QuantuMoonLight: A Low-Code Platform to Experiment with Quantum Machine Learning

Francesco Amato[a], Matteo Cicalese[a], Luca Contrasto[a], Giacomo Cubicciotti[a], Gerardo D'Ambola[a], Antonio La Marca[a], Giuseppe Pagano[a], Fiorentino Tomeo[a], Gennaro Alessio Robertazzi[a], Gabriele Vassallo[a], Giovanni Acampora[c], Autilia Vitiello[c], Gemma Catolino[b], Giammaria Giordano[a], Stefano Lambiase[a], Valeria Pontillo[a], Giulia Sellitto[a], Filomena Ferrucci[a], Fabio Palomba[a]

[a]*Software Engineering (SeSa) Lab—University of Salerno, Italy*
[b]*Jheronimus Academy of Data Science & Tilburg University, The Netherlands*
[c]*University of Naples, Italy*

Abstract

Nowadays, machine learning is being used to address multiple problems in various research fields, with software engineering researchers being among the most active users of machine learning mechanisms. Recent advances revolve around the use of *quantum* machine learning, which promises to revolutionize program computation and boost software systems' problem-solving capabilities. However, using quantum computing technologies is not trivial and requires interdisciplinary skills and expertise. For such a reason, we propose QuantuMoonLight, a community-based low-code platform that allows researchers and practitioners to configure and experiment with quantum machine learning pipelines, compare them with classic machine learning algorithms, and share lessons learned and experience reports. We showcase the architecture and main features of QuantuMoonLight, other than discussing its envisioned impact on research and practice.

Keywords: Quantum ML, Low-Code Platforms, Artificial Intelligence.

1. Motivation and Significance

Machine Learning (ML) is now, more than ever, one of the primary mechanisms employed to solve real-world problems. Among the various research communities which are benefiting from its use, the software engineering research one has been employing it to support practitioners under several perspectives like the analysis of source code naturalness [1], code smell detection [2, 3], defect prediction [4], and test code quality (e.g., test flakiness [5]

and test case effectiveness [6]), to name a few. The adoption of quantum computing technologies represents the next frontier of the research on Machine Learning in software engineering [7–9], as it can help solve some of the limitations affecting classical computing—*e.g.*, long training times. Quantum technologies have the potential to exponentially increase processing capabilities, enabling more efficient and faster algorithms for tasks such as detection and pattern recognition [10], leading to better quality software. Moreover, quantum machine learning can allow deeper analyses of large or complex data sets, being particularly useful in the field of scientific research. Big firms like IBM, Google, and Microsoft are investing in quantum hardware, providing access to some of their resources for experimental usage by researchers and practitioners. This is the case of the *IBM Quantum* platform [11], which allows users to gain access to quantum machines via a cloud-based API and lets them design, implement, and execute their quantum applications on IBM hardware. Similar solutions are provided by Microsoft Azure,[1] D-Wave,[2] and Xanadu.[3] Recently, Grossi *et al.* [12] proposed a highly extendable framework to build quantum-based web apps. Moreover, Di Marcantonio *et al.* [13] proposed QUASK, a library to integrate quantum machine learning algorithms in traditional programs. However, exploiting quantum computing technologies is still challenging and requires inter-disciplinary skills, other than the basic knowledge about the underlying technology, namely how quantum circuits are defined and work [14–16]. In the case of classic machine learning, a plethora of ready-to-use tools and guidelines are currently available to allow non-experts to exploit the technology as a black-box, leveraging its functionalities without necessarily understanding the inner-working of ML algorithms. This is not the case for quantum machine learning, as users must have knowledge of the quantum engine to experiment with the computing platforms provided by large companies owning the hardware.

To address this gap, we propose QuantuMoonLight, a low-code web application designed to fulfill the following requirements of researchers and practitioners: (1) configure and experiment with quantum machine learning algorithms; (2) compare quantum solutions with canonical machine learning algorithms; and (3) openly discuss experience and share solutions through a community-inspired blog. The goal of QuantuMoonLight is to create a level of abstraction that hides the intrinsic complexity of quantum circuits, providing users with a graphical user interface through which they can inter-

[1] Azure Quantum: `https://azure.microsoft.com/it-it/services/quantum/`
[2] D-Wave: `https://www.dwavesys.com/`
[3] Xanadu: `https://xanadu.ai/`

act with quantum machines and run quantum machine learning algorithms. In contrast with existing solutions for running quantum algorithms, that are designed to be used by experts in the domain, QuantuMoonLight is designed for non-expert users, that are allowed to experiment with quantum machine learning algorithms without the underlying knowledge about how quantum circuits work. QuantuMoonLight is available as a web application,[4] and is accompanied by a demonstration video.[5]

In this paper, we describe the architecture, features, and potential impact of QuantuMoonLight. Moreover, we report on a preliminary assessment of the tool, which focused on two software engineering tasks such, as those of code smell and flaky test prediction.

2. Software Description

2.1. Overview of the Tool

QuantuMoonLight is a web application—already deployed on the web—to experiment with quantum machine learning. It relies on the *IBM Quantum Computing platform* [11]—*i.e.*, a framework that allows quantum computers to be programmed on the cloud—and, from a software perspective, uses the *Qiskit* framework[6]—*i.e.*, an open-source SDK for working with quantum computers. The tool was designed to allow for (1) the configuration of quantum machine learning models and (2) the comparison between quantum and classic machine learning solutions.We provided the tool with a user-friendly graphical interface to abstract the users from the natural complexity of the quantum mechanisms employed.

2.2. Software Features and Architecture

QuantuMoonLight allows users to perform the following operations:

1. Configuration and validation of quantum machine learning models;
2. Comparison of quantum and classic machine learning solutions developed on the platform;
3. Sharing knowledge and results about quantum machine learning solutions through a community-inspired blog.

[4] QuantuMoonLight site: `https://sesaquantumoonlight.ngrok.io/`
[5] QuantuMoonLight demo video: `https://youtu.be/xhXj1uZ7P1M`
[6] *Qiskit*: `https://qiskit.org/`

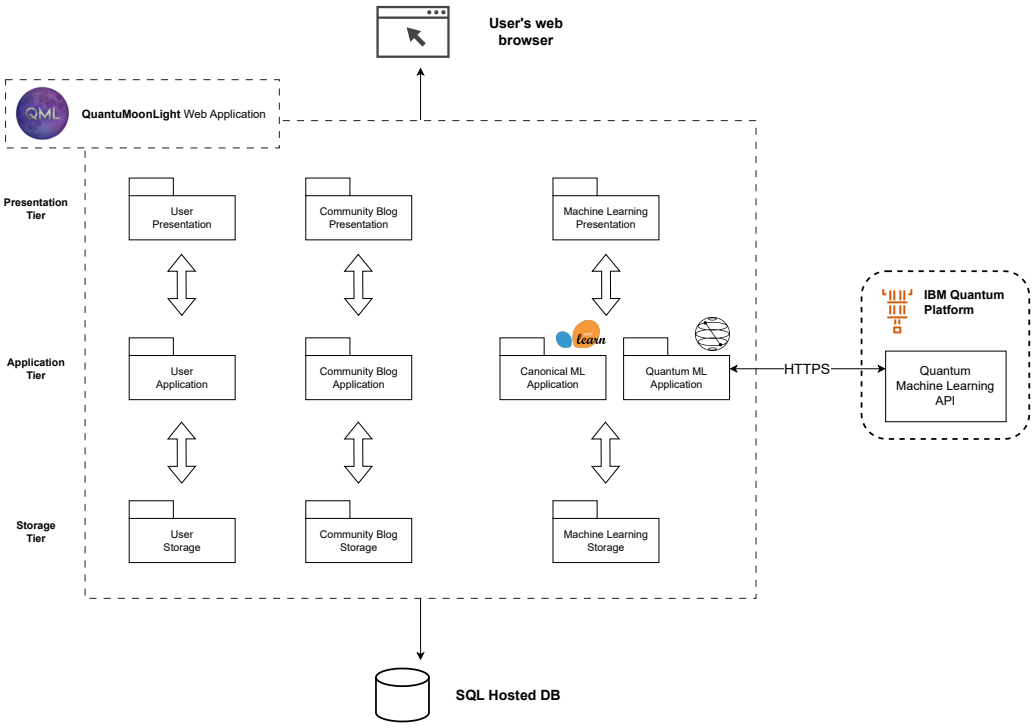

Figure 1: QuantuMoonLight tool architecture.

We designed the tool to make it suitable for empirical research aiming at evaluating the performance of quantum algorithms. Indeed, QuantuMoonLight can help researchers with data preprocessing (*e.g.*, data cleaning and normalization), hyper-parameters configuration, feature selection, validation strategy choice, and evaluation metrics set to compute in an experimentation. It is important to note that the tool does not automatically select the best pipeline for a task but requests the user to do that. Moreover, the blog feature can let researchers and practitioners discuss about the experiments conducted, hence (1) increasing the awareness of the potential of quantum machine learning and (2) lowering the entry barriers faced by newcomers approaching such a complex theme. The tool requires users to be registered to acquire data about their permissions over the IBM Quantum platform [11].

Figure 1 reports the architecture of the tool. It was developed as a web application, implementing a three-tier architecture. Moreover, we split QuantuMoonLight into three core subsystems:

- **User**: It manages the users account, login, and registration processes;
- **Community blog**: It implements the community blog features;

- **Machine learning**: It implements the application's functions related to machine learning—both quantum and canonical.

In the tool, data flows vertically through the three tiers, from the presentation to the storage. All the subsystems are decoupled and independent. Each subsystem is horizontally distributed through the three tiers. The presentation tier manages the interaction with users by means of the graphical user interfaces. We developed three different sets of GUIs, one for each subsystem.

Regarding the application tier, while the User and Community Blog implement basic functionalities—*e.g.*, login—the machine learning one is the most interesting. This subsystem enables the creation of both classic and quantum machine learning models from an input dataset. The quantum machine learning part consists of an adapter that implements the communication—over an HTTP protocol—with the Web API of the *IBM Quantum Computing* platform [11]—implemented using the REST API of the platform.[7] The adapter has been designed to be highly independent from the IBM platform, allowing future integrations of new quantum learners, configuration steps, and back-ends. As for the classic machine learning part, our web application implements on-premise solutions based on *scikit-learn*.[8]

The platform uses a database—implemented as a cloud SQL database—to collect historical data and implement the community-oriented blog. From the implementation side, we used Python and the well-known Flask framework for the web-deployment infrastructure.

2.3. Quantum Machine Learning with QuantuMoonLight

To enable experimentations, QuantuMoonLight relies on the APIs provided by *Qiskit*. Specifically, the tool allows users to exploit algorithms for data classification and quantum support vector regressors.[9] As for the classifiers, the tool implements the *Quantum Support Vector Machine*, the *Quantum Support Vector Classifier*, the *Quantum Neural Network Classifier*, and the *Pegasos Quantum Support Vector Classifier* algorithm defined by Shalev-Shwartz *et al.* [17]. As for the regressors, the tool implements the *Support Network* and *Variational Quantum Regressor*.

To use this tool, an account (and a token) on the *IBM Quantum Computing* [11] platform is required. The *IBM Quantum* platform provides different

[7] REST API of IBM Quantum: `https://cloud.ibm.com/apidocs/quantum-computing`

[8] *Scikit-learn*: `https://scikit-learn.org/stable/`

[9] *Qiskit* Machine Learning API reference: `https://qiskit.org/documentation/machine-learning/apidocs/qiskit_machine_learning.algorithms.html`

back-end systems with different hardware potentiality to perform the quantum operations.[10] For example, the *ibm_washington* system is characterized by a high number of qubits (127), while the *ibm_lagos* has fewer (7). Our application allows users to select the desired back-end to perform ML tasks, but with limitations based on the type of logged user. Specifically, "standard" registered users can select among seven systems equipped with a number of qubits ranging from 1 to 5. Users with a research license can select all the systems for "standard" users plus three machines provided with 7 qubits. Moreover, being that *QuantuMoonLight* relies on IBM quantum machines, the queue to access such machines depends on the status of the *IBM Quantum* platform. Specifically, the policy of IBM stipulates that each back-end has a separate queue with a first-in-first-out logic; this means that if the selected back-end is busy, QuantuMoonLight informs the user that the computation time could be longer.

3. Illustrative Examples

QuantuMoonLight provides its features through a four-page website. In the following, we describe the page representing the core functionalities of the developed tool—more details are available in our online appendix [18].

Figure 2 shows the GUI of the web page enabling to configure and run experiments with QuantuMoonLight. The first step consists in loading the desired dataset into the tool (Sub-figure 2.1). QuantuMoonLight allows uploading the training, test, and prediction sets, depending on the specific experiment the user would like to run. The tool also provides the possibility to quickly setup experiments; the user can indeed check whether they would like to apply default configurations in terms of feature extraction, feature selection, and validation techniques. In particular, the latter is implemented using a percentage split that automatically assigns to the training set 80% of the instances in the uploaded dataset and 20% in the test set. The user can then select the quantum machine learner to use among the supported ones (Sub-figure 2.2).

A user can already execute quantum experiments with the first two configuration steps. However, further customization is allowed through the *"Advanced Options"* menu, showed in Sub-figures 2.3-2.7. In terms of validation, the tool provides an additional setting, *i.e.*, the k-fold cross-validation. The platform also enables the automation of a number of data pre-processing

[10] *IBM Quantum* back-end: `https://quantum-computing.ibm.com/services?services=systems`

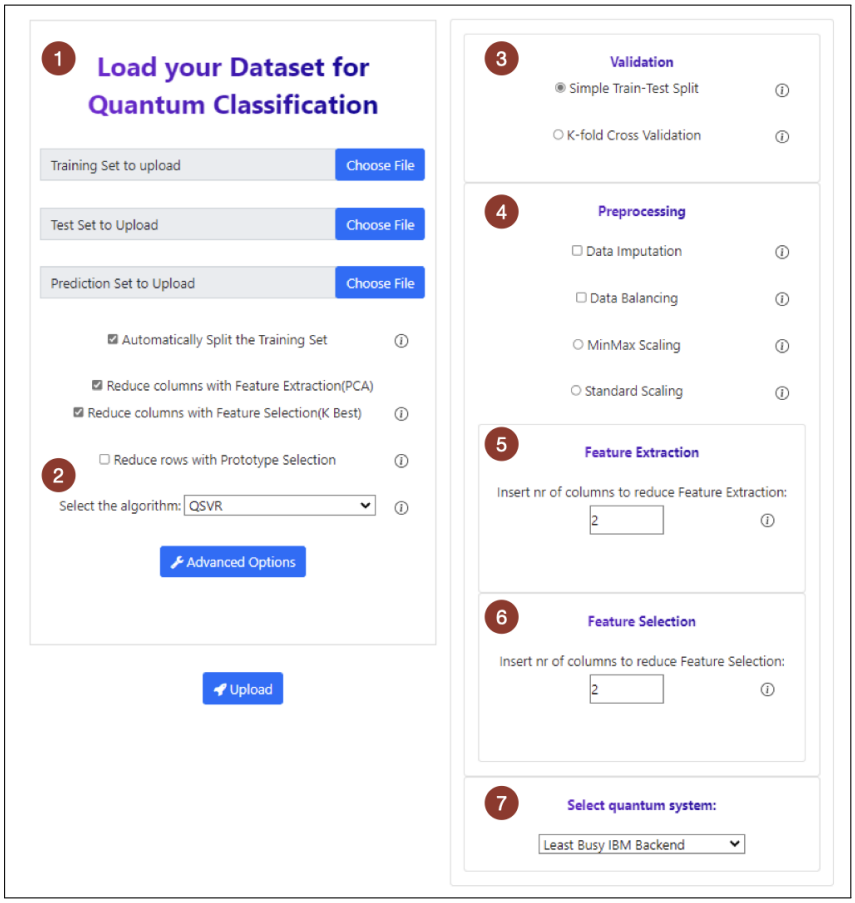

Figure 2: The GUI provided by QuantuMoonLight to run experiments.

steps, such as data balancing or standardization. The user can manage features by employing Principal Component Analysis [19] for feature extraction and choosing to rely on the k-best features [20] for prediction. In both cases, the user can select the number of features to work with. Finally, the user has to select the back-end, namely the characteristics of the quantum hardware.

After user confirmation, QuantuMoonLight will connect to the IBM services, start the execution, and inform the user via e-mail once the experiment is concluded. Once the results are available, the user can request the comparison with the performance of canonical machine learners through the dedicated *analysis* page—described in the online appendix [18].

4. Evaluation of the Tool

We preliminarily assessed QuantuMoonLight on two software engineering tasks, *i.e.*, predicting (1) code smells [21] and (2) flaky test [22]. The

former are suboptimal implementation choices applied by programmers, the latter are non-deterministic tests exhibiting both a passing and failing behavior. This focus is due to our expertise and willingness to experiment with such a new powerful technology in the context of our own research. Also, we assessed the usability of the tool [23].

Code Smells and Flaky Tests Prediction. As training data for prediction tasks, we used product metrics, *e.g.*, the number of lines of code and lack of cohesion. With respect to the dataset size, our experiment can be considered large-scale. Both datasets are indeed quite large when considering the typical studies conducted in the field of software engineering. As for the code smell prediction case, we used a dataset composed of 25,000 instances provided by Palomba et al. [21]; it focuses on the analysis of five large-scale software projects, taking into account a set of five code smell types of different granularity. As for the flaky test prediction case, we analyzed 9,785 test cases, of which 670 were flaky [24]: in literature, other datasets have similar sizes, e.g., in their seminal work, Bell *et al.* [25] considered datasets of 412 and 423 flaky tests, respectively.

For both tasks, we used QuantuMoonLight to prepare the datasets, configure the hyper-parameters, and optimize the pipeline according to our domain knowledge and the literature available on the matter. Specifically, all the choices conducted in the study were based on (1) the paper by Pecorelli *et al.* [26] for the code smell prediction study and (2) the paper by Pontillo *et al.* [24] for the flaky test prediction study. Finally, we employed 10-fold cross-validation and the *Quantum Support Vector Classifier*—detailed configurations and results are available in the online appendix [18].

We measured the performance using accuracy and training time. The evaluation metrics were chosen based on previous papers published in the field, which showed that these are among the recommended aspects to consider when assessing quantum machine learning solutions [27–29].

When considering the code smells prediction task, the results achieved were in line with those reported in previous work [30], yet we observed a reduction in the training time. As for the task of flakiness prediction, we noticed that the performance achieved was *lower* compared to what showed by existing approaches [5, 24], while the computational cost drastically decreased. While the accuracy obtained by quantum machine learning in the two use cases considered was relatively low, we believe that our results would be helpful to researchers to (1) initially estimate the potential of quantum machine learning when applied to the software engineering domain and (2) use the features made available by QuantuMoonLight to investigate other

Table 1: Iterative Usability Test.

#	Task Description
1	Perform a quantum regression task using the website.
2	Compare the quantum regression task with a canonical one.
3	Access the community blog and post the results of your tasks.

research problems. Hence, the tool may impact research since it can support researchers when experimenting with quantum machine learning, other than empirical comparisons with traditional machine learning solutions.

Usability Improvement. We evaluated the usability of the tool by applying *iterative usability testing* [23], thus implementing an iterative process to get continuous feedback from users and keep improving the user experience of the tool. We conducted the test with 12 students of the course *"Introduction of Machine Learning"* at the Jheronimus Academy of Data Science (The Netherlands); participants were voluntary. Students had to perform three tasks—described in Table 1—during each iteration, sharing feedback on the tool's usability. After each iteration, we interviewed participants to assess the tool's usability in terms of *learnability*, *efficiency*, and *satisfaction*. We improved the user interface of the tool according to the feedback received. Overall, we conducted three iterations before reaching saturation. During the first iteration, students identified several areas for improvement, including the size of graphical elements, which some users found too small, and the need for more guidance when interacting with the tool. Based on this feedback, we changed the size of the graphical elements and included info boxes to guide users through the tool's features. In the second iteration, we continued to collect students' feedback and made further changes to the user interface based on them. We found that users were still struggling to understand certain tool features, so we made additional changes to the website content and included more detailed feedback to guide users through those features, *e.g.*, parameters configuration. Finally, in the third iteration, we conducted a final round of user testing and found that students could use the tool more effectively and with fewer issues, thus reaching saturation.

5. Impact

Quantum technology is still far from being exploitable for everyday tasks. However, QuantuMoonLight can contribute in this regard, facilitating its adoption by researchers and practitioners.

Impact on Research. The increasing interest in the quantum machine learning efficiency field [29, 31] stimulates researchers to conduct more and more empirical studies to compare quantum-based solutions and classic ones. QuantuMoonLight was designed to be a low-code platform, allowing researchers to interact with quantum machine learning through a user interface that would allow them to experiment with multiple configurations of quantum machine learning algorithms and compare a number of classic solutions. In this sense, the tool is impactful as it eases the investigation of quantum machine learning capabilities. Perhaps more importantly, the community-inspired blog allows researchers to share lessons learned, experiences, and reflections that may increase awareness about quantum computing, driving the community to grow.

Impact on Practice. QuantuMoonLight opens a "window" on the quantum world that can allow practitioners—*e.g.*, data scientists and developers—to exploit such a technology to make decisions faster. Practitioners can indeed use the platform to verify the suitability of quantum computing solutions for a large plethora of tasks, *e.g.*, software engineering, and make informed decisions on their adoption in practice. At the same time, practitioners can read about successful experiences through the blog, learning how to configure quantum machine learning solutions.

6. Concluding Remarks

QuantuMoonLight is a web application to experiment with quantum machine learning design to be extremely usable and lower the entry barrier to quantum computing for most users. We evaluated the tool on two prediction tasks, *i.e.*, flaky tests and code smells prediction, other than its usability through iterative usability testing.

We aim at extending the set of implemented quantum algorithms with more advanced ones—*e.g.*, the *Variational Quantum Eigensolver (VQE)* and the *Quantum Boltzmann Machine (QBM)*. Furthermore, we plan to extend our application from an integration point of view, including solutions from other providers—*e.g.*, Microsoft Azure Quantum and Xanadu. Finally, the openly available implementation will allow researchers to use the tool in other contexts and/or different, wider experimentation on quantum machine learning.

Acknowledgements

We acknowledge the use of IBM Quantum services for this work. The points of view expressed are those of the authors and do not reflect the

official policy or position of IBM or the IBM Quantum team. The work is partially supported by the Swiss National Science Foundation (SNF Project No. PZ00P2_186090) and by the EMELIOT national research project (MUR Project No. 2020W3A5FY).

References

[1] M. Allamanis, E. T. Barr, P. Devanbu, C. Sutton, A survey of machine learning for big code and naturalness, ACM Computing Surveys (CSUR) 51 (2018) 1–37.

[2] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, A. Marino, Comparing and experimenting machine learning techniques for code smell detection, Empirical Software Engineering 21 (2016) 1143–1191.

[3] M. I. Azeem, F. Palomba, L. Shi, Q. Wang, Machine learning techniques for code smell detection: A systematic literature review and meta-analysis, Information and Software Technology 108 (2019) 115–138.

[4] S. Hosseini, B. Turhan, D. Gunarathna, A systematic literature review and meta-analysis on cross project defect prediction, IEEE Transactions on Software Engineering 45 (2017) 111–147.

[5] A. Alshammari, C. Morris, M. Hilton, J. Bell, Flakeflagger: Predicting flakiness without rerunning tests, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 1572–1584.

[6] G. Grano, F. Palomba, H. C. Gall, Lightweight assessment of test-case effectiveness using source-code-quality indicators, IEEE Transactions on Software Engineering 47 (2019) 758–774.

[7] P. Benioff, The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines, Journal of Statistical Physics 22 (1980) 563–591. doi:10.1007/BF01011339.

[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd, Quantum machine learning, Nature 549 (2017) 195–202.

[9] G. Acampora, A. Vitiello, Tssweb: a web tool for training set selection, in: 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, 2020, pp. 1–7.

[10] Y. Zhang, Q. Ni, Recent advances in quantum machine learning, Quantum Engineering 2 (2020) e34. doi:https://doi.org/10.1002/que2.34.

[11] IBM Quantum, https://quantum-computing.ibm.com/, 2021.

[12] M. Grossi, L. Crippa, A. Aita, G. Bartoli, V. Sammarco, E. Picca, N. Said, F. Tramonto, F. Mattei, A serverless cloud integration for quantum computing, 2021.

[13] F. Di Marcantonio, M. Incudini, D. Tezza, M. Grossi, Quask–quantum advantage seeker with kernels, arXiv preprint arXiv:2206.15284 (2022).

[14] H. Li, F. Khomh, M. Openja, et al., 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), IEEE, 2021, pp. 343–354.

[15] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, A. De Lucia, Software engineering for quantum programming: How far are we?, Journal of Systems and Software 190 (2022) 111326.

[16] A. J., A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'malley, D. Oyen, S. Pakin, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. J. Swart, J. G. Wendelberger, B. Yoon, R. Zamora, W. Zhu, S. Eidenbenz, A. Bärtschi, P. J. Coles, M. Vuffray, A. Y. Lokhov, Quantum algorithm implementations for beginners, ACM Transactions on Quantum Computing 3 (2022). doi:10.1145/3517340.

[17] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: Primal estimated sub-gradient solver for svm, Mathematical programming 127 (2011) 3–30.

[18] F. Amato, M. Cicalese, L. Contrasto, G. Cubicciotti, G. DÁmbola, A. La Marca, G. Pagano, F. Tomeo, G. Catolino, G. Giordano, S. Lambiase, V. Pontillo, G. Sellitto, F. Ferrucci, F. Palomba, QuantumMoonLight: A low-code platform to experiment with quantum machine learning — online appendix, 2022. doi:10.6084/m9.figshare.20108336.

[19] H. Abdi, L. J. Williams, Principal component analysis, Wiley Interdisciplinary Reviews: Computational Statistics 2 (2010) 433–459.

[20] E. R. Dougherty, J. Hua, C. Sima, Performance of feature selection methods, Current genomics 10 (2009) 365–374.

[21] F. Palomba, G. Bavota, M. D. Penta, F. Fasano, R. Oliveto, A. D. Lucia, On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation, Empirical Software Engineering 23 (2018) 1188–1221.

[22] Q. Luo, F. Hariri, L. Eloussi, D. Marinov, An empirical analysis of flaky tests, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 643–653.

[23] A. Genov, Iterative usability testing as continuous feedback: A control systems perspective, Journal of Usability Studies 1 (2005) 18–27.

[24] V. Pontillo, F. Palomba, F. Ferrucci, Static test flakiness prediction: How far can we go?, Empirical Software Engineering 27 (2022) 1–44.

[25] J. Bell, O. Legunsen, M. Hilton, L. Eloussi, T. Yung, D. Marinov, Deflaker: Automatically detecting flaky tests, in: Proceedings of the 40th International Conference on Software Engineering, 2018, pp. 433–444.

[26] F. Pecorelli, F. Palomba, D. Di Nucci, A. De Lucia, Comparing heuristic and machine learning approaches for metric-based code smell detection, in: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 93–104. doi:10.1109/ICPC.2019.00023.

[27] H. Gupta, H. Varshney, T. K. Sharma, N. Pachauri, O. P. Verma, Comparative performance analysis of quantum machine learning with deep learning for diabetes prediction, Complex & Intelligent Systems 8 (2022) 3073–3087.

[28] C. Havenstein, D. Thomas, S. Chandrasekaran, Comparisons of performance between quantum and classical machine learning, SMU Data Science Review 1 (2018) 11.

[29] R. D. M. Simões, P. Huber, N. Meier, N. Smailov, R. M. Füchslin, K. Stockinger, Experimental evaluation of quantum machine learning algorithms, IEEE Access (2023).

[30] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, A. De Lucia, Detecting code smells using machine learning techniques: are we there yet?, in: IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER 2018), IEEE, 2018, pp. 612–621.

[31] A. Zeguendry, Z. Jarir, M. Quafafou, Quantum machine learning: A review and case studies, Entropy 25 (2023) 287.

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1
C2	Permanent link to repository	`https://github.com/Robertales/QuantuMoonLight`
C3	Permanent link to Reproducible Capsule	NA
C4	Legal Code License	Common Development and Distribution License 1.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, IBM Quantum
C7	Compilation requirements, operating environments	$Python \geq 3.7$, ANACONDA $\geq$ 2021.11, MYSQL ≥ 7.0
C8	Link to developer documentation/manual	`https://github.com/Robertales/QuantuMoonLight`
C9	Support email for questions	`sesalab@unisa.it`

Table 2: Code metadata.

Current executable software version

Nr.	(Executable) software metadata description	Please fill in this column
S1	Current software version	v1
S2	Permanent link to executables of this version	`https://sesaquantumoonlight.ngrok.io/`
S3	Permanent link to Reproducible Capsule	NA
S4	Legal Software License	Common Development and Distribution License 1.0
S5	Computing platforms/Operating Systems	web-based application
S6	Installation requirements & dependencies	*Python* $\geq$ 3.7, ANACONDA $\geq$ 2021.11, MYSQL $\geq$ 7.0
S7	Link to user manual	`https://github.com/Robertales/QuantuMoonLight`
S8	Support email for questions	`sesalab@unisa.it`

Table 3: Software metadata.