/t/pennylane-0-19-and-lightning-qbit/1479>
- [85] “qml.mps offset control,” 4 2023. [Online]. Available: <https://discuss.pennylane.ai/t/qml-mps-offset-control/2890>
- [86] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, “Bug characteristics in open source software,” *Empirical software engineering*, vol. 19, pp. 1665–1705, 2014.
- [87] D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi, “How do bugs surface? a comprehensive study on the characteristics of software bugs manifestation,” *Journal of Systems and Software*, vol. 113, pp. 27–43, 2016.
- [88] A. Di Franco, H. Guo, and C. Rubio-González, “A comprehensive study of real-world numerical bug characteristics,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 509–519.
- [89] “One wire with multi-observables,” 5 2021. [Online]. Available: <https://discuss.pennylane.ai/t/one-wire-with-multi-observables/1032/4>
- [90] O. Chaparro, C. Bernal-Cárdenas, J. Lu, K. Moran, A. Marcus, M. Di Penta, D. Poshyvanyk, and V. Ng, “Assessing the quality of the steps to reproduce in bug reports,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 86–96.