Peel | Pile? Cross-Framework Portability of Quantum Software

Manuel Schönberger, Maja Franz
Technical University of Applied Sciences Regensburg, Germany
manuel.schoenberger@othr.de,
maja.franz@st.othr.de

Stefanie Scherzinger
Chair of Scalable Database Systems
University of Passau
Passau, Germany
stefanie.scherzinger@uni-passau.de

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg
Siemens AG, Corporate Research
wolfgang.mauerer@othr.de

***Abstract*—In recent years, various vendors have made quantum software frameworks available. Yet with vendor-specific frameworks, code portability seems at risk, especially in a field where hardware and software libraries have not yet reached a consolidated state, and even foundational aspects of the technologies are still in flux. Accordingly, the development of vendor-independent quantum programming languages and frameworks is often suggested. This follows the established architectural pattern of introducing additional levels of abstraction into software stacks, thereby *piling on* layers of abstraction. Yet software architecture also provides seemingly less abstract alternatives, namely to focus on hardware-specific formulations of problems that *peel off* unnecessary layers. In this article, we quantitatively and experimentally explore these strategic alternatives, and compare popular quantum frameworks from the software implementation perspective. We find that for several specific, yet generalisable problems, the mathematical formulation of the problem to be solved is not just sufficiently abstract and serves as precise description, but is likewise concrete enough to allow for deriving framework-specific implementations with little effort. Additionally, we argue, based on analysing dozens of existing quantum codes, that porting between frameworks is actually low-effort, since the quantum- and framework-specific portions are very manageable in terms of size, commonly in the order of mere hundreds of lines of code. Given the *current* state-of-the-art in quantum programming practice, this leads us to argue in favour of *peeling off* unnecessary abstraction levels.**

I. Introduction

In recent years, academia (*e.g.*, [1]) and vendors have made frameworks for developing quantum software available, such as IBM Qiskit [2], Pennylane [3], TensorFlow Quantum (TFQ) [4], and D-Wave Ocean [5]. Oftentimes, these vendors also provide access to quantum processing units (QPUs). Since the current implementations of quantum algorithms are inherently coupled to the framework used, the development of a vendor-independent quantum programming language as part of a hardware-independent quantum processing framework has been suggested [6]. A high-level quantum programming language may moreover be beneficial from a quantum software engineering perspective [7], [8]. Some quantum frameworks enabling abstraction, to an extent, are already available, like QC Ware's Quasar library [9] and the Atos Quantum Learning Machine [10]. Moreover, methods for automatically proposing suitable quantum hardware for specific problems, formulated in a vendor-independent language, have been proposed [6], [11]. The idea of making quantum software development hardware-independent and thus *piling on* new layers of abstraction, is particularly attractive, as both quantum hardware and software libraries are still evolving. More broadly, tools such as GitHub Copilot [12] successfully demonstrate how a higher degree of abstraction increases developer efficiency: In Copilot, an AI engine proposes entire lines of code automatically.

Since quantum software development is still at an early stage, the research community still lacks insights into the available systems and the specific properties that make them suitable for certain quantum algorithms. Much like QPUs themselves, these properties are evolving. Developing a sufficiently accurate and future-proof automated selection process of quantum hardware is therefore difficult to accomplish, at the current stage.

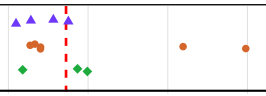
Moreover, the actual benefits of further abstraction levels in quantum software development are still unclear: they strongly depend on the effort required for migrating existing implementations onto another framework. This effort needs to outweigh both the expense of creating an additional abstraction layer and the abstraction effort regarding existing implementations. Otherwise, the prospects for a vendor-independent programming language will be limited, as demonstrated by the lack of adoption of the programming language Ada, once developed with similar aspirations [13].

We argue that we need to thoroughly understand the specific characteristics of the quantum frameworks. Previously, LaRose et al. [14] investigated four quantum frameworks and compared a variety of aspects, such as the available QPUs and library support. Moreover, a comparison framework for quantum frameworks was presented by Viez et al [15]. Yet we find the software implementation perspective underexplored.

We need to gain an understanding whether *piling on* new layers of abstraction is advisable, or whether we should rather focus our joint efforts on hardware-specific implementations and thus *peel off* unnecessary layers instead. Accordingly, we systematically compare four popular quantum frameworks [16]–[22]: (1) Qiskit, (2) Pennylane, (3) TFQ, which incorporates Google's Cirq, and (4) D-Wave Ocean.

MS, MF, and WM were supported by the German Federal Ministry of Education and Research (BMBF), funding program "quantum technologies—from basic research to market", grant number 13N15645. Our systematic search for open source quantum programming projects on GitHub using Google BigQuery, was supported by Google Cloud.

TABLE I
QUANTUM CODE VOLUME OF QUANTUM-CLASSICAL APPLICATIONS (FILES IMPORTING A QUANTUM FRAMEWORK LIBRARY ARE CONSIDERED QUANTUM CODE). DASHED RED LINE: CROSS-DOMAIN AVERAGE (434).

Domain	Lines of Code (LoC)	References
Optimisation		[21], [24]–[32]
ML		[16]–[18], [33]–[38]
Simulation		[39]–[42]

Contributions. Our contributions are as follows:

- We size up the quantum-specific code for several hybrid quantum-classical algorithms, all implemented by third parties. Our key insight is that the solutions studied require less than two thousand lines of code for encoding the quantum-specific parts. The quantum-specific code is thus small and manageable, comparable in volume to small personal programming projects.
- We compare the software development process given different development frameworks. We focus on two specific and rather novel application cases, namely reinforcement learning and multi-query optimisation in databases engines. While reformulating these problems to quantum algorithms is conceptually challenging, we again find that the actual implementation effort for all frameworks is very manageable, to the point of straightforward.
- We compare the portability of quantum software across frameworks. Specifically, we assess how strongly an implementation is coupled to the underlying framework. While the framework-specific implementation effort varies between applications, only small-scale code portions are involved in general. This renders cross-framework porting a task that involves little effort.

We then discuss the peel vs. pile trade-off given these results.

Structure. Chapter II introduces problem domains for which quantum computing is particularly promising, and investigates the code volume of the quantum-specific part of existing implementations. Chapter III studies cross-framework portability for two specific application use cases. Chapter IV concludes.

II. DIMENSIONS OF HYBRID QC APPLICATIONS

We size up the quantum-specific code for realistic, hybrid quantum-classical algorithms by considering applications where quantum computing promises speedups. The *Quantum Application and Technology Consortium* (QuTAC) comprises ten multi-national companies from different sectors, in particular automotive, chemistry, insurance, and technology. The consortium identifies relevant application domains [23]: optimisation, machine learning (ML), and simulation.

Table I references previous work and code repositories related to these problem domains. The material was collected by manually reviewing existing literature, but also by a keyword search on a current snapshot of open source projects on GitHub, using Google BigQuery. These and all further steps of our analysis are fully reproducible [43] using our provided reproduction package[1]. The table lists the total lines of code of all files within a project, where the files import some quantum framework library. As such, this is a generous over-approximation of the share of the quantum-specific code. Other possible metrics include code maintainability or readability. However, the former is hardly applicable for software with only several hundred lines of code, whereas the latter is hard to quantify.

Similarly to classical software, the lines of code metric (LoC) has been suggested as useful for evaluating the size of quantum software and the process [44] and development effort [45]. Other metrics correlate with lines of code [46], rendering them a suitable proxy metric. In the applications analysed, we also found that the number of commits containing changes to quantum-related code correlates with the LoC metric.

We observe that these numbers are small, which we attribute to the representation pattern for many of these problems, particularly optimisation problems: typically, they are reformulated as quadratic unconstrained binary optimisation (QUBO) problems [21], [24]–[27], to leverage existing implementations of quantum algorithms and solvers. Consequently, the bulk of development time is actually spent on finding suitable reformulations (rather than implementing complex control flow paths). This suggests that for these problems, the number of framework-specific and quantum-related implementation steps might be limited, since they mostly consist of calling existing quantum subroutines. Similar observations can be made for machine learning and simulation problems.

To verify or refute this indication, we analyse two specific and practically relevant problems representing two of the discussed domains in more detail. Specifically, we investigate implementation complexity and the quantum-specific steps.

III. CROSS-FRAMEWORK PORTABILITY

We critically evaluate and compare the gate-based frameworks Qiskit, Pennylane, TFQ, and the quantum annealing framework D-Wave Ocean, regarding the software implementation process. For our analysis, we choose two specific application scenarios that are subject to current research: (A) reinforcement learning [16]–[19] and (B) multi-query optimisation [20]. Due to the restriction to QUBO problems, we do not consider the Ocean framework for RL. More specifically, for each implementation we evaluate its size (in terms of lines of code) and its complexity with respect to the available documentation and library support. Moreover, we investigate the framework-specific and quantum-related implementation steps, which determine how strongly the implementations are tied to the quantum frameworks. Based on our findings, we discuss the benefits a new abstraction layer may provide.

A. Reinforcement Learning

Most reinforcement learning (RL) formulations centre around a Markov Decision Process (MDP) [48]: An *agent* interacts with an *environment* to maximise a cumulative reward $G_t =$

[1]Zenodo: https://doi.org/10.5281/zenodo.5898296

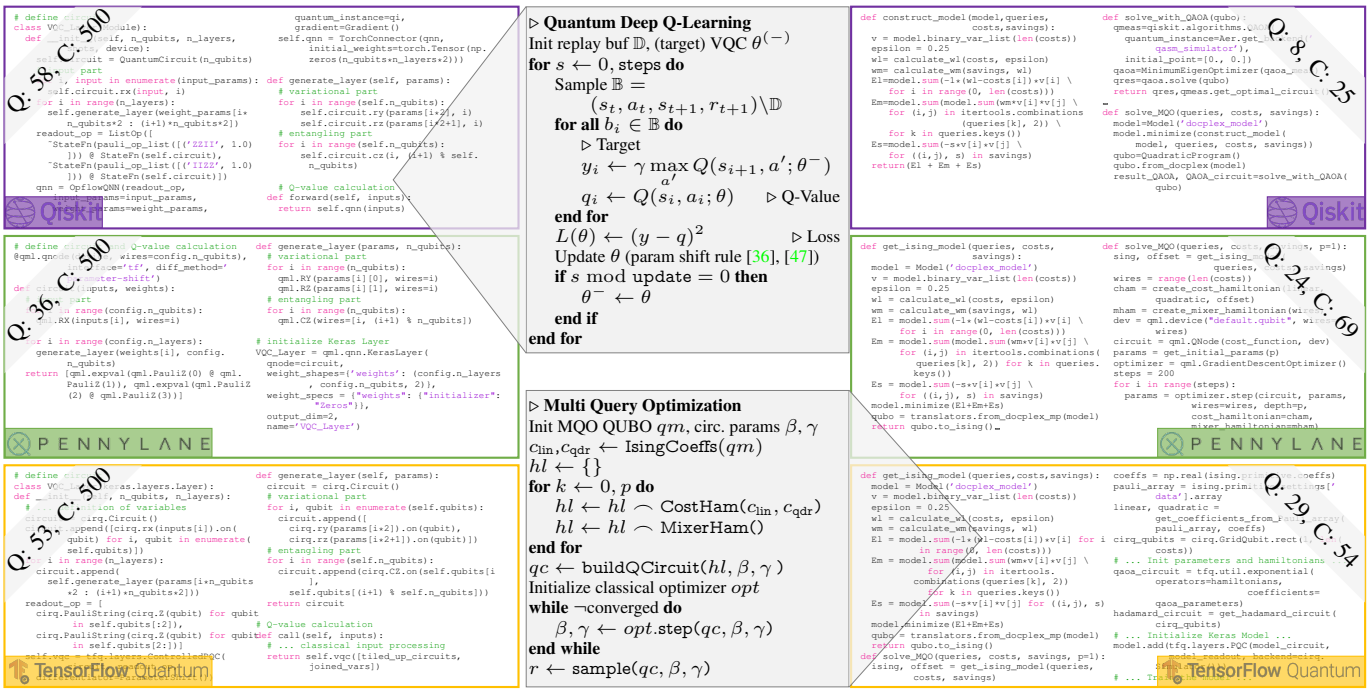


Fig. 1. (Best viewed in colour.) Pseudocode capturing the essential structure of quantum algorithms for reinforcement learning (top centre) and multi-query optimisation (bottom centre), and their concrete implementations in three frameworks (left: RL, right: MQO), together with approximate lines of code for quantum-specific components (Q: $\langle N \rangle$), and classical (C: $\langle N \rangle$) contributions (based on a manual classification by the authors). The python code, although functional, is not meant to be read, but merely gives a sense of scale. Regardless of the framework, the concrete implementations are close in size to the abstract pseudo-code representation, indicating that further abstraction layers or domain-specific quantum programming languages have very limited potential for additional reduction in size, and increase in expressivity. We only show classical code that is directly interrelated with quantum code components in the figure.

$\sum_{t'=t}^{T} \gamma^{t'} R_{t'}$ until a terminal state S_T is reached, with $R_{t'}$ being the reward at time step t' and a discount factor γ [49]. We focus on Deep Q-Learning [50], [51], where the idea is to learn the optimal *action-value function*, also referred to as *Q-function*: $Q_*(s,a) = \max_\pi \mathbb{E}\left[G_t | S_t = s, A_t = a, \pi\right]$. It represents the return, or accumulative reward G_t, expected when taking an action a in the environment's state s, then following a policy π in future states. An optimal policy π_* can be recovered by taking the action that maximises future *Q-values*: $\pi_*(s) = \arg\max_a Q_*(s,a)$. In classical Deep Q-Learning, this is achieved by training a neural network to satisfy the *Bellman Optimality Equation* [48] that relates the values of a state-action pair to the value of the next state:

$$Q_*(s,a) = \mathbb{E}\Big[R_t + \gamma \max_{a'} Q_*(S_{t+1}, a') \,|S_t = s, \quad A_t = a\Big]$$

In quantum-based RL, the neural network can be replaced by a *variational quantum circuit* (VQC) [16]–[18], [36], parameterised by weights θ. The algorithm sketched in Figure 1 (top centre) employs the Double Q-Learning approach as suggested in [52], which calculates targets with a *target network* or a *target VQC* in the quantum domain, parameterised by θ^-.

As a first step, we implemented classical RL with a neural network, using TensorFlow [53] and PyTorch [54]. We chose these frameworks because Pennylane offers an interface for both, TFQ builds upon TensorFlow, and Qiskit provides a machine learning library based on PyTorch. Since common quantum Q-Learning algorithms [16]–[18] do not examine annealing approaches, we only consider gate-based frameworks.

After confirming the correctness of our implementations, we replaced the classical neural network with a VQC based on standard framework patterns, in particular using python to represent a VQC for machine learning with trainable parameters. It essentially comprises three code elements: (1) the VQC definition, (2) calculation and processing of Q-values, and (3) calculation of gradients in a quantum-classical back-propagation procedure [36], [47]. All frameworks offer similar library classes for this purpose. In each case, we could easily isolate the quantum-based steps from the classical algorithm.

As we discuss in Sec. III-C, the framework-specific implementation steps are almost interchangeable[2]. An abstraction layer at this level could be beneficial, since TFQ is coupled with TensorFlow, and Qiskit with PyTorch. Hence, it could ease porting between ML and quantum frameworks.

B. Multi Query Optimisation

Multi query optimisation (MQO) is a longstanding problem in database research. It seeks to determine a globally optimal set of execution plans for a batch of database queries, minimising the overall execution cost by reusing common

[2]We did observe major differences in run-time: Calculating gradients for one batch takes 47 748 ms in Qiskit, 1 212 ms in Pennylane, and 659 ms in TFQ; the differences substantially impact practical utility (measurements are averaged over 100 batches and conducted on the same device.)

subexpressions [55]. The problem has been addressed on a D-Wave quantum annealer, based on a reformulation into a QUBO problem [20]. QUBO problems and their equivalent Ising formulations [56] can also be solved on gate-based QPUs with variational hybrid quantum-classical algorithms [57], such as the quantum approximate optimisation algorithm (QAOA) [58]. We can solve MQO problems on gate-based frameworks [59], [60] and therefore on all considered quantum frameworks using the reformulation approach presented in Ref. [20].

We again discuss the implementation procedure for each framework. The Ocean implementation for D-Wave is straightforward—it suffices to apply the QUBO reformulation proposed in [20]. Using framework-provided classes, we create a quadratic model that serves as input for all solvers. The Ocean implementation is the most compact among all frameworks.

For gate-based frameworks, we use the same reformulation approach to solve the problem with QAOA. The algorithm is sketched in Figure 1 (bottom centre). Here, we first searched the available documentation and libraries for artefacts related to QAOA. We found that Qiskit offers a library that fully encapsulates all QAOA steps. Much like the Ocean implementation, Qiskit requires only a limited number of steps: We use the IBM DOcplex tool [61] to apply the QUBO transformation using mathematical expressions. We then use classes and methods provided by Qiskit to create a quadratic model based on the DOcplex model. Finally, we determine an optimal solution for the quadratic problem using a provided optimiser, which transforms the QUBO to an Ising model and which we configure to internally use the available QAOA solver. No explicit QAOA circuit specification is necessary for Qiskit.

We did not find any comparable libraries for Pennylane and TFQ. However, for Pennylane, a library containing utility functions for QAOA (e.g., for applying the cost and mixer layers) exists, which simplifies the process of creating the QAOA circuit. In both cases, we create QAOA circuits by simply following demo code from the documentation.

The QAOA circuit consists of an alternating sequence of repeating cost and mixing operators, where the number of repetitions is given by a parameter p [58]. To create the cost Hamiltonian needed for the cost operators, we need the Ising coefficients of our problem formulation. Therefore, we base the Pennylane and TFQ implementations on our Qiskit implementation, which allows us to use available methods for converting QUBOs into an Ising model. As before, the framework-specific steps are few and simple: they consist of creating the operators and parameterised quantum circuits based on the Ising coefficients. The parameters are classically optimised in an outer loop (e.g., by gradient descent methods).

We found weak framework coupling for all MQO implementations. Ocean and Qiskit mostly require QUBO transformation, and resulting models serve as input for the solvers offered by frameworks. The implementations for Qiskit, Pennylane, and TFQ are near-identical up to determining the QUBO model or the respective Ising coefficients. The remaining framework-specific steps for Pennylane and TFQ are largely independent of the concrete problem, and moreover straightforward to implement through the use of existing libraries and available documentation. The effort for porting the MQO implementation across frameworks is minor, and an additional abstraction layer at implementation level provides little benefit.

C. Application Scenario Lineup

Figure 1 shows the quantum-relevant code for the use-cases in the gate-based frameworks. It allows for side-by-side comparison. In the centre, we show the abstract pseudo-code for RL and MQO. To the left and right, we show quantum-specific code for each framework (the python code is not meant to be readable, but merely to provide a sense of scale).

The implementations are of limited LoC size, and well below multi-million LoC typically considered in software engineering [62]. In fact, the quantum-specific python code ranges between just 8 and 58 lines of code.

Most importantly, they are comparable in size to the mathematical pseudo-code representation. This suggests limited potential for further abstraction. Porting between frameworks is mostly a direct substitution of APIs without structural code changes, indicating that the expressivity is essentially optimal.

IV. Discussion and Conclusion

Programming quantum computers is, at the current state of technology, often perceived as a very low-level task, comparable to programming early-generation classical machines. We have studied potentials and limitations for extending the state-of-the-art with higher-level abstractions and device-independent presentation of quantum algorithms using two means: (a) by learning from existing quantum programs, and (b) by implementing two advanced use-cases for multiple, popular quantum programming frameworks, and judging similarities across frameworks, with an abstract pseudo-code representation. For RL, we isolated the quantum-specific implementation details from the classical algorithm. For MQO, we used a QUBO reformulation applicable for all frameworks.

In all cases, quantum-specific portions are small, and the level of abstraction is not much different between pseudo-code and all frameworks. We see no reason to assume much difference between the considered problems and others of similar size and problem domains, in this regard. All scenarios are orders of magnitude away from problem sizes considered challenging in software architecture and engineering practice and research.

Our findings suggest that in general, introducing new abstraction layers by crafting framework-independent programming languages holds limited promise. Still, our selected application cases represent problem domains which are considered promising candidates for quantum speedups. Other problems of these domains are usually solved with similar patterns and paradigms. For instance, optimisation problems are typically reformulated to leverage established quantum algorithms. This might change once new quantum algorithms and paradigms are discovered. However, progress related to quantum algorithms has been moderate—for instance, key algorithms like Grover search [63] have been known for more than two decades. Therefore, the

familiar quantum patterns and paradigms are likely to persist for the foreseeable future.

Ultimately, when deciding between piling on new abstraction layers or peeling off existing ones, our results suggest the latter.

References

[1] W. Mauerer, *Semantics and simulation of communication in quantum programming*, 2005. arXiv: quant - ph / 0511145 [quant-ph]. [Online]. Available: https://arxiv.org/abs/quant-ph/0511145.

[2] IBM, *Qiskit: An open-source framework for quantum computing*, 2021. [Online]. Available: https://qiskit.org/.

[3] V. Bergholm, J. Izaac, M. Schuld, *et al.*, *Pennylane: Automatic differentiation of hybrid quantum-classical computations*, 2020. arXiv: 1811.04968 [quant-ph]. [Online]. Available: https://arxiv.org/abs/1811.04968.

[4] M. Broughton, G. Verdon, T. McCourt, *et al.*, "Tensorflow quantum: A software framework for quantum machine learning," arXiv: 2003.02989 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2003.02989.

[5] D-Wave Systems Inc, *Documentation for the Ocean SDK for solving problems on D-Wave quantum computers*, 2021. [Online]. Available: https://docs.ocean.dwavesys.com/en/stable/.

[6] Frank Leymann, Johanna Barzen, Michael Falkenthal, *et al.*, "Quantum in the cloud: Application potentials and research opportunities," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, 2020, pp. 9–24. DOI: 10.5220/0009819800090024.

[7] J. Zhao, "Quantum software engineering: Landscapes and horizons," 2020. arXiv: 2007.07047 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2007.07047.

[8] T. Krüger and W. Mauerer, "Quantum annealing-based software components: An experimental case study with sat solving," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 445–450, ISBN: 9781450379632. DOI: 10.1145/3387940.3391472. [Online]. Available: https://doi.org/10.1145/3387940.3391472.

[9] QC Ware, *Quasar library for creating and evaluating quantum computing circuits*, 2022. [Online]. Available: https://qcware-quasar.readthedocs.io/en/latest/.

[10] Atos, *Quantum learning machine*, 2020. [Online]. Available: https://atos.net/wp-content/uploads/2020/11/Quantum-Learning-Machine-Brochure.pdf.

[11] B. Weder, J. Barzen, F. Leymann, *et al.*, "Automated quantum hardware selection for quantum workflows," *Electronics*, vol. 10, no. 8, p. 984, 2021. DOI: 10.3390/electronics10080984.

[12] A. Ziegler, *A first look at rote learning in GitHub Copilot suggestions*, 2021. [Online]. Available: https://docs.github.com/en/github/copilot/research-recitation.

[13] J. D. Ichbiah, B. Krieg-Brueckner, B. A. Wichmann, *et al.*, "Rationale for the design of the Ada programming language," *ACM Sigplan notices*, vol. 14, no. 6b, pp. 1–261, 1979.

[14] R. LaRose, "Overview and comparison of gate level quantum software platforms," *Quantum*, vol. 3, p. 130, 2019. DOI: 10.22331/q-2019-03-25-130.

[15] D. Vietz, J. Barzen, F. Leymann, *et al.*, "On decision support for quantum application developers: Categorization, comparison, and analysis of existing technologies," in *Proceedings ICCS 2021*, 2021, pp. 127–141. DOI: 10.1007/978-3-030-77980-1_10.

[16] Owen Lockwood and Mei Si, "Reinforcement learning with quantum variational circuit," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, pp. 245–251, 2020, ISSN: 2334-0924.

[17] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, *et al.*, "Variational quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141 007–141 024, 2020. DOI: 10.1109/ACCESS.2020.3010470.

[18] A. Skolik, S. Jerbi, and V. Dunjko, *Quantum agents in the gym: A variational quantum algorithm for deep q-learning*. arXiv: 2103.15084 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2103.15084.

[19] M. Franz, L. Wolf, M. Periyasamy, *et al.*, "Uncovering instabilities in variational-quantum deep q-networks," 2022. arXiv: 2202.05195 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2202.05195.

[20] I. Trummer and C. Koch, "Multiple query optimization on the D-Wave 2X adiabatic quantum computer," *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 648–659, 2016. DOI: 10.14778/2947618.2947621.

[21] S. Feld, C. Roch, T. Gabor, *et al.*, "A hybrid solution method for the capacitated vehicle routing problem using a quantum annealer," *Frontiers in ICT*, vol. 6, p. 13, 2019. DOI: 10.3389/fict.2019.00013.

[22] S. Khairy, R. Shaydulin, L. Cincio, *et al.*, "Reinforcement-learning-based variational quantum circuits optimization for combinatorial problems," arXiv: 1911.04574 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1911.04574.

[23] A. Bayerstadler, G. Becquin, J. Binder, *et al.*, "Industry quantum computing applications," *EPJ Quantum Technology*, vol. 8, no. 1, p. 25, 2021. DOI: 10.1140/epjqt/s40507-021-00114-x.

[24] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, p. 5, 2014. DOI: 10.3389/fphy.2014.00005.

[25] R. Martonák, G. E. Santoro, and E. Tosatti, "Quantum annealing of the traveling-salesman problem," *Physical Review E*, vol. 70, no. 5, p. 057 701, 2004. DOI: 10.1103/PhysRevE.70.057701.

[26] T. Hogg, "Adiabatic quantum computing for random satisfiability problems," *Physical Review A*, vol. 67, no. 2, p. 022 314, 2003. DOI: 10.1103/PhysRevA.67.022314.

[27] T. Gabor, S. Zielinski, S. Feld, *et al.*, "Assessing solution quality of 3SAT on a quantum annealing platform," in *Quantum Technology and Optimization Problems*, S. Feld and C. Linnhoff-Popien, Eds., Springer International Publishing, 2019, pp. 23–35, ISBN: 978-3-030-14082-3. DOI: 10.1007/978-3-030-14082-3_3.

[28] P. Pierre-Louis, H. Tong, and J. McFarland, *Portfolio optimization*, 2021. [Online]. Available: https://github.com/dwave-examples/portfolio-optimization.

[29] T. Krüger and W. Mauerer, "Quantum annealing-based software components: An experimental case study with SAT solving," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 445–450. DOI: 10.1145/3387940.3391472.

[30] I. Sax, S. Feld, S. Zielinski, *et al.*, "Approximate approximation on a quantum annealer," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 108–117. DOI: 10.1145/3387902.3392635.

[31] V. Goliber, H. Tong, and R. Stevanovic, *Antennas selection*, 2021. [Online]. Available: https://github.com/dwave-examples/antenna-selection.

[32] ——, *Maximum cut*, 2021. [Online]. Available: https://github.com/dwave-examples/maximum-cut.

[33] Owen Lockwood and Mei Si, *Reinforcement learning with quantum variational circuit*, 2020. [Online]. Available: https://github.com/lockwo/quantum_computation.

[34] A. Skolik, S. Jerbi, and V. Dunjko, *Quantum agents in the gym: A variational quantum algorithm for deep q-learning*. [Online]. Available: https://github.com/askolik/quantum_agents.

[35] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, *et al.*, *Variational quantum circuits for deep reinforcement learning*, 2021. [Online]. Available: https://github.com/ycchen1989/Var-QuantumCircuits-DeepRL.

[36] K. Mitarai, M. Negoro, M. Kitagawa, *et al.*, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, p. 032 309, 2018.

[37] G. Hellstern, "Analysis of a hybrid quantum network for classification tasks," *IET Quantum Communication*, 2021. DOI: 10.1049/qtc2.12017.

[38] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," 2018. arXiv: 1802.06002 [quant-ph]. [Online]. Available: https://arxiv.org/abs/1802.06002.

[39] S. Barison, D. E. Galli, and M. Motta, "Quantum simulations of molecular systems with intrinsic atomic orbitals," 2020. arXiv: 2011.08137 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2011.08137.

[40] ——, "Quantum simulations of molecular systems with intrinsic atomic orbitals," 2020. [Online]. Available: https://github.com/StefanoBarison/quantum_simulation_with_IAO.

[41] J. Copenhaver, A. Wasserman, and B. Wehefritz-Kaufmann, "Using quantum annealers to calculate ground state properties of molecules," 2021. [Online]. Available: https://github.com/jcopenh/Quantum-Chemistry-with-Annealers.

[42] ——, "Using quantum annealers to calculate ground state properties of molecules," *The Journal of Chemical Physics*, vol. 154, no. 3, p. 034 105, 2021. DOI: 10.1063/5.0030397.

[43] W. Mauerer and S. Scherzinger, *1-2-3 reproducibility for quantum software experiments*, 2022. arXiv: 2201.12031 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2201.12031.

[44] R. Ramsauer, D. Lohmann, and W. Mauerer, "The list is the process: Reliable pre-integration tracking of commits on mailing lists," in *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*, (Montreal, QC, Canada), May 2019, pp. 807–818. DOI: 10.1109/ICSE.2019.00088.

[45] Jianjun Zhao, "Some size and structure metrics for quantum software," in *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, 2021, pp. 22–27. DOI: 10.1109/Q-SE52541.2021.00012.

[46] M. A. A. Mamun, C. Berger, and J. Hansson, "Correlations of software code metrics: An empirical study," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, ACM, 2017, pp. 255–266. DOI: 10.1145/3143434.3143445.

[47] M. Schuld, V. Bergholm, C. Gogolin, *et al.*, "Evaluating analytic gradients on quantum hardware," *Physical Review A*, vol. 99, no. 3, p. 032 331, 2019.

[48] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 4 1957, ISSN: 0022-2518.

[49] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 2018, ISBN: 0262039249.

[50] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing Atari with deep reinforcement learning," 2013. arXiv: 1312.5602 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1312.5602.

[52] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with Double Q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[53] T. Developers, *Tensorflow*, version v2.6.2, Nov. 2021. DOI: 10.5281/zenodo.5645375. [Online]. Available: https://doi.org/10.5281/zenodo.5645375.

[54] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, *et al.*, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[55] P. Roy and S. Sudarshan, "Multi-query optimization," in *Encyclopedia of Database Systems, Second Edition*, L. Liu and M. T. Özsu, Eds., Springer, 2018. DOI: 10.1007/978-1-4614-8265-9_239. [Online]. Available: https://doi.org/10.1007/978-1-4614-8265-9_239.

[56] Z. Bian, F. Chudak, W. Macready, *et al.*, "The Ising model: Teaching an old problem new tricks," D-Wave Systems Inc, Tech. Rep., 2010.

[57] J. R. McClean, J. Romero, R. Babbush, *et al.*, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023 023, Feb. 2016. DOI: 10.1088/1367-2630/18/2/023023.

[58] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014. arXiv: 1411.4028 [quant-ph]. [Online]. Available: https://arxiv.org/abs/1411.4028.

[59] M. Schönberger, "Applicability of quantum computing on database query optimization," unpublished, M.S. thesis, Technical University of Applied Sciences Regensburg, 2021.

[60] T. Fankhauser, M. E. Solèr, R. M. Füchslin, *et al.*, "Multiple query optimization using a hybrid approach of classical and quantum computing," 2021. arXiv: 2107.10508 [cs.DB]. [Online]. Available: https://arxiv.org/abs/2107.10508.

[61] IBM, *IBM decision optimization CPLEX modeling for python*, 2021. [Online]. Available: https://ibmdecisionoptimization.github.io/docplex-doc/.

[62] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd. Addison-Wesley Professional, 2012, ISBN: 0321815734.

[63] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866.