Analyzing the Evolution and Maintenance of Quantum Software Repositories

Krishna Upadhyay*, Vinaik Chhetri*, A.B. Siddique†, Umar Farooq*
* Louisiana State University, † University of Kentucky
Email: kupadh4@lsu.edu, vchhet2@lsu.edu, siddique@cs.uky.edu, ufarooq@lsu.edu

Abstract—Quantum computing is rapidly advancing, but quantum software development faces significant challenges, including a steep learning curve, high hardware error rates, and a lack of mature engineering practices. This study conducts a large-scale mining analysis of over 21,000 GitHub repositories, containing 1.2 million commits from more than 10,000 developers, to examine the evolution and maintenance of quantum software. We analyze repository growth, programming language and framework adoption, and contributor trends, revealing a 200% increase in repositories and a 150% rise in contributors since 2017. Additionally, we investigate software development and maintenance practices, showing that perfective commits dominate (51.76%), while the low occurrence of corrective commits (18.54%) indicates potential gaps in bug resolution. Furthermore, 34% of reported issues are quantum-specific, highlighting the need for specialized debugging tools beyond conventional software engineering approaches. This study provides empirical insights into the software engineering challenges of quantum computing, offering recommendations to improve development workflows, tooling, and documentation. We are also open-sourcing our dataset to support further analysis by the community and to guide future research and tool development for quantum computing. The dataset is available at: **https://github.com/kriss-u/QRepoAnalysis-Paper**.

Index Terms—Quantum Software Engineering, Software Repositories, Software Mining, Software Evolution, Software Maintenance.

I. INTRODUCTION

Quantum computing has the potential to transform industries by solving problems that are currently beyond the capabilities of classical computers. Quantum computers can perform calculations exponentially faster in certain applications using quantum mechanics and facilitate breakthroughs in many fields such as cryptography, material science, and optimization. Shor's algorithm threatens modern encryption by efficiently factoring large integers [45], while Grover's algorithm accelerates unstructured database searches [25]. The economic impact is expected to be significant, with projections suggesting that the global quantum computing market could surpass $125 billion by 2030 [46]. This potential has driven major investments, such as the $1.2 billion U.S. National Quantum Initiative [1], along with contributions from industry including IBM, Google, and Microsoft [43]. However, quantum software development faces unique challenges compared to classical computing, such as the probabilistic nature of quantum operations, high error rates in hardware, and a steep learning curve for developers [26].

Despite rapid advancements in quantum hardware, quantum software development remains challenging and underdeveloped. Frameworks like Qiskit [2] and Cirq [4] are still maturing and lack the robustness of classical software ecosystems [2], [18]. Developers face a steep learning curve due to the complexity of quantum models, the probabilistic nature of computations, and the need for a strong grasp of quantum mechanics [17], [32]. The field also suffers from a lack of standardized tools, debugging environments, and educational resources, which collectively hinder progress [26]. These challenges highlight the need for analyzing real-world repositories to better understand current quantum programming practices and identify key areas for improvement.

State-of-the-Art. Prior research on quantum software has largely concentrated on small-scale or qualitative analyses of developer challenges. Paltenghi and Pradel [38] examined 18 quantum projects to identify recurring bug patterns, while Zhao et al. [49] analyzed 36 bugs in Qiskit, highlighting key debugging challenges. In a separate study, Zhao and others [48] explored 391 quantum machine learning issues across 22 projects. More recently, Chen et al. [16] investigated quantum-specific code smells in 15 programs. Other studies have analyzed discussions from Stack Exchange, GitHub issues, and Xanadu Discussion Forums [19], [47], offering qualitative insights into quantum software engineering. However, these works are either constrained by their limited dataset size or focus on specific frameworks, leaving a gap in understanding the broader landscape of quantum software development.

Overview of This Work. Unlike prior work, this work conducts a systematic mining analysis of over 21 thousand GitHub repositories, which contain more than 1.2 million commits contributed by more than 10,000 unique developers. By leveraging this large dataset, we extract broad trends, analyze software activity patterns, and identify key challenges in quantum computing development. This data-driven approach provides a comprehensive perspective on the state of quantum software and offers previously unexplored quantitative insights. Our analysis reveals a 200% increase in the number of repositories and a 150% rise in contributors since 2017, indicating the rapid expansion of quantum computing thanks to the availability of frameworks like Qiskit [2], Cirq [4], and PennyLane [6]. We further examine trends in programming languages and frameworks and find out that Python and Qiskit dominate but that there is also a shift toward specialized tools. Additionally, we discover that quantum computing projects are focused on simulation, machine learning, and algorithm development, reinforcing the field's evolving priorities and the

1

need for robust infrastructure and quantum-specific tools.

While analyzing the state of maintenance, we examine commit patterns and note an increase in both commit volume and size over time, which reflects growing project complexity and heightened developer engagement. A deeper analysis of reported issues reveals that over 34% of the issues are specific to quantum computing. Furthermore, our analysis of commits identifies that perfective commits, aimed at refining functionality, are the most common, while fewer corrective commits suggest areas for improvement in bug resolution. Last but not least, we categorize repositories by activity levels and demonstrate that high-activity repositories typically feature frequent updates, larger codebases, and greater community involvement, in contrast to low-activity projects, which exhibit sporadic updates.

Our findings highlight challenges and opportunities in quantum software engineering, emphasizing the need for improved development workflows, enhanced debugging tools, and more structured maintenance practices. The dominance of perfective commits over corrective commits suggests a gap in bug resolution, indicating that better issue-tracking mechanisms and automated testing frameworks could strengthen software reliability. The high proportion of quantum-specific issues indicates the necessity for specialized tooling, such as domain-aware debuggers and enhanced simulators, to support developers in tackling the unique complexities of quantum programming. Moreover, the disparity between high- and low-activity repositories suggests that community-driven initiatives, improved documentation, and more accessible learning resources could help sustain engagement in quantum software projects.

In summary, this work establishes a solid foundation to drive targeted advancements in quantum software and to promote sustained growth and technical progress. Through an in-depth analysis of development patterns, community structure, and maintenance practices, we provide actionable insights for enhancing quantum programming tools, documentation, and community resources. We are open-sourcing our dataset to facilitate additional analysis by the community to guide future research and inspire the development of specialized tools for the unique demands of quantum computing.

II. BACKGROUND AND RELATED WORK

A. Quantum Software Development

Quantum software encompasses quantum platforms that provide the tools and environments developers need to write and execute quantum programs. Developers use these platforms to create algorithms that run on quantum hardware or simulators, bridging theoretical quantum concepts with practical applications.

Quantum platforms generally have three main components: a programming language, a compiler, and an execution environment. Quantum programming languages include both API-based libraries and stand-alone languages. For instance, Qiskit provides quantum programming abstractions as a Python library, making it accessible within a well-known host language [28]. In contrast, other languages, such as Q# [36],

Silq [11], and Quipper [23], are stand-alone languages specifically designed for quantum programming, offering dedicated syntax and semantics to better align with quantum computing paradigms. These languages help developers create complex quantum operations through high-level abstractions like qubits, gates, and circuits. The compiler component translates high-level quantum code into low-level instructions optimized for execution on quantum hardware, often using intermediate representations like QASM [12]. This process standardizes translation, applies optimizations, and reduces error-prone operations, which is crucial given the sensitivity of quantum systems to noise. Finally, the execution environment supports running quantum programs on either actual quantum devices or simulators that emulate quantum conditions. Simulators are especially useful during development, as they allow testing under controlled conditions with noise models that approximate real hardware behavior [21].

B. Related Work

Mining software repositories research has produced a vast array of studies across different domains, including machine learning, programming language adoption, mobile apps, and blockchain, among others. Research on language trends, such as [9], [42], reveal how shifts impact software quality. Studies on bug prediction leverage historical data to improve defect prediction [31], [41]. Eyolfson et al. [20] explored how developer experience and commit timing affect bug introduction, shedding light on factors impacting software reliability. Mining repositories in Android development has exposed issues like API misuse and permission risks [10], [33], while blockchain security research identifies common vulnerabilities in smart contracts [39]. Additionally, Buse and Weimer [13] automated documentation from commit messages to enhance traceability. These studies exemplify how mining repositories capture critical insights from version control data to advance software quality, security, and development practices.

Existing studies on quantum software development primarily focus on small datasets, often limited to specific platforms or a narrow subset of quantum issues. For example, Paltenghi and Pradel [38] conducted an empirical study of bugs in quantum computing platforms by analyzing 223 bugs in 18 open-source projects, including major platforms such as Qiskit, Cirq, and Q#. Although this study provides critical insights into quantum-specific bugs and recurring bug patterns, its scope remains limited to specific platforms, offering a narrower perspective on the quantum software landscape. Similarly, Zhao et al. [49] collected a dataset of 36 bugs exclusively from the Qiskit platform, highlighting issues unique to quantum programming but restricting the analysis to a single framework.

This study seeks to fill this gap by analyzing a large dataset of GitHub repositories related to quantum computing. Through this analysis, we aim to provide a snapshot of the current state of quantum computing development, identify key trends, and offer insights into the collaborative dynamics within the community. Our findings will contribute to a deeper under-

TABLE I: File patterns used to extract package dependencies by programming language.

Language	File Patterns
Python	`Pipfile`, `pyproject.toml`, `*requirements*.txt`, `requires*.txt`, `required*.txt`, `setup.py`, `setup.cfg`
C++	`CMakeLists.txt`, `Makefile`, `conanfile.txt`
C	`CMakeLists.txt`, `Makefile`
Lisp	`Makefile`
Julia	`Project.toml`
Rust	`Cargo.toml`

TABLE II: Languages and their package names patterns.

Language	File Patterns
Python	`cirq`, `amazon-braket-sdk`, `qiskit`
C++	`cuda-quantum`, `qpp` ,`qrack`
C	`forest-sdk`, `qrack`
Lisp	`forest-sdk`, `qcl`
Julia	`PastaQ`, `Yao`
Rust	`quil-rs`

standing of the landscape of quantum computing research and development and inform future efforts to advance the field.

III. METHODOLOGY

A. Study Objective and Research Questions

Following the Goal Question Metric (GQM) guidelines [14], our research goal is structured as follows: Analyze *quantum computing projects* for the purpose of *exploring and categorizing* with respect to *their current status, evolution, and maintenance* from the point of view of quantum computing researchers and developers in the context of GitHub repositories.

Two main research questions (RQ) arise from this goal. We explore quantum computing repositories to understand their development, popularity, and maintenance.

> **RQ1**. *What is the current status and evolution of the Quantum Computing Community?*

- RQ1.1: How has the popularity of quantum computing changed?
- RQ1.2: How have programming languages and framework usage trends evolved?
- RQ1.3: What is the developers' distribution for different frameworks and how do developers collaborate?
- RQ1.4: What trends and insights can be derived from topics and project categories?

> **RQ2**. *How can we assess and classify the maintenance status of Quantum Computing repositories on GitHub based on their commit and issues data?*

- RQ2.1: What commit patterns reveal over time?
- RQ2.2 How do the size and frequency of commits evolve over time?
- RQ2.3 What are the prominent topics in the reported issues?
- RQ2.4: How do various commit types (perfective, corrective, adaptive) influence the maintenance of repositories?
- RQ2.5: How can commit data be used to categorize the activity status of individual repositories?
- RQ2.6: How do repository characteristics vary across different activity levels?

B. Dataset Construction

1) Data Collection: We collected data from the Quantum Open Software Foundation (QOSF) list [8] and expanded it using GitHub's Code Search API [5], focusing on package dependencies in standard Python project files (e.g., `requirements.txt`, `setup.py`) and keyword searches in relevant files (see Table I). Due to the API's 1,000-result limit, we refined our search scope and manually inspected selected repositories. We also included repository forks, resulting in 95,011 repositories (2,799 originals and 92,212 forks).

2) Preprocessing: Since GitHub Code Search matches any substring, we need to make sure identified packages are genuinely used and not false positives. During the process, we manually inspected 1,300 search results to verify quantum-related repositories. We identified two common false-positive patterns: (1) repositories that copied content instead of forking and (2) monorepos containing both quantum-specific and unrelated libraries, such as the Azure SDK for .NET [3]. To address this, we compiled a list of such repositories for exclusion.

Next, we shallow-cloned the repositories and analyzed standard files for package occurrences. We used `tokei` to detect programming languages and then searched for relevant package names in corresponding files (Table I). We also maintained a mapping of programming languages to quantum framework package names (Table II). Since `tokei` does not detect certain languages like Q#, we used GitHub Linguist [22] as a fallback. This process yielded 24,374 repositories.

For commit analysis, we removed redundant commits from forks, focusing only on new developments. We also excluded tagged version commits (`ref: tag`), bot-authored commits, and anomalously high-volume commits. After multiple iterations of data cleaning and manual inspection, we retained 24,122 repositories and 1,232,828 commits from 10,697 users.

C. Dataset Analysis

Next, we describe the dataset and explain how various data points help address the research questions (RQs) outlined earlier. Our dataset includes 24,122 quantum computing repositories from GitHub, containing over 1.2 million commits, and 157,471 issues. These repositories cover diverse topics, such as simulation, quantum machine learning, algorithms, and frameworks, providing a foundation for analyzing the current state and evolution of quantum software.

1) Repository Characteristics: The dataset captures key information on project scale, community engagement, and activity levels, supporting RQ1 on the status and growth of quantum software. Typical repositories in the dataset have around 3.2 stars and 1.1 forks, though popular repositories like `Qiskit/qiskit` attract significantly more attention, with 5,167 stars and 2,353 forks, reflecting high community interest. Repositories contain 132,707 lines of code on average,
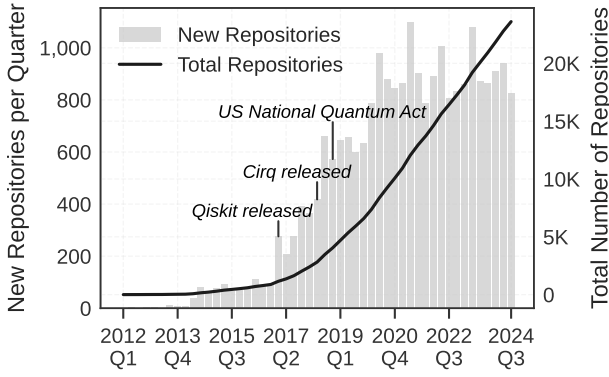
Fig. 1: Quantum Computing repository growth over time.

Fig. 2: Contributors' growth in Quantum computing projects.

contributing to a total of over 3.2 billion lines, underscoring the substantial codebase supporting quantum software.

2) Issues and User Engagement: The issue data provides insights for RQ2.3 which focuses on the issue patterns of the repositories. The dataset includes 157,471 reported issues with 278,089 comments from 9,132 unique users, showing active community engagement in identifying and addressing challenges. Of these issues, 15,405 remain open, highlighting ongoing development needs. The issues cover both general software engineering concerns and quantum-specific topics, such as quantum experiments and matrix operations, helping us understand the unique demands within quantum software.

3) Commit and Maintenance Analysis: The commit data provides comprehensive details essential for answering RQ2.1, RQ2.2, RQ2.4, RQ2.5 and RQ2.6 on maintenance practices. With 1,232,828 commits, each entry includes information on the author, lines added or deleted, files changed, and commit timestamp, offering detailed insights into project evolution. Using a fine-tuned DistilBERT model, we categorized commits into perfective, adaptive, and corrective types, allowing us to analyze how different maintenance actions influence software development. This classification reveals maintenance priorities, such as functionality enhancement, adaptability to new requirements, and bug resolution practices.

IV. RESULTS

A. Current Status and Evolution of Quantum Computing Community (RQ1)

1) How has Quantum Computing popularity changed?: To evaluate the growth in popularity of quantum computing, we analyzed trends in both the number of repositories and the number of contributors over time.

To find the number of quantum projects started, we analyzed the start dates of projects; this allows us to track the emergence of new repositories as well as the cumulative total of quantum computing projects. Figure 2 presents both the count of newly created repositories each quarter and the overall growth in the number of projects over time. The data reveals a gradual increase in new projects from 2012 through 2016, reflecting early and moderate interest in quantum computing. However, around 2017, there is a noticeable acceleration in the number of new projects, suggesting a growing interest in quantum
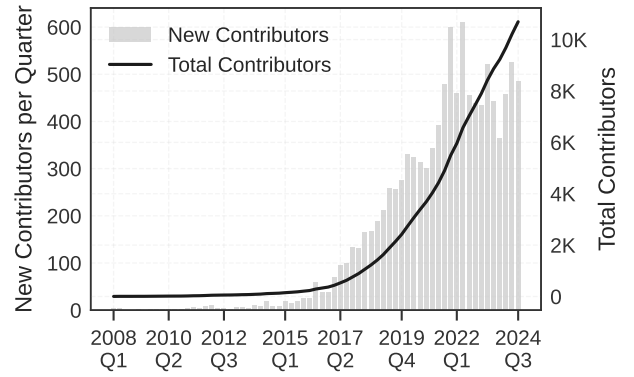
computing during this period. This increase becomes more prominent between 2018 and 2021, with especially sharp rises around 2019 and 2020. This surge is likely correlated with the release and adoption of accessible open-source quantum platforms, such as Qiskit (released in March 2017) and Cirq (released in July 2018), which lowered the barriers for new developers to enter the field.

By 2023, the cumulative number of quantum computing repositories nears 20,000, underscoring the field's expanding popularity and developer engagement. The steady increase in both new projects and total repository count reflects a rapidly growing ecosystem, indicating not only rising interest but also a commitment to developing resources and tools for quantum computing. This growth provides a strong indicator of quantum computing's increasing prominence in both academics and industry.

To further understand, how many developers are contributing to quantum projects, we analyze all the contributors to existing quantum projects and community growth over time.

Complementing the repositories trend, the contributor growth (in Figure 2) reveals a similar pattern, with new contributors joining steadily over time and a marked increase starting around 2018. The total number of contributors continues to grow, reaching 10,697 by 2024. This surge in active developers indicates a robust, expanding community contributing to quantum computing projects on GitHub.

> **Finding 1.1**. *Quantum computing's popularity has shown a significant upward trend, as reflected in the steady increase in new repositories, total projects, and active contributors over time. The sharp rise beginning around 2017 underscores growing community engagement and sustained interest in quantum computing.*

2) How have programming languages and framework usage trends evolved?: To explore how programming language and framework usage trends have evolved in quantum computing, we examined project repositories over time, specifically focusing on the growth in the use of various languages and frameworks.

Framework Usage. Figure 3 shows the evolution of popular quantum computing frameworks, with a strong dominance by
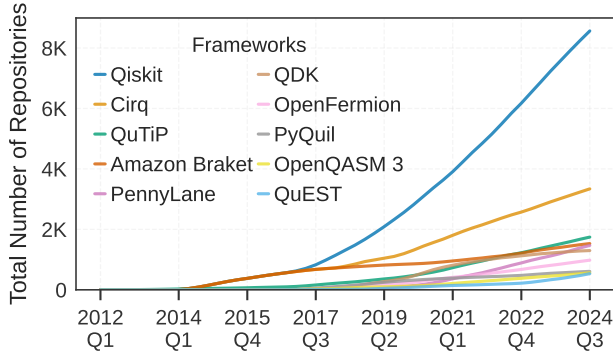
Fig. 3: Quantum computing framework usage over time, Qiskit remains the most popular framework.

Fig. 4: Usage of programming languages in Quantum repositories, Python remains the most popular language.

Qiskit. Starting around 2017, Qiskit [2] rapidly gained traction, reflecting its widespread adoption and IBM's continued support and community engagement. Cirq [4] and QuTiP [29] also display steady growth, although at a slower pace compared to Qiskit. Notably, other frameworks, such as Amazon Braket SDK [7] and PennyLane [6], began appearing around 2020, indicating newer entries into the quantum ecosystem, likely supported by commercial cloud providers like Amazon.

This trend reflects a shift toward using comprehensive frameworks that simplify quantum computing development, offering specialized libraries and tools tailored to different quantum hardware and software needs. The rise in the variety of frameworks also underscores the diversification of quantum software development, with developers opting for tools that meet specific project or hardware requirements.

Programming Languages Usage. Figure 4 presents a clear trend toward Python as the most widely used language for quantum computing, consistent with the popularity of quantum frameworks such as Qiskit, Cirq, and PennyLane that support Python APIs. The widespread use of Python is evident, with a significant lead over other languages, illustrating its role as the primary language for quantum software development. C++ has a strong but comparatively smaller presence, probably because of its use in performance-critical components or low-level operations.

Emerging languages, such as Q# from Microsoft [36] and OpenQASM [12], show steady growth around 2018, suggesting an increasing focus on languages explicitly designed for quantum applications. Julia and Rust have also begun to appear in recent years, reflecting a gradual diversification in language preferences as developers explore options that may offer performance benefits or specific features relevant to quantum computing.

Finding 1.2. *Programming language and framework usage in quantum computing has evolved notably, with Python and Qiskit dominating the landscape. However, recent growth in frameworks such as QuTiP and Amazon Braket SDK, along with emerging quantum-specific languages such as Q# and OpenQASM, reflects a diversifying ecosystem tailored to specialized quantum applications.*
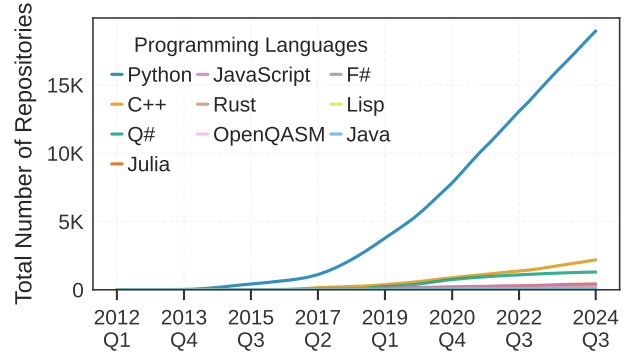
3) What is the developers' distribution for different frameworks and how do developers collaborate?: The distribution of developers across quantum computing frameworks reveals varying levels of reliance and engagement. As shown in Table III, Qiskit is the most widely adopted, with 8,561 repositories and 2,231 contributors relying on it for their projects, making it a central resource in the quantum computing ecosystem. Cirq and QuTiP follow, with 3,338 and 1,743 repositories, respectively, each supported by active communities of over 1,200 and 670 contributors. Other frameworks, such as Penny-Lane and Amazon Braket, are also widely utilized, with 1,464 and 1,525 repositories, backed by substantial contributor bases. Frameworks like OpenFermion and PyQuil, while linked to fewer repositories, maintain dedicated followings, suggesting that developers often choose frameworks based on specific applications or functionalities they offer.

TABLE III: Distribution of repositories using different frameworks and associated contributors for repositories.

Framework	# of repositories	# of contributors
Amazon Braket	1,525	292
Cirq	3,338	1,205
OpenFermion	979	318
OpenQASM 3	561	227
PennyLane	1,464	558
PyQuil	610	239
QDK	1,299	286
Qiskit	8,561	2,231
QuEST	530	243
QuTiP	1,743	673

Finding 1.3.1. *Qiskit is the most widely used quantum computing framework, followed by Cirq and QuTiP, with each framework attracting distinct developer communities based on specific functionalities and applications.*

To explore the collaboration among authors in popular quantum computing frameworks, we employ the Louvain algorithm [40] to build a network of developers that have contributed to the most prominent frameworks found in RQ 1.2. We identify highly collaborative authors as those with significant participation across multiple frameworks. Figure 5 illustrates the developers' network using six clusters, where each
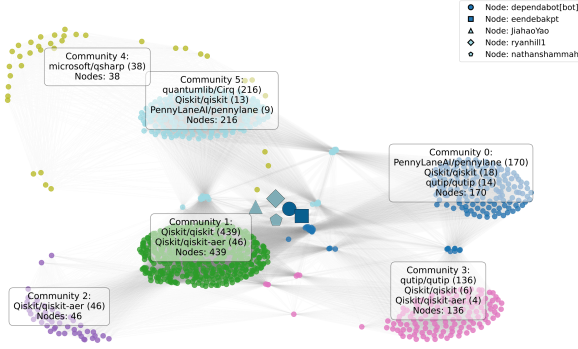
5

Fig. 5: Network of developers working with popular frameworks, highlighting the top 3 frameworks used in each cluster.

represents one of the prominent frameworks and highlights the top three frameworks within each cluster. For brevity, we have only displayed the top-5 most collaborative authors, identified by the highest cross-framework participation, represented as the larger nodes in Figure 5. The top-5 most collaborative authors, along with their respective cross-framework participation, are as follows: dependabot[bot] (5), eendebakpt (4), JiahaoYao (4), ryanhill1 (3), and nathanshammah (3). The most collaborative author in the network is a GitHub bot, dependabot[bot], due to its role in automating the management of project dependencies and ensuring they are consistently updated.

Figure 5 also shows the cross-framework collaboration in quantum computing. Specifically, the percentage of contributors who engage in cross-framework collaboration is 0% for `Community 2`, 10% for `Community 1`, 19% for `Community 0`, 0% for `Community 4`, 7% for `Community 3`, and 10% for `Community 5`.

TABLE IV: Breakdown of developer backgrounds contributing ≥3 prominent Quantum Frameworks.

Researchers	Quantum Developers	Physicists	Grad. Students	Bot
6.25%	31.25%	18.75%	37.5%	6.25%

To further understand developers appearing as collaborators, we manually analyzed the GitHub profiles of the developers that contribute to three or more prominent frameworks. As shown in Table IV, the majority of contributors are graduate students (37.5%), followed by quantum-focused developers (31.25%), physicists (18.75%), and researchers (6.25%), with bots accounting for 6.25%. This mix highlights a diverse community, where graduate students and specialized quantum developers are the most active collaborators, contributing essential expertise from both academia and applied quantum software development.

> **Finding 1.3.2.** *Collaboration patterns show limited cross-framework interaction, concentrated around a few key contributors. Graduate students and specialized quantum developers are the primary drivers of cross-repository collaboration.*
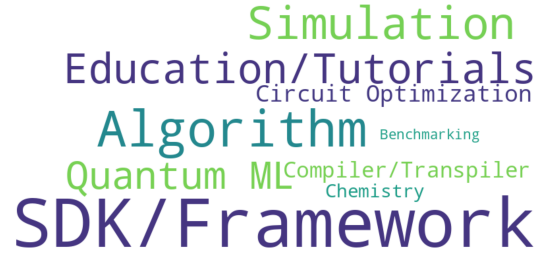

Fig. 6: Word cloud of topics from Quantum repositories.

4) What trends and insights can be derived from project topics?: To investigate project topics, we labeled repositories based on their primary focus (as shown in Figure 6) and analyzed trends in these topics over time. The timeline of repository growth across categories (in Figure 7) reveals distinct patterns: SDK/API and Quantum Simulation projects exhibit the most substantial growth, indicating that toolkits and simulation capabilities are key areas of interest in the quantum computing field. Quantum Machine Learning and Algorithm Implementation also show steady growth, reflecting rising demand for quantum algorithms and their integration with machine learning.
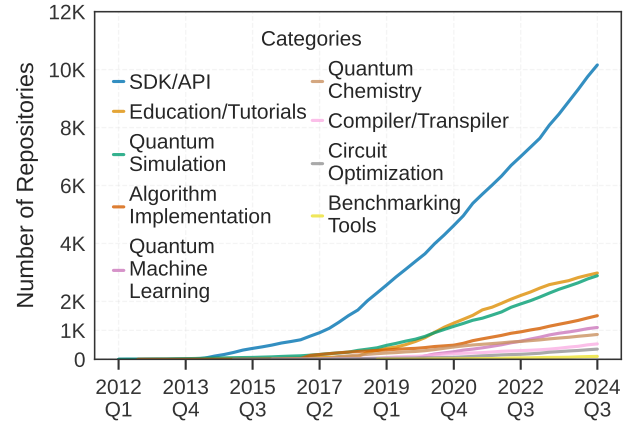

Fig. 7: Growth of different topics in Quantum repositories.

The word cloud of repository categories further emphasizes the prominence of these areas, with terms related to Simulation, Machine Learning, and Compiler/Transpiler frequently appearing. This trend suggests a strong focus on tools that support both foundational quantum research and applied quantum computing projects, particularly in algorithm development and machine learning integration.

> **Finding 1.4.** *The quantum computing community is not only focused on foundational frameworks and simulators but is also increasingly exploring applications in machine learning, chemistry, and advanced algorithm design.*
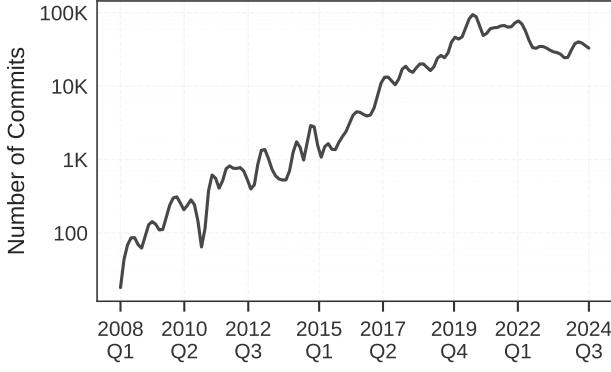
6

Fig. 8: Number of new commits in Quantum repositories show a consistent growth.

B. Maintenance Analysis (RQ2)

1) What commit patterns reveal over time?: We analyzed the overall commit activity in quantum computing repositories, as shown in Figure 8. The graph illustrates commit counts overtime on a log scale, capturing trends from as early as 2008 through to 2024.

The data reveals a gradual increase in commits during the initial years, reflecting modest activity and slow growth in the quantum computing development landscape. However, starting around 2017, there has been a rise in commit frequency, likely driven by increased community engagement and the emergence of more accessible quantum development frameworks and tools. Again, similar to the repository growth, commit growth also significantly increases between 2020 and 2023, indicating increased development efforts, community contributions, and perhaps an expansion in the scope of quantum projects.

> **Finding 2.1**. *The sharp rise in recent commit activity indicates that quantum computing is evolving from niche research into a collaborative, mature field, marked by consistent improvements and a growing number of contributors and projects.*

2) How does the size and frequency of commits evolve?: We examine the trends in code volume and file changes. Figure 9 and Figure 10 show a steady increase in both the number of lines added/deleted and files modified, with a significant spike around 2020-2023.

This surge in file changes corresponds to a noticeable jump in activity due to large-scale code reformatting, which temporarily elevated the number of files modified. This pattern aligns with the previously observed trend of accelerated commit activity post-2017, reflecting a period of heightened development and restructuring in quantum computing projects. These spikes indicate both growing project complexity and increased focus on code organization and standardization as the field matures.
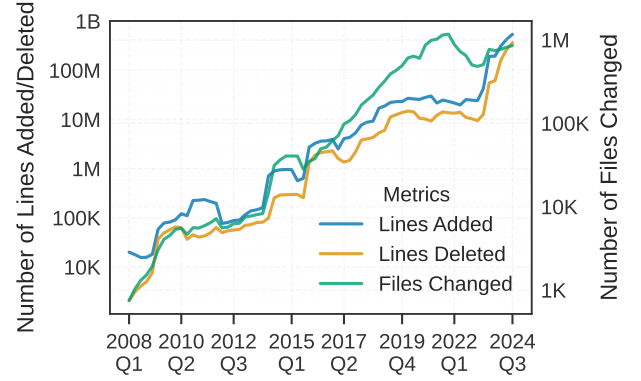

Fig. 9: Code changes trend in Quantum software, in terms of the number of lines added, deleted, and files changed.
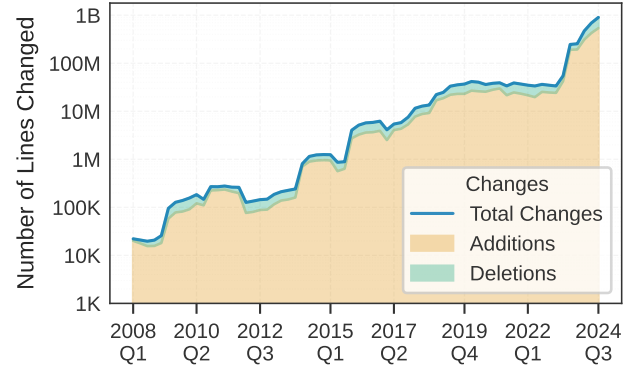

Fig. 10: Volume of code changes in Quantum repositories, the number of lines added, deleted, and files changed.

> **Finding 2.2**. *Commit size and frequency have risen significantly, with a spike from large-scale code reformatting around 2020-2023, indicating increasing project complexity and a move toward standardized development in quantum computing.*

3) What are the prominent topics in the reported issues?: We identified key topics in quantum computing repository issues using BERTopic [24]. However, because unsupervised BERTopic can generate noisy or inconsistent topics that are difficult to interpret, we adopted a semi-supervised approach to improve its reliability. Specifically, we initialized BERTopic with seed topics – keywords selected based on their high TF-IDF scores – to guide the model toward coherent and meaningful clusters rather than arbitrary groupings.

To assess the effectiveness of our approach, we conducted a qualitative validation focusing on two key aspects: (1) whether BERTopic formed coherent and meaningful topics, and (2) whether it generalized beyond the explicit seed terms. We manually reviewed a diverse subset of issues from each cluster to evaluate whether BERTopic not only grouped semantically similar issues in the same cluster – even when they did not explicitly contain the seed topic keywords – but also identified a broader range of topics beyond those initially defined by the seed terms. Our qualitative analysis confirmed that the seeded BERTopic approach facilitated the discovery of

TABLE V: Description of the key features in the issues for Quantum Computing repositories.

Topic	Components	Frequency(%)
Dependency Management and Updates	deprecate, remove, bump, version, dependency, update, pypi, pip	14.76
Quantum Computing Terminology	qubit, quantum, pulse, hamiltonian, matrix, gate, pauli, phase, rotation	11.33
Code Maintenance and Refactoring	rename, update, refactor, fix, remove, typo, change	9.85
Quantum Computing Software and Frameworks	qasm, qaoa, openqasm, qobj, qnode, qpu, qiskit, qutip, ibmq	7.94
Fixing Bugs	error, fix, fail, bug, warning, log, issue	6.44
Documentation	doc, documentation, readme, docstring, tutorial, docs, sphinx	5.75
Quantum Computing Hardware Interfaces	driver, backend, device, channel, job, add, instrument, executor, gpu	5.72
Continuous Integration and Testing	test, ci, travis, coverage, build, workflow, run, testing, integration	5.68
Quantum Computing Experiments	measurement, calibration, benchmark, performance, experiment, noise	5.59
Version Control	merge, update, release, link, branch, master, version, main, pr	4.66
Matrix Operations	state, gradient, solver, matrix, vqe, observable, adjoint, sparse	4.08
Python Syntax	import, module, json, dataset, serialization, kwarg, decorator	2.66
Data Visualization and Machine Learning Libraries	matplotlib, tensor, graph, plot, tensorflow, notebook, lattice, torch, visualization	2.16
Other	circuit, pass, drawer, transpiler, register, operator, wire, datum, seed, simulator, rb, shot, sampler, optimizer, numpy, python, rust	13.37

meaningful clusters beyond the original TF-IDF terms while improving topic cohesion and reducing noise. This approach enabled us to uncover various issue categories relevant to quantum software development, summarized in Table V.

Our analysis revealed that a significant portion of issues (around 47.14%) are related to classic software engineering challenges, including dependency management, code maintenance, refactoring, documentation, continuous integration and testing, and version control. These issues reflect common needs in software projects, such as managing package versions, fixing bugs, improving code structure, and ensuring seamless integration and deployment processes. The prevalence of these topics underscores that, like other software fields, quantum computing projects require robust tools and practices for maintaining code quality and collaboration.

About 34.66% of the reported issues are specific to quantum computing, focusing on specialized topics such as quantum computing basics (e.g., qubits, rotation), hardware interfaces, quantum computing experiments, matrix operations, and quantum-specific software frameworks. These issues highlight the unique challenges in quantum software, including the need for tools that enable the simulation of quantum experiments and support the complex mathematical operations essential for quantum algorithms.

Finding 2.3. *Approximately 47% of issues in quantum computing repositories relate to classic software engineering challenges, while 34% are quantum-specific like quantum experiments and matrix operations. This indicates a need for specialized tools to address the unique requirements of quantum software development, as well as a demand for overall robust software infrastructure to support quantum computing.*

4) Types of Commits: Understanding the proportions of different types of commits is important to understand advances in quantum computing. To identify the different commit types in our repositories, we used the DistilBERT model fine-tuned by [15] on GitHub commit messages [44]. This model classified the commit messages into categories based on their purpose: perfective (improvements), adaptive (adjustments to changes), and corrective (bug fixes).

As shown in Table VI, the majority of commits are perfective (51.76%), this finding reflects an emphasis on improving and refining code to enhance functionality and performance. Adaptive commits, at 22.58%, indicate frequent adjustments to accommodate new requirements or technologies, while corrective commits, at 18.54%, suggest a relatively lower occurrence of bug fixes. The presence of combined categories, such as adaptive-perfective and corrective-perfective, suggests overlapping maintenance efforts, where developers enhance functionality while adapting or fixing the code.

TABLE VI: Percentage of Frequency for Each Classification

Classification	Percentage (%)
Perfective	51.76
Adaptive	22.58
Corrective	18.54
Adaptive Perfective	4.97
Corrective Perfective	1.23
Corrective Adaptive	0.46

Finding 2.4. *The majority of commits in quantum computing repositories focus on code improvements, followed by adjustments, while corrective commits are less frequent. This suggests an emphasis on enhancing functionality over bug fixing, with potential gaps in tracking and addressing bugs.*

5) Classification of Repository Activity-Levels using Commit Data: To account for potential duplication in forked repositories, we analyzed only original commits not present in parent repositories, resulting in 8,548 repositories being included in our analysis. We examine the factors that differentiate high-activity repositories from those with lower activity through K-means clustering [34], [35] of repository commit attributes, including commit frequency per month, average gap between commits, number of commits, number of contributors, and project duration. Specifically, we apply K-means clustering to assign labels to the repositories and then perform 2D Principal Component Analysis (PCA) [27], [30] to visualize the clusters, as shown in Figure 11. Figure 11 illustrates that the dashed red line, passing through the rightmost lower-activity point (red circle), effectively separates high-activity repositories (blue) from lower-activity ones (yellow). Additionally, we observe that high-activity repositories tend
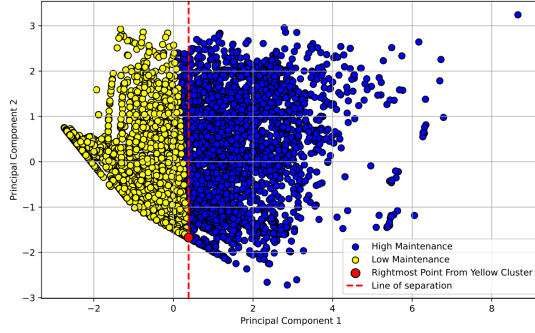
Fig. 11: Clustering Quantum repositories into 2 classes: high and low activity repositories.

to exhibit greater variance, as evidenced by more spread-out scatter points and outliers.

TABLE VII: Mean centroids of repository activity categories.

Category	Frequency	Avg. Days b/w Commits	Commits	Authors	Months
High	20.05	4.65	187.838	4.523	3.9
Low	2.86	387.535	9.351	1.296	1.47

Table VII displays the attributes of the centroids for the two clusters, highlighting that high-activity repositories are more actively managed, as evidenced by their higher commit frequency, shorter commit gaps, larger number of commits, more contributors, and longer duration of maintenance.

> **Finding 2.5**. *Highly active quantum repositories have more contributors and are sustained over longer periods. However, there are fewer high-activity repositories (3,562) compared to those with low-activity (4,986), highlighting the disparity in the level of development and support in the quantum computing ecosystem.*

6) How do different repository characteristics differ between maintenance activity levels?: To explore differences between high- and low-activity repositories, we examined various attributes and commit categories, using the Mann-Whitney U Test for statistical significance [37].

TABLE VIII: Mann-Whitney U Test results for continuous variables.

Variable	High Activity Mean	Low Activity Mean	p-value
Update Duration	281.9597	147.6592	<0.001
Stars	6.5573	0.61462	<0.001
Forks	2.1212	0.20753	<0.001
Size	40581.0595	32940.8206	<0.001
Contributors	53.2758	62.6904	0.044
Lines of code	468798.7614	115685.0642	<0.001

The results in Table VIII reveal notable distinctions between these repository types. High-activity repositories show a significantly longer update duration, averaging 281.96 days compared to 147.66 days in low-activity repositories (p < 0.001), which suggests more consistent, ongoing updates over time. They also tend to attract more stars and forks, with high-activity repositories averaging 6.56 stars and 2.12 forks versus 0.61 stars and 0.21 forks for low-activity ones, indicating
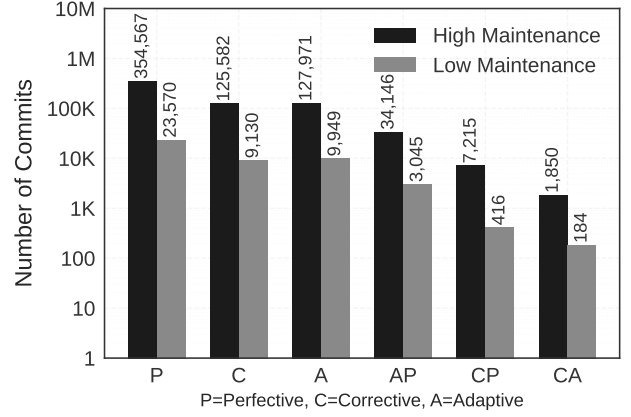

Fig. 12: Distribution of different commit classes for high and low activity repositories.

greater community interest and engagement. In terms of size, high-activity repositories have substantially more lines of code (468,798 vs 115,685) and overall project size, reflecting more complex codebases that likely demand intensive upkeep. Interestingly, low-activity repositories have a slightly higher mean contributor count (62.69 vs 53.28, p = 0.044), suggesting low-activity projects may receive occasional contributions without sustained commitment.

The distribution of commit types across activity levels, shown in Figure 12, further supports these findings. High-activity repositories show a higher frequency of perfective and adaptive commits, focusing on ongoing improvements and adaptations to changes, while low-activity repositories exhibit fewer updates across all commit categories. These insights highlight that high-activity repositories are generally larger, attract more engagement, and emphasize continuous enhancement and adaptability, while low-activity repositories may experience sporadic contributions without regular development efforts.

> **Finding 2.6**. *High-activity repositories have frequent updates, larger codebases, and greater community engagement, while low-activity repositories show irregular updates and smaller codebases.*

V. IMPLICATIONS

Our findings highlight key challenges in quantum software engineering and outline actionable steps to improve development practices, tooling, and education.

- (a) *Bug Resolution and Software Reliability.* The prevalence of perfective commits (51.76%) over corrective commits (18.54%) suggests that defect resolution is underemphasized. Prior work by Paltenghi and Pradel [38] found that quantum software exhibits recurring bug patterns, yet our findings indicate that corrective actions remain limited. Enhancing automated testing, issue-tracking systems, and debugging frameworks is critical for improving software reliability.
- (b) *Quantum-Specific Tooling Needs.* With 34% of reported issues being unique to quantum computing, there is a

clear need for quantum-aware debugging tools, enhanced simulators, and automated static analysis techniques. Zhao et al. [49] identified debugging as a major challenge in Qiskit, and our findings reinforce the necessity for specialized tooling to address quantum-specific issues, such as circuit optimization and error mitigation.

(c) *Sustaining Software Development.* The divide between *high- and low-activity repositories* indicates challenges in long-term software maintenance. Studies on repository evolution [19], [47] suggest that engagement in quantum projects is often short-lived, which aligns with our observation that many repositories exhibit sporadic contributions. Encouraging community-driven contributions, better documentation practices, and sustainable governance models can improve repository longevity.

(d) *Standardization and Interoperability.* As quantum computing frameworks evolve, ensuring consistent APIs, cross-framework compatibility, and adherence to software engineering best practices will enhance usability and reduce fragmentation. Prior work on quantum programming languages [48] has shown that the ecosystem remains highly fragmented, and our findings confirm that while Python and Qiskit dominate, newer frameworks, such as Cirq and PennyLane are gaining traction, emphasizing the need for standardization.

To drive advancements in quantum software engineering, we open-source our dataset, providing the broader software engineering community with empirical insights to inform tool development, maintenance strategies, and best practices in quantum software development.

VI. Threats to Validity

Internal Validity. A potential threat to internal validity lies in the criteria used for selecting quantum-related repositories. While we aimed to ensure relevance by filtering for repositories with quantum-specific keywords and technologies, the initial reliance on GitHub metadata alone may have introduced some irrelevant or incomplete projects. To mitigate this, we performed extensive data cleaning as described in the processing section, where we removed unchanged forks, retained only repositories demonstrating meaningful activity, and ensured the presence of quantum package usage. These steps helped refine the dataset to better represent genuine quantum software development projects. However, there remains a risk that some valid but low-activity repositories could have been excluded, possibly omitting some niche or experimental quantum projects.

External Validity. Our findings may have limited generalizability beyond GitHub to other platforms where quantum software may be developed, such as proprietary or institution-hosted repositories. This study focuses exclusively on open-source projects on GitHub, which could lead to insights that reflect primarily the open-source quantum development community. This limitation suggests that proprietary or enterprise-level practices may not be fully captured here, as they might involve different workflows, tools, or challenges. Thus, while the results provide valuable insights into open-source quantum software, extending the study to include other sources could yield a broader understanding of quantum software practices.

Construct Validity. The construct validity of this study could be impacted by limitations inherent to mining software repositories. Metrics such as commit counts, issue reports, and collaboration structures provide quantitative insights into development activities but may overlook qualitative factors such as developer expertise, code quality, or algorithm complexity. The absence of standardized best practices and tools in quantum programming complicates the interpretation of certain development patterns and issues. To address this, we incorporated additional filtering and categorization criteria based on quantum package usage, commit activity, and algorithm types, helping to focus the dataset on relevant aspects of quantum software engineering.

Reliability. The reliability of our findings is based on the reproducibility of the data mining and cleaning techniques applied. Future changes to GitHub's API or to repository metadata could impact the replicability of aspects of our methodology. Additionally, reliance on automated processes like natural language processing (NLP) for issue and commit classification may introduce some errors in categorization. To reduce this risk, we detailed the data extraction and cleaning process and cross-validated classifications where feasible to ensure consistency.

By recognizing these limitations, we aim to provide transparency about the study's scope and the validity of our findings within this context.

VII. Conclusions

This study provides a comprehensive analysis of the current state, evolution, and maintenance practices in the quantum computing community by examining over 21,000 GitHub repositories, containing more than 1.2 million commits from over 10,000 unique contributors. Through this large-scale mining effort, we evaluate community growth, programming trends, and maintenance practices as well as offer insights into the strengths and areas for improvement in quantum software development. Our findings show a rapidly growing community, with a 200% increase in the number of repositories and a 150% rise in contributors since 2017, underscoring the growing interest in quantum computing. However, our commit analysis reveals a focus on perfective updates, with fewer corrective commits, which suggests potential gaps in bug resolution. Additionally, a third of the issues emphasize the need for specialized tools alongside general software infrastructure.

Based on our analysis of development patterns, community structure, and maintenance practices, we recommend investment in specialized tooling and systematic maintenance frameworks for quantum computing. Enhanced support for high-maintenance repositories could drive further innovation. Additionally, extending this analysis to other platforms may provide broader insights into the quantum software landscape.

REFERENCES

[1] National quantum initiative. https://www.quantum.gov/, 2018. Accessed: 2024-09-17.

[2] Qiskit - world's most popular software stack for quantum computing. https://www.ibm.com/quantum/qiskit, 2022. Accessed: 2024-09-17.

[3] Azure sdk for .net. https://github.com/Azure/azure-sdk-for-net, 2024. Accessed: 2024-09-17.

[4] Cirq. https://quantumai.google/cirq, 2024. Accessed: 2024-09-17.

[5] Github code search api. https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28#search-code, 2024. Accessed: 2024-09-17.

[6] Pennylane – quantum programming software. https://pennylane.ai/, 2024. Accessed: 2024-09-17.

[7] Python sdk for interacting with quantum devices on amazon braket. https://github.com/amazon-braket/amazon-braket-sdk-python, 2024. Accessed: 2024-09-17.

[8] Qosf awesome quantum software. https://github.com/qosf/awesome-quantum-software, 2024. Accessed: 2024-09-17.

[9] Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. In *2013 10th working conference on mining software repositories (MSR)*, pages 207–216. IEEE, 2013.

[10] Ali Alshehri, Pawel Marcinek, Abdulrahman Alzahrani, Hani Alshahrani, and Huirong Fu. Puredroid: Permission usage and risk estimation for android applications. In *Proceedings of the 2019 3rd international conference on information system and data mining*, pages 179–184, 2019.

[11] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 286–300, New York, NY, USA, 2020. Association for Computing Machinery.

[12] Lev S Bishop. Qasm 2.0: A quantum circuit intermediate representation. In *APS March Meeting Abstracts*, volume 2017, pages P46–008, 2017.

[13] Raymond PL Buse and Westley R Weimer. Automatically documenting program changes. In *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*, pages 33–42, 2010.

[14] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, pages 528–532, 1994.

[15] J. Castano, S. Martinez-Fernandez, X. Franch, and J. Bogner. Analyzing the evolution and maintenance of ml models on hugging face. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, pages 607–618, Los Alamitos, CA, USA, apr 2024. IEEE Computer Society.

[16] Qihong Chen, Rúben Câmara, José Campos, André Souto, and Iftekhar Ahmed. The smelly eight: An empirical study on the prevalence of code smells in quantum computing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 358–370, 2023.

[17] Antonio D Córcoles, Abhinav Kandala, Ali Javadi-Abhari, Douglas T McClure, Andrew W Cross, Kristan Temme, Paul D Nation, Matthias Steffen, and Jay M Gambetta. Challenges and opportunities of near-term quantum computing systems. *Proceedings of the IEEE*, 108(8):1338–1352, 2019.

[18] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S Bishop, Steven Heidel, Colm A Ryan, Prasahnt Sivarajah, John Smolin, Jay M Gambetta, et al. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):1–50, 2022.

[19] Mohamed Raed El aoun, Heng Li, Foutse Khomh, and Moses Openja. Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 343–354, 2021.

[20] Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, 2011.

[21] Mark Fingerhuth, Tomáš Babej, and Peter Wittek. Open source software in quantum computing. *PloS one*, 13(12):e0208561, 2018.

[22] https://github.com/github-linguist/linguist.

[23] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 333–342, 2013.

[24] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[25] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[26] Thomas Häner, Damian S Steiger, Krysta Svore, and Matthias Troyer. A software methodology for compiling quantum programs. *Quantum Science and Technology*, 3(2):020501, 2018.

[27] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[28] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J Wood, Jake Lishman, Julien Gacon, Simon Martiel, Paul D Nation, Lev S Bishop, Andrew W Cross, et al. Quantum computing with qiskit. *arXiv preprint arXiv:2405.08810*, 2024.

[29] J.R. Johansson, P.D. Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012.

[30] Ian Jolliffe. Principal component analysis. *Encyclopedia of statistics in behavioral science*, 2005.

[31] Sunghun Kim, Thomas Zimmermann, E James Whitehead Jr, and Andreas Zeller. Predicting faults from cached history. In *29th International Conference on Software Engineering (ICSE'07)*, pages 489–498. IEEE, 2007.

[32] Ryan LaRose. Overview and comparison of gate level quantum software platforms. *Quantum*, 3:130, 2019.

[33] Li Li, Tegawendé F Bissyandé, Haoyu Wang, and Jacques Klein. Cid: Automating the detection of api-related compatibility issues in android apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 153–163, 2018.

[34] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[35] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*, 1967.

[36] Microsoft. Introduction to the quantum programming language q#. https://learn.microsoft.com/en-us/azure/quantum/qsharp-overview, 2019. Accessed: 2024-09-17.

[37] Nadim Nachar et al. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology*, 4(1):13–20, 2008.

[38] Matteo Paltenghi and Michael Pradel. Bugs in quantum computing platforms: an empirical study. *Proc. ACM Program. Lang.*, 6(OOPSLA1), April 2022.

[39] Peng Qian, Zhenguang Liu, Qinming He, Butian Huang, Duanzheng Tian, and Xun Wang. Smart contract vulnerability detection technique: A survey. *arXiv preprint arXiv:2209.05872*, 2022.

[40] Xinyu Que, Fabio Checconi, Fabrizio Petrini, and John A Gunnels. Scalable community detection with the louvain algorithm. In *2015 IEEE international parallel and distributed processing symposium*, pages 28–37. IEEE, 2015.

[41] Foyzur Rahman, Daryl Posnett, Israel Herraiz, and Premkumar Devanbu. Sample size vs. bias in defect prediction. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pages 147–157, 2013.

[42] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, pages 155–165, 2014.

[43] Jonathan Ruane, Andrew McAfee, and William Oliver. Quantum computing for business leaders. https://hbr.org/2022/01/quantum-computing-for-business-leaders, 2022. Accessed: 2024-09-17.

[44] Muhammad Usman Sarwar, Sarim Zafar, Mohamed Wiem Mkaouer, Gursimran Singh Walia, and Muhammad Zubair Malik. Multi-label classification of commit messages using transfer learning. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 37–42. IEEE, 2020.

[45] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[46] Grand View. Quantum computing software market size, share & trends analysis report by component, by deployment, by technology, by application, by end-use, by region, and segment forecasts, 2024 - 2030, 2024. Accessed: 2024-09-17.

[47] Jake Zappin, Trevor Stalnaker, Oscar Chaparro, and Denys Poshyvanyk. When quantum meets classical: Characterizing hybrid quantum-classical issues discussed in developer forums, 2024.

[48] Pengzhan Zhao, Xiongfei Wu, Junjie Luo, Zhuo Li, and Jianjun Zhao. An empirical study of bugs in quantum machine learning frameworks. In *2023 IEEE International Conference on Quantum Software (QSW)*, page 68–75. IEEE, July 2023.

[49] Pengzhan Zhao, Jianjun Zhao, and Lei Ma. Identifying bug patterns in quantum programs. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, pages 16–21, 2021.