Integration and Evaluation of Quantum Accelerators for Data-Driven User Functions

Thomas Hubregtsen [1,2], Christoph Segler [1,3], Josef Pichlmeier [2],
Aritra Sarkar [2], Thomas Gabor [4], and Koen Bertels [2]
[1] BMW Group Research, New Technologies, Innovations, Garching bei München, Germany
[2] Delft University of Technology, Delft, Netherlands
[3] Technical University of Munich, Department of Informatics, Garching bei München, Germany
[4] LMU Munich, München, Germany

Abstract— Quantum computers hold great promise for accelerating computationally challenging algorithms on noisy intermediate-scale quantum (NISQ) devices in the upcoming years. Much attention of the current research is directed to algorithmic research on artificial data that is disconnected from live systems, such as optimization of systems or training of learning algorithms. In this paper we investigate the integration of quantum systems into industry-grade system architectures. In this work we propose a system architecture for the integration of quantum accelerators. In order to evaluate our proposed system architecture we implemented various algorithms including a classical system, a gate-based quantum accelerator and a quantum annealer. This algorithm automates user habits using data-driven functions trained on real-world data. This also includes an evaluation of the quantum enhanced kernel, that previously was only evaluated on artificial data. In our evaluation, we showed that the quantum-enhanced kernel performs at least equally well to a classical state-of-the-art kernel. We also showed a low reduction in accuracy and latency numbers within acceptable bounds when running on the gate-based IBM quantum accelerator. We, therefore, conclude it is feasible to integrate NISQ-era devices in industry-grade system architecture in preparation for future hardware improvements.

Keywords— Quantum machine learning, quantum architectures, quantum devices, quantum modelling, hybrid quantum/classical algorithms

I. Introduction

The industry today is facing major problems that are being solved by digital solutions. Optimization problems increase efficiency, data-driven functionalities provide forecasts and personalization; simulations provide a way to explore new compounds in numbers previously unseen. Still, a major holdback is in the amount of data that can be processed. Just as with the advance of deep neural network through the acceleration of GPUs, we are about to enter a new era of computation, being the era of quantum computation. Some problems that were previously thought to be inefficient in the general case, due to their computational complexity being labeled as non-deterministic polynomial-time (NP), are now becoming amenable to the quantum-efficient class called bounded quantum polynomial-time (BQP). Furthermore, extensive research is being performed into the hypothesis that quantum systems can represent probability distributions that can not be matched efficiently on classical systems [1]. Algorithms already exist, such as the prime-factorization [2] that threatens current-day security, and real-world experimental quantum systems are now accessible through the cloud.

In this paper, a system architecture for the integration of quantum systems in industry-grade environments is proposed. The components and considerations are investigated. A working proof of concept is built to evaluate the performance in terms of timing and accuracy. For this, we implement a data-driven function, which is a function that can be trained to make predictions. We also evaluate the quantum-enhanced kernel proposed by Havlicek et al. [3], previously only tested on artificial data on real-world data.

For this work, the following research questions are posed:

RQ1 How to integrate quantum systems into an industry-grade system architecture?
RQ2 How do current-day state-of-the-art quantum accelerators perform when integrated in the production environment of real-world industry data-center?
RQ3 How does the quantum-enhanced kernel perform on real-world data?

This paper provides the following scientific contributions but also contributes to the state of practice:

(i) Proposed system architecture for integration of various quantum systems.
(ii) Evaluation of quantum accelerators, both from a software and hardware perspective, in a real-world scenario.
(iii) Evaluation of the quantum-enhanced kernel on real-world data.

The remainder of this paper is structured as follows: Section II summarizes related work followed by the primary approach of this work in section III. Section IV presents the evaluation and section V the results obtained, followed by their discussion in section VI. Finally, the paper in concluded in VII.

II. Related work

We split the related work into the areas of industry use-cases and full-stack implementations, including system architectures that are evaluated.

A. Industry use-cases

The introduction of the D-Wave System's Quantum Annealer has inspired many scientists to implement real-world use-cases on an experimental quantum device [4–6]. For instance, the applicability of the D-Wave 2X for traffic flow optimization [7] based on real-world data has been investigated. In their work, the underlying optimization problem was defined, such that the lowest energy solution minimizes the number of cars that take the same route. Nevertheless, the experiment was performed on a small subset of the original problem. Similarly, the D-Wave 2000Q has been used to approximate solutions for the flight gate assignment problem [8]. An optimal solution of the derived Quadratic Unconstrained Binary Optimization (QUBO) problem would minimize the transit time for passengers at an airport. However, due to the limited problem size that can be processed with the D-Wave Quantum Annealer, only special subsets of the original problem have been used for the test. These subsets were derived during a pre-processing step.

In contrast to the listed work, which takes a large problem and cut out a subspace, we have selected a use-case that fits the current state-of-the-art quantum hardware, as well as applied proven methods for reducing the number of calls, such as feature selection and cross-compilation. Furthermore, a system architecture is proposed, and metrics relevant to the integration of this hardware are reviewed, such as latency and queuing times.

B. Quantum computing stack

To use a quantum processing unit (QPU) for a particular application, there are many encapsulating layers that bridge the physical platform with the algorithmic logic [9]. These include expressing the application as an algorithm [10] consisting of both classical and quantum kernels. Typically the QPU is accessed as an accelerator to the host CPU, with specific logic offloaded when it can harness quantum phenomena (e.g., superposition, entanglement, interference) to a computational advantage. The quantum kernels in high-level logic are translated to assembly-level instructions for a gate-based quantum computer by the compiler [11]. This can be directly executed on a classical quantum-circuit simulator for functional verification [12]. For targeting real QPU specifications (e.g., supported gateset, qubit connectivity, fidelity), further hardware constraints like gate decomposition, scheduling, mapping, routing, and error-correction coding are applied on the logic. The resultant QPU-dependent assembly [13] is then passed to the micro-architecture layer [14, 15]. This takes care of the precise instruction issue timings to the analog waveform generators connected to the qubit control lines of the QPU chip.

Other research groups [16–18] have echoed this view of pursuing a system's view in the quantum hardware-software co-design for scaling up and encapsulating the QPU in a programmer agnostic accelerator framework. Such a full-stack approach is crucial for large-scale simulators [19] as well as real quantum accelerators based on annealers [20] and gate-based models [21].

III. Approach

An overview of the proposed system architecture is depicted in **Figure 1**. The architecture consists out of four major components: (i) The vehicle, (ii) the backend of the original equipment manufacturer (OEM) (i.e., the car maker), (iii) the hardware accelerator for the training of the data-driven function (cf. green boxes), and (iv) the hardware accelerator for the deployment of the data-driven function (cf. red boxes). At first, the vehicle provides the data which is required to develop the data-driven function. The data from the vehicle is sent to the OEM's backend infrastructure, where the vehicle's data is received, preprocessed, and stored. For the training of the data-driven function, the training features are selected by feature selection and then only the selected subset is sent to the hardware accelerator for the training of the data-driven function provided by a cloud service. The type of the accelerator can vary between different hardware designs, and in our case also including quantum accelerators. The developed data-driven function can then either be deployed on accelerating hardware in the backend or onboard of the vehicle. In the following section, each component is further described:

A. Vehicle

In today's vehicles, most of the functions are realized as a mechatronic system that is executed on an electrical control unit (ECU). Current vehicles are equipped with up to 70 ECUs [22]. These ECUs are each connected to sensors and actuators in order to measure environmental as well as internal technical information and interact with their environment. Each of the ECUs provides their data to a gateway distributing the data inside the vehicle or sending the information to the backend of the OEM.

B. OEM backend

The vehicle data arrives through an application programming interface (API) in the OEM's backend infrastructure. In the first step the data pre-processing, the data is cleaned (removal of invalid data, augmentation of context) and transformed (resampling, normalization). Thereafter, the data is stored within the backend. To resolve the "curse of dimensionality" [23], posed by the high amount/dimensionality of vehicle signals, a feature selection component selects a subset of the most descriptive features/signals for the specific data-driven function. In our case, a supervised filter feature selection algorithm is applied in order to identify this subset of features. These feature selection algorithms calculate a score for each feature and rank them according to their score. Only the features showing the highest scores are then used for the training of the data-driven function in the cloud service. This is ideal for quantum computation, as current state-of-the-art systems have a limited number of input features that can be processed.

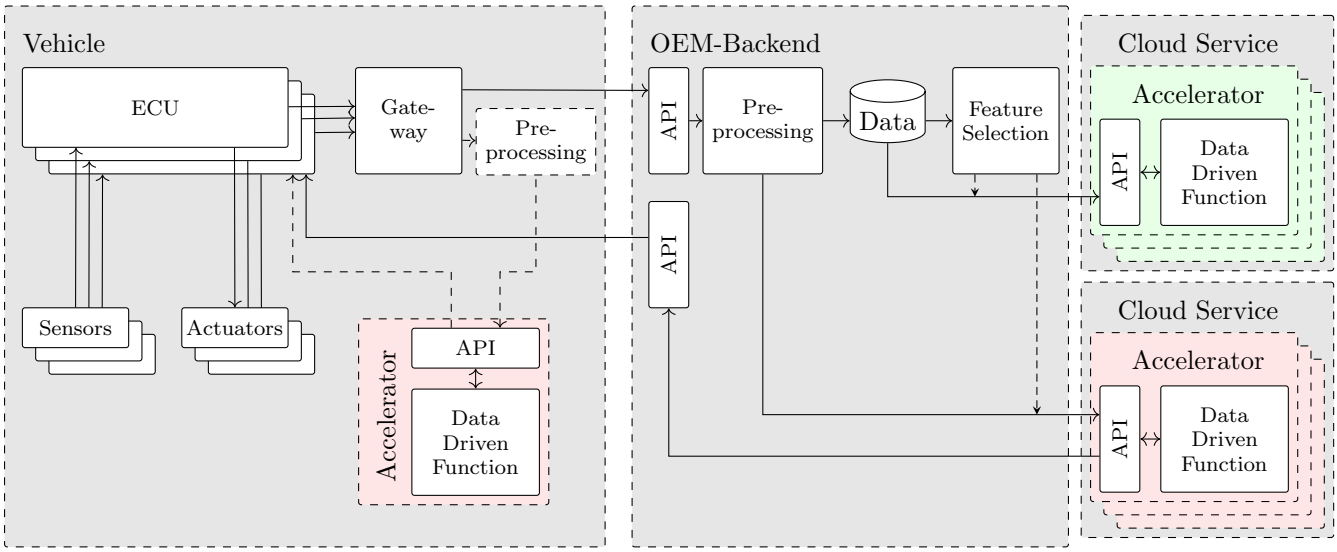

Fig. 1
Proposed system architecture

C. Cloud service

Data arrives in the accelerating cloud service at an API on a classical node. It is then forwarded to the accelerating hardware, where a data-driven function (often referred to as machine learning function) is trained on the pre-selected features and collected data. Each type of accelerating hardware has specific characteristics but often works together with a host CPU in a hybrid configuration. Typical hardware accelerators are GPUs, FPGAs, and ASICs. Roughly speaking, the GPU can perform many simple computations in parallel. The functions are given while the parameters can be changed. The FPGA can be reprogrammed to calculate for more complex functions, such as accelerating square roots efficiently. In this case, the developer defines both the function and the parameters. These functions can then be put together onto an ASIC. The ASIC does not allow changes to the function but can be further optimized both for performance and cost of mass production. In this case, only the parameters can be changed.

Similarly, one can incorporate a quantum accelerator with the accompanying software. One possible quantum accelerator which is currently heavily explored by industry is the special-purpose quantum annealer. This system allows for the acceleration of QUBO problems by providing a hardware platform and program, where the user can modify certain parameters. Alternative quantum accelerators are based on gate-based quantum systems. These are less mature technologies, but more generic and hold more promise. These systems provide the hardware where the user can define both the program and the parameters along with the offered functionality of the hardware. Examples (in various stages of maturity) are the superconducting systems (e.g., Google, IBM [24]), trapped-ion systems (e.g., IonQ [25]), photonic systems (e.g., Xanadu [26]) and topological systems (e.g., Microsoft [24]). A mixture of quantum ideologies implemented on classical hardware also exist, in the form of quantum-inspired classical systems (Fujitsu [27], Hitachi [28]). It is currently an open question if these systems provide an advantage over conventional classical systems.

After training, the model can be deployed. The deployment of the model for prediction would typically happen on a similar type of hardware accelerator, but one could consider functions that share a common mathematical formula that shares the weight to cross deploy on different hardware accelerators or simulators. For example, one could use a hybrid quantum-classical system to train the weights of a function, and deploy it on a classical system. In the case of the classical system, the prediction model could also be deployed directly in the vehicle architecture. When the data-driven function is deployed within the vehicle, the data pre-processing must also be performed onboard the vehicle (cf. **Figure 1**).

IV. Evaluation

A. Test case

As an exemplary test case for the evaluation, the prediction of an user preference settings is selected, being seat-heating. The current state of the driver's seat-heating considering the states *on* and *off* is observed. The data-driven function is then trained on the historical data collected from the user in order to predict the seat-heating state. By this, a proactive seat-heating personalized on the user's behavior can be achieved.

B. Data

Using a single car, 79 drives worth of raw data has been collected. The raw data has a sampling rate of 5 seconds from the start till the end of every drive. In total, 116 features and 20458 samples were collected. These are augmented with 5 extra features, relating to the time of day and time elapsed since the start of the drive. Every drive

is additionally labeled with a majority vote primarily seat heating on or off. Out of the 79 drives, 52 drives were performed with the seat heating off, 27 drives with the seat heating on.

C. Feature selection

In order to select the most relevant features as input for the data-driven function, the supervised filter feature selection algorithm *Fisher Score* [29] is run on the data. As a result the two most descriptive features are identified: the current outside temperature and the current temperature of the evaporator. Here, the evaporator is a part of the vehicle's Heating Ventilation Air Conditioning (HVAC).

D. Data-driven function

To predict the label for the chosen use-case, seat-heating, various functions are employed. The correlations in our data set allow for computationally less complex systems, such as decision trees, support vector machines (SVMs), and nearest neighbor. For our study, a support vector machine is deployed to find the best fit. A support vector machine takes the input data and maps it into a higher dimensional feature space, typically through a non-linear transformation called kernel function. Within this space, a linear separating hyperplane can be constructed to perform the classification of previously unseen data.

SVM are well-researched for classical hardware and are currently being explored for quantum accelerators. We implement this on a classical CPU, a hybrid quantum/classical system involving a gate-based system from the IBM cloud [30], and a hybrid quantum/classical system involving a specific-purpose quantum annealer with 2000 qubits developed by D-Wave Systems [31]. All classical calculation is performed on a single core of an Intel Xeon Gold 5222 CPU at 3.8 GHz.

D.1 SVM on a classical CPU

We implement a SVM on a classical CPU. This is programmed in python and uses the package Scikit-learn [32]. In particular, the fit and predict methods of *sklearn.svm.SVC* with gamma set to auto are used. The following kernels are evaluated: linear, polynomial, radial basis function (RBF), and sigmoid. The model with the best performing kernel is used as a baseline model.

D.2 Gate-based quantum system

To accelerate the SVM quantum mechanically, various feature maps can be used. These are realized through specific state preparation circuits [33]. Various circuits already exist, we choose the one proposed by Havlicek et al. [3] together with their optimization method, as this entails a non-linear mapping from the classical data to the Hilbert space along with a distribution that can currently not be replicated in poly-time classically. The state preparation circuit does not require any optimization of its parameters. A similar approach on a different quantum system using a different encoding was also proposed by Schuld and Killoran [33]. They used a continuous-variable system that uses a feature map based on squeezed vacuum states.

The proposed system works as follows: The prediction is performed by a parameterized quantum circuit, also called a variational quantum circuit. The task of the quantum circuit is to find an optimal cutting hyperplane in a multi-dimensional Hilbert space. The training of the SVM consists of finding the right parameters for the quantum circuit. This is done in a hybrid quantum/classical approach. The quantum circuit, initialized with default values, is used to predict labels on the training data. Based on the resulting probability distribution and the known labels, a cost value is computed, and fed into the classical optimizer for proposing new parameters for the quantum circuit. This is repeated till convergence is achieved, or a maximum iteration limit is reached.

Predicting a label for a set of input values entails preparing the quantum state and running the quantum circuit with the parameters found by the optimizer. This is repeated various times, as the measurement of a quantum circuit collapses the quantum state, resulting in a probabilistic outcome. The appropriate label is assigned to the value with the highest probability.

The implementation of the circuit is available through the IBM qiskit library in a class called vqc [30]. Both the training and the classification can independently be run on either a classical system using the IBM QASM simulator or on a quantum circuit in the IBM backend through a subscription service. A classical system with the IBM QASM simulator is used for training and the freely available subscription service for the quantum system in the IBM cloud is used for prediction [34]. The system with the smallest queue is selected, as our quantum circuit only requires two qubits [3].

D.3 Specific-purpose quantum annealer

For the specific-purpose quantum annealer, we use the quantum accelerator built by D-Wave Systems, which specializes in approximating optimization problems which are given as an instance of a QUBO.

In order to accelerate the construction of an SVM the problem needs to be translated into a QUBO, which was done according to the approach shown by Willsch et al. [35]. The constructed QUBO can be written as a quadratic matrix of size $n \cdot k \times n \cdot k$ where n is the amount of training data samples the SVM is based on and k is the precision used for the discretization of the (originally real-valued) coefficients that define the SVM's decision boundary.

V. Results

All evaluation runs are performed three times. The data was split into 80% training data, 10% test data, and 10% validation data by splitting data based on the individual drives. It was made sure no samples of the same drive is present in another data set. Hyper-parameters were trained on the training data and tested on the test data.

TABLE I
EVALUATION RESULTS — TIMING AND ACCURACY OF VARIOUS METHODS AND HARDWARE ACCELERATORS

Setup		**Training**		**Validation**		
Method	*Kernel*	*Hardware*	*Time*	*Hardware*	*Time*	*Accuracy*
Classical	Rbf	Classical	74 ±2 ms	Classical	485 ±5 ms	93.2±0.0%
Gate-based	Q. enhanced	Simulator	267 ±0 m	Simulator	52 ±0 s	93.1±0.1%
Gate-based	Q. enhanced	Simulator	267 ±0 m	Quantum	218 ±65 m	91.2±0.7%

Full experiments were performed by combining the training and test data and validating on the validation set.

The models for the full experiment were trained using 10% of the train and test set. This resulted in 1813 samples, 1057 samples of which had the seat heating off, and 756 samples had the seat heating on. The models were verified on 100% of the validation data, being 2327 samples. These 2327 samples contained 1830 samples with the seat heating off and 497 samples with the seat heating on. An overview of all results can be seen in **Table I**.

A. Classical SVM

The training of the SVM on the classical system took between 72 to 76 milliseconds. The prediction by using the test set took between 480 and 490 milliseconds. The RBF kernel was selected during tests on the test data. Using the RBF kernel, the accuracy achieved on the validation set was 93.2%.

B. Quantum Annealer

In the early phase of our work, the feasibility and potential of both the gate-based approach and the annealing-based approach were evaluated. However, the translation of the SVM construction to QUBO does not scale well and creates a graph that far exceeds the capabilities of the current generation quantum annealers. Tools exist to solve QUBOs that large classically or even with the support of the quantum annealer on specific sub-problems [36], but it was noticed that run-times jumped from sub-minute performance to over an hour. On a classical machine, the QUBO encoding has a quadratic disadvantage with respect to the number of data points used. Then solving QUBOs is NP-complete—while typical SVMs can be trained in cubic time—thus employing an algorithm (like TABU search) that is much more powerful and resource-intensive than necessary. These drawbacks cannot be overcome with the quantum annealer accelerating relatively small sub-problems compared to the full problem instance. For this reason, we choose to spend our remaining focus on the gate-based quantum accelerator.

C. Gate-based quantum accelerator

The training of the system on the classical hardware, consisting of tuning the parameters of the quantum circuit, took approximately 4.5 hours. The parameters found from training were used during validation. Using the quantum simulator on classical hardware took 52 seconds and resulted in accuracy between 93.1% and 93.3%. This experiment was also performed on the IBMQ quantum accelerator. This took between 2.5 hours and 5 hours, depending on the queue size. This resulted in accuracy between 90.5% and 91.9%. The system reported error rates for its gates between 10^{-1} to 10^{-3}.

The evaluation of the 2327 samples on the IBMQ quantum accelerator involved running 2327 quantum circuits that needed to be repeated 1000 times each (called shots). Every call to the IBMQ API packed 75 circuits, and got queued for several seconds up to an hour. A total of 32 calls to the API were needed. Every pack of 75 circuits which ran 1000 times each, consumed 88 to 90 seconds of QPU time. Every single run of a circuit, therefore, completes in approximately 1.2 milliseconds. Sending data up for prediction from the vehicle to the cloud and back takes anywhere from sub-second time to several seconds, assuming the vehicle is in an area of connectivity. Sending a pro-active update is done once a minute.

VI. Discussion

The accuracy of the quantum-enhanced kernel holds up in simulation to the accuracy of the classical RBF kernel with an accuracy of around 93%. Even though the quantum system is subjective to error rates in the order of 10^{-1} to 10^{-3}, whereas classical systems lie around 10^{-12}, the performance decrease for the low-depth minimally entangled circuit is only 2% points with a range of ±1% point. The authors of the method already showed with artificially created data that the algorithm works well with NISQ-era devices; with our findings, we show that this statement also holds on our real-world data. Future work would include investigating data with more features, multi-class labels, and various classes of use-cases, as well as exploring different feature maps.

The quantum system, assuming an exclusive subscription to a machine, can classify a single sample with 1000 shots in 1.2 seconds. As the update loop in the car is on a minute-basis, we showed that a quantum system could be integrated for the prediction of in-vehicle systems using the architecture we proposed yielding near-similar accuracy as classical systems.

Even though classical systems can perform classification on this small number of features quicker, with 32 microseconds per classification, and with similar accuracy, it is still expected that NISQ devices will eventually outperform classical systems for specific tasks similar to ours [24]. First hints of nearing this computational barrier can already be

seen, such as the recent experiment at Google [37]. *Threats to the validity:* The analysis was performed on data from one user on one use-case, being seat-heating. However, the focus of this paper is on the integration of the quantum system with regards to timing and reduction of accuracy due to decoherence.

Additionally, the control parameters were not optimized. This included settings such as a number of shots and runs till convergence. This would be generated faster performance, but not change our argument, as the results are already within the minute-domain.

VII. Conclusion

Quantum computing is an emerging field with great potential. In this paper, we proposed an architecture integrating quantum accelerators in an industry-grade system. We implemented this architecture and evaluated the performance of various SVM implementations on real-world data for the prediction of the status of seat-heating. In our work, we were not searching for quantum advantage. The goal was to evaluate feasibility for the integration of NISQ devices in preparation for expected improvements in the next 5-10 years. These improvements would lay in more parallel processing power, and higher expressibility resulting in higher accuracy.

The accuracy of the SVM with quantum-enhanced kernel on the gate-based quantum simulator performed similarly to a classical SVM with RBF kernel, both achieving an accuracy of 93%. The same circuit run on the IBM quantum accelerator dropped 2%-points in accuracy. This shows that the quantum-enhanced kernel, previously only tested on artificial data, also performs well on our real-world data. It also shows that our selected algorithm running on a NISQ-era device with high error rates can still perform close to its theoretical accuracy.

The latency of the quantum system, assuming exclusive subscription to avoid lengthy and non-deterministic queues, can be parallelized to provide results in under a minute. This makes it feasible to predict for our seat-heating use-case, which requires updates every minute.

Based on the low reduction in accuracy and latency numbers within acceptable bounds, we conclude it is feasible to integrate NISQ-era devices in an industry-grade system architecture in preparation for future hardware improvements.

References

[1] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. In *32nd Computational Complexity Conference (CCC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[2] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review 41.2 (1999): 303-332.*, 1999.

[3] Vojtech Havlicek, Antonio D. Crcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum enhanced feature spaces. 2018.

[4] Yongcheng Ding, Lucas Lamata, Jos D. Martn-Guerrero, Enrique Lizaso, Samuel Mugel, Xi Chen, Romn Ors, Enrique Solano, and Mikel Sanz. Towards prediction of financial crashes with a d-wave quantum computer, 2019.

[5] Kazuki Ikeda, Yuma Nakamura, and Travis S. Humble. Application of quantum annealing to nurse scheduling problem, 2019.

[6] Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, November 2019.

[7] Florian Neukart, Gabriele Compostella, Christian Seidel, David von Dollen, Sheir Yarkoni, and Bob Parney. Traffic flow optimization using a quantum annealer, 2017.

[8] Tobias Stollenwerk, Elisabeth Lobe, and Martin Jung. Flight gate assignment with a quantum annealer, 2018.

[9] K Bertels, A Sarkar, A.A Mouedenne, T Hubregtsen, A Yadav, A Krol, and I Ashraf. Quantum computer architecture: Towards full-stack quantum accelerators. *arXiv preprint arXiv:1903.09575*, 2019.

[10] Aritra Sarkar, Zaid Al-Ars, Carmen G Almudever, and Koen Bertels. An algorithm for dna read alignment on quantum accelerators. *arXiv preprint arXiv:1909.05563*, 2019.

[11] Nader Khammassi, Gian G Guerreschi, Imran Ashraf, Justin W Hogaboam, Carmen G Almudever, and Koen Bertels. cqasm v1. 0: Towards a common quantum assembly language. *arXiv preprint arXiv:1805.09607*, 2018.

[12] Nader Khammassi, I Ashraf, X Fu, Carmen G Almudever, and Koen Bertels. Qx: A high-performance quantum computer simulation platform. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 464–469. IEEE, 2017.

[13] Xiang Fu et al. eqasm: An executable quantum instruction set architecture. In *2019 IEEE International Symposium on HPCA*, pages 224–237. IEEE, 2019.

[14] L Riesebos, X Fu, AA Moueddenne, L Lao, S Varsamopoulos, I Ashraf, J van Someren, N Khammassi, CG Almudever, and K Bertels. Quantum accelerated computer architectures. In *IEEE ISCAS*, pages 1–4. IEEE, 2019.

[15] X Fu, L Lao, K Bertels, and CG Almudever. A control microarchitecture for fault-tolerant quantum computing. *Microprocessors and Microsystems*, 70:21–30, 2019.

[16] Rodney Van Meter and Clare Horsman. A blueprint for building a quantum computer. *Communications of the ACM*, 56(10):84–93, 2013.

[17] Frederic Chong. Closing the gap between quantum algorithms and hardware through software-enabled vertical integration and co-design. In *APS Meeting Abstracts*, 2018.

[18] Margaret Martonosi and Martin Roetteler. Next steps in quantum computing: Computer science's role. *arXiv preprint arXiv:1903.10541*, 2019.

[19] Cupjin Huang, Mario Szegedy, Fang Zhang, Xun Gao, Jianxin Chen, and Yaoyun Shi. Alibaba cloud quantum development platform: Applications to quantum algorithm design. *arXiv preprint arXiv:1909.02559*, 2019.

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

[21] David McKay et al. Qiskit backend specifications for openqasm and openpulse experiments. *arXiv preprint arXiv:1809.03452*, 2018.

[22] Manfred Broy. Challenges in automotive software engineering. In Leon J. Osterweil, Dieter Rombach, and Mary Lou Soffa, editors, *Proceeding of the 28th international conference on Software engineering - ICSE '06*, page 33, New York, New York, USA, 2006. ACM Press.

[23] Richard Bellman. *Dynamic Programming*. Dover Books on Computer Science. Princeton University Press and Dover Publications, Princeton, NJ, USA, 1 edition, 1957.

[24] John Preskill. Quantum computing in the nisq era and beyond. *Quantum 2 (2018): 79.*, 2018.

[25] K. Wright et al. Benchmarking an 11-qubit quantum computer. *Nature Communications*, 10(1), November 2019.

[26] Stefano Pirandola, Bhaskar Roy Bardhan, Tobias Gehring, Christian Weedbrook, and Seth Lloyd. Advances in photonic quantum sensing. 2018.

[27] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7, April 2019.

[28] Research and Ltd. Development Group, Hitachi. Research team

led by the hitachi cambridge laboratory demonstrates an innovative hybdrid circuit for quantum computers, 2019.

[29] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification*. Wiley-Interscience publication. Wiley, New York NY u.a., 2. ed. edition, 2001.

[30] Héctor Abraham et al. Qiskit: An open-source framework for quantum computing, 2019.

[31] Catherine C McGeoch, Richard Harris, Steven P Reinhardt, and Paul I Bunyk. Practical annealing-based quantum computing. *Computer*, 52(6):38–46, 2019.

[32] F.Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] M. Schuld and N. Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters 122.4*, 2019.

[34] Karol Bartkiewicz, Clemens Gneiting, Antonn ernoch, Kateina Jirkov, Karel Lemr, and Franco Nori. Experimental kernel-based quantum machine learning in finite feature space, 2019.

[35] Dennis Willsch, Madita Willsch, Hans De Raedt, and Kristel Michielsen. Support vector machines on the d-wave quantum annealer. *arXiv preprint arXiv:1906.06283*, 2019.

[36] Michael Booth, Edward Dahl, Mark Furtney, and Steven P Reinhardt. Abstractions considered helpful: a tools architecture for quantum annealers. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–2. IEEE, 2016.

[37] Arya K. Babbush R. et al. Arute, F. Quantum supremacy using a programmable superconducting processor. *Nature 574, 505510 (2019)*, 2019.