Works on My QPU: Reproducibility in Quantum Computing Research

Dominik Köster*, Maja Franz*, Benjamin Zec*, Nicole Hoess*, Ralf Ramsauer*, Wolfgang Mauerer*†
*Technical University of Applied Science Regensburg, Germany,
{dominik.koester, maja.franz, benjamin.zec, nicole.hoess, ralf.ramsauer, wolfgang.mauerer}@othr.de
†Siemens AG, Foundational Technology, Munich, Germany

***Abstract*—Quantum computing research increasingly depends on complex software stacks, yet the reproducibility of published results does not receive the priority and longevity mandated by recommendations of large international scientific bodies and best practices in software-centric systems research. In this paper, we present a combined manual and automated large-scale analysis of the reproducibility landscape in quantum computing research, quantify shortcomings, and derive actionable steps forward.**

We manually evaluate a curated sample of 127 papers using a five-question framework that covers code availability, environment specification, documentation, hardware description, and executability. To place these findings in a broader context, we conduct an automated large-scale screening of nearly 5000 quantum computing papers for the same reproducibility indicators. Our manual analysis reveals that only 24.4% of the sampled papers provide code artefacts, and among those, 64.5% fail to execute successfully in a clean environment. This assessment is corroborated by a large-scale automated analysis that yields a consistent code availability rate of 26.8%. Further, it shows that approximately one-third of the papers with accessible code lack machine-readable environment specifications.

The results in this paper indicate that reproducibility is not yet consistently achieved in quantum computing research. In response, we outline a set of practical recommendations that address the observed failure modes and illustrate how reproducibility can be improved in practice.

***Index Terms*—Reproducibility, Quantum computing, Quantum software, Research artefacts, Software environments, Empirical study**

Reproducibility has become a cornerstone of scientific progress [1], [2]. Ensuring reproducible research artefacts remains challenging across disciplines [3], [4], [5]. In quantum computing (QC), this challenge is dominated by the rapid dynamics of development processes, evolving tooling, and experimental hardware infrastructures [6] that are often not designed for long-term stability and availability, including cloud-based systems [7], [8]. As QC research relies on complex software environments, the reproducibility of experimental results depends not only on algorithmic descriptions [9], but also on the availability and precise configuration of software dependencies.

A natural assumption is that reproducibility issues primarily arise from environment drift over time, such as outdated dependencies or deprecated APIs. However, during our reproduction attempts of publicly available QC artefacts, we observed that many projects failed to execute even in freshly provisioned environments shortly after publication. These failures occurred independently of long-term dependency changes, and suggest that some artefacts may never have been fully reproducible outside the original development setup. This observation points to a broader issue: implicit assumptions about the development environment, such as unreferenced locally installed dependencies, or undocumented configuration steps. These remain unrecorded and therefore hinder reproducibility.

Publishing reproducibility artefacts (*e.g.*, code) alongside research is generally highly commendable and a necessary precondition for reproducibility in QC [10]. In this work, we conduct an empirical study of a filtered sample drawn from a cross-section of two scientific databases. For papers within this sample that provide publicly available code artefacts, we attempt reproduction in a clean environment, following only the provided documentation. We record reproduction outcomes and perform a qualitative analysis of observed failures by identifying recurring patterns that reveal implicit environmental assumptions, including incomplete dependency specifications, undocumented system-level requirements, reliance on transient container images, and hidden local state. While we observe these patterns in the underexamined field of QC, several are not specific to it and can arise in software projects more broadly.

Our findings indicate that reproducibility challenges in QC are not solely caused by temporal environmental drift, but frequently stem from incomplete or implicit environment specifications. Existing approaches such as virtual environments, dependency files, or containerisation [2], [10] mitigate parts of the problem but often remain insufficient to capture the full execution context. Based on our observations, we discuss declarative environment specifications as a promising mitigation strategy. In particular, we highlight how languages and tools such as Nix and devenv.sh enable explicit and reproducible environment definitions [11] that reduce implicit assumptions, and allow for long-term reproducibility even in the face of evolving dependencies and hardware.

In this paper, we claim the following contributions:

(1) A large-scale semi-automatic analysis of the availability and quality characteristics of nearly 5000 QC papers from 2021 to 2026, which provides a quantitative estimate of the proportion of recent QC works that include reproduction packages.
(2) The manual analysis of a subset of these QC papers with an attempt to reproduce the claimed results, in which we identify common failure modes of reproduction packages.

(3) An actionable list of recommendations to improve the resilience against temporal environmental changes and implicit assumptions of development environments.
(4) A reproduction package that operationalises these recommendations into a reusable and extendable template for future studies.

I. Background and Related Work

The challenge of providing reproducible research artefacts has previously been addressed in classical software engineering. For instance, early studies propose a minimal process with related elements required for empirical studies [1], while later approaches present actionable guidelines to build self-contained Docker images. Beyond code repositories and virtual environments, these images include an automated end-to-end study pipeline with all relevant dependencies [2] and can be provided in long-term repositories such as Zenodo. Despite these advances, research artefacts are still often not published at all, only in parts, or as a diverse [3], [5], possibly unusable collection of scripts [4].

In QC, further limitations in reproducibility arise due to characteristics of software stacks. User-written quantum programs are compiled into an abstract instruction sequence [12], such as a quantum circuit [13], [14], [15], defining a logical schedule [16]. As quantum computers vary significantly in their physical implementation [12], [14], this instruction sequence is then transpiled into a physical schedule for a specific target QPU [17] and brought to execution via control pulses [18]. Although uncertainty is also introduced at the lower layers — for instance during hardware-dependent optimisations [17] or varying noise across and within noisy intermediate-scale quantum (NISQ) devices [7], [19], [20], [21] — this study focuses on reproducibility from a software perspective.

The predominant programming language of popular quantum software frameworks such as Qiskit [22], PennyLane [23], Cirq [24] and pyQuil [25] is Python. While offering convenience [26] for user-written programs, the Python ecosystem is short-lived with rapid deprecation cycles and limited backward compatibility [27]. Typical issues such as version conflicts, long dependency chains [28], and dependency drift, referring to unexpected changes in system behaviour due to updates, therefore also apply to quantum software [29]. For instance, studies report that migrating to a new QC framework version caused incompatibilities, faults [30] and even substantially different results, for instance due to hidden changes in circuit hyperparameter defaults [31]. The authors report significant efforts to identify, debug and resolve root causes [30], [31], which may also limit the ability to integrate new research projects into the rapidly evolving main frameworks. In addition, such issues call into question the validity and generalisability of study results, especially when considered in conjunction with potentially missing research artefacts and restricted hardware access [10].

Reproducibility should therefore receive more attention in QC research. First approaches propose knowledge graphs to document as many workflow and hardware details as possible [9] and present a Docker meta-container template for quantum software experiments [10]. While such self-contained images ensure study reproducibility over time, they may not be sufficient when the objective is to *modify* or build an environment from scratch. The Nix package manager allows for building environments from source [32] and has previously been proposed for classical high-performance computing (HPC) settings to automate workflows and ensure reproducibility across environments and systems [33]. While recent studies demonstrate its effectiveness on classical systems [11], [32], its adoption in the context of QC has not yet been explored.

TABLE I: Reproducibility Assessment Framework for QC Papers.

ID	Criterion	Questions to Answer
RQ1	Code Availability	Is source code published? Where (GitHub, Zenodo, supplementary)? Complete reproduction package or only snippets? Runnable or illustrative only?
RQ2	Environment	Is the environment specified? Framework mentioned? Are `requirements.txt`, `Dockerfile`, or `environment.yml` available?
RQ3	Build & Run Documentation	Are instructions provided? Are logs and outputs self-explanatory?
RQ4	Hardware Specification	Are hardware specifications available? Calibration snapshot?
RQ5	Executability	Does the package run without errors? Are specified dependencies installable and run without conflicts?

II. Quantitative Reproduction Study

The reproducibility of a scientific paper in QC typically involves computational artefacts that cannot be perfectly captured by a textual description alone. For this reason, we expect a reproduction package to be provided alongside these artefacts. In this section, we describe the setup and results of a quantitative analysis of the reproducibility landscape in current QC research. Notably, our pipeline for the reproducibility assessment is itself reproducible, with the code provided on GitHub[1].

A. Reproducibility Criteria

A reproduction package typically includes documentation on the structure and use of the package, as well as a clear and complete definition of the environment needed to generate these artefacts.

Reproducing quantum-specific artefacts may additionally require execution on actual quantum hardware, which cannot be replicated by a reproduction package per se due to associated costs and limited availability. In this case, at least the execution logs as well as hardware specifications, such as calibration snapshots, should be provided. Given these prerequisites, we ask whether the package is executable and the artefact can be

[1] https://github.com/lfd/qce26_repro_analysis

recreated. These reproducibility criteria lead to the research questions summarised in Tab. I, which we aim to answer for a selection of QC research papers in this section.

B. Method

Fig. 1 provides an overview of the paper sampling, filtering, and manual validation pipeline, together with the corresponding quantitative results at each stage. Each of these stages is summarised in the following.

1) Paper Sampling: We constructed a dataset of QC research papers by querying the databases of two scientific literature tools, *arXiv* [34] and *Semantic Scholar* [35]. To identify relevant publications, we formulated a structured query targeting publications in the QC software and algorithm domain: Specifically, we required the presence of the following terms in the title or abstract: (1) *quantum computing*, and (2) *algorithm* or *software*. To avoid secondary literature, papers, which include the terms *survey* and *review* in title or abstract were explicitly excluded. Due to our focus on the QC research community, the search was further restricted to the arXiv category *quant-ph* and to publications between January 1, 2021 and March 27, 2026. We aggregated results from both sources based on identifiers such as DOIs, as well as title, and authorlist, yielding a unified de-duplicated paper corpus of 4966 papers.

2) Human-validated Analysis Pipeline: To make the size of the paper corpus manageable for manual analysis, we focus on NISQ-era research only. Particularly, we adapt the query on the two databases to contain the keyword *NISQ* in title or abstract, which results in 648 subject papers.

a) Paper Filtering: As we are only interested in papers that contain experiments or numerical evaluations, we further filtered the paper corpus using a full-text search. Specifically, all papers not containing the keywords that indicate code or one of the phrases *experiments*, *experimental result*, or *numerical* were discarded. To increase the reliability of the sampled papers, we further restricted the dataset to (1) papers with a DOI (as a proxy for peer review), and (2) papers that were found in both databases. This step resulted in a set of 127 potentially experimental papers, which forms the starting point of the human-validated analysis pipeline.

b) Automated Screening: To identify candidate papers with reproducibility artefacts, we performed another full-text scan of the available PDFs. The scan searched for reproducibility indicators such as references to source code platform (*i.e.*, (1) *git(hub|lab|ee)*, (2) *bitbucket*, or (3) *zenodo*), or explicit statements of code availability (*i.e.*, (1) *source[-]code*, (2) *repository*, (3) *code availab(le|ility)*, (4) *data availab(le|ility)*, (5) *download code*, (6) *reproduc(ible|tion)*, or (7) *code to reproduce*), reflecting common best practices. This automated screening resulted in 66 out of 127 papers that exhibited at least one reproducibility indicator.

c) Manual Validation: As the automated screening favours a larger number of false positives, we performed an additional manual validation on each candidate paper from the previous step. For the subset of confirmed papers, which do have some reproduction package linked, we further analysed the characteristics of the provided artefacts. In particular, we assessed the artefacts that satisfy RQ1 (code availability) according to the questions RQ2 to RQ4 in Tab. I. For those papers that satisfied both, RQ2 (environment specification) and RQ3 (documentation), we further tested for RQ5 (executability).

TABLE II: Reproducibility indicators across the quantum computing research landscape. **Manual analysis**: 127 considered NISQ-era papers. Automated large-scale analysis: 4966 papers in the quantum algorithm/software domain. Bold values correspond to the **manual analysis**.

Indicator	Yes	Partially	No	N/A
Code Availability	**24.4%**	-	**67.7%**	**7.9%**
	26.8%	-	73.2%	-
Environment Spec.	**13.4%**	**2.4%**	**76.3%**	**7.9%**
	14.6%	3.0%	82.4%	-
Documentation	**15.0%**	**6.3%**	**70.8%**	**7.9%**
	20.9%	1.9%	77.2%	-
Hardware Spec.	**2.4%**	**1.6%**	**69.2%**	**26.8%**
	6.3%	11.6%	73.2%	8.9%
Executability	**8.7%**	**0.8%**	**82.6%**	**7.9%**
	-	-	73.2%	26.8%

3) Automated Large-Scale Analysis Pipeline: The human-validated study provides a detailed, qualitative analysis of a sample of the QC research landscape, which is necessarily constrained by our sampling and filtering criteria. To contextualise these findings and assess the broader state of reproducibility across the QC research landscape, we further conduct a larger-scale, automated analysis on the full paper corpus of 4966 papers. For each PDF, we applied a full-text scan for the same reproducibility indicators as in the manual screening. Additionally, we apply the following normalisation steps to reduce false positives and analyse corresponding repositories, which (1) detect repository URLs with repair heuristics for line breaks and trailing punctuation introduced by PDF text extraction, (2) verify the accessibility of the repositories via HTTP requests, and (3) check for environment specific files (*e.g.*, requirements.txt, Dockerfile) and documentation indicators (*e.g.*, README.md). Based on the extracted metadata, each paper was automatically assessed against the first four questions of our framework (see Tab. I), while RQ5 was not evaluated due to the infeasibility of large-scale reproduction attempts.

C. Results

The results for the human-validated manual and large-scale analysis are summarised in Tab. II and described below. Overall, both analyses paint a consistent picture of the current state of reproducibility practices in QC research. The alignment in code availability rates between the curated manual sample (24.4%) and the broader automated corpus (26.8%) is not coincidental: the manual sampling filters for experimental

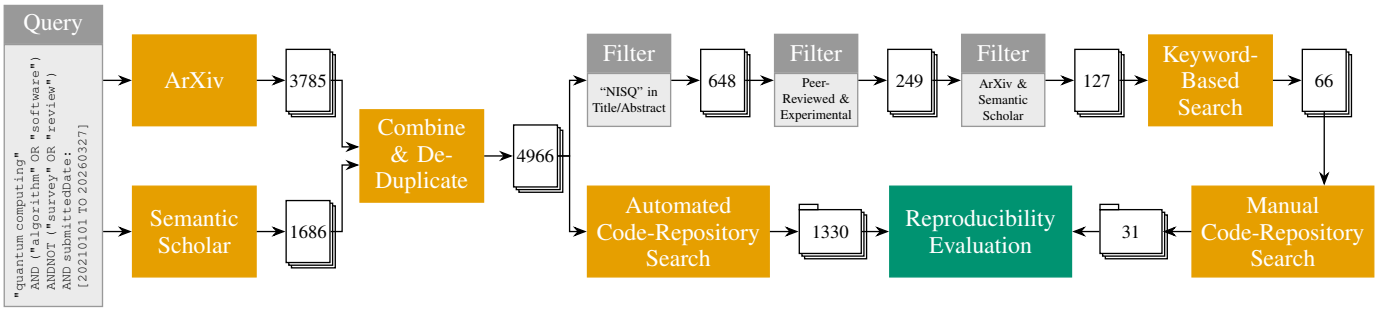

Fig. 1: Papers are retrieved via a structured query from arXiv and Semantic Scholar, merged and deduplicated. Subsequently resulting papers are either filtered through multiple stages, identified via keyword-based screening and finally subjected to manual validation, or directly submitted to automated screening for code-repositories. The results for the final reproducibility evaluation are summarised in Tab. II.

papers with a DOI and cross-database presence, criteria that are orthogonal to whether authors share their code. Consistent rates across both samples therefore suggest that neither filter systematically selects for higher or lower code-sharing behaviour.

1) RQ1 – Code Availability: Across both datasets, code availability remains limited. In the manual analysis, 31 out of 127 papers (*i.e.*, **24.4%**) provide source code, with an additional 7.9% offering code only upon request. This aligns closely with the large-scale analysis, where 26.8% of papers include accessible repository links and a further 10.8% mention code availability but do not provide an accessible link, contain explicit "available upon request" disclaimers instead of public code artefacts, or contain repository links that are no longer reachable. Taken together, these findings indicate that only about one quarter of papers provide directly usable code artefacts, with a non-negligible fraction relying on restricted or unavailable access mechanisms.

2) RQ2 – Environment Specification: A similar pattern can be observed for environment specification. In the manual analysis, 17 out of the 31 papers (*i.e.*, **54.8%**) that have code available also provide environment specifications, while an additional 9.7% of the repositories provide only partial environmental information. The large-scale analysis yields comparable results, with 54.4% of the found repositories providing machine-readable environment files, and 11.1% offering partial specifications (*e.g.*, framework mentions). This suggests that even if code is available, an environment specification is not consistently provided in a fully reproducible form.

3) RQ3 – Documentation: The manual analysis showed that **61.3%** of all available repositories have some form of documentation, while an additional 25.8% provide minimal descriptions for the code. The large-scale analysis confirmed that documentation is mostly consistently available, with even 78.0% of the considered papers including a README or similar documentation, and 7.1% having a partial minimal description. Taken together, these results indicate that documentation is widely available, although its completeness varies.

4) RQ4 – Hardware Specification: Most of the considered papers that satisfy RQ1 do not rely on real quantum hardware but instead use classical simulations, accounting for 24 out of 31 repositories (**77.4%**).

Among the remaining seven papers that both describe the use of quantum hardware and provide code artefacts, only three specify the target hardware in sufficient detail, two provide only partial specifications, and the remaining two provide no hardware specification.

Although the small sample size limits the strength of conclusions from the manual analysis, the large-scale results indicate a similar overall trend. Conditional on satisfying RQ1, 23.5% of papers provide a complete hardware specification, 43.5% provide partial information (*e.g.*, mentioning a platform without calibration details), and the remaining 33.1% either omit hardware information entirely or do not rely on quantum hardware.

5) RQ5 – Executability: The manual study revealed that of the examined papers with code availability, only about **35.5%** were executable, while the remaining 64.5% failed to execute due to the following failure modes:

(1) There is no explicit or correct specification of library dependencies and programming language versions (*e.g.*, Python), resulting in floating dependency drifts.
(2) Required licenses and hardware are either not available or difficult to obtain, which hinders testing the provided code.
(3) The provided code contains hardcoded local file paths, as well as missing path definitions and required files, preventing successful code execution.
(4) Non-trivial API incompatibilities are present, obstructing code execution.
(5) Although the code can be brought to a runnable state, execution remains error-prone and insufficient documentation makes it difficult to verify whether the results are correctly reproduced.
(6) Dockerfiles use `latest` tags instead of stable pinned versions, leading to unreliable container executions.

While the automated analysis cannot assess the quality, ex-

ecutability or reproducibility of the artefacts, the combination of both analyses suggests that reproduction artefacts are not only scarce, but also often of insufficient quality to enable reproducibility, even in the short term.

III. Discussion

Our results reveal a substantial gap between the availability of reproducibility artefacts and their actual executability. While approximately one quarter of the analysed papers provide source code (RQ1), and a subset of these further includes environment specifications (RQ2) and documentation (RQ3), the majority of artefacts (64.5%) fail to execute successfully in a clean environment (RQ5). This discrepancy suggests that the presence of code, documentation, and environment descriptions alone is not a reliable indicator of reproducibility in practice.

A closer inspection of the observed failure modes indicates that many issues arise from incomplete or implicit assumptions about the execution environment. Common problems include missing or unpinned dependencies, unspecified programming language versions, hardcoded local file paths, and incompatibilities with evolving APIs. Notably, these failures frequently occur even when some form of environment specification or documentation is provided, suggesting that such artefacts often remain incomplete. Taken together, these observations point towards a central challenge: critical aspects of the execution context are not fully captured or communicated, hindering reproducibility.

Several factors may contribute to this phenomenon. From a technical perspective, software is often developed and tested in local environments that implicitly encode assumptions about system configuration, available dependencies, and data layout. These assumptions may not be apparent to the original developers and therefore remain undocumented in shared artefacts. From a process-oriented perspective, it is plausible that time constraints in publication cycles, the absence of widely adopted reproducibility standards, and the lack of structured reproducibility checklists contribute to incomplete artefact descriptions. We emphasise that these factors represent plausible interpretations of our observations rather than quantitatively validated causes.

Our findings further suggest that commonly used approaches to environment management only partially address these challenges. Tools such as dependency files (*e.g.*, `requirements.txt`), virtual environments, and containerisation frameworks are frequently used to support reproducibility. However, our results indicate that these approaches often fail to capture the full execution context. For instance, dependency specifications may omit system-level requirements, while container-based approaches can suffer from implicit assumptions introduced by base images or the use of non-pinned versions (*e.g.*, `latest` tags), which aligns with other recent findings on the limited build reproducibility of Docker images [36]. More broadly, no single approach has emerged as a widely adopted standard for fully specifying reproducible environments in QC research.

These challenges are particularly pronounced in the context of quantum computing. Compared to other domains, QC research is characterised by rapidly evolving software frameworks, frequent API changes, and a strong dependence on specialised hardware or cloud-based platforms [37]. Access to quantum hardware may be limited or subject to change, and detailed hardware configurations, including calibration states, are often not fully documented. As a result, QC artefacts are especially susceptible to both software and infrastructure-related reproducibility issues. While similar challenges exist in other computational fields, such as machine learning or high-performance computing, the combination of rapid tool evolution and constrained hardware access amplifies these issues in QC.

In this context, prior work in other domains has explored declarative approaches to environment specification, such as those enabled by *GNU Guix* [38], [39] or *Nix* [40], [41]. These approaches aim to describe complete execution environments in a machine-readable and reproducible manner, thereby reducing implicit assumptions. By capturing entire dependency graphs, including system-level components, they directly address several of the failure modes observed in this study. While such techniques have shown promising results in areas such as bioinformatics and high-performance computing, their adoption remains limited, and their applicability to QC workflows has not yet been systematically explored.

At the same time, declarative approaches introduce practical challenges. In particular, they may interact non-trivially with existing ecosystem-specific tooling, such as Python virtual environments or JavaScript package managers, leading to integration inconsistencies in heterogeneous software stacks. However, similar tensions can be observed in container-based workflows, where multiple dependency management layers must be orchestrated. Furthermore, declarative environment approaches have not yet seen widespread adoption in quantum computing workflows. As a result, while they represent a technically promising direction, further work is required to assess their practical integration into existing research practices.

Overall, our study indicates that reproducibility challenges in quantum computing are not solely a matter of artefact availability, but rather of how completely and explicitly the execution environment is specified. Addressing this gap may require both improved tooling and broader community practices that emphasise explicit, machine-readable, and reproducible environment descriptions.

IV. Recommendations

Ensuring reproducibility requires making implicit assumptions about the execution environment explicit. Our findings indicate that many failures arise not from the absence of artefacts, but from incomplete or underspecified environments. To address this, authors should provide complete, machine-readable environment specifications that capture all relevant aspects of the execution context, including programming language versions, library dependencies, and system-level requirements. In addition, dependencies should be pinned to

stable versions to avoid variability introduced by drifting or floating specifications. Reproducibility can be further improved by offering minimal, well-defined entry points into the intended execution environment, for example through a single command that instantiates a development shell and allows the execution of a representative example with expected behaviour. Importantly, such artefacts should be validated across independent environments, for instance by testing on different machines or by multiple contributors, in order to expose hidden assumptions. Aligning the development and reproduction environments can further reduce discrepancies, as environment specifications are then implicitly maintained as part of the development process.

Beyond software environments, reproducibility in quantum computing is inherently constrained by dependencies on specialised hardware and cloud-based platforms. Access to quantum devices is often limited or changes over time, and cloud APIs and service interfaces may evolve or be deprecated. As a result, even fully specified software environments cannot guarantee long-term reproducibility. To mitigate these challenges, authors should document hardware requirements as precisely as possible and provide alternative execution paths, such as simulator-based fallbacks, where applicable. In addition, experimental results should be preserved at the lowest feasible level of abstraction, including raw measurement data and sufficient metadata to reconstruct full post-processing pipelines. Capturing the chain from raw data to final results improves transparency and enables partial reproduction, even when the original hardware or interfaces are no longer available.

To illustrate these recommendations, we provide a reproduction package that serves as a blueprint for reproducible QC artefacts online. The package includes a minimal entry point based on a `Makefile` and containerised execution via Docker, as well as a fully specified container image archived with a persistent identifier. In addition to enabling direct execution, the environment can be rebuilt from source, ensuring transparency and long-term accessibility. The package further integrates the complete experimental pipeline, allowing the paper, including all figures and results, to be recompiled from the underlying data. As an exploratory extension, the package also includes a declarative environment specification based on a Nix flake-based configuration. While this component is not yet intended as a complete solution, it demonstrates how declarative approaches can be used to make environment assumptions explicit and reproducible.

V. Conclusion

Reproducibility remains a key challenge in QC research. In this work, we analysed a large corpus of QC papers and found that only around one quarter of publications provide accessible code artefacts (24.4% in the manual study, 26.8% in the large-scale analysis). Among these, only a minority include complete environment specifications, and the majority of artefacts (64.5%) fail to execute successfully in a clean environment. These results highlight a substantial gap between artefact availability and actual executability.

Our findings indicate that these failures are primarily driven by incomplete and implicit assumptions about the execution environment rather than by the absence of artefacts. Even when code and documentation are available, critical details such as dependency versions, system-level requirements, or data layout are often underspecified. This is further compounded by domain-specific characteristics of quantum computing, including rapidly evolving software ecosystems and dependencies on specialised hardware and cloud-based platforms, which introduce additional challenges for consistent and long-term reproducibility.

While the practical recommendations outlined in this paper address common failure modes, they do not fully eliminate the underlying issue of incomplete environment specification. Declarative approaches to environment management represent a promising direction in this context; however, their applicability to QC workflows remains largely unexplored. Future work will investigate their potential for improving the robustness and longevity of reproducible research artefacts.

Acknowledgements We acknowledge partial support by the German Research Foundation, grant MA 9739/1-1, by the High-Tech Agenda of the Free State of Bavaria, the German Federal Ministry of Research, Technology and Space (BMFTR), funding program 'Research Program Quantum Systems', grant number 13N17387, the European Regional Development Fund (ERDF) and the Free State of Bavaria as part of the project AIM-SMEs (Grant No. 2506-014-3.2), co-funded by the European Union.

References

[1] J. M. González-Barahona and G. Robles, "On the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Empirical Software Engineering*, 2012.

[2] W. Mauerer, S. Klessinger, and S. Scherzinger, "Beyond the badge: Reproducibility engineering as a lifetime skill," in *Proceedings of the 4th International Workshop on Software Engineering Education for the Next Generation*, ser. SEENG '22, Association for Computing Machinery, 2022, DOI: 10.1145/3528231.3528359

[3] F. Trautsch, S. Herbold, P. Makedonski, et al., "Addressing problems with replicability and validity of repository mining studies through a smart data platform," *Empirical Software Engineering*, 2018.

[4] J. M. Gonzalez-Barahona and G. Robles, "Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories," *Information and Software Technology*, 2023.

[5] J. T. Liang, C. Badea, C. Bird, et al., "Can GPT-4 Replicate Empirical Software Engineering Research?" *Proc. ACM Softw. Eng.*, 2024.

[6] T. Yue, W. Mauerer, S. Ali, et al., "Challenges and opportunities in quantum software architecture," in *Software Architecture: Research Roadmaps from the Community*, P. Pelliccione, R. Kazman, I. Weber, et al., Eds., Springer Nature Switzerland, 2023, DOI: 10.1007/978-3-031-36847-9_1

[7] P. Senapati, Z. Wang, W. Jiang, et al., "Towards Redefining the Reproducibility in Quantum Computing: A Data Analysis Approach on NISQ Devices," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2023, DOI: 10.1109/QCE57702.2023.00060

[8] C. Carbonelli, M. Felderer, M. Jung, et al., "Challenges for quantum software engineering: An industrial application scenario perspective," in *Quantum Software: Aspects of Theory and System Design*, I. Exman, R. Perez-Castillo, M. Piattini, et al., Eds., Springer Nature, 2024, DOI: 10.1007/978-3-031-64136-7_12

[9] T. Munasinghe, K. A. Cornell, J. Hendler, et al., "A Knowledge-Based System for Managing Hardware Dependency and Reproducibility in Quantum Machine Learning Workflows," in *2025 IEEE International Conference on Big Data (BigData)*, 2025, DOI: 10.1109/BigData66926.2025.11401505

[10] W. Mauerer and S. Scherzinger, "1-2-3 Reproducibility for Quantum Software Experiments," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, DOI: 10.1109/SANER53432.2022.00148

[11] J. Malka, S. Zacchiroli, and T. Zimmermann, "Reproducibility of Build Environments through Space and Time," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER'24, Association for Computing Machinery, 2024, DOI: 10.1145/3639476.3639767

[12] R. Ramsauer and W. Mauerer, "Towards system-level quantum-accelerator integration," in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2025, DOI: 10.1109/QCE65121.2025.10286 arXiv: 2507.19212 [quant-ph].

[13] E. Younis and C. Iancu, "Quantum Circuit Optimization and Transpilation via Parameterized Circuit Instantiation," in *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE Computer Society, Sep. 2022, DOI: 10.1109/QCE53715.2022.00068

[14] F. Greiwe, T. Krüger, and W. Mauerer, "Effects of imperfections on quantum algorithms: A software engineering perspective," in *IEEE International Conference on Quantum Software (QSW)*, IEEE, 2023, DOI: 10.1109/QSW59989.2023.00014

[15] T. Krüger and W. Mauerer, "Out of the Loop: Structural Approximation of Optimisation Landscapes and non-Iterative Quantum Optimisation," *Quantum*, Nov. 2025. DOI: 10.22331/q-2025-11-06-1903

[16] L. Schmidbauer, E. Lobe, I. Schaefer, et al., "It's quick to be square: Fast quadratisation for quantum toolchains," *ACM Transactions on Quantum Computing*, 2026. DOI: 10.1145/3800943

[17] I. M. Veiga and E. Hänggi, *Reproducible builds for quantum computing*, 2025. arXiv: 2510.02251 [quant-ph].

[18] Y. Shi, P. Gokhale, P. Murali, et al., "Resource-Efficient Quantum Computing by Breaking Abstractions," *Proceedings of the IEEE*, 2020. DOI: 10.1109/JPROC.2020.2994765

[19] S. Dasgupta and T. Humble, "Impact of Unreliable Devices on Stability of Quantum Computations," *ACM Transactions on Quantum Computing*, Oct. 2024. DOI: 10.1145/3682071

[20] S. Thelen, H. Safi, and W. Mauerer, "Approximating under the influence of quantum noise and compute power," in *IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2024, DOI: 10.1109/QCE60285.2024.10291 arXiv: 2408.02287 [quant-ph].

[21] S. R. Maschek, J. Schwittalla, M. Franz, et al., "Make some noise! measuring noise model quality in real-world quantum software," in *Proceedings of the IEEE International Conference on Quantum Software (QSW)*, IEEE, 2025, DOI: 10.1109/QSW67625.2025.00010 arXiv: 2506.03636 [quant-ph].

[22] A. Javadi-Abhari, M. Treinish, K. Krsulich, et al., *Quantum computing with qiskit*, 2024. arXiv: 2405.08810 [quant-ph].

[23] V. Bergholm, J. Izaac, M. Schuld, et al., *Pennylane: Automatic differentiation of hybrid quantum-classical computations*, 2022. arXiv: 1811.04968 [quant-ph].

[24] C. Developers, *Cirq*. Zenodo, Aug. 2025. DOI: 10.5281/ZENODO.4062499

[25] R. S. Smith, M. J. Curtis, and W. J. Zeng, *A practical quantum instruction set architecture*, 2017. arXiv: 1608.03355 [quant-ph].

[26] M. Schulz, M. Ruefenacht, D. Kranzlmüller, et al., "Accelerating HPC With Quantum Computing: It Is a Software Challenge Too," *Computing in Science & Engineering*, 2022. DOI: 10.1109/MCSE.2022.3221845

[27] Z. Zhong, S. He, H. Wang, et al., "An Empirical Study on Package-Level Deprecation in Python Ecosystem," in *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ser. ICSE '25, IEEE Press, 2025, DOI: 10.1109/ICSE55347.2025.00046

[28] D. B. Bose, T. Chan, M. Trimble, et al., "AutoPyDep: A Recommendation System for Python Dependency Management Utilizing Graph-Based Analytics," in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, ser. FSE Companion '25, Association for Computing Machinery, 2025, DOI: 10.1145/3696630.3728576

[29] M. Fernández-Osuna, R. Pérez-Castillo, J. A. Cruz-Lemus, et al., "Exploring design patterns in quantum software: A case study," *Computing*, Apr. 2025. DOI: 10.1007/s00607-025-01467-2

[30] L. J. Kitt and M. B. Cohen, "MorphQ++: A Reproducibility Study of Metamorphic Testing on Quantum Compilers," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ser. ASEW '24, Association for Computing Machinery, 2024, DOI: 10.1145/3691621.3694959

[31] J. Cardinal, I. Benzarti, G. E. boussaidi, et al., *Migrating QAOA from Qiskit 1.x to 2.x: An experience report*, 2026. DOI: 10.48550/arXiv.2512.08245

[32] J. Malka, S. Zacchiroli, and T. Zimmermann, "Does Functional Package Management Enable Reproducible Builds at Scale? Yes.," in *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, 2025, DOI: 10.1109/MSR66628.2025.00115

[33] A. Devresse, F. Delalondre, and F. Schürmann, "Nix based fully automated workflows and ecosystem to guarantee scientific result reproducibility across software environments and systems," in *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*, ser. SE-HPCCSE '15, Association for Computing Machinery, 2015, DOI: 10.1145/2830168.2830172

[34] Open Archives Initiative, *arXiv API*, 2026.

[35] R. Kinney, C. Anastasiades, R. Authur, et al., *The semantic scholar open data platform*, 2025. arXiv: 2301.10140 [cs.DL].

[36] J. Malka, S. Zacchiroli, and T. Zimmermann, *Docker does not guarantee reproducibility*, 2026. arXiv: 2601.12811 [cs.SE].

[37] M. Schönberger, M. Franz, S. Scherzinger, et al., "Peel — pile? cross-framework portability of quantum software," in *IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, 2022, DOI: 10.1109/ICSA-C54293.2022.00039

[38] N. Vallet, D. Michonneau, and S. Tournier, "Toward practical transparent verifiable and long-term reproducible research using guix," *Scientific Data*, 2022.

[39] L. Courtès and R. Wurmus, "Reproducible and user-controlled software environments in hpc with guix," in *European Conference on Parallel Processing*, Springer, 2015,

[40] M. Hausch, S. Hauser, and B. Uekermann, "Improving reproducibility of scientific software using nix/nixos: A case study on the precice ecosystem," *Electronic Communications of the EASST*, 2025.

[41] E. Dolstra, M. De Jonge, E. Visser, et al., "Nix: A safe and policy-free system for software deployment.," in *LISA*, 2004,