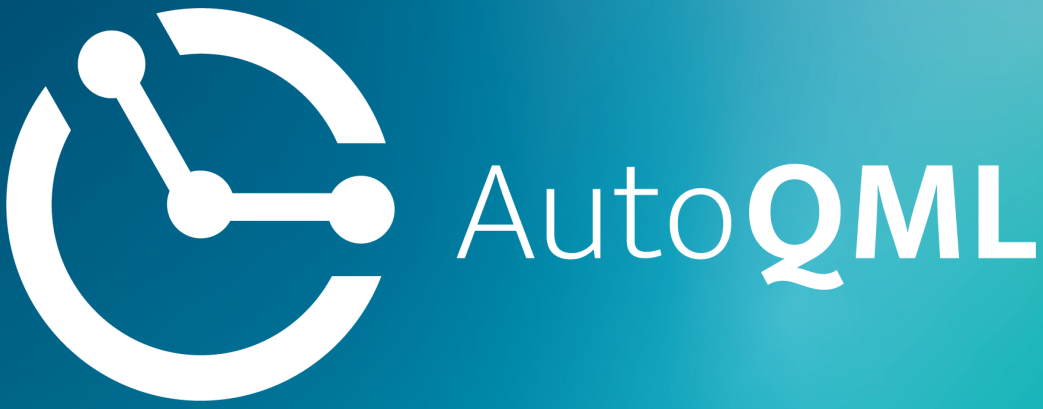
Fraunhofer IAO



Dennis Klau[1] | **Marc Zöller**[2] | **Dr. Christian Tutschku**[1]

[1] *Fraunhofer IAO, NobelstraSSe 12, 70569 Stuttgart, Germany*
[2] *USU Software GmbH, Rüppurrer Str. 1, 76137 Karlsruhe, Germany*

Bringing Quantum Algorithms to Automated Machine Learning

A Systematic Review of AutoML Frameworks Regarding Extensibility for QML Algorithms

Abstract

Quantum Computing (QC) is becoming an increasingly promising technology for modern computation, especially in the field of data driven approaches like simulation and machine learning. With the high momentum of research and development of new hard- and software, QC holds a big promise in redefining many state-of-the-art approaches in computation today.

On the other hand, the nowadays well-established field of machine learning (ML) faces challenges like the discrepancy of demand by industry and availability of ML experts, reproducibility, and efficiency in prototyping. To overcome some of these issues, several frameworks have been created for automating the process of pipeline construction, data preprocessing, model training and hyperparameter optimization (HPO), many of them open source. In most cases, these Automated Machine Learning (AutoML) frameworks implement a fixed subset of known approaches and algorithms, or encapsulate an established ML backend, that defines the available algorithms.

This work describes the selection approach and analysis of existing AutoML frameworks regarding their capability of a) solving a set of industrial use-cases with different ML problem types and b) incorporating Quantum Machine Learning (QML) algorithms into this automated solving approach. For that, available open-source tools are condensed into a market overview and suitable frameworks are selected on a multi-phase, multi-criteria approach considering software selection aspects [1], as well as in terms of the technical perspective of AutoML [2, 3].

The requirements for framework selection are divided into hard and soft criteria regarding their software and ML attributes. Additionally, a classification of AutoML frameworks is made into high- and low-level types inspired by the findings of [4]. Finally, we select Ray and AutoGluon as the suitable low- and high-level frameworks respectively, because they best fulfil the requirements and got the best feedback from our use-case study.

Based on this work, we will build QC specific pipeline steps and decision characteristics for hardware and software constraints into these frameworks to enable them for Automatic Quantum Machine Learning (AutoQML).

Contents

1. Introduction & Motivation

Building Machine Learning (ML) applications is a complex task, that usually requires involvement from both ML and domain experts. Additionally, the development of such systems is an iterative process, often driven by trial and error. So far, most of the required work to build ML applications is carried out by those experts manually: designing and implementing the individual steps from data cleaning, pre-processing and feature engineering, model tuning and hyper-parameter selection to evaluation and deployment. Consequently, building good pipelines can be a time and cost intensive task [2].

A similar situation is present in the Quantum Computing (QC) domain: Designing and tailoring quantum algorithms to a specific problem and testing different parametrizations on a simulated or real backend is usually done in an iterative manner by highly skilled experts, that require both, extensive domain and QC knowledge. Additionally, hardware handling is especially delicate still when computing on real backends: finding suitable quantum hardware for a specific problem and data depends on multiple factors like current error rates and calibration, the coupling map, fidelity and decoherence times, as well as the speed of gates.

Automated Machine Learning (AutoML) is a potential solution to overcome some of the abovementioned challenges that researchers, developers and adopters face [4, 3, 5]. AutoML is an end-to-end approach for building ML applications, which aims to improve the iteration speed of and reduce the required knowledge for building ML pipelines via full or partial automation and encapsulation of algorithmic implementation details. This enables domain experts to create functioning first-draft ML pipelines, as well as increases efficiency of ML researchers and developers by automating tedious tasks like hyperparameter optimization (HPO) [6].

The general AutoML paradigm of knowledge and usability separation, as well as the increase of prototyping efficiency is also desired in the area of QC algorithm and software development, especially in the emerging subfield of Quantum Machine Learning (QML). This field follows the same concept of learning statistics from data as classical ML and is inspired by existing algorithms and implementations. For that, approaches from the ML domain are transferred to the QC paradigm, where equivalent, hybrid, or new versions of existing algorithms like kernel methods (e.g., QKE, QSVM [7]) or neural networks (QNN, QCNN [8]) are developed.

To utilize the benefits of AutoML also in the QML domain, a review, classification and evaluation of already existing AutoML frameworks regarding their capabilities and extensibility to quantum algorithms is mandatory. This includes the collection of the most prominent open-source frameworks to date with associated meta-data, the categorization of their degree of automation, optimization formulation and abstraction level, as well as an evaluation of the most promising frameworks by solving specific use-cases and conducting expert interviews.

This work is structured as follows: We first give an overview over the existing techniques and approaches in AutoML, together with a new classification of the existing frameworks, in

Section 2. We do this to build a common understanding of this technology for the different target groups of AI and QC developers and researchers. In Section 3, we introduce the desired extensions to AutoML for the QC domain and introduce the terminology of AutoQML, followed by the framework selection approach and result the core of our work in Section 4. The selection approach is backed by the accompanied use-case study in Section 5, where identified frameworks are tested against multiple use-cases with different ML problem types. We conclude with a summary and future work perspective in Section 6.

2. Systematic Review of AutoML

AutoML creates an abstraction layer between the goal formulation of an ML problem (e.g., image classification) with its associated data, and the knowledge level and programmatic details required to implement, train, and fine-tune a specific or range of different ML algorithms.

Following the definition of [4], we classify the subproblems addressed by AutoML according to a tree-based structure as shown in Figure 1. Going from a narrow to the broadest definition of tasks, the following subproblems can be partially or fully addressed by AutoML:

- *Hyperparameter Optimization (HPO)*: This is the smallest encapsulation of automation in the AutoML context. In HPO, the hyperparameters of a given algorithm are optimized [2, 9]. This can only be done outside of the actual training loop of the algorithm and requires a full or partial training cycle to assess the quality of the chosen hyperparameters. HPO is an computationally expensive task, thus in addition to naive approaches like grid search (GS) or random search (RS), also more sophisticated optimizers like Bayesian Optimization (BayesOpt) or evolutionary algorithms (e.g., DEHB [10]) have been developed, that promise better convergence and less evaluations of the target function [11].

- *Algorithm Selection (AS)*: Automatic AS is a fundamental aspect of pipeline construction, since no single algorithm defines the state-of-the-art for a given problem and usually a set of ML models with complementary strengths are tested. In the AutoML context with a given set of algorithms, AS is usually formulated as the per-instance algorithm selection problem [12].

- *Combined Algorithm Selection and Hyperparameter Optimization (CASH)*: is the natural extension of the optimization problem by combining AS and HPO. In the CASH setting, a set of available ML algorithms and their corresponding hyperparameters form a hierarchical search space - in the following denoted as Λ - that can be traversed by an optimization algorithm, as shown in Figure 2. An addition in CASH is the introduction of a search space structure in the sense that *a priori* knowledge for HP combinations is available (i.e., that when optimizing the HPs of a specific ML algorithm, the parameters of other algorithms are irrelevant and thus dont need to be tested). This structure must be supported by the optimization algorithm, so that only valid and meaningful HP candidate sets are selected.

- *Structure Search*: Since an ML pipeline normally involves many operations from data validation and cleaning, pre-processing, feature engineering and model execution, finding the optimal structure of a ML pipeline is a hard problem on its own. In addition to the pre-processing requirements of individual algorithms, also arbitrary ensemble models can be constructed which then lead to parallel pipeline paths. To find good ensemble configurations for specific problems, usually genetic algorithms or pre-defined pipeline structures are used.

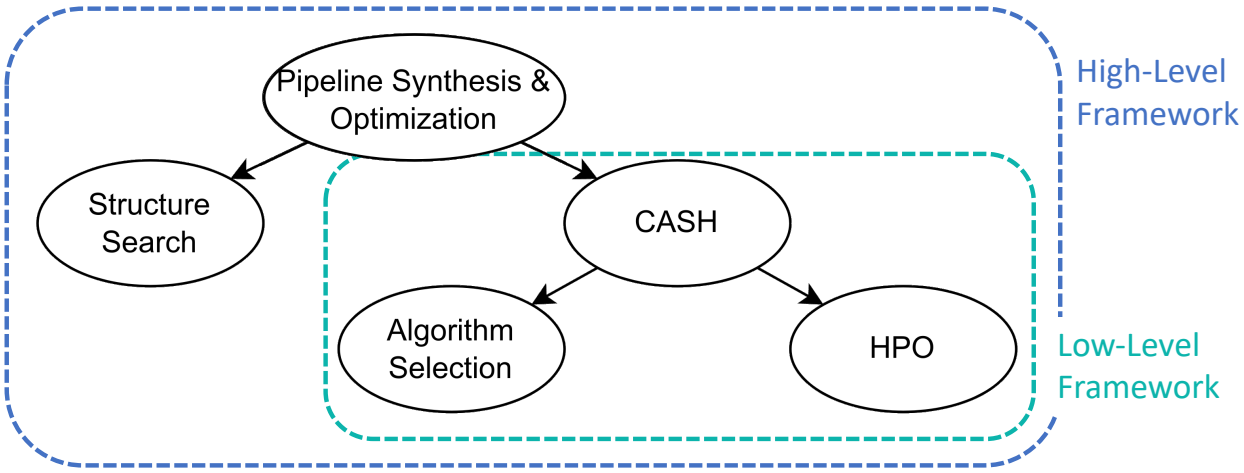


Figure 1: *Stages and abstraction levels in the AutoML context. Parent nodes include the associated underlying automation and optimization tasks respective to the AutoML problem formulation. While the high-level frameworks (dark blue) try to optimize the whole ML pipeline, including its structure, appropriate pre-processing and cleaning steps for each algorithm and possible ensemble configurations, the low-level frameworks (green) assess mostly the CASH (combined algorithm selection and hyper-parameter optimization) or HPO (hyper-parameter optimization) paradigms. Taken from [4].*

- *Pipeline Synthesis and Optimization (PSO)*: is the outer scope of the AutoML problem. It includes all previously described optimization approaches, i.e., performing Structure Search and CASH, of an ML pipeline as shown in Figure 1.

To evaluate which frameworks address which scope of the AutoML cntext and are best suited for extension with QML algorithms, we additionally classify the considered frameworks with respect to their underlying abstraction and automation level into two types:

- *Low-level* frameworks require substantial expertise in data science and specifically machine learning. Such solutions need user-defined and (partially) structured search spaces, like algorithms to test and their respective hyperparameters, as well as already cleaned and preprocessed input data. These type of frameworks mostly build directly ontop of well-established ML frameworks (e.g., Scikit-Learn [13], PyTorch [14], Tensorflow [15]) and wrap the basic algorithms inside of an automation routine with a scheduler and optimizer to efficiently apply approches from HPO up to CASH.

- *High-level* frameworks, extend the approach of the low-level frameworks by attempting to solve the whole pipeline creation problem. This includes the low-level tasks (CASH or individual sub-tasks) as well as assembling a complete ML pipeline automatically. Since a ML pipeline consists of several dependent steps with high degrees of freedom, the search space has not only a predefined structure and is much larger but also more complex. The tasks for PSO usually include the inquiry of the architecture itself, selection of algorithms and their hyperparameters, as well as appropriate pre-processing and necessary data cleaning steps to some extend for each algorithm.

The approaches of each framework type are shown in Figure 1, as well as encapsulated steps in an exemplary ML pipeline in Figure 3.

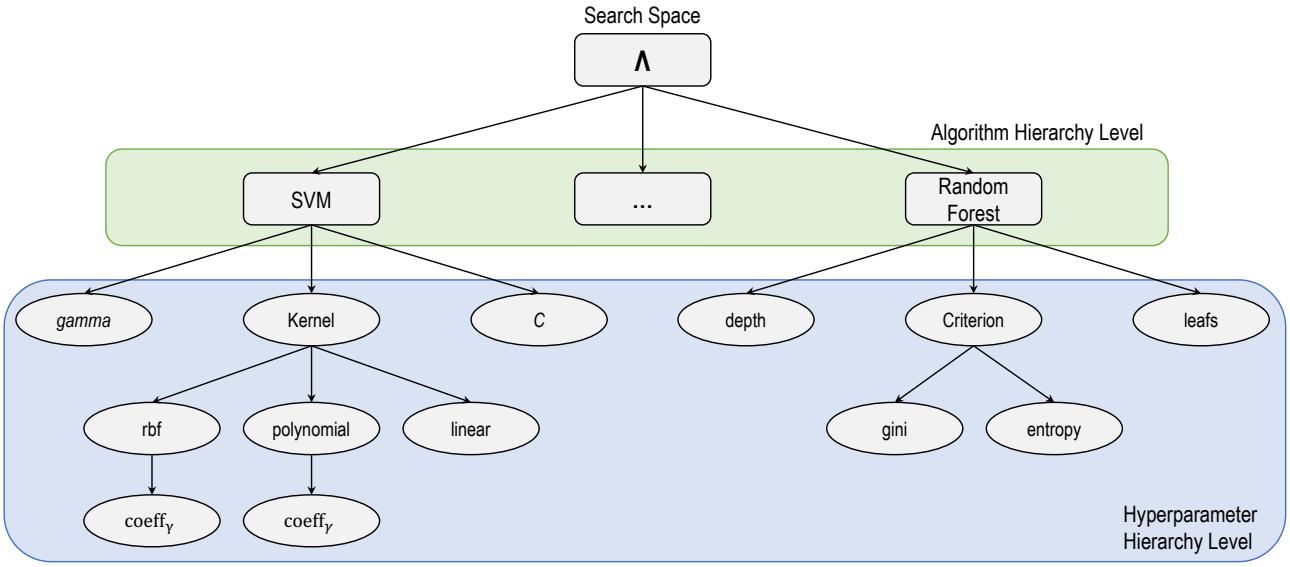


Figure 2: *Tree-like hierarchical parameter structure for the Combined Algorithm Selection and Hyperparameter Optimization (CASH) formulation. From a defined global search space Λ with all available hyperparameters, a hierarchical order must be constructed, that defines the subsets of meaningful hyperparameters to optimize during each iteration.*

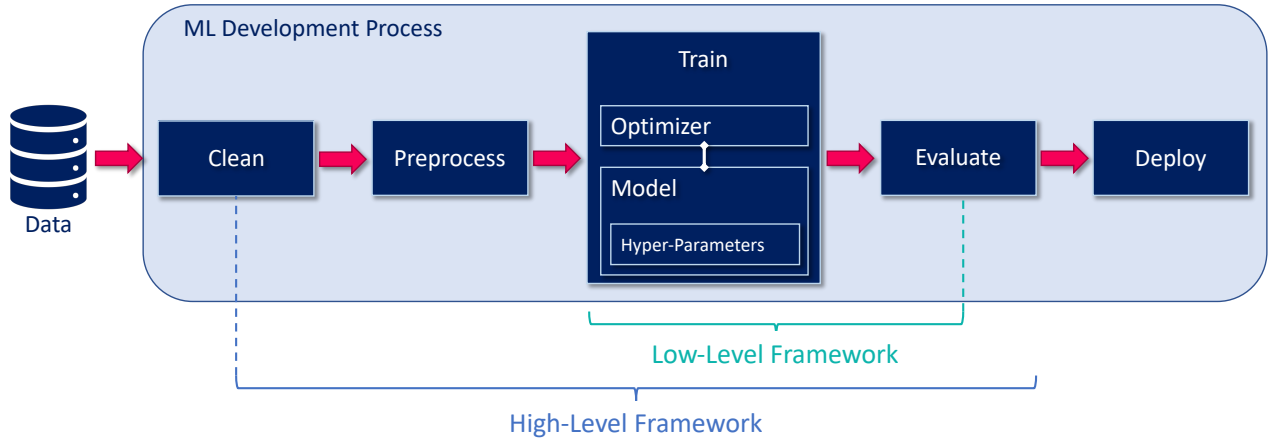


Figure 3: *Depiction of a standard ML development process from data cleaning to model deployment. The different abstraction levels of AutoML address certain processing and evaluation steps in one pipeline. The low-level framework usually automates the classical training step by testing different combinations of algorithms and hyper-parameters. The high-level framework also encapsulates algorithm-dependent data pre-processing, as well as parts of the data cleaning procedure. Note that for simplicity, the construction of multiple vertically or horizontally stacked pipelines or pipeline parts (a technique that most high-level frameworks also utilize) is not depicted here.*

3. Automated Quantum Machine Learning (AutoQML)

The synergy of integrating and utilizing QC within the AutoML context can be seen in multiple scenarios: (1) Using AutoML to select and configure suitable QML algorithms, (2) using quantum algorithms to improve the optimization process within AutoML, and (3) developing and providing decision criteria tailored for QML algorithms. These approaches have different requirements inside of the AutoML modelling and thus require different types of algorithms and pipeline elements, as well as frameworks that provide access those elements.

1. **Using AutoML to select and configure QML algorithms**
   For this approach, the user provides a dataset to the AutoML framework. This dataset could contain features from any domain, such as finance, health, or physics. As in the classical case, the framework is expected to analyse this dataset and draw conclusions about helpful preprocessing and feature engineering steps. Afterwards, QML algorithms are employed to train on the data. The framework is expected to select the most appropriate QML algorithm for the task and optimize its hyperparameters in an efficient and meaningful way. It then returns an optimized pipeline, which is a sequence of data processing elements, for the given problem.

   This approach describes the classical usage scenario of AutoML in the *high-level* abstraction layer defined earlier. The user can solely provide the data and problem type for the task without the need to understand the intricate details of the underlying processing steps, applied QML algorithms, or the QC backends. This is highly advantageous for individuals who might not have expertise in ML or QC, as they can still utilize its capabilities without diving into the complexities.

2. **Using quantum algorithms to improve AutoML optimization**
   In the second application, the focus shifts to using quantum computing to optimize the AutoML process itself. Since hyperparameters are crucial in determining the performance of an algorithm, AutoML frameworks usually search through a vast parameter search space Λ (c.f. Figure 2) to find the best combination. Quantum computing has the potential to significantly expedite this search. By developing and testing quantum algorithms such as Quantum Bayesian Optimization [16], the AutoML framework can be enriched by alternative optimization approaches.

   This approach changes the optimizer used by the AutoML framework, which we normally do not have access to in the high-level frameworks. Thus, this use-case necessitates a more low-level interaction with the AutoML library. Here, the focus is on the optimization process itself and the user needs to understand (Q)ML algorithms, search spaces and hyperparameters.

3. **Developing decision characteristics for QML**
   Besides the integration of QC into the AutoML context on the technical level, another important aspect of AutoQML is the essential aspect of determining how to make educated decisions on QML algorithms. This can be done by supporting developers with selecting QC algorithms and backends based on their characteristics, establishing meta-learning pipelines, tuning hyperparameters, and providing them with the tools to efficiently test and evaluate their quantum algorithms. The potential synergies of combining such automation approaches with the QC domain are:

   - QML algorithms have inherent attributes that make them more suitable for specific tasks like solving of linear systems, searching or combinatorial optimization due to the unique properties of quantum mechanics, such as superposition, entanglement, and quantum interference. Automatic decision criteria that recognize these properties can efficiently choose between classical and quantum algorithms for a given task.

   - Applying aspects of meta-learning like best-practice pipelines, where *a priori* knowledge is either available or the system extracts it from previous tasks, the development of QML algorithms can benefit from past experiences. By studying prior QC runs and training sessions, the AutoML framework can either use refined and generalized rules or predict the best pipeline for new datasets, thus minimizing the computational cost and increasing efficiency.

   - Using an established AutoML framework to tune hyperparameters of QML algorithms ensures that the optimization itself is well-tested, continuously revised, and has in general a high quality.

   - As the QC field progresses, there is a growing need for tools that enable developers to test, evaluate, and benchmark their QML algorithms effectively and in a standardized way. This includes the ability to automatically run a set of (classical and QML) algorithms in their problem tailored configuration without much programming overhead, the comparison of performance metrics of those algorithms, and highlighting potential areas of improvement e.g. in terms of speed, accuracy, and resource consumption.

4. Framework Selection Approach

Software selection in general is a challenging process due to either the abundance or oversupply of options, limited time, and the complexity involved in understanding and evaluating these options. Integration with existing applications and meeting diverse requirements from predefined use-cases add to the complexity. Moreover, it's imperative to build consensus in tool handling and acceptance among the different user groups (AI and QC developers) as their differing expectations and requirements can influence the success of the chosen framework.

For selecting the best-fitting framework, we draw inspiration from the software selection recommendations of Capgemini [1] and use a four-phase approach, visualized in Figure 4. The phases gradually increase in the number and detail of requirements to the evaluated frameworks and consist of the following steps:

- **Phase 1**: Following [1], we compile a *market overview* of existing AutoML tools, which is a candidate list of available frameworks. Additionally, basic programmatic and technical requirements from both user sides are collected: the classical AI developers familiar with the ML algorithms and AutoML setting but lack in-depth knowledge in the quantum computing domain, and QC developers which usually have oppsite knowledge levels. These requirements are merged and added to the list as additional features. For each entry, we have classified the features into two categories: *hard* and *soft* features, which are distinct in the difficulty of acquisition and used to filter down the list in the following phases. This is done using a ranking system where each framework can collect points to reach a score between 1 (worst) and 5 (best). At the time of writing, no existing AutoML framework had native support for the special requirements of QC algorithms (described in phase 4), which will be implemented as wrappers around or inside the chosen frameworks.

- **Phase 2**: The overview of frameworks is roughly filtered in a first step by excluding those tools which do not meet the requirements of the hard feature set. Hard features are purely based on absolute metrics, that can be directly compared between the selected frameworks. Those features include the popularity (ranked by GitHub Stars), novelty and support (time of last commits and release, number of contributors), programming language and type of licence. Decision criteria for the hard features are: (1) must be under active development, (2) must have an active community, (3) must be open-source, (4) must have a Python interface, (5) must support the requirements of all considered real-world use-cases (described in Section 5), (6) must have a permissive license. Frameworks which do not meet the requirements are assigned a score of 1, the rest are added to a *long list* of candidates.

- **Phase 3**: The long list of frameworks is then further reduced using the collection soft features. Soft features are more diverse and include information only available after closer review of the documentation and / or source code. Those features are also not always directly comparable between the libraries, since e.g., different technology backends

or algorithms are not necessarily better or worse than another solution. The collected features include implemented ML backends, supported input data types (e.g., tabular, images, etc.), supported ML problem types (e.g., classification, regression, etc.), type and size of the search space, implemented optimizers, and the categorization into the high- and low-level frameworks introduced in Section 2. Depending on how aligned they are to these requirements, frameworks are assigned a score between 2 - 5. Frameworks ranking the highest are added to the *short list* of candidates.

- **Phase 4**: Frameworks included in the filtered short candidate list are all in principle a suitable choice, because they fulfill the collected hard and soft requirements. In addition to the prerequisites of the previous phases, two aspects are considered in the last phase: (1) feedback from the use-case studies regarding the experiences made with different applicable frameworks (see Section 5) and (2) implications of current framework limitations for QC. Since the objective is the integration of QC-native or -infused QML algorithms into the AutoML paradigm, specific challenges for quantum algorithms are collected from experts interviews and an estimation is made how well these challenges can be addressed by the frameworks included in the short candidate list. Current challenges for QC that shall be addressed and automated by AutoQML are mostly stemming from the fact, that QC today is still in a very early state, described as the noisy intermediate-scale quantum (NISQ) era. ML methods for todays quantum hardware have to deal with a considerable amount of noise along with a limited number of qubits and short circuit depths. That necessitates feature reduction in pre-processing, post-processing in form of error mitigation, as well as extensive computation times and queuing on the real QC backends.
  As decision criteria, we consider support and integration capabilities for QC-specific data encoding, backend selection and analysis utility, automatic and meaningful feature selection (e.g., by dimesionality reduction methods), support for the current asynchronic nature of QC and algorithm queueing as well as hardware constraints from the quantum computers themselves.

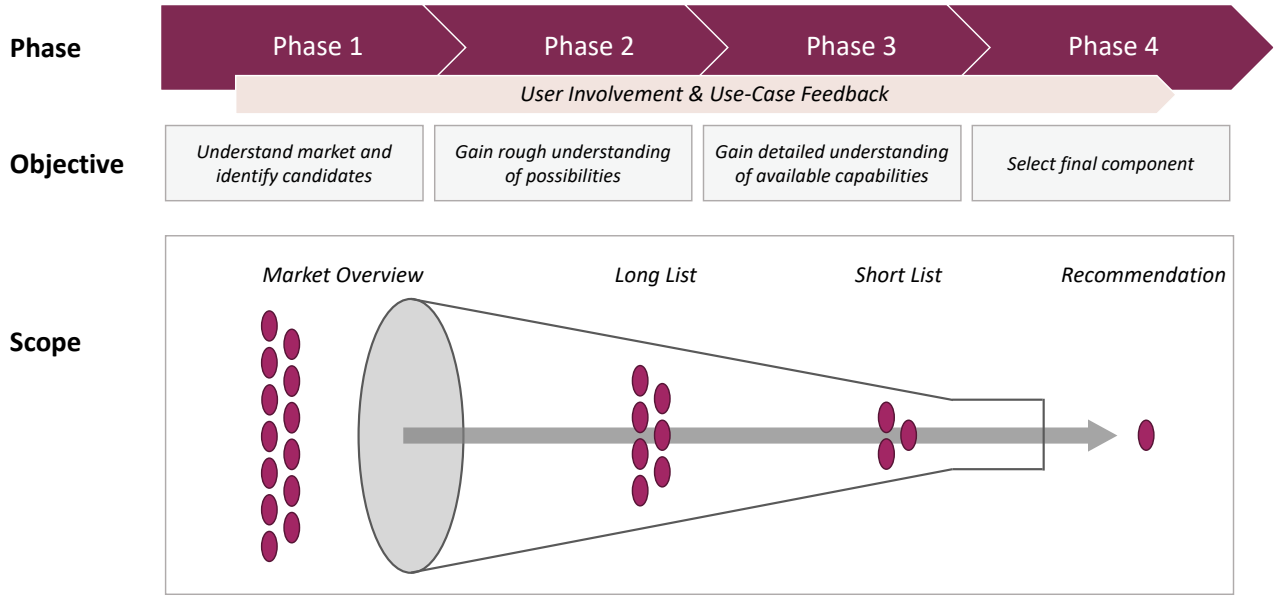


Figure 4: *Framework selection approach following [1]. The approach consists of four phases which gradually narrow down the available software tools to an appropriate recommendation. The different phases are described in detail in the text.*

Selection Results

Our results of the phases 1 to 3 - the collection of candidates as a market overview and the ranking of frameworks according to the hard and soft criteria - are exemplary shown in Table 3 and the link to the full candidate list is provided in Appendix 7.1. Additional considerations are made to later integrate the final framework into a platform ecosystem like PlanQK[1], which is not specifically represented in the candidate list.

Not all aspects of Section 3 can be completely mapped to either a high- or low-level framework only. Thus, both framework types are required and the phases 3 and 4 are executed for both types respectively, resulting in a short list and final decision for both framework types. In our case, the short list after phase 3 consists of the following frameworks, which got a score of ≥ 4:

Table 1: *Final short list of frameworks after the* 3^{rd} *phase for low- and high-level respectively.*

Low-level Frameworks	**High-level Frameworks**
Ray	FLAML
syne-tune	NNI
Optuna	AutoGluon
SMAC3	

Afterwards, feedback from the participating use-case companies is collected regarding the required abstraction level of AutoML, usability, interpretability, framework constraints and performance of the tested candidate frameworks, described in detail in Section 5. Additional consideration of the QC-specific requirements from phase 4 and feedback from the QC developer interviews described in Section 5.2 result in the final framework choices:

- **Ray** for the low-level framework and
- **AutoGluon** for the high-level framework.

With an origin in parallel computation, Ray provides a highly flexible and performant solution for the CASH formulation with many optimizers to choose from in a plug-and-play style. Additionally, the framework is extremely popular, well maintained and documented. As a framework maintained and developed by Amazon, AutoGluon provides a similar level of support as well as state of the art PSO approaches like multi-level ensemble construction, which are missing in many other high-level frameworks.

For the further development and extension for the chosen frameworks, we make use of the widely adopted design patterns of the Scikit-Learn library [17], used nowadays in many high-level libraries, adapters, and wrappers.

[1]PlanQK (Plattform und Ökosystem für Quantenapplikationen, `https://planqk.de`) is a German initiative for providing a platform for researchers and industry to exchange and provide quantum software solutions in an app-store like way.

5. Use-Case Study

In addition to the technical prerequisites and objectives for framework selection described in Section 4, we conduct different use-case studies. The reason for that is two-fold: (1) collect real-world requirements from end-users for the hard feature set of phase 2 in the selection approach, and (2) validate the short list of candidates by implementing use-case specific proof-of-concepts (POCs) using one or more of the candidate frameworks and give further recommendations for the final decision in phase 4. For that, each participating company is asked to implement a classical ML solution for their use-case using at least one of the AutoML frameworks identified in phase 3 in Section 4. Afterwards, an interview was conducted with the ML experts who implemented the POCs, and a standardized questionnaire is filled over the course of the interview. The full questionnaire is available in Appendix 7.2.

5.1. Use-Case Description

In total, four studies with companies from different backgrounds have been conducted[2]. Additional to the ML problem formulation and input data requirements, all use-case POC's are realised with one or multiple candidate frameworks. In the following, the partners and their respective use-case are described briefly, and their general ML requirements are listed in Table 2.

5.1.1. IAV GmbH Ingenieursgesellschaft Auto und Verkehr

IAV is an engineering consultancy service provider in the mobility and IT sector. The use-case is the modelling and simulation of forces appearing at tires of cars, which from the ML perspective is a timeseries prediction task in the engine control unit. A digital twin shall be modeled that implements the behavior of the real unit. In a reduced problem formulation, there are 24 input and six output features. A time series prediction model, that operates as a filling prediction forecaster, is implemented.

5.1.2. KEB Automation KG

KEB is a system solver for automated drive technologies, also in the fields of mechanical and plant engineering. KEB Automation operates a transport system (AGILOX) for intralogistics tasks. Each AGILOX vehicle is a driverless vehicle that navigates through plants and warehouses to transport items. Occasionally, a vehicle may fail for operational or technical reasons, causing an immediate stop that requires human intervention and causes delays in operations. The AGILOX system generates data about the vehicle itself and order status and makes it available to other services. The goal is to develop an ML solution that classifies specific events on time series data.

[2]The use-case partners are also introduced here: https://www.autoqml.ai/en/use-cases

Table 2: *Overview of the requirements and ML formulations of each use-case partner. Each partner implemented their use-case with multiple candidate frameworks, also resulting in the deduction of the required AutoML abstraction level introduced in Section 2 for their specific use-case.*

	IAV GmbH	**KEB Automation KG**	**TRUMPF**	**Zeppelin GmbH**
Prerequisites	Own model definition must be possible	Commodity hardware	A CNN model shall be used	The solution should be easy to implement
Functional Requirements	High quality documentation should be available	Real-time inference	Deep learning and accelerator hardware must be supported	Framework feedback and interpretability is important
Data Type(s)	Time series (numerical)	Time series (numerical and categorical)	Images (greyscale)	Tabular (numerical and categorical)
ML Problem	Time Series Forecast	Time Series Classification	Image Classification	Tabular Regression
Tested Frameworks	Ray Optuna	Ray Optuna SMAC Auto-Sklearn	Ray AutoGluon	Auto-Sklearn FLAML AutoGluon Optuna
Resulting AutoML Abstraction Level	Low-level	Low-level	Low-level	High-level

5.1.3. TRUMPF Werkzeugmaschinen GmbH + Co. KG

TRUMPF is a market and technology leader in machine tools and lasers, including their software solutions for industrial manufacturing as well as in the smart factory field. In the context of AutoQML, TRUMPF wants to optimize sheet metal processing by a laser cutting machine. Before the process of cutting, a metal sheet is first placed on the support bars of the pallet. The pallet is then moved into the machine room, where the cutting head is located and cuts the parts out of the sheet metal. The challenge is, that cut parts may tip over and cause a collision with the cutting head, interrupting the cutting process and potentially damaging the head. To avoid tilting, the cut parts should be stabilized by support bars. Those bars must be detected beforehand from a machine-attached camera using image data with convolutional neural networks (CNN) on the classical side.

5.1.4. Zeppelin GmbH

The Zeppelin Group offers solutions in the fields of construction, propulsion, and energy, as well as engineering and plant construction. Zeppelin Baumaschinen GmbH sales and services construction machinery. They also actively trade new and used machines. Trading with used machines requires the individual appraisal and evaluation of each machine that is to be bought or sold. This requires a high level of expertise, coupled with knowledge of current price developments, and usually leads to a valuation process lasting several hours. To support this process, an (Q)ML-powered price prediction tool is developed based on daily updated market

data. This tool implements a regression model to predict the selling price of used construction machines. The importance of AutoML in this context was also shown previously in [6].

5.2. Study Evaluation

Here, we present some key messages and findings from the interview evaluation. The complete study feedback (in German) is referenced in Appendix 7.2.

We evaluate the feedback from the POC's over all use-cases and with respect to their individual ML problem types stated in Table 2. The answers to some questions can be subjective due to the fact of diverse prerequisites of each use-case, but also because of different familiarity levels in AutoML of the individual use-case experts and their preferences. Thus, the shown rankings are not directly generalizable to different use-cases and applications with the same ML problem type since prerequisites and functional requirements for each use-case partner also played a role in their feedback. This is also the reason, why the same framework can occur on both sides of the ranking spectrum (see e.g., the Ray framework in the predictive power plot of Figure 5).

The frameworks NNI and syne-tune were not tested by the use-case partners. The reason for that is that NNI only works on neural network algorithm classes, which only the image classification use-case had as requirement but decided to focus on the low-level optimization frameworks, and syne-tune is the underlying native optimizer for the high-level framework AutoGluon, which was a prominent choice already.

5.2.1. Quality Aspects

In general, the predictive quality of the resulting models created by the tested frameworks was rated as good or very good by the ML experts in 70% of all tested frameworks and use-case implementations, confirming the general potential of AutoML as a good approach for initial ML pipeline implementation. The high-level framework AutoGluon was reported to perform very bad in the image classification tasks in terms of resulting model performance and framework configuration possibilities, as shown in Figure 5. This can mainly be attributed to the prerequisite of the use case: the experts perceived the task of incorporating and enforcing the use of specific models in the framework as difficult and partially not possible. On the other hand, the Ray framework was reported to achieve good overall results in every shown quality metric except for the model performance in the time series forecasting setting. Training times of the frameworks and inference speed of the resulting models were generally reported to be more than sufficient for the users among all use-cases. Only auto-sklearn and Optuna had bad training time requirements in the tabular regression setting. Our findings are in line with [6].

Especially the tested low-level frameworks, which only do HPO or CASH, comply with the predefined budget limit. PSO frameworks like auto-sklearn did not always meet that requirement. In general, apart from sub-optimal implementation, we suspect this effect can occur due to the construction of whole pipelines which take longer to evaluate and makes violation of, e.g., time restrictions easier. From the evaluated use-case implementations, many partners reported that Optuna, while easy to use with a clean code basis, is too simple with too few SOTA optimizations for AutoML to deliver good results (both in configurability and model performance).

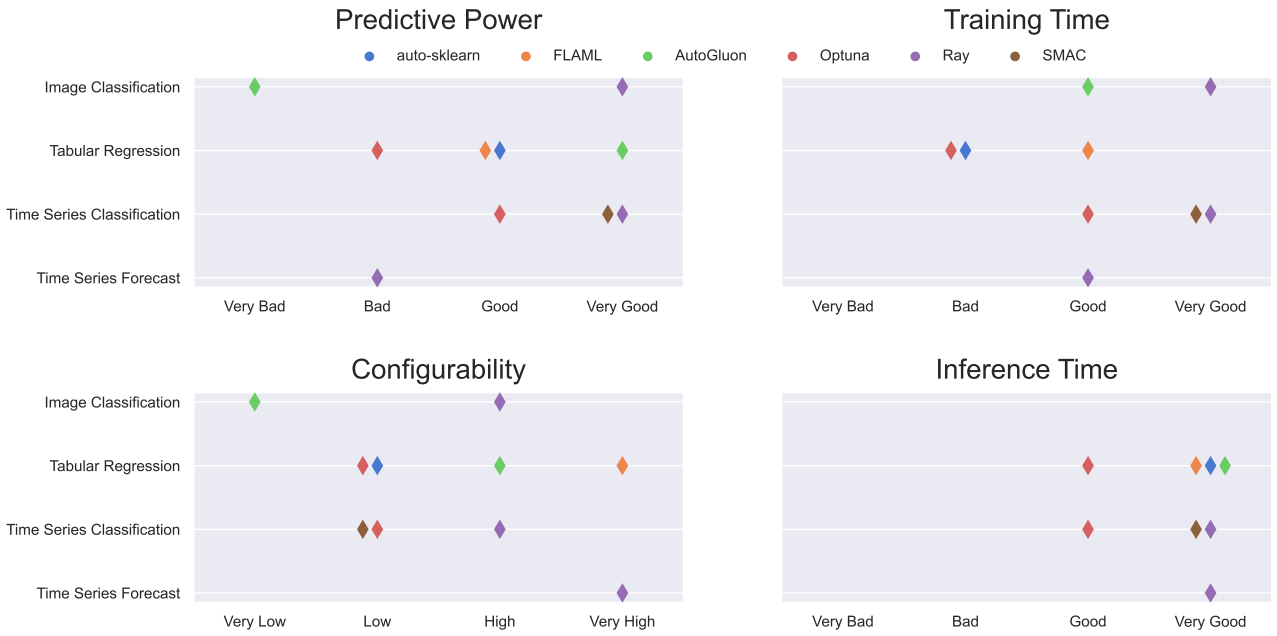


Figure 5: *Evaluation results from the quality related assessments of the use-case study. Feedback is depicted for the final model performance (top left) and degree of configurability (bottom left), as well as the required training (top right) and inference time (bottom right) for each framework. Ratings are shown w.r.t. the respective ML problem type of each interview partner and their tested frameworks. If problem types have no or missing data points (e.g. inference time for the image classification task), the users reported that this has no relevance for their use-case and framework.*

5.2.2. Resource and API aspects

Some computing resource demands (CPU, RAM / VRAM) are reported in Figure 6 (top left and right respectively). Resource consumption for AutoML frameworks is in general difficult to compare between different frameworks. Due to their mostly parallelized evaluation of many candidate configurations, they tend to be resource intensive programs with high demands for hardware, especially when optimizing deep learning models that utilize accelerator hardware. This was, however, expected or known by all developers. While mostly SMAC and Optuna were perceived to have unjustified high CPU and (V)RAM requirements, especially well established or recent frameworks have very good resource utilizations. The aspects of model size or complexity were mostly not relevant for the experts.

In general, the documentation, code examples and community help have been reported having high quality. The API of the frameworks was perceived easy to use, with less boilerplate or glue code necessary to create a working solution for the use-cases, except for Ray in some respect. This is shown in Figure 6 (bottom left). Particularly, low-level frameworks that require much more user interaction and adaption, or frameworks that did not fit to the use-case prerequisites, were reported to be more difficult to use.

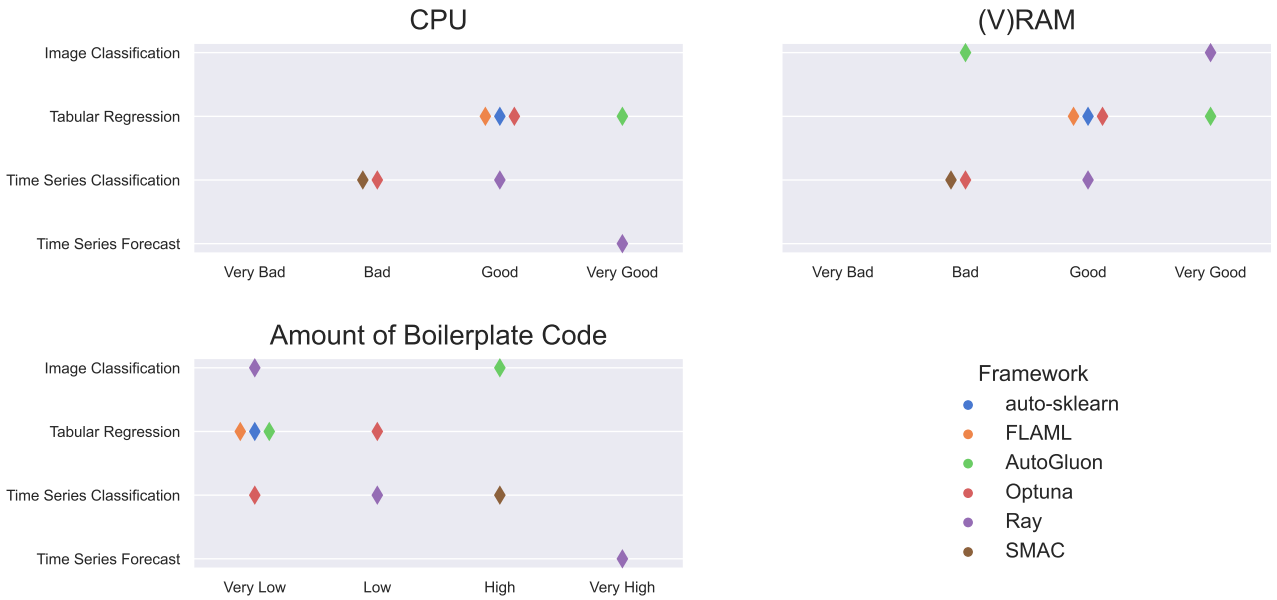


Figure 6: *Interview feedback regarding resource and efficiency aspects of the different frameworks, classified by each ML use-case type. User ratings are shown in terms of CPU and (V)RAM utilization during training (top left and right). If a framework uses accelerator hardware, findings are reported for the combination of RAM and VRAM utilization. Additionally, the required amount of boilerplate / glue code is shown (bottom left), that was required by the developers to implement their use-case solution for each framework and the given requirements from Table 2. Not all use-cases gave feedback on every aspect or framework. A missing data point in the plot means that this aspect was not relevant for the implementation of the respective use-case.*

5.2.3. Quantum Aspects

An individual interview was conducted with quantum computing and software development experts from Fraunhofer[3] regarding their future requirements when implementing quantum algorithms and utility functions into a chosen framework. A specification is, for example, that QML algorithms must be able to be included via wrapper functions into the existing framework, which also support HP configuration. Additionally, simulators must be supported as well as real hardware backends, while providing utility methods to automatically handle long queueing times and session handling. It should also be possible to include external tool for evaluating the economical, execution and hardware restrictions of the submitted code. QC libraries that should be supported include IBM Qiskit[4] and Xanadus PennyLane[5], together with standard Python data science and AI packages.

These requirements mainly address the extensibility and code quality aspects of the reviewed frameworks and were additionally considered in the final choice as explained in Section 4.

5.2.4. Additional Findings

In general, the usability of all frameworks is high. While almost all the candidates in the short list are well documented, they come with some initial setup costs. The libraries usually

[3]Interviews were conducted with experts from the Fraunhofer instituts for "Industrial Engineering (IAO)" and "Manufacturing Engineering and Automation (IPA)"
[4]https://www.ibm.com/quantum/qiskit-runtime
[5]https://pennylane.ai

have a high number of external dependencies making an integration into existing software environments more difficult. This holds especially for the high-level frameworks since they often include multiple backends and additional utility libraries. However, most of the use-case partners reported, that integration was quite easy due to the possibility to install the framework via package and dependency managers like *pip* or *conda*, which takes care of most of the package and version conflicts automatically.

The study also confirmed the automation motive of AutoML of [6] in terms of required knowledge levels for different ML and data science tasks necessary to manually implement a ML pipeline: High-level frameworks like AutoGluon, FLAML and Auto-Sklearn dont require the users to be proficient in the used ML algorithms or optimization strategies. The low-level frameworks like Ray, SMAC, and Optuna instead demand a more profound understanding of the individual algorithms, according to the interview feedback. The required knowledge for data preprocessing was reported high or very high for all tested frameworks, emphasizing the fact that every use-case had to implement additional preprocessing techniques and could not rely exclusively on the already available functionality, even in the high-level libraries. The necessary basic programming skills for each framework were reported with a high variance, leading to our conclusion that this is a highly subjective topic depending on the individual skills of the interview partner.

Table 3: *Excerpt of the framework candidate list. Frameworks, that didnt meet the hard feature requirements from got a rating value of 1 and were not analysed further. For example, Hyperband and Advisor were considered not under active development and community participation anymore due to their years of inactivity. H2O is written in Java and thus doesnt provide the desired technology stack. The remaining frameworks were analysed in more detail and rated according to their suitability for the considered use-cases and extensibility.*

Framework	GitHub Stars	Last Commit	Last Release	Nr. of Contrib-utors	Language	License	Supp. ML Libraries	Input Types	ML Problems	Search Space	Optimizer	Abstract. Level	Rating
Hyperband	572	13.09.2017		1	Python	BDS 2	-	-	-	-	-	-	1
Advisor	1500	11.11.2019	18.10.2018	9	Python	Apache 2.0	-	-	-	-	-	-	1
H2O 3	5900	04.08.2022	03.08.2022	160	Java	Apache 2.0	-	-	-	-	-	-	1
Auto-PyTorch	1700	21.07.2022	18.07.2022	15	Python	Apache 2.0	PyTorch	Tabular Data Time Series	Classification Regression Time Series Forecast	Neural Networks	Random Search BayesOpt Portfolios	High	2
TPOT	8700	29.07.2022	06.01.2021	73	Python	LGPL 3.0	sklearn	Tabular Data	Classification Regression	Preprocessing Classifier Regressor	Evolutionary	High / Low	2
Auto-Sklearn	6400	03.08.2022	18.02.2022	77	Python	BSD 3	sklearn	Tabular Data	Classification Regression	Preprocessing Classifier Regressor	smac	High	2
Hyperopt	6300	29.11.2021	17.11.2021	90	Python	Custom	Arbitrary	Arbitrary	Arbitrary	Arbitrary	BayesOpt	Low	3
Ray	21400	04.08.2022	09.06.2022	709	Python	Apache 2.0	Arbitrary	Arbitrary	Arbitrary	Arbitrary	Grid Search Random Search Evolutionary Other AutoML libs	Low	5
AutoGluon	4700	03.08.2022	29.07.2022	81	Python	Apache 2.0	sklearn PyTorch	Tabular Data Images Text	Classification Regression	Classifier Regressor Neural Networks	syne-tune ray-tune	High	5
Optuna	6700	03.08.2022	13.06.2022	175	Python	MIT	Arbitrary	Arbitrary	Arbitrary	Arbitrary	Grid Search Random Search BayesOpt Evolutionary	Low	5

6. Summary & Future Work

We present our systematic approach for the multi-criteria selection and assessment process of AutoML frameworks regarding their capability of solving real-world use-cases and extensibility with quantum computing algorithms. For that, we classify the existing AutoML frameworks into high- and low-level types and follow the software selection approach proposed by Capgemini [1] and adapt each phase to our specific needs. While collecting requirements from both end-user sides (ML and QC developers), we in parallel conduct a use-case study with four different use-cases and assess the fitness of candidate frameworks by conducting expert interviews with the developers of each POC. The types of use-cases are in the ML domains of classification, regression and forecasting, and work with tabular, time series and image input data. We evaluate the candidate frameworks during the selection phases on the use-cases based on quality aspects of the produced solution (e.g. predictive power, configurability, time requirements) and efficiency dimensions (e.g. hardware utilization, quality-of-life aspects).

Our results show, that there is a need for having functionalities from both: high- and low-level framework types and especially the resulting performance quality is satisfactory for most use-cases and tested frameworks. After consideration of all requirements and the study feedback, the final choices for the AutoQML framework are Ray (low abstraction level) and AutoGluon (high abstraction level) due to their fulfillment of all requirements during the selection phases and overall best performance in the use-case evaluation. Supplementary material is provided on GitHub at this URL (`https://github.com/AutoQML/Whitepaper-Framework-Selection`).

Following this assessment, an open-source AutoQML framework will be implemented that combines AutoML and QC. For that, stand-alone QML algorithms will be developed which can be later integrated directly or as an independent backend into an existing AutoML library. Additionally, the chosen AutoML framework(s) need to be extended by QC-specific preprocessing, search space and training requirements, which can either be integrated into the open-source software, or included as a wrapper library which utilizes the underlying automation functions.

Acknowledgements

We thank the companies and especially the developers for implementing the POC's and providing valuable feedback during the use-case study.

This work was created in the context of the AutoQML project, funded by the Federal Ministry for Economic Affairs and Climate Action. The project website can be found at this URL: `https://www.autoqml.ai/en`

7. Appendix

7.1. Candidate list of open-source frameworks

The full candidate list is too long to be included in this document. Thus, the full Excel list of candidates, together with the hard and soft selection features and the framework scores can be accessed in this GitHub repository[6].

7.2. Questionnaire and results of the ML expert interviews

This is the survey used in the ML expert interviews with the use-case partners created with Microsoft Forms. The questionnaire, together with the respective evaluation results, is also available in the GitHub repository[7].

[6] https://github.com/AutoQML/Whitepaper-Framework-Selection/blob/main/doc/AutoML-Framework-Overview.xlsx

[7] https://github.com/AutoQML/Whitepaper-Framework-Selection/blob/main/doc/Interviews/Questionnair_Framework-Use-Case-Study.pdf

References

[1] F. Middendorf and G. Kamann. *Software Selection: Managing the complexity of choosing the right software*. Capgemini Consulting, 2016.

[2] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[3] M. A. Zöller, T.-D. Nguyen, and M. Huber. Incremental search space construction for machine learning pipeline synthesis. In *International Symposium on Intelligent Data Analysis*, 2021.

[4] M. A. Zöller and M. Huber. Benchmark and Survey of Automated Machine Learning Frameworks. *Journal of artificial intelligence research*, pages 409–472, 2021.

[5] Y. Quanming, W. Mengshuo, C. Yuqiang, D. Wenyuan, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Yu Yang. *Taking Human out of Learning Applications: A Survey on Automated Machine Learning*. arXiv:1810.13306, 2018.

[6] 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.

[7] P. Rebentrost, M. Mohseni, and S. Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13), 2014.

[8] K. Beer, D. Bondarenko, T. Farrelly, T. J. Osborne, R. Salzmann, D. Scheiermann, and R. Wolf. Training deep quantum neural networks. *Nature Communications*, 11(1):808, 2020.

[9] M. Feurer and F. Hutter. Hyperparameter optimization. In *Automatic Machine Learning: Methods, Systems, Challenges*, pages 3–38. Springer, 2019.

[10] N. Awad, N. Mallik, and F. Hutter. *DEHB: Evolutionary Hyperband for Scalable, Robust and Efficient Hyperparameter Optimization*. preprint arXiv:2105.09821, 2021.

[11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, (24), 2011.

[12] P. Kerschke, H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computations*, 27(1):3–45, 2019.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, and O. Grisel. Scikit-

learn: Machine learning in python. *Journal of Machine Learning Research*, (12):2825–2830, 2011.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, and J. Bradbury. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, and G. S. Corrado. In *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, pages 265–283, 2015.

[16] F. Rapp and M. Roth. *Quantum Gaussian Process Regression for Bayesian Optimization.* arXiv:2304.12923, 2023.

[17] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013.