AutoQML

A Framework for Automated Quantum Machine Learning

D. Klau[1] | H. Krause[2] | D. A. Kreplin[3] | M. Roth[3] | C. Tutschku[1] | M. Zöller[4*]

[1]*Fraunhofer IAO, Nobelstraße 12, 70569 Stuttgart, Germany*
[2]*GFT Integrated Systems GmbH, Reichenaustraße 39A, 78467 Konstanz, Germany*
[3]*Fraunhofer IPA, Nobelstraße 12, 70569 Stuttgart, Germany*
[4]*USU GmbH, Rüppurrer Str. 1, 76137 Karlsruhe, Germany*
[*]*Authors are listed in alphabetical order.*

Abstract

In this work, we present AutoQML, a framework that seamlessly integrates Quantum Machine Learning (QML) algorithms into Automated Machine Learning (AutoML). Leveraging the advantages of the AutoML paradigm, the framework is intentionally designed with a high level of abstraction, eliminating the need for users to possess extensive experience in both Machine Learning (ML) and Quantum Computing (QC). The tool automates the entire process of constructing typical ML pipelines including data cleaning and preprocessing as well as model selection, optimization, and evaluation. Additionally, it automatizes QC-specific aspects as for example selection of quantum backends and execution management on real quantum hardware. AutoQML utilizes *Ray* as its underlying AutoML optimization framework and employs the in-house developed QML library *sQUlearn* for providing QML algorithms. Both of these components provide low-level functionality and can be used as standalone solutions. Finally, we delve into the integration steps required to incorporate the framework into the Quantum Computing-as-a-Service platform, PlankQK.

Contents

1 Introduction & Motivation

In today's quantum computing (QC), the advancement and assessment of quantum algorithms demand a substantial understanding of both the specific application domain and intricate quantum hardware configuration. The proper configuration of quantum hardware is also delicate, given that the selection of the appropriate backend and algorithm hinges on factors such as error rates, calibration, coupling maps, fidelity, and more. Similar to traditional Machine Learning (ML), Quantum Machine Learning (QML) combines the statistical learning techniques of ML with QC, transforming conventional algorithms into innovative forms, such as quantum kernel methods [1] or quantum neural networks [2]. Despite the existence of open research questions, there is a large potential for enhancing algorithmic performance [3]. In both traditional and quantum ML, the process of developing and implementing solutions typically follows an iterative approach, guided by trial and error, given the absence of clearly defined rules for creating optimal pipelines. Experts must manually handle tasks such as data cleaning, preprocessing, feature engineering, model tuning, and evaluation, rendering the process slow and resource-intensive.

Automated Machine Learning (AutoML) holds promise in overcoming challenges associated with constructing ML applications [4, 5, 6, 7]. The research field is dedicated to streamlining the development process and reducing the required level of expertise by automating the creation of ML pipelines. AutoML simplifies the training of ML models for domain experts and enhances the efficiency of ML professionals by automating repetitive tasks, such as hyperparameter optimization (HPO) or algorithm selection (AS) [8]. The fundamental concept of AutoML, which separates technical complexity from user experience and facilitates quick prototyping, is an attractive proposition, particularly in the domain of QML, for which a shortage of experts exists.

This study expands the AutoML paradigm into the quantum domain by integrating optimization and integration routines specifically designed for QML algorithms with the automation capabilities of traditional AutoML frameworks. We outline the requirements and architecture of a novel AutoQML framework, along with potential integration options into the QC-as-a-Service platform provided by PlanQK.

The following document is structured as follows: In Chapter 2, we outline the requirements and illustrate our rationale for selecting the traditional AutoML framework that serves as the backbone of our library. Chapter 3 introduces the framework architecture and its current implementation status, with a specific focus on two key aspects: the integration and management of QML algorithms and their associated management layer (Sec.3.1), and the design of the overall AutoQML framework architecture (Sec.3.2). Chapter 4 outlines our initiatives to integrate the framework into the PlanQK platform, a quantum software ecosystem, aiming to enhance accessibility for researchers and practitioners. In the final Chapter 5, we provide an overview of the current state of the framework and offer insights into its future outlook.

2 Framework Requirements

After systematically evaluating, selecting, and benchmarking traditional AutoML frameworks for their extension to QML in our prior work [9], we have proceeded to implement the AutoQML framework. This software library augments a suitable existing AutoML framework with functionalities designed to optimize and streamline the development of QML algorithms. While the framework is still in active development, it already incorporates the majority of functionalities related to both AutoML and QML algorithms. This chapter outlines the functional and technical requirements for the chosen traditional AutoML framework, which serves as the functional backend, along with the expected utility of the implemented AutoQML framework, as described in Section 3.

2.1 AutoML Requirements

AutoML frameworks for traditional ML can be categorized into high and low abstraction levels [9, Ch. 2]. Depending on their abstraction level, these tools encompass various aspects of an ML development procedure and resulting pipeline, as illustrated in Figure 1. Low-level frameworks often demand a high level of proficiency in data science, particularly requiring familiarity with ML algorithms, as users must define and structure search spaces. In contrast, high-level frameworks aim to address the overarching challenge of pipeline creation. These tools leverage the automation and optimization features of low-level frameworks, expanding them with additional preprocessing, ensemble, and evaluation steps. This enables high-level frameworks to automatically generate entire ML pipelines.

The AutoML optimizer for the AutoQML framework must fulfill the following requirements:

(1) Have a Python interface.

(2) Be open-source.

(3) Be under active development.

(4) Have an active community.

(5) Support different common ML problems (classification, regression) and input data types (e.g., tabular, image, time series).

(6) Have a permissive license.

These criteria are essential for the frameworks under consideration to be included in the final selection. Additionally, we assess the framework based on the following aspects:

- Number and types of available ML backends.
- Size and structure of the provided search space.
- Supported optimizers/search algorithms (such as random search, Bayesian Optimization, evolutionary approaches, etc.).

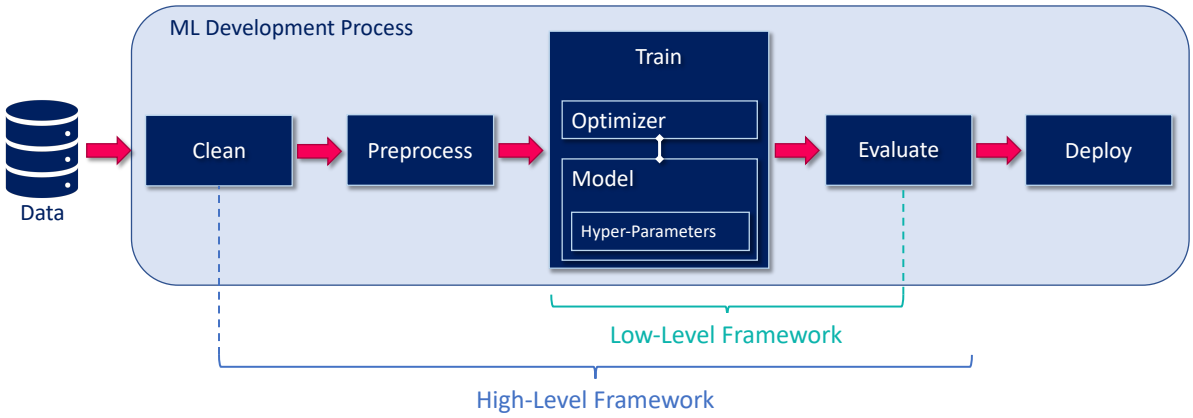

Figure 1: *Illustration of a typical ML development process. Different abstraction levels of AutoML correspond to specific processing steps within a single pipeline. A low-level AutoML framework typically automates the traditional training step by exploring various combinations of algorithms and hyperparameters. In contrast, a high-level framework not only automates the training step but also encompasses algorithm-dependent data preprocessing and potentially includes segments of the data cleaning procedure. The figure is obtained from [9, Fig. 3].*

The detailed discussion of the selection process for the aforementioned requirements is provided in our previous publication, referenced as Klau *et al.* [9, Ch. 4].

2.2 Quantum Machine Learning Requirements

Through interviews with QC developers and researchers affiliated with the Fraunhofer Institutes for "Manufacturing Engineering and Automation" (IPA) and "Industrial Engineering" (IAO), three general integration benefits and synergies with AutoML were identified [9]. These have been incorporated as desired functionalities in the framework specification:

- Selection and configuration of QML algorithms with AutoML approaches: Utilizing established traditional ML and optimization methods, the framework aims to identify the optimal QML algorithm and configure hyperparameters tailored to a given task. This necessitates the framework's ability to handle datasets with diverse or multimodal input data, execute preprocessing steps such as data analysis and feature engineering, choose and optimize the most suitable QML algorithms, and ultimately generate an end-to-end QML pipeline.

- Quantum-enhanced AutoML optimization: The framework should offer the flexibility to implement custom HPO approach. QC can also enhance optimization efficiency during the hyperparameter search for traditional ML algorithms. Given that AutoML frameworks often navigate high-dimensional parameter spaces, quantum optimization and search algorithms, such as quantum Bayesian optimization [10], hold the potential to improve this process.

- Incorporating decision characteristics and meta-learning for QML pipelines: Analogous to their traditional counterparts, different QML models exhibit advantageous settings or failure modes contingent on the specific task. This knowledge gained through the empirical application of these algorithms can be integrated into

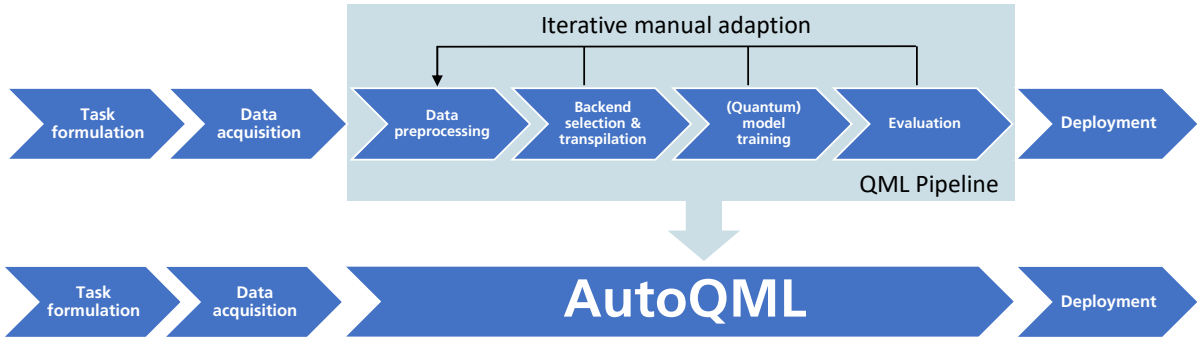

Figure 2: *Visualization depicting the stages in the QML development process addressed by the AutoQML framework. The framework streamlines algorithm and hardware optimization tasks throughout the QML pipeline, covering specialized data preprocessing, evaluation of quantum hardware constraints, and automatic model training and assessment.*

the decision criteria to help developers in an a priori selection of well-performing algorithms and QC backends for their data and use case. A prerequisite for the AutoQML framework is the incorporation of these rules and meta-learning approaches into the pipeline creation process.

On the software development front, the integration of QML algorithms into the AutoQML framework should be facilitated through wrapper functions that support hyperparameter configuration, enhancing compatibility with existing implementations. The framework also should support both simulators and real QC hardware backends. Furthermore, it is crucial to provide utility methods for the automatic management of lengthy queues and job submissions, tailored to the specific requirements of different quantum computers. Additional tools may be deemed necessary to evaluate the economic, operational, and hardware constraints of the written code. The supported QC libraries should encompass the two primary frameworks widely used in QML: IBM Qiskit [11] and Xanadu's PennyLane [12]. Additionally, the framework should exhibit compatibility with popular Python-based data science and AI tools.

2.3 Decision Criteria & Framework Selection

In our approach, we opt for a low-level framework and develop a customized high-level automation utility around it. This strategy guarantees increased flexibility in handling both traditional and QML algorithms, while also minimizing overhead in the implementation and management of separately maintained frameworks.

Building upon the insights from [9], the traditional framework chosen for extension and integration with both traditional and QC-specific automation procedures is *Ray* [13]. The decision is grounded on the following considerations:

- Ray, classified as a low-level framework, possesses distinct attributes:
  - It is more straightforward to programmatically expand or encapsulate than its higher-level counterparts. This simplifies the definition of QC-specific processing steps and pipeline components.

  - Ray exclusively addresses the CASH problem (combined algorithm selection and hyperparameter optimization) by defining (structured) search spaces. This feature facilitates the seamless incorporation of novel algorithm classes and different frameworks.
  - Pipeline stages specific to QC, not typically considered in traditional AutoML frameworks (such as backend analysis and selection), can be directly implemented. This is possible because Ray does not rely on predefined preprocessing stages.

- In a comprehensive real-world evaluation across the four distinct use cases and QC problem types [9, Ch. 5.2] investigated in the project, Ray achieved the best results. It either matched or surpassed frameworks within the same category in terms of final model prediction quality, training and inference efficiency, utilization of computing resources, and configurability of framework behavior.

- Ray supports a variety of traditional libraries for implementing ML models, making it a flexible tool to train and compare traditional ML algorithms with their quantum counterparts.

- Ray also receives high ratings for future security and quality of life aspects. The framework's emphasis on parallelism and efficient computing renders it widely applicable across various software development use cases. With a substantial community of around 900 contributors and an extensive user base (28.5k GitHub Stars), along with nearly 6000 individual documentation URLs, Ray remains under active development and improvement. This makes it a fitting option for a high-performance backend[1].

[1] The numbers are sourced from the project's GitHub page https://github.com/ray-project/ray.

3 AutoQML Framework Specifications

Building upon the framework requirements identified in the previous chapter, we present an overview of the actual implementation of the framework that integrates QML into AutoML. Initially, we introduce a library designed to standardize QML algorithms within a scikit-learn-like interface, referred to as *sQUlearn* [14]. Subsequently, we introduce *AutoQML* to enhance sQUlearn with AutoML capabilities.

3.1 QML Integration

We have developed sQUlearn [14], a standalone Python library ready for QML on Noisy Intermediate-Scale Quantum (NISQ) computers. This library features a scikit-learn interface [15] to seamlessly integrate QML algorithms into an AutoML framework. The design ensures compatibility with many traditional ML tools in general. The library adopts a dual-layer approach, offering high-level implementations of QML methods. Additionally, it provides low-level implementations for advanced users interested in developing and researching new QML methods. The high-level implementations cover quantum neural networks (QNN)[16] and quantum kernel methods[17] in various forms, such as quantum support vector machines (QSVM)[18] and quantum Gaussian processes[19]. All these tools can be utilized for both classification and regression tasks. Figure 3 illustrates the high- and low-level modules of the sQUlearn library. The high-level methods at the top of the figure are seamlessly integrated into the AutoQML framework.

QNNs employ parameterized quantum circuits with data-dependent inputs and generate outputs by evaluating the expectation values of observables. The circuit parameters and observables are adjusted through gradient-based optimization to minimize a loss function, similar to the training process of traditional artificial neural networks. The QNN engine in sQUlearn incorporates arbitrary differentiation concerning parameters or data inputs, achieved through the successive application of the parameter shift rule [20]. Data structures facilitating the computation of necessary arithmetic for expectation values are also implemented. In contrast to other QML libraries like Qiskit, sQUlearn allows parameterization and training of the observables defining the QNN output, enabling more versatile models. Additionally, it supports caching of both derivative circuits and obtained results. Furthermore, sQUlearn offers an efficient execution that minimizes the number of circuit evaluations on the quantum computer. Our developed variance regularization, aimed at reducing shot noise, and an adaptive shot control are available in the high-level QNN programs [21]. These features can also be controlled through hyperparameters.

Quantum kernel methods leverage the exponentially large Hilbert space accessible by a quantum computer to embed input data into a high-dimensional representation. In this expansive space, a machine learning model can be constructed using linear regression or classification through conventional kernel methods. These methods utilize the fidelity between quantum states as the kernel matrix. However, the use of fidelity-type quantum

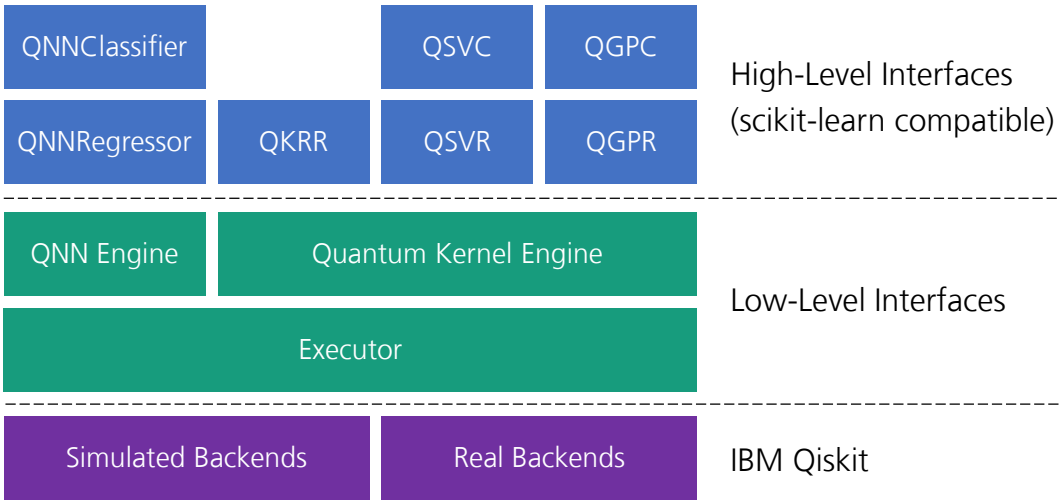

Figure 3: *Schematic overview of sQUlearn: The top level displays the diverse high-level implementations available in sQUlearn that can directly be integrated into the AutoQML framework. These implementations build upon the low-level modules of the quantum kernel and QNN engine. The executor is employed to conduct experiments on simulated and real backends, utilizing the Qiskit environment.*

kernels in such extremely large spaces may result in exponentially vanishing fidelity [22]. To circumvent this curse of dimensionality, projected kernels have been developed [23]. These kernels first project the embedded states back into a lower-dimensional traditional representation before performing the kernel evaluation. In sQUlearn, both fidelity-type and projected quantum kernels are supported. The implementation of the projected quantum kernel utilizes the QNN implementation to compute the projection back to the real space. The outcome is then incorporated into an outer kernel, offering various options such as the Radial Basis Function (RBF) Kernel or the Matern kernel. The options for constructing the projected kernel can be set as hyperparameters, which allows tuning by the AutoQML framework. Additionally, specialized regularization and mitigation techniques, adjustable through hyperparameters, are available in the high-level implementations.

Moreover, sQUlearn places a significant emphasis on NISQ-compatibility and end-to-end automation, featuring black-box implementations of QML algorithms capable of running on currently available quantum hardware. This is achieved by centralizing the execution of all quantum jobs. The execution engine is responsible for automatically selecting the suitable backend, submitting and restarting failed jobs, as well as renewing sessions of the quantum provider. Presently, the execution and circuit management rely on Qiskit [24]. Consequently, all algorithms can be easily executed on IBM hardware and Qiskit-based simulators. However, an extension to other hardware providers is planned.

3.2 Framework Design

In the following section, we elaborate on the AutoQML framework, which enhances QML algorithms with AutoML capabilities. Our goal is to achieve end-to-end automation for QML. Consequently, AutoQML not only selects a fine-tuned QML algorithm but also constructs a complete QML pipeline with appropriate preprocessing to ensure high-quality

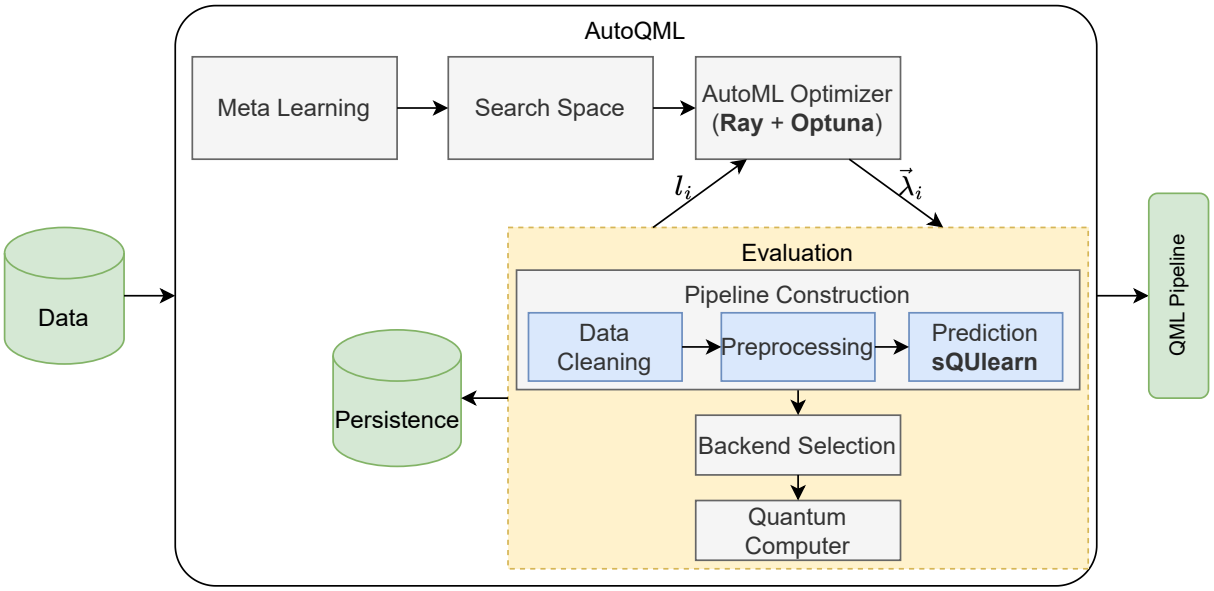

Figure 4: *Architecture overview of the AutoQML framework.*

predictions. Figure 4 provides a high-level overview of the AutoQML architecture. As of the time of writing, the framework is still under development. While core modules such as Search Space Selection, AutoML Optimizer, and Evaluation are already functional, other modules like Meta Learning and Persistence are still currently developed. The fully functional framework will be available by the end of the project. In the remainder of this section, we will provide a more detailed explanation of the architecture.

In general, the AutoQML architecture is primarily inspired by state-of-the-art AutoML, such as [25]. The user is required to provide an input dataset with a task description, more specifically classification or regression.[2] Similar to AutoML for traditional machine learning, we generate a search space for automated optimization. This search space encompasses the necessary steps to create an end-to-end pipeline for QML predictions, which can be divided into three phases:

Data Cleaning This phase is responsible for eliminating potential defects from the input data. It includes imputation of missing values, outlier removal, and encoding of categorical features.

Preprocessing The preprocessing phase transforms the data into a format suitable for quantum computing. It involves dimensionality reduction, down-sampling, and rescaling. An extension with basic feature engineering is planned for future work.

Prediction The prediction phase employs classification or regression algorithms to generate the actual predictions. The available QML algorithms are implemented via sQUlearn.

Most steps in the search space can be freely combined or skipped to create a diverse set of pipelines.

[2] While an extension to other learning tasks like unsupervised learning is possible, the focus of this work is on supervised learning.

The actual AutoML optimization is conducted using Ray Tune [26] in conjunction with Optuna [27]. With the complete search space description, Ray Tune iteratively draws new test configurations, denoted as $\vec{\lambda}_i$. A configuration represents an abstract specification of a particular QML pipeline. The configuration $\vec{\lambda}_i$ is then passed to the evaluation function, which returns a score $l_i \in [0, 1]$. Based on the $(\vec{\lambda}_i, l_i)$ tuple, a probabilistic model of the loss function is generated to create the next test candidate $\vec{\lambda}_{i+1}$. This process is repeated until a user-provided budget, such as a time limit, is exhausted. Ray Tune is employed to learn the relation between $\vec{\lambda}_i$ and l_i, guiding the optimization toward a well-performing region in the search space.

The evaluation function transforms the abstract configuration into an actual QML pipeline. Subsequently, the determination of the quantum computer on which the pipeline will be executed becomes crucial. Therefore, an algorithm for automatic QC backend selection is integrated into the framework. This algorithm, given the required qubits of the QML circuits, searches for a backend with sufficient qubits and low expected error rates. Once the backend is fixed, the pipeline is scheduled for execution. The selected preprocessing steps are computed on traditional hardware, and the intermediate dataset is passed to the chosen QML implementation to calculate the performance l_i of the configuration. All evaluated pipelines are persisted with meta-information. When the optimization budget is exhausted, the best-performing configuration $\vec{\lambda}^\star$ and the corresponding fitted pipeline are returned to the user.

Note that, prior to search space generation, the concept for the final framework incorporates also a meta-learning module. Here, the input data is analyzed to capture its general characteristics in various *meta-features*. These meta-features contain simple yet crucial information about the dataset, such as the number of features, number of samples, or presence of missing values. These features can then be used to warm-start the optimization process. The fundamental idea is that meta-learning can be employed to directly propose a set of QML pipelines that are *well-suited* for the provided input data. In the context of QC, well-suited pipelines should include necessary preprocessing steps to enable execution on quantum hardware. In the NISQ era of QC, these steps include appropriate downsampling and feature reduction to account for the limited number of qubits. For traditional computing, meta-learning is typically performed by evaluating a large set of pipelines on various datasets to learn a mapping [28]. In the context of QC, the details of such a mapping are an ongoing research question within the project. Given that calls to actual QC hardware are expensive in many aspects, the inclusion of a meta-learning module is expected to significantly reduce the overall optimization cost.

4 PlanQK Integration

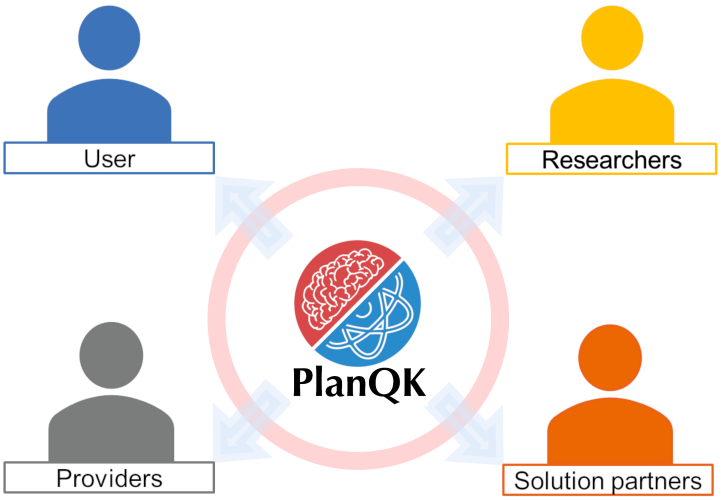

Figure 5: *Stakeholders of the PlanQK platform.*

To fully leverage the potential of Quantum Computing (QC) in the near-term future, the integration of the AutoQML framework into everyday processes, both technically and economically, is essential. This integration will be facilitated through PlanQK (Plattform und Ökosystem für Quantenapplikationen), a German initiative designed to offer a platform for researchers and industry to exchange and provide quantum software solutions in an app-store-like manner.[3]. PlanQK serves a dual purpose: providing developers, scientists, and businesses with a platform for collaboration, and functioning as a provider for QC as a service.

To integrate tools into the PlanQK platform, users submit their software solution in form of a docker container as a service that becomes available on the platform. Customers can then access the service through an app that manages billing and documentation of the provided functionality. Input and output data are stored in data pools on the platform, available for use by the services, while the scheduling and queuing of the quantum

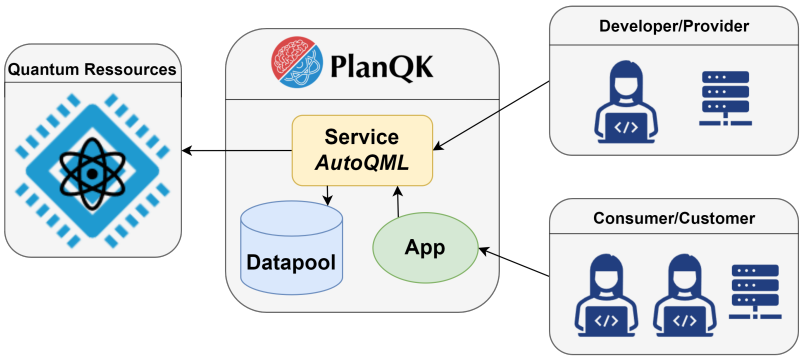

Figure 6: *Architecture example for automated QML as service*

[3]For more details, see https://planqk.de

workload to the quantum hardware are handled by the PlanQK platform.

By the end of the project, the AutoQML framework will be added as a container template to PlanQK, enabling the platform to offer the high-level features of the AutoQML framework as a service. Figure 6 shows how a service is integrated into a production environment and communicates with stakeholders.

To enhance the interaction with the PlanQK platform, the Python library *pyplanqk* has been developed as a project helper to abstract and wrap the PlanQK REST API. This simplifies and makes the interaction with PlanQK more developer-friendly. *pyplanqk* will be open source and published as a community-maintained project.

5 Summary and Outlook

In this work, we introduced AutoQML – a Python framework that extends AutoML to quantum-based machine learning algorithms. The library is built upon the low-level AutoML library Ray. The QML implementation is achieved through sQUlearn, an in-house-developed QML library offering various NISQ friendly QML algorithms. The interface between Ray and sQUlearn is based on scikit-learn. AutoQML provides an end-to-end solution, enabling the automatic creation of the entire pipeline – from preprocessing and algorithm selection to hyperparameter optimization of the QML methods. Furthermore, the hardware selection of supported quantum computers and execution management are also automatized when real quantum computers are employed.

AutoQML aims to bridge the gap between QML research and industrial application by automating the use of cutting-edge QML methods. Leveraging the foundation of a conventional AutoML library, AutoQML seamlessly accommodates both QML and traditional ML. This flexibility enables the selection of the overall best-performing algorithm for a given task, irrespective of its nature – be it traditional or quantum. The library is intended to co-develop with the technological advances in the field of QC, adapting to the state of the underlying hardware. Consequently, we anticipate that QML will emerge as the preferred choice for specific use cases as technology matures. This holds true even though traditional ML outperforms most of the existing QML alternatives today.

Acknowledgements

We thank the companies IAV GmbH Ingenieursgesellschaft Auto und Verkehr, KEB Automation KG, TRUMPF Werkzeugmaschinen GmbH + Co. KG, Zeppelin GmbH for their valuable contributions to the project Additionally, we appreciate the close and excellent collaboration with PlanQK. Furthermore, we acknowledge HQS Quantum Simulations GmbH for their contributions to hardware selection and execution processes.

This work was conducted within the framework of the AutoQML project, funded by the Federal Ministry for Economic Affairs and Climate Action. For more information about the project, please visit the project website at this URL: https://www.autoqml.ai/en.

References

[1] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," Physical Review Letters **113**, 5 (2014).

[2] K. Beer, D. Bondarenko, T. Farrelly, T. J. Osborne, R. Salzmann, D. Scheiermann, and R. Wolf, "Training deep quantum neural networks," Nature Communications **11**, 808 (2020).

[3] Y. Liu, S. Arunachalam, and K. Temme, "A rigorous and robust quantum speed-up in supervised machine learning," Nature Physics **17**, 1013–1017 (2021).

[4] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning - Methods, Systems, Challenges* (Springer, 2019).

[5] M.-A. Zöller and M. F. Huber, "Benchmark and survey of automated machine learning frameworks," Journal of artificial intelligence research , 409–472 (2021), arXiv:1904.12054.

[6] M.-A. Zöller, T.-D. Nguyen, and M. F. Huber, "Incremental search space construction for machine learning pipeline synthesis," in *International Symposium on Intelligent Data Analysis* (arXiv:2101.10951, 2021).

[7] Y. Quanming, W. Mengshuo, C. Yuqiang, D. Wenyuan, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang, "Taking human out of learning applications: A survey on automated machine learning," (arXiv:1810.13306, 2018).

[8] H. Stühler, M. A. Zöller, D. Klau, A. Beiderwellen-Bedrikow, and C. Tutschku, "Benchmarking automated machine learning methods for price forecasting applications," in *Proceedings of the 12th International Conference on Data Science, Technology and Applications* (Rome, Italy, 2023).

[9] D. Klau, M.-A. Zöller, and C. Tutschku, "Bringing quantum algorithms to automated machine learning: A systematic review of automl frameworks regarding extensibility for qml algorithms," (2023), arXiv:2310.04238.

[10] F. Rapp and M. Roth, "Quantum gaussian process regression for bayesian optimization," (2023), arXiv:2304.12923.

[11] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," (2023).

[12] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi,

S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," (2022), arXiv:1811.04968 [quant-ph] .

[13] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging AI applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (USENIX Association, Carlsbad, CA, 2018) pp. 561–577.

[14] D. A. Kreplin, M. Willmann, J. Schnabel, F. Rapp, and M. Roth, "sQUlearn - A Python Library for Quantum Machine Learning," (2023), 10.48550/arXiv.2311.08990.

[15] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," (2013), arXiv:1309.0238 [cs.LG] .

[16] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational quantum algorithms," Nature Reviews Physics (2021), 10.1038/s42254-021-00348-9.

[17] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," Phys. Rev. Lett. **122**, 040504 (2019).

[18] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," Nature **567**, 209–212 (2019).

[19] F. Rapp and M. Roth, "Quantum gaussian process regression for bayesian optimization," arXiv preprint arXiv:2304.12923 (2023).

[20] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," Phys. Rev. A **98**, 032309 (2018).

[21] D. A. Kreplin and M. Roth, "Reduction of finite sampling noise in quantum neural networks," (2023), arXiv:2306.01639 [quant-ph] .

[22] S. Thanasilp, S. Wang, M. Cerezo, and Z. Holmes, "Exponential concentration and untrainability in quantum kernel methods," (2022), arXiv:2208.11060 [quant-ph] .

[23] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. Mcclean, "Power of data in quantum machine learning," Nature Communications **12** (2021), 10.1038/s41467-021-22539-9.

[24] Qiskit Community, "Qiskit: An open-source framework for quantum computing," (2017).

[25] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenber, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *International Conference on Neural Information Processing Systems*, edited by C. Cortes, D. D. Lee, M. Sugiyama, and R. Garnett (MIT Press, 2015) pp. 2755–2763.

[26] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," arXiv preprint arXiv:1807.05118 (2018).

[27] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," (Association for Computing Machinery, 2019) pp. 2623–2631.

[28] J. Vanschoren, "Meta-learning," in *Automatic Machine Learning: Methods, Systems, Challenges* (Springer, 2019) pp. 35–61.