Toolchain for experiment tracking in iterative quantum software development

Otso Kinanen
University of Jyväskylä
Jyväskylä, Finland
otso.j.r.kinanen@jyu.fi

Mahee Gamage
University of Jyväskylä
Jyväskylä, Finland
mahee.s.hewagamage@jyu.fi

Vlad Stirbu
University of Jyväskylä
Jyväskylä, Finland
vlad.a.stirbu@jyu.fi

***Abstract*—As quantum software engineering (QSE) matures, practitioners are adopting classical software engineering practices. This paper proposes a quantum experiment tracking infrastructure that aids quantum researchers in making informed decisions while developing quantum software incrementally and iteratively. By leveraging tools and best practices from classical machine learning communities, we provide a mature environment for quantum software practitioners to enhance their practices. Our approach involves collecting a comprehensive quantum provenance dataset from each experiment execution for analysis and decision-making. To validate our approach, we developed an experimental environment where a program was iteratively developed on quantum simulators and an IQM computer, and we demonstrate how the development workflow backed by the tracking infrastructure meets expectations for quantum experiment reporting.**

***Index Terms*—Quantum software engineering, development workflow, methodology**

I. Introduction

The field of quantum computing has evolved rapidly over the past decade, fostered by improvements in quantum computer hardware. These improvements allow researchers to develop increasingly complex quantum algorithms with a growing impact on numerous scientific and industrial application domains. To support this progress, the emerging field of Quantum Software Engineering (QSE) has started to adopt practices inspired from classical software engineering. However, QSE diverges from classical software engineering due to the inherent uncertainty of quantum computers, but also practical limitations of current hardware such as measurements prone to errors, and limited accessibility of quantum hardware.

These unique challenges require a new well-defined software development lifecycle for QSE. The lifecycle needs to be adapted with tools that will assist researchers and developers to understand the context in which their programs run. In current practice, practitioners begin their quantum application development from simulated environments, and move to actual quantum computers in the later stage of the development, while keeping track of each experimental run. For them to make educated decisions in the process they need to keep track of quantum circuits, state of the targeted hardware, compilation, and executions. This necessitates suitable tooling to help collect, store, organise, and analyse the data.

In this paper, we propose a quantum experiment tracking infrastructure. The solution assists quantum researchers in making informed decisions while developing quantum software incrementally and iteratively. By leveraging tools and best practices widely adopted in classical machine learning communities, we provide a mature environment that quantum software practitioners can adopt to enhance their practices.

This article is structured as follows. We start with background in Section II, followed by the motivation and objectives of our work in Section III. Next we introduce our proposed solution to support software developer in quantum computing with experiment tracking tooling in Section IV. Then we provide an example of how the enhanced process can be used in practice in Section V. We conclude with the discussion in Section VI and concluding remarks in Section VII

II. Background

A. Quantum Software Development and Engineering

Software engineering for quantum computing and quantum classical hybrid computing is an emerging topic as the recent advancements in quantum hardware have turned the theory into reality. At best, the computational advantage from quantum algorithm may be phenomenal, as with Shor's factoring algorithm and quantum Fourier transform providing exponential speed-up over traditional methods, or as with Grover's search algorithm providing a less drastic but still noticeable quadratic speed-up in various search-based problems. The novel computing paradigm demands new practices and adaptations to classical methods, and affects many areas of software engineering [1]. All advancement towards practical quantum computing is increasing this said need to adapt and create new development methods specific for quantum software [2]. In the current state of quantum computing the dominant programming paradigm is circuit programming model, where each wire in the circuit presents one qubit, for which quantum gates are then applied on to perform computations, and finally qubits are measured [3]. Quantum software development should follow iterative methodologies [4], in which the developer begins implementation at a suitable scale, considering circuit complexity and the available execution hardware. The process starts with local

This work has been supported by the Academy of Finland (project DEQSE 349945), Business Finland (EM4QS 155/31/2024), Finnish Ministry of Education and Culture through the Quantum Doctoral Education Pilot Program (QDOC VN /3137/2024-OKM-4) and the Research Council of Finland through Finnish Quantum Flagship project (359240, JYU).

979-8-3315-5736-2/25/$31.00 ©2025 IEEE
DOI 10.1109/QCE65121.2025.10317

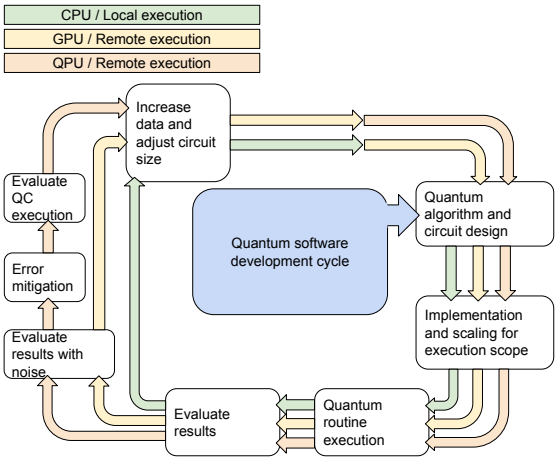

Fig. 1. Iterative and incremental software development model for quantum software.

execution on a small dataset and environment setup, then incrementally moves to larger-scale classical simulators with increased data and circuit complexity, and finally to quantum platforms—evaluating execution and results at each iteration [5]. The model is presented in Fig. 1.

B. Quantum software middleware

Current quantum computing systems are based on layered stack design, abstraction presented in Fig. 2. Typically the systems are built as follows. Top layer of the stack offers a user interface, for example to submit a quantum workload for execution in a cloud platform. Under the user interface layer, is quantum application layer for interacting with the quantum code. Beneath that lays logical compilation and optimization layer. Followed by quantum error correction software. The compilation from circuit to the executable code is very important layer in the process, while containing contains the qubit mapping process, which is one computationally hard task in the process and will effect drastically the execution quality [6], [7]. Generally, the software developer has access and control up to four top layers of the quantum computing stack. The following layers, hardware compilation, quantum firmware and the actual physical qubit implementation is usually beyond the reach of the developer [2]. Post execution, developers may also apply different quantum error mitigation (QEM) techniques, aimed to improve the results and mitigate the noise [8].

C. Quantum execution

The selection of quantum hardware available is getting wider, with new providers and services being offered for the public. While the availability is getting better the high operational cost, current error levels and the inability to perform step by step debugging create a need to use classical quantum simulators in the development process [4], [9]. Therefore, in the quantum software development process classical simulators remain as important tools by offering step-by-step

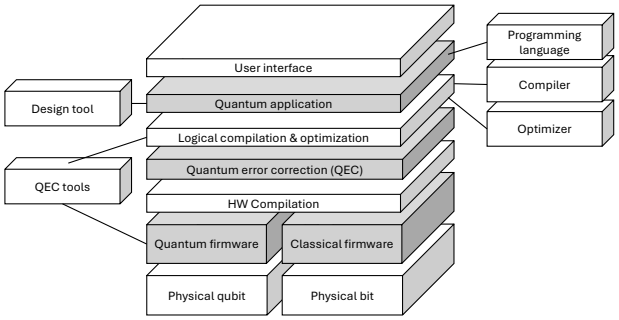

Fig. 2. Quantum computing stack components overview [2].

execution and exact computation outputs. Simulators also have their limitations, as the circuit grow in depth and width the memory of any classical platform will eventually run out, and the execution times will become longer than could be considered practical.

D. Experiment tracking

In machine learning and AI development, experiment tracking is the systematic process of recording, managing, and analyzing all relevant information from each experimental run. This includes capturing key artifacts such as datasets, model parameters, predictions, and evaluation metrics. Comprehensive experiment tracking enables reproducibility, facilitates model comparison across different approaches, and provides transparency throughout the ML workflow, making it an essential practice for reliable and scalable machine learning development. [10]. MLflow[1] is an open-source platform developed to streamline this entire ML lifecycle, addressing four major challenges in ML development: experiment tracking, reproducibility, the multitude of tools, and production deployment.

The experimental workflow of current quantum software development resembles the classical ML workflows, making MLflow a strong candidate to facilitate experiment tracking capabilities. Its flexible, open-interface design supports tracking of any Python-based experiments, including those in quantum software development.

E. Quantum Provenance

Data provenance methods and practices has been identified in classical computing literature for decades and has it's clear demand in both academic and in business domains [12], [13]. Quantum computing, with different hardware architecture and programming paradigm has it's unique demands and specialities when coming to provenance. In their article Weder et al. have [11] recognized the important items to be considered in quantum provenance and composed a model for categorising them accordingly, see Fig. 3. Quantum provenance categories and items considered are the following. Quantum Circuit, including the gates used and available gate set, measurement

[1]https://mlflow.org/docs/latest/ml/

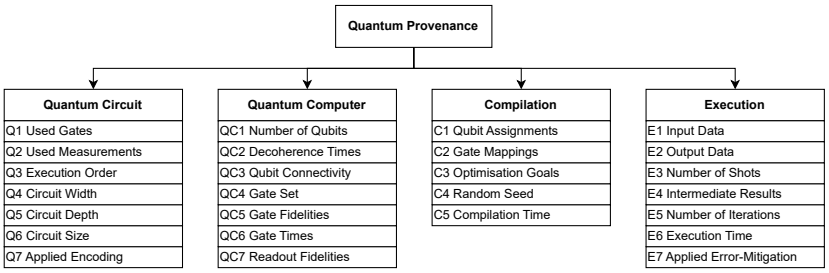

Fig. 3. Categories of quantum provenance attributes. Adapted from [11]

protocols, circuit dimensions (width, and depth), and encoding methods applied. Quantum Hardware details including number of qubits, decoherence times, qubit connectivity, native gate set, gate fidelities and times, and readout fidelity. Compilation process, including qubit assignments, gate mappings, optimization objectives, random seeds used, and total compilation time. Execution details, input and output data, number of measurement shots, any intermediate results captured, iteration counts, total execution time, and error mitigation techniques employed.

III. Motivation and objectives

The workflow model for quantum software development presented in II-A offers a clear model to follow in the development process, with incremental progress. Yet it demands evaluation of the execution and results as a part of the progress. To evaluate the execution developer needs to have suitable tooling to access and analyse the experiment data. The data collected and later analysed should be considered to match items introduced in II-E. The developer needs to take in consideration, not only the results form the execution, but also the direction of development in following iterations, and multiple data points from execution process. In our research we have noticed that in the current development tools that otherwise do support well the development process, there are limited tools for data collection and analysing to support decision making. For our research and development work we have set our objective as:

O1: How to enable data driven decision making in quantum development workflow?

Our approach on achieving this objective is to develop a comprehensive solution for tracking quantum computing experiments, encompassing data collection and evaluation processes. This holistic approach will empower data-driven decision-making capabilities. To achieve this objective, we have used design science research methodology (DSRM) and have selected to use objective centred approach [14], method and process targets presented in Fig. 4.

IV. Quantum development workflow supported by experiment tracking

Following the workflow requires a developer to evaluate the execution and the quantum code on each iteration, and further more make informed decisions on the direction to continue with. To perform the evaluation the developer will need suitable tooling to collect and to analyse the data. As the quantum computing has many crucial components that are not yet mature and prone to errors, the necessity of evaluation is not limiting to execution results and quality. Experiment tracking tooling, will help the developer in both tasks, collecting and analysing the data during the development. In line with the development workflow, the points that will demand data driven decisions in the development workflow are, *Implementation and scaling for execution scope* e.g. user has scale the circuit width and depth accordingly to the available number of qubits and errors rates of gates and readout fidelities in the platform, *Evaluate results* and *Evaluate results with noise* e.g. user should be able to compare results and execution quality of each execution finding the limits of the given hardware and to possibly improve the results by changing qubit assignment and routings, and *Evaluate the QC execution* e.g. developer should evaluate the output of the execution and evaluate applied error mitigation efficiency.

Using an experiment tracking tool, such as MLFlow adapted accordingly to quantum computing, the user will be able to make data-driven and educated decisions on the development process on each iteration of the development process. Quantum programs can be run on a multitude of execution environments, identified at each stage of the development workflow. The client library logs the experiment data and artifacts to the server, which can be viewed via the web-based user interface. The tracked experiments can be queried later for further analysis that supports the decision-making process. The experiment tracking infrastructure is depicted in Fig. 5.

V. Development process in practice

A. Evaluation environment

To evaluate how the experiment tracking facilitates decision making process in an iterative quantum software development process, we have devised an experimental environment in which a developer performs iteratively a series of development experiments. The experiment is executed first on its own laptop, then on a remote cluster that has GPU computation capabilities (e.g. Nvidia GTX 4090), and, finally, on a IQM quantum computer accessed via the QX service operated by

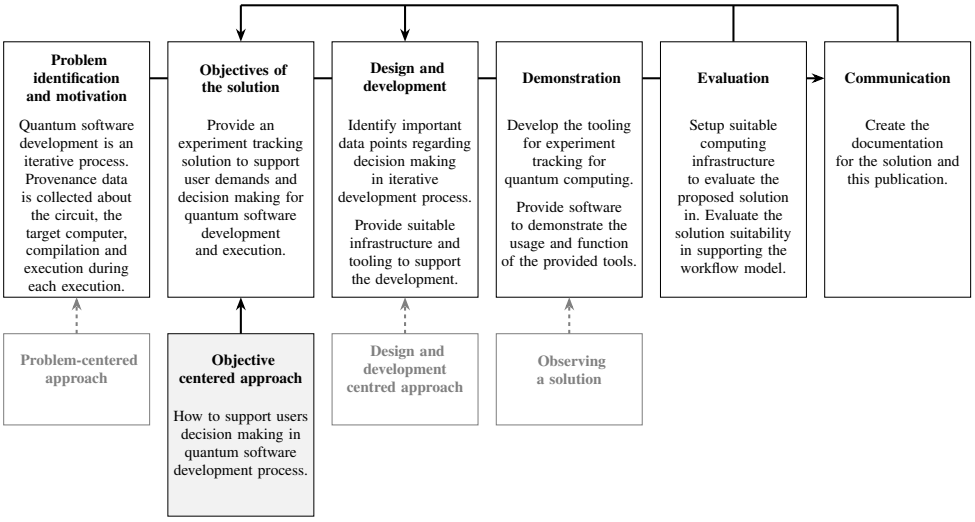

Fig. 4. Design science research methodology applied to improve the quantum software development process.

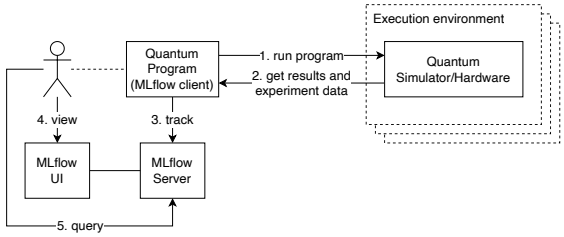

Fig. 5. Experiment tracking infrastructure assisting quantum development process

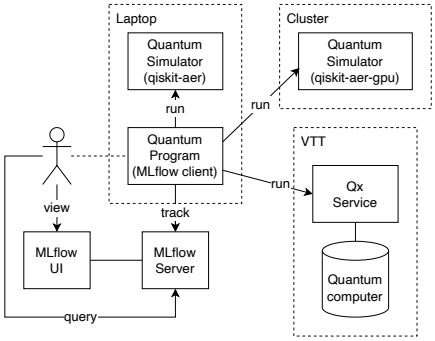

Fig. 6. Experiment setup: the user executes a program on the on a quantum computer exposed via the VTT QX service and tracks the experiment results to the MLflow tracking server

VTT. The MLflow server and UI components are available in the environment. The evaluation environment is depicted in Fig. 6.

B. Tracking experiment data

The MLflow client library allows application developers to collect a multitude of data related to their experiments, see the Listing 1. The developer can name the experiment (e.g. line 8), allowing different iteration runs to be grouped. Various information about the software used can be collected as experiment *tags*, seen in line 10. The input parameters used at various stages of the execution can be collected as *params*, seen in lines 24 or 35. The MLflow library allows collection of other run artifacts, allowing the developers to collect images or JSON dictionaries, as seen in line 37 and 42, respectively.

The data collected to the MLflow server by the samle program can be mapped to all categories mentioned by the quantum provenance attributes (QProv): *quantum circuit* – Q5 circuit depth, *compiler* – C3 optimisation goals, *execution* – E2 output data, and *quantum computer* – QC2 decoherence time, QC5 gate fidelities, or QC7 readout fidelities. This information is collected programatically from the quantum software development kit or from third-party systems, such as the Qiskit SDK[2] or the VTT QX service API[3].The program included in this paper demonstrates what is achievable within the experimental setup, extended versions are available in our online repository[4].

C. Decision-making support

Tracking data from individual experiment runs is crucial, but often, decision-making relies on analysing data from multiple runs. The MLflow provides facilities to search for relevant experiments. The returned data is presented as Python datagrams, which can be utilised according to the application's specific needs to make appropriate decisions, depending on

[2]https://docs.quantum.ibm.com/api/qiskit
[3]https://qx.vtt.fi/docs/
[4]https://github.com/qubernetes-dev/q8s-examples/blob/main/qiskit-iqm-mlflow/

```python
from qiskit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit.visualization import plot_histogram
from iqm.qiskit_iqm import IQMProvider
import mlflow

mlflow.set_tracking_uri("https://mlflow.example.com")
mlflow.set_experiment("Quantum iterative development")

mlflow.set_tag("QDK", "qiskit")

with mlflow.start_run():
    provider = IQMProvider("https://qx.vtt.fi/api/devices/q50")
    backend = provider.get_backend()

    abstract = QuantumCircuit(2, 2)
    abstract.h(0)
    abstract.cx(0, 1)
    abstract.measure([0, 1], [0, 1])

    optimization_level = 3
    pm = generate_preset_pass_manager(backend, optimization_level=optimization_level)
    # QProv-C3
    mlflow.log_param("optimization_level", optimization_level)

    physical = pm.run(abstract)
    # QProv-Q5
    mlflow.log_text("circuit_depth", abstract.depth())

    shots = 500

    job = backend.run(physical, shots=shots)
    result = job.result()
    # QProv-E3
    mlflow.log_param("shots", shots)
    # QProv-E2
    mlflow.log_figure(plot_histogram(result.get_counts()), "results.png")

    calibration_set_id = result.results[0].calibration_set_id
    calibration_data = get_calibration_data(backend.client, calibration_set_id)
    # QProv-QCx
    mlflow.log_dict(calibration_data, "calibration_data.json")
```

Listing 1: Tracking experiment data with MLflow. The implementation of `get_calibration_data` function has been omitted for brevity.

```python
import mlflow
import pandas as pd
import matplotlib.pyplot as plt
experiment_name = "Quantum iterative development"
exp = mlflow.get_experiment_by_name(experiment_name)

# Data-frame can be further processed with pandas and matplotlib
runs_df = mlflow.search_runs(experiment_ids=[exp.experiment_id])
...
```

Listing 2: Processing experiments data with MLflow

the stage of the development workflow. The Listing 2 offers a template for retrieving experiment data.

VI. Discussion

We set our objective O1 for this research as "How to enable data driven decision making in quantum development workflow?". To answer to this objective, we have identified the data points necessary for the developer from literature, and developed solution based on open-source classical software components. In this paper, we introduce the solution, present our experimental setup, and provide practical guidance through example code. We suggest that using an experiment tracking tool to collect quantum provenance data, combined with the workflow model, the developer will be able to make effective data-driven decisions in the development process.

The study by Moguel et al. [15] offers an evaluation guideline and a checklist for reporting quantum computing experiments, holding items that should be collected and presented when presenting experimental quantum computing research. The items have been divided into six sub categories, context, experimental planning, experimental design, execution, analysis, threats to validity. Our suggested solution MLflow adopted for quantum combined with the presented workflow model supports the user to reach all items listed, presented in Table I. For the context category items, the experiment tracking collects data from the execution platforms and stores the quantum circuits. The experimental planning is part of the workflow process, including the scaling of quantum circuit accordingly to the target, experimental design items such as shots executed and the reasoning for that, also user should use data collected from earlier executions to support the decisions. The items from execution category are partly collected by the experiment tracking tool, e.g. experiment tracking tool may store precision and calibration data of the hardware, but software version data and dependencies should be collected and managed in specialised dependency management tools as part of the development process. Analysis demands the developer to have access to experiments output data and applied QEM methods, in the workflow model this is covering steps evaluating results. Finally, the threats to validity demands again coherent and thoroughly collected data from multiple executions, experiment tracking collects this data and helps the developer to access and evaluate it at a later time.

In our current implementation, MLflow is used to collect the data necessary for coherent provenance data, and helps the developer to conduct controlled and replicable experiments on quantum computing. Yet, in its current form, the solution requires the user to have wide knowledge of the topic for user to be able to pick the data points to collect and to use. This affects negatively the usability, demands the code to collect the data to be developed, and increases the workload of the developer. For classical machine learning experiments, within the MLflow ecosystem, there are toolkit specific tools that are integrated into experiment tracking and automatically perform tasks like logging functionalities for metrics, parameters, model signature, artefacts and dataset. We plan to research and implement similar automatic logging solutions for quantum computing.

The experiment tracking infrastructure based on MLflow provides a solid foundation for making informed decisions during quantum software development following our iterative and incremental workflow. Further, popular quantum software development kits (e.g. Qiskit) and hardware providers (e.g. VTT QX) provide facilities to collect execution data that conforms with the QProv attributes. Finally, the proposed toolchain and the workflow are suitable and support the quantum experiment reporting content and formats expected in the field. We consider the objective O1 to be achieved.

TABLE I
CHECKLIST FOR QUANTUM COMPUTING EXPERIMENTS REPORTING BACKED BY THE WORKFLOW AND EXPERIMENT TRACKING INFRASTRUCTURE

Category	Item	Experiment tracking	Workflow
Context	Min Q-bits	✓	
	Gate requirements	✓	
	Circuit depth (if appropriate)	✓	
	Connectivity between qubits	✓	
	Limitations imposed by the execution platform (max execution time, max qbits, etc.)	✓	
Experimental Planning	Bootstrapping needs		✓
	Target platform execution limitations		✓
	Target platform planning limitations		✓
Experimental Design	Justification for the shots and shots to execute		✓
	Techniques used to initialize quantum state		✓
Execution	Precision	✓	
	Calibration data for the qubits used during the experiment	✓	
	Classical pre-processing and post-processing steps	✓	
	Quantum programming language and framework (e.g., Qiskit, Cirq, etc.)		✓
	The version of the software framework and any dependencies		✓
	Random seeds used in case of simulation-based experiments	✓	
	Measurement settings and readout error mitigation techniques	✓	
Analysis	Statistical modelling of the distributions of outputs		✓
	Quantum error correction or mitigation techniques employed		✓
Threats to Validity	Analysis of how the results might vary with different quantum hardware		✓
	Description for how the experiment might be modified or extended to other quantum computing platforms		✓

VII. CONCLUSIONS

In this article we discuss the relation of current suggested quantum software development workflow and its relation to data provenance in quantum computing experiments. We have recognised that developer needs to evaluate multiple data points on each iteration of software development, and to do so efficiently they will need suitable tooling. The experimental setup built around MLflow shows that experiment tracking tools are necessary and will provide support for data driven decision in the development process. To support this claim we have validated the suggested solution with a checklist for quantum computing experiment reporting.

REFERENCES

[1] A. A. Khan, A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, T. Mikkonen, and P. Abrahamsson, "Software architecture for quantum computing systems—a systematic review," *Journal of Systems and Software*, vol. 201, p. 111682, 2023.

[2] M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, and M. Piattini, "Quantum software components and platforms: Overview and quality assessment," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–31, 2022.

[3] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[4] A. A. Khan, M. Fahmideh, A. Ahmad, M. Waseem, M. Niazi, V. Lahtinen, and T. Mikkonen, "Embracing iterations in quantum software: a vision," in *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, ser. QP4SE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 11–14. [Online]. Available: https://doi.org/10.1145/3549036.3562057

[5] O. Kinanen, A. D. Muñoz-Moller, V. Stirbu, J. M. Murillo, and T. Mikkonen, "Toolchain for faster iterations in quantum software development," *Computing*, vol. 107, no. 4, pp. 1–28, 2025.

[6] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 9, no. 1, 2018, pp. 138–142.

[7] M. Maronese, L. Moro, L. Rocutto, and E. Prati, "Quantum compiling," in *Quantum Computing Environments*. Springer, 2022, pp. 39–74.

[8] Z. Cai, R. Babbush, S. C. Benjamin, S. Endo, W. J. Huggins, Y. Li, J. R. McClean, and T. E. O'Brien, "Quantum error mitigation," *Reviews of Modern Physics*, vol. 95, no. 4, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1103/RevModPhys.95.045005

[9] M. Golec, E. S. Hatay, M. Golec, M. Uyar, M. Golec, and S. S. Gill, "Quantum cloud computing: Trends and challenges," *Journal of Economy and Technology*, 2024.

[10] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[11] B. Weder, J. Barzen, F. Leymann, M. Salm, and K. Wild, "Qprov: A provenance system for quantum computing," *IET Quantum Communication*, vol. 2, no. 4, pp. 171–181, 2021. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/qtc2.12012

[12] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer, 2001, pp. 316–330.

[13] Y. L. Simmhan, B. Plale, D. Gannon *et al.*, "A survey of data provenance techniques," *Computer Science Department, Indiana University, Bloomington IN*, vol. 47405, p. 69, 2005.

[14] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007. [Online]. Available: https://doi.org/10.2753/MIS0742-1222240302

[15] E. Moguel, J. A. Parejo, A. Ruiz-Cortés, J. Garcia-Alonso, and J. M. Murillo, "Quantum software experiments: A reporting and laboratory package structure guidelines," 2024. [Online]. Available: https://arxiv.org/abs/2405.04192