

AutoQML: A Framework for Automated Quantum Machine Learning

Marco Roth,^{1,*} David A. Kreplin,^{1,†} Daniel Basilewitsch,² João F. Bravo,³ Dennis Klau,³
Milan Marinov,⁴ Daniel Pranjic,³ Horst Stuehler,⁵ Moritz Willmann,¹ and Marc-André Zöller⁴

¹Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Nobelstraße 12, 70569 Stuttgart, Germany

²TRUMPF SE + Co. KG, Quantum Applications Group, Johann-Maus-Straße 2, 71254 Ditzingen, Germany

³Fraunhofer Institute for Industrial Engineering IAO, Nobelstraße 12, 70569 Stuttgart, Germany

⁴USU GmbH, Rüppurrer Str. 1, 76137 Karlsruhe, Germany

⁵Zeppelin GmbH, Graf-Zeppelin-Platz 1, 85748 Garching, Germany

(Dated: March 3, 2025)

Automated Machine Learning (AutoML) has significantly advanced the efficiency of ML-focused software development by automating hyperparameter optimization and pipeline construction, reducing the need for manual intervention. Quantum Machine Learning (QML) offers the potential to surpass classical machine learning (ML) capabilities by utilizing quantum computing. However, the complexity of QML presents substantial entry barriers. We introduce *AutoQML*, a novel framework that adapts the AutoML approach to QML, providing a modular and unified programming interface to facilitate the development of QML pipelines. AutoQML leverages the QML library *sQULearn* to support a variety of QML algorithms. The framework is capable of constructing end-to-end pipelines for supervised learning tasks, ensuring accessibility and efficacy. We evaluate AutoQML across four industrial use cases, demonstrating its ability to generate high-performing QML pipelines that are competitive with both classical ML models and manually crafted quantum solutions.

I. INTRODUCTION

A key factor in the success and democratization of machine learning (ML) has been the development of increased abstraction levels, which facilitate rapid prototyping and lower entry barriers. Automated machine learning (AutoML) aims to automate the predominantly manual process of ML pipeline construction, representing a significant progression in this development [1]. This approach has proven successful by enhancing the efficiency of specialists, allowing them to focus more on modeling business problems rather than on implementation details [2]. Furthermore, the reduced expertise required to use these tools democratizes ML methods, allowing companies with less experience to integrate ML-based solutions into their workflows [3]. This is a particularly relevant consideration in market environments that are increasingly impacted by labor shortages, notably in specialized domains such as ML [4].

Quantum machine learning (QML) employs quantum computers to develop ML algorithms that harness quantum mechanical principles, with the goal of expanding the capabilities of ML beyond the classical limits [5–7]. As an interdisciplinary field requiring expertise in quantum computing, ML, and computer science, the entry barriers to this technology are particularly high. In this work, we introduce the framework *AutoQML*¹, which aims to transfer the success of AutoML from the classical to the quantum realm, making QML accessible to a broad audience in science, technology and industry.

Using quantum computers for ML instead of classic hardware presents a unique set of challenges. Providing a diverse suite of QML algorithms that can be orchestrated using AutoML techniques, such as combined algorithm selection and

hyperparameter optimization (CASH) [8] requires a modular implementation with a unified programming interface. Additionally, a seamless transition from classical preprocessing and simulation to real quantum computers needs to be ensured. AutoQML builds on the QML library *sQULearn* [9], which offers a variety of QML algorithms with a scikit-learn [10] programming interface to create a set of modular and diverse QML methods. The library leverages PennyLane [11] and Qiskit [12], enabling the execution of algorithms on multiple simulators and quantum computers, such as IBM Quantum [13] and various backends available through Amazon Braket [14].

AutoQML creates end-to-end QML pipelines for various supervised learning scenarios, such as time series classification, tabular regression, and image classification. Using the open-source libraries *Optuna* [15] and *Ray Tune* [16], the framework offers fully optimizable pipelines, including quantum-specific preprocessing, to make QML accessible to non-experts. In designing AutoQML, we have anticipated future developments in quantum computing and focused particularly on modularity, allowing for easy extension of the algorithm pool.

In this work, we outline the architecture of the AutoQML framework. We benchmark the framework on four distinct industrial use cases involving time series and image classification, as well as tabular and time-series regression. The results are compared to classical solutions and manual quantum computing pipelines. This study aims to demonstrate the capability of AutoQML in facilitating the development of effective QML solutions in various domains.

The remainder of this work is structured as follows. Following the related work in Sec. IA, Sec. II outlines the architecture of the AutoQML framework. Section III describes the industrial use cases to evaluate the performance of the framework. Section IV presents the experimental results, comparing AutoQML-generated pipelines with both manually crafted QML pipelines and classical ML approaches. Finally, Secs. VI and VII conclude with a summary and discussion of our findings and potential future research directions.

* marco.roth@ipa.fraunhofer.de

† david.kreplin@ipa.fraunhofer.de

¹ Code available at <https://github.com/AutoQML/autoqml>

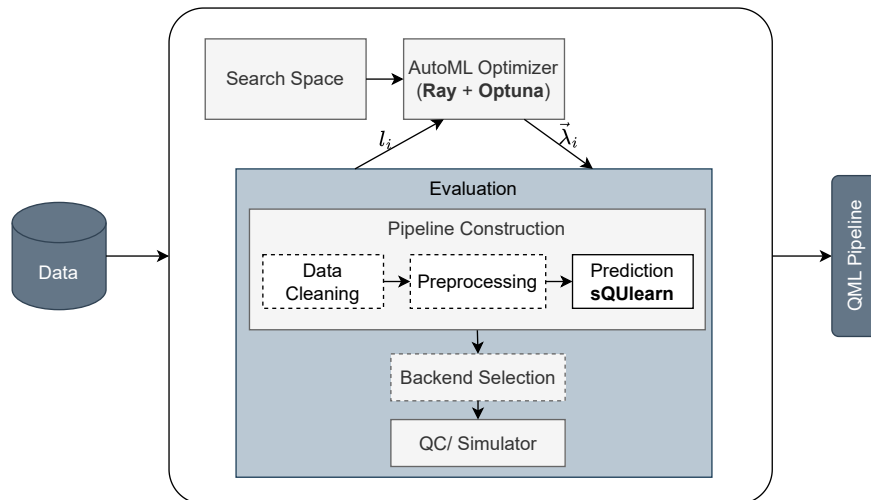


FIG. 1. Architecture overview of the AutoQML framework. Data is supplied by the user. Using Ray and Optuna, AutoQML constructs a pipeline that is optimized over a preconfigured search space. A loss value l_i is obtained for each configuration λ_i , which consists of data cleaning, preprocessing, and a model with hyperparameters evaluated using a simulator or a real quantum computer (QC). After a given budget is exhausted, the best-performing pipeline is returned to the user. Optional pipeline steps are indicated as dashed boxes.

A. Related Work

The term AutoQML has been used previously, primarily emphasizing the optimization of specific QML models rather than addressing the optimization of the complete pipeline, which includes preprocessing and the selection from different QML models. This term has been applied to hyperparameter optimization for a fixed model [17], as well as in the context of architecture search for optimizing the encoding of data into quantum states [18, 19]. In these architecture search efforts, the QML model remains fixed while the encoding circuit is optimized for the dataset, aiming to improve the performance of the given model [20–23]. Our work distinguishes itself from these prior approaches by presenting, to the best of our knowledge, the first comprehensive framework that constructs and optimizes end-to-end QML pipelines. Unlike previous efforts that focused on optimizing specific QML models or encoding circuits, our approach emphasizes the integration of diverse QML methods, enabling the development of adaptable and effective QML solutions tailored to various application scenarios.

II. FRAMEWORK

The foundation of AutoQML are the open-source optimization, parallelization, and scheduling frameworks Ray Tune and Optuna. The algorithm pool is constituted by scikit-learn and sQULearn, with extended functionalities designed to optimize and streamline the development of QML pipelines. Neural network-based approaches such as autoencoders for preprocessing are implemented using PyTorch [24].

A. Pipeline Creation

The framework follows a standard AutoML approach known as *pipeline synthesis and optimization* [1, 25], designed to create complete (quantum) ML pipelines. This involves searching for the optimal pipeline structure and performing CASH optimization. For a given problem type such as tabular regression, AutoQML constructs pipelines based on best-practice templates, where the sequence of steps is predefined but the specific methods are. Each pipeline explores various data cleaning and preprocessing steps, along with testing available models and their hyperparameters. The available templates can be easily extended to accommodate additional scenarios.

Figure 1 provides a high-level overview of the AutoQML architecture. It is primarily inspired by state-of-the-art AutoML approaches, such as those introduced in Hutter *et al.* [26] and Feurer *et al.* [27], and adheres to the design principles of scikit-learn [28] to provide a well-known, standardized programming interface. Users are required to provide an input dataset and choose a suitable problem and data type, such as tabular regression.² Similar to AutoML for non-quantum ML, the AutoQML framework samples (quantum) ML pipelines from a pre-defined search space for automated optimization. This search space includes the necessary steps to create an end-to-end pipeline for (quantum) ML predictions, which can be categorized into three steps:

1. *Data Cleaning* is responsible for eliminating potential defects from the input data. It includes imputation of

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

missing values, outlier removal, and encoding of categorical features.

2. *Preprocessing* transforms the data into a format suitable for quantum computing. This involves dimensionality reduction, down-sampling, feature-centric and rescaling.
3. *Prediction* uses classification or regression algorithms to generate the actual predictions. The available QML algorithms are implemented via sQLearn. In addition, a subset of classical ML methods are optionally available.

Most steps in the pipeline are facultative and can be combined in various ways to create diverse of ML pipeline configurations.

The set-up for the optimization is as follows. Given a complete search space description, the optimizer iteratively draws new test configurations i , denoted as $\vec{\lambda}_i \in \Lambda$, where Λ is a configuration space. A configuration encapsulates an abstract specification of a particular (quantum) ML pipeline. Following the CASH optimization procedure, each configuration $\vec{\lambda}_i$ jointly describes algorithms for (quantum) ML models, data cleaning, preprocessing, and prediction steps, as well as their associated hyperparameters. More specifically, for a given data set $\mathcal{D} = (X, y)$, with features $X = (\vec{x}_1, \dots, \vec{x}_N)$ from some feature space $\mathcal{X} \ni \vec{x}_j$ and labels $y \subset \mathbb{R}$, we solve the following optimization problem [1]

$$\vec{\lambda}^* = \arg \min_{\vec{\lambda} \in \Lambda} l(\mathcal{D}, \vec{\lambda}). \quad (1)$$

Here, l is a suitable loss function. AutoQML thus not only optimizes over quantum models and their hyperparameters but also over the type of preprocessing algorithms (e.g. PCA) and their configuration (e.g. the number of retained principal components). Pipeline examples are shown in Tab. I.

To solve Eq. (1) AutoQML utilizes probabilistic models to guide the search for high-performing configurations. It leverages Bayesian optimization using tree-structured Parzen Estimator (TPE) [29]. The search algorithm constructs a Gaussian mixture model to approximate the loss (e.g., validation loss or accuracy) based on the results of previous evaluations. For a given ML problem, the optimizer performs the following steps: (i) For a new pipeline i , a configuration $\vec{\lambda}_i$ is drawn from Λ by sampling close to the optimal point of a probabilistic model, which favors promising regions of the search space. (ii) The proposed configuration $\vec{\lambda}_i$ is passed to the evaluation function, yielding a performance score l_i . This score reflects how well the pipeline performs on a validation set. (iii) The tuple $(\vec{\lambda}_i, l_i)$ is used to update the probabilistic model of the loss function. The model is refined to more accurately represent the relationship between configurations and their performance. (iv) The process repeats until a user-provided budget, such as a time limit, is exhausted. When the optimization process is finished, the best-performing configuration $\vec{\lambda}^*$ and the corresponding fitted pipeline are returned to the user. This optimization process enables AutoQML to efficiently discover high-performing (quantum) ML pipelines tailored to the specific problem at hand. A code example is shown in Fig. 2.

The fixed pipeline templates in AutoQML account for quantum computing specific preprocessing. Particularly, the limited

```
# Set up pipeline with fixed order
autoqml = TabularRegression()

# provide data and options
cmd = AutoQMLFitCommand(
    X=X_train,
    y=y_train,
    time_budget=timedelta(seconds=100),
    backend='pennylane',
    configuration="quantum_regression")

# fit the pipeline
autoqml = autoqml.fit(cmd)

# predict using the best performing pipeline
y_pred = autoqml.predict(X_test)
```

FIG. 2. Example code for fitting a tabular regression pipeline. Here, it is assumed that the training data is supplied as X_{train} with corresponding targets y_{train} . Within AutoQML, the data is split into a test and validation set. Options such as the time budget `timedelta` for the optimization or the backend for execution of the QML algorithms can be specified. In the example, the preset configuration "quantum_regression" is used to restrict the search space to quantum computing based regression algorithms only.

size of current quantum computer requires a significant dimensionality reduction, e.g., through principal component analysis (PCA) or autoencoders. Additionally, the encoding of classical data in AutoQML is done using pre-defined encoding circuits based on angle-encoding. This usually requires scaling the features to avoid non-injective maps.

B. QML Integration

The QML algorithms are provided by sQLearn. The library offers several high-level methods such as quantum neural networks (QNN) [30], quantum reservoir computing (QRC) [31], and various kernel methods such as quantum kernel ridge regression (QKRR) [32], quantum support vector machines (QSVM) [33], and quantum Gaussian processes (QGPR) [34]. These methods are accessible through a scikit-learn programming interface, allowing for user-friendly implementation. High-level methods can be modularly configured using a variety of pre-implemented quantum encoding circuits. Each circuit's configuration can be tailored by adjusting hyperparameters such as the number of qubits and the number of layer repetitions. In QNNs and QRC, the form and quantity of observables used to compute the output can also be customized by hyperparameters. Additionally, fidelity kernels and projected quantum kernels are available for quantum kernel methods. Together with classical hyperparameters, such as regularization strength for the QSVM or optimization parameters for QNNs, these degrees of freedom form the configuration space for the QML predictors.

The QML methods can be executed using PennyLane and Qiskit simulators, as well as IBM Quantum computers and several quantum computing backends provided by Amazon Braket. In the case where real quantum computers are used for execution, communication with the quantum hardware providers is managed by sQLearn. For IBM quantum backends, an

automated hardware selection routine that can prioritize either speed or accuracy is available.

III. USE CASES

We benchmark AutoQML on four scenarios based on real-world use cases from the domains of manufacturing and automotive. For each, we derive a supervised learning problem. The learning problems have been chosen such that their (effective) dimensionality and data set size are small enough so that they can be processed by current quantum computers (or simulators) while still retaining an adequate level of difficulty. For some use cases this required creating synthetic data.

In the following, we briefly describe each use case and the data sets used to benchmark the framework. For some problems, the data sets are preprocessed before being inputted into AutoQML. In these cases, the preprocessing steps are described in the corresponding section for each use case.

A. Time Series Classification

We consider a time series classification task in which sensor data is collected from an autonomous vehicle to identify several states of the vehicle, for example, the execution of individual tasks or abnormal activity. To this end, we synthetically generate vibration sensor data, i.e., a univariate time series containing 7 individual states which are each associated with a unique label for the classification task. From the time series, we generate a spectrogram that is then divided into 2-dimensional tiles of size $[7 \times 30]$. Each tile is flattened into a one-dimensional vector of dimension $d = 210$ and a label corresponding to the states present in the tile is assigned. In total, the data set contains $N = 3291$ samples that are divided into $N_{\text{training}} = 758$ training points $N_{\text{test}} = 2533$ testing points. Since most tiles are associated with no sensor activity, the training set is stratified so that all classes occur roughly with the same frequency.

B. Image Classification

In sheet metal processing using laser cutting machines, metal plates rest on a bed of supporting slats during processing. These slats are manually positioned by the machine operator in fixed socket positions ahead of time, with the specific configuration depending on the cutting task. For optimal operation, it is beneficial to know the positions of the slats in advance [35]. The associated task is a binary image classification problem, where the goal is to determine whether a given image contains a supporting slat. To ensure enough image data for training, we use a proprietary synthetic pipeline to automatically generate artificial yet authentic slat images using accurate CAD models of all employed parts a 3D rendering framework. These rendered images feature entirely randomized configurations of slat positions. Since each rendered image depicts multiple slats, it is subsequently divided into smaller image

snippets of size 80×200 pixels, each focusing on exactly one slat position, which is either occupied or not. Although this methodology allows for the generation of an arbitrary number of images, to keep computational requirements moderate, we restrict the data set under study to 500 image snippets, divided into $N_{\text{training}} = 400$ training and testing points $N_{\text{test}} = 100$. Moreover, we perform feature reduction via PCA in advance, reducing each image snippet to $d = 8$ features. This reduction is necessary to maintain the confidentiality requirements of the use case. We treat the resulting latent space vectors as input images. The data set has been presented in more detail by Basilewitsch *et al.* [36].

C. Tabular Regression

Accurate price forecasting is essential for companies managing pre-owned assets, whose values fluctuate with spatial and temporal variations in supply and demand. This is particularly relevant for heavy construction equipment dealers and rental companies, who depend on precise price predictions to optimize asset management. Assessing the current and future residual value of their fleets enables these companies to determine the ideal time to resell individual pieces of machinery. By collecting data from seven major online construction equipment portals, we create a data set with $N = 165$ data points ($N_{\text{training}} = 132$ and $N_{\text{test}} = 33$). Each data point represents one *Caterpillar* type 308 construction machine. The features are the construction year, the working hours, the current location and the model extension. The target values are the prices. The location and model extension are categorical variables with 9 and 3 unique values, respectively. The resulting data set has six dimensions (16 if one-hot-encoded). The data set is a subset of the data used by Stühler *et al.* [37].

D. Time Series Forecasting

Engineering control technology systems for the automotive sector heavily depend on the ability to model or simulate sensor time series data. This is particularly relevant for dynamic situations, such as when accelerating a vehicle. Producing accurate and precise forecasts of physical quantities can significantly influence the quality of system control. To manipulate, compare, and analyze the computed models effectively, we select the relative cylinder filling of an internal combustion engine as an application where we can easily control the dimensionality. The time series encodes complex non-linear dynamics. Using a sliding-window approach, the problem is formulated as a regression task, and the number of time steps utilized for the forecast can be freely chosen. Our real industrial data set consists of 10000 time steps, each covering a period of 10ms. The final data set is a resampled version with $N_{\text{train}} = 556$ and $N_{\text{test}} = 140$, where the features are lagged versions of the time series from the four previous time steps. Training and test data are derived from different parts of the time series.

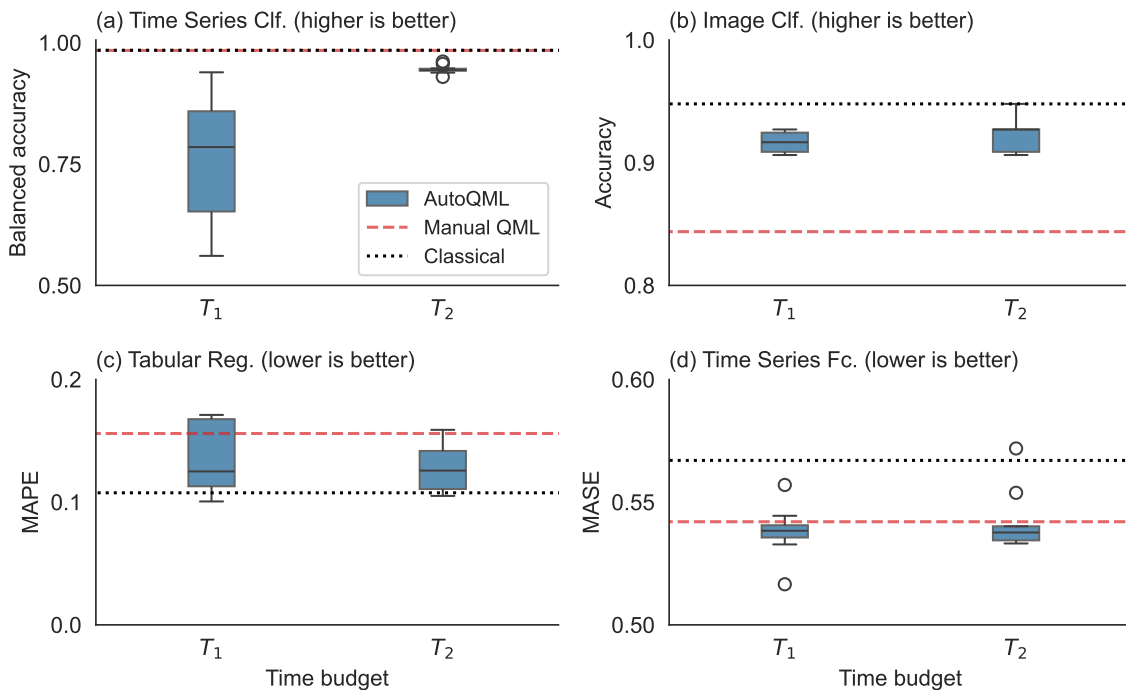


FIG. 3. Performance of AutoQML (boxes) for two different time budgets T_1 and T_2 . Additionally, manual QML pipelines (red, dashed) and classical models (black, dotted) are depicted. (a) shows the balanced accuracy (higher is better) for the time series classification. (b) shows the accuracy (higher is better) for the image classification. (c) shows the mean absolute percentage error (MAPE, lower is better) for the tabular regression, and (d) shows the mean absolute scaled error (MASE, lower is better) for the time series forecasting. For the box plots, points that are outside $1.5\times$ the inter-quartile range are shown as circles, and the lines inside the boxes denote the sample median.

IV. RESULTS

The performance of the AutoQML framework is evaluated using the four use cases described in Sec. III. The results are shown in Fig. 3. For each use case, we fit AutoQML for ten different seeds with two time budgets, $T_1 = 10000$ s and $T_2 = 50000$ s. Note that although AutoQML can optimize over a joint algorithm pool consisting of classical and quantum ML algorithms, we are interested in the performance of the QML algorithms in particular and thus only include quantum methods in the search space for this benchmark.

We compare our results with manually created QML pipelines (red, dashed lines). For two use cases, we sourced manual solutions from previous studies [36, 37], while for the other two, we constructed custom pipelines tailored to the specific use cases. Details of the manual models and the process used to obtain them are provided in Appendix A. Since these models have been crafted by quantum computing specialists, they require significantly more expertise and time compared to the corresponding AutoQML solutions. All quantum models are evaluated using the PennyLane statevector simulator. Additionally, to better gauge the quality of the results, we compare them with the performance of classical models. These are shown as dotted lines in Fig. 3, indicating the performance of the best model among random forests, XGBoost [38], and support vector machines. For regression, Gaussian process regression is also included. For the kernel methods, RBF ker-

nels were used, and the hyperparameters of all models were optimized using Optuna. The preprocessing pipeline is the same as for the manually created QML models.

When evaluating the manually obtained QML pipelines against the AutoQML pipelines, we observe that the performance is comparable. For three out of the four use cases, the AutoQML pipeline outperforms the manual QML pipeline on average, with a slight improvement in the tabular regression and time series forecasting use cases (b, c) and a clear advantage in the image classification use case (d). However, for the time series classification use case (a), the manual pipeline provides a superior solution.

In all use cases, we observe a median improvement when granting a larger time budget. Specifically, for the time series classification use case (a), there is a significant performance increase with the budget T_2 compared to T_1 . However, for the other use cases, the performance gains are only marginal. Additionally, the drastically reduced variance with time budgets T_2 suggests that budget the T_1 is insufficient for the time series classification use case, while for other use cases, the pipeline search appears to be nearing convergence even with T_1 . This difference in convergence is likely due to the much larger dimensionality of the time series classification use case compared to the other use cases.

When comparing the QML models with the classical models, we observe that the best AutoQML pipeline outperforms the best classical solution in three out of the four use cases (b–d),

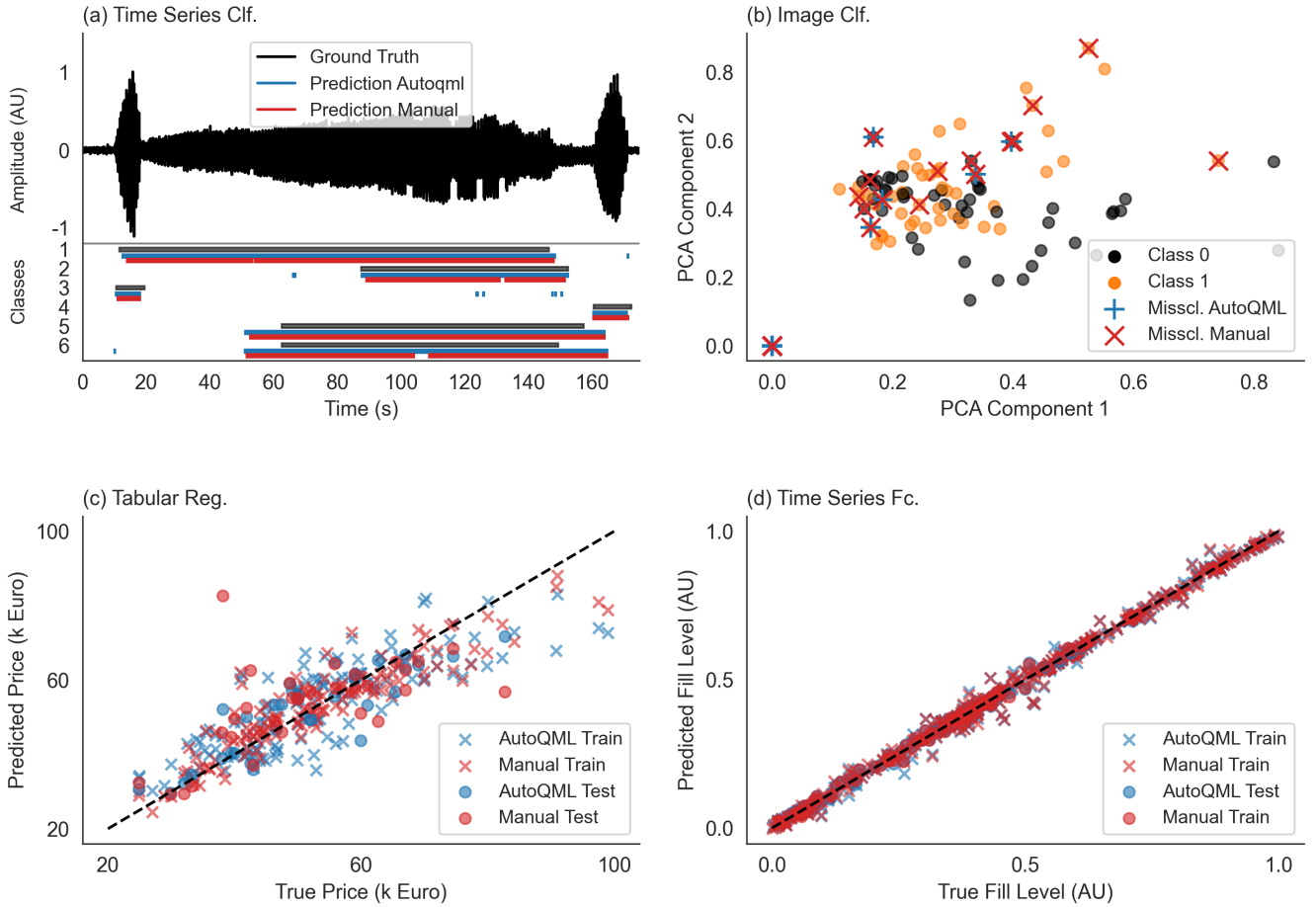


FIG. 4. Application of the best AutoQML pipelines (blue) from Fig 3 on the respective use cases. The application of the manually created models is shown in red. (a) shows signal in the upper part of the figure. The bars below show the presence or absence of events which are classified by the models. (b) shows the two principal components with the largest singular values of the test set of the image classification use case. The points that have been misclassified by the AutoQML (cross, blue) or the manual pipeline (plus, red) are shown in addition. The classes 0 (no slat) and 1 (slat) are shown in different colors. Figure (c) and (d) depicts the prediction vs. the true target values of the tabular regression and time series forecasting, respectively.

while for the time series classification (a), the classical solution achieves the highest score. Overall, the classical solutions are similar to the AutoQML solutions, and neither demonstrates a clear advantage over the other. The comparable quality of results across all models supports the validity of the manual QML models and provides evidence for the effectiveness of AutoQML.

The pipelines with the highest scores in Fig. 3 are shown in Tab. I. The rows show the choices for the corresponding pipeline step. Some preprocessing steps such as one hot encoding are only relevant for specific use cases and are omitted in the table. The final pipelines are diverse in terms of models and preprocessing. Notably, all models are quantum kernel models. This is, at least partially, expected since training QNNs is significantly more time consuming than training quantum kernel methods. Therefore, QNNs are underrepresented in the optimization process. Estimators based on QRC were the best optimizers in several runs. However, in none of the use cases was QRC the best among the ten runs (i.e., the ten optimiza-

tion with different seeds per use case). Three out of the four kernel methods use projected quantum kernels (PQK). Interestingly, all PQKs in the best performing models do not employ the commonly used RBF-type outer kernel function. This is in agreement with other studies which found that the outer kernel function in PQKs should be treated as an additional hyper-parameter [32].

Figure 4 shows the application of the best AutoQML pipelines (blue) from Fig. 3 together with the models from the manually crafted pipelines. Overall, the pipelines solve the use cases well. In the time series classification (a), both pipelines are able to determine the classes, although classes 4 and 5 seem to be more difficult as both pipelines do not classify them optimally. In the image classification (b), the pipelines are applied to the full data set but only the first two principal components are shown for visualization. It can be seen that the first two principal components are not sufficient to linearly separate the model. This aligns with the result of the best AutoQML pipeline, which uses all 8 principal components present

TABLE I. Summary of different model pipelines. Fidelity quantum kernels are denoted by FQK. The row *Observables* depicts the measurement observables for projected quantum kernels (PQK) or QNNs. Here, X_i, Y_i, Z_i denotes the respective Pauli operator on qubit i where $i = 1, \dots, n$ runs across all qubits. The description of the outer kernels in cases where PQKs have been used (in the present cases pairwise, Mattern, dot product), and the encoding circuits can be found in the sQULearn documentation [39].

	Time Series Classification	Image Classification	Tabular Regression	Time Series Regression
Dim. Red.	PCA	UMAP	–	–
Scaling	Normalization	Standardization	Normalization	Normalization
Observable	–	$\{X_i, Z_i\}$	$\{X_i, Y_i, Z_i\}$	$\{X_i, Y_i\}$
Model	QSVM; FQK	QSVM; PQK(Mattern)	QKRR; PQK(DotProduct)	QGPR; PQK(pairwise)
Encoding Circuit	[40]	Multi-Control	Multi-Control	YZ-CX [41]
Num. Qubits	8	8	8	8
Num. Layers	1	2	3	3

in the data set. Most of the misclassifications happen in the region where the classes overlap in the first two components, indicating that even with the full dimensionality, there might still be some overlap of the data classes. The application of the pipelines to the regression problems (c) and (d) is in line with the expectations from the performance metrics in Fig. 3. Overall, the tabular regression use case (c) is more difficult than the time series forecasting problem because the tabular regression problem has higher dimensionality and more noise compared to the relatively simple one-step ahead problem in (d). This difference in difficulty is reflected in the larger deviations of the predicted values from the true values in (c) compared to (d). Comparing the training performance to the test performance, no significant shortcomings, i.e., overfitting or underfitting, can be observed.

V. DISCUSSION

The results in this paper have been performed with state-vector simulations. As quantum computing matures, a significant portion of the model evaluation will have to be done on quantum computers. Although we have tested this prototypically, a full pipeline optimization on current quantum hardware is currently infeasible due to both financial and time constraints. Since simulation techniques co-evolve with the hardware [42, 43], we foresee that in the upcoming years the pipeline search will most likely involve a combination of advanced simulation techniques and evaluation on real devices with a requirement to be efficient in QPU time as much as possible. Furthermore, the current evaluation time for QML models is notably longer than for classical ML models, necessitating a greater time budget for pipeline optimization. This underscores a trade-off between human developer time and computational time, which is more significant than in classical AutoML frameworks. Addressing these challenges and integrating novel QML developments into AutoQML will be crucial for advancing the framework’s efficacy and versatility.

Although we have tested AutoQML on a diverse set of problems, the framework in its current form is only designed for supervised ML problems. Extensions to unsupervised problems like clustering are conceivable. Through its modular design and the encapsulation of the quantum computing-facing

modules in sQULearn, such extensions can be implemented easily. This is also true for incorporating novel developments in QML, such as new models.

Currently, AutoQML only supports a fixed, predefined library of circuits. These circuits can be further customized by the automation process. Nevertheless, recent work indicates that tailoring QML models to the dataset, rather than relying on generic hardware-efficient circuits, might be required to retain trainability as the models grow in depth and width [44, 45]. Incorporating automated approaches to quantum circuit design [23] into the framework is thus left for future work.

VI. CONCLUSION

We have introduced AutoQML, an innovative framework for automated QML, and evaluated its performance through comprehensive benchmarking on a diverse set of problems derived from four distinct industrial applications, specifically two classification and two regression tasks. Our results demonstrate that AutoQML is capable of effectively generating QML pipelines that incorporate QML-specific preprocessing, model selection, and hyperparameter optimization. Notably, the performance of the generated pipelines was competitive with that of manually constructed ones, which were, wherever feasible, derived from existing literature to reduce subjective biases. These findings indicate that AutoQML is a valuable tool for addressing machine learning challenges in QML, requiring minimal expertise in quantum computing. Additionally, the capabilities of AutoQML underscore its potential as a powerful prototyping and benchmarking resource for QML researchers and practitioners.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Economic Affairs and Climate Action through the project AutoQML (grant no. 01MQ22002A). We thank Frederic Rapp, Timoteo Lee and Khaled Al-Gumaei for their contributions to the time series classification use case. We thank Peter Schichtel and Jaroslav Vondrejč for their contribution to the time series forecasting use case. We thank Giorgio Silvi for the im-

plementation of the automated backend selection and Dennis Kleinhans for the implementation of tests for the framework. The authors disclose the use of LLM-based tools for grammar and spelling.

Appendix A: Manual QML Models

In this section, we briefly describe the pipelines for the manual solutions and the process by which they were obtained.

1. Time Series Classification

To reduce the dimensionality of the data set, we perform a PCA with 5 components and scale the output to the interval $[-1, 1]$. The manual model is a QSVM using a projected quantum kernel with the feature map from [40] with 5 qubits and 6 layers. The model is obtained using hyperparameter optimization over the regularization parameters of the QSVM, as well as the number of layers and qubits. The model used in the benchmark is the best performing from optimizing over a set of two quantum feature maps [40, 46].

2. Image Classification

The manually created pipeline contains no preprocessing steps in addition to those described in Sec. III B. The model is a QNN classifier with 8 qubits and an Ising-type cost operator. The model is the best performing model presented

by Basilewitsch *et al.* [36]. Details on how the circuit has been determined can be found there.

3. Tabular Regression

The reference quantum model is obtained from Ref. [37], in which it demonstrates optimal performance on a similar data set for a different type of construction machinery. In line with the original study, categorical features are one-hot encoded, resulting in a dataset with a dimensionality of $d = 15$. These features are then scaled to the interval $[-1, 1]$. The reference model in Fig. 3 is a QSVM that utilizes a Fidelity Quantum Kernel. The encoding circuit has been obtained from Fig. 5 of Ref. [37], and employs 15 qubits (one qubit per feature). The hyperparameters of the underlying SVM have been optimized through a dedicated hyperparameter optimization process.

4. Time Series Forecasting

The manually created pipeline contains no additional preprocessing steps. The model is a 4-qubit quantum reservoir regressor with 54 random measurement operators which are fed into a linear regression model. The used reservoir is the result of a search over the number of qubits, the number of layers used in the encoding, the number of observable, and the architecture of the encoding circuit. The search is performed using Optuna.

-
- [1] M.-A. Zöller and M. F. Huber, Benchmark and survey of automated machine learning frameworks, *J. Artif. Int. Res.* **70**, 409–472 (2021).
- [2] D. Wang, Q. V. Liao, Y. Zhang, U. Khurana, H. Samulowitz, S. Park, M. Muller, and L. Amini, *How much automation does a data scientist want?* (2021), [arXiv:2101.03970 \[cs.LG\]](https://arxiv.org/abs/2101.03970).
- [3] F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning - Methods, Systems, Challenges* (Springer, 2019).
- [4] S. Miller and D. Hughes, *The quant crunch: how the demand for data science skills is disrupting the job market* (2017), viewed 28 Feb 2025.
- [5] Y. Liu, S. Arunachalam, and K. Temme, A rigorous and robust quantum speed-up in supervised machine learning, *Nature Physics* **17**, 1013 (2021).
- [6] H.-Y. Huang, M. Broughton, J. Cotler, S. Chen, J. Li, M. Mohseni, H. Neven, R. Babbush, R. Kueng, J. Preskill, and J. R. McClean, Quantum advantage in learning from experiments, *Science* **376**, 1182 (2022).
- [7] J. Liu, M. Liu, J.-P. Liu, Z. Ye, Y. Wang, Y. Alexeev, J. Eisert, and L. Jiang, Towards provably efficient quantum algorithms for large-scale machine-learning models, *Nature Communications* **15**, 434 (2024).
- [8] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, Auto-weka: combined selection and hyperparameter optimization of classification algorithms, in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13 (Association for Computing Machinery, New York, NY, USA, 2013) p. 847–855.
- [9] D. A. Kreplin, M. Willmann, J. Schnabel, F. Rapp, and M. Roth, *sqlearn: A python library for quantum machine learning*, *IEEE Software*, 1 (PrePrints 5555).
- [10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, Scikit-learn: Machine learning in python, *Journal of Machine Learning Research* **12**, 2825 (2011).
- [11] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, and others, PennyLane: Automatic differentiation of hybrid quantum-classical computations (2022), [arXiv:1811.04968 \[quant-ph\]](https://arxiv.org/abs/1811.04968).
- [12] Qiskit Community, *Qiskit: An open-source framework for quantum computing* (2017).
- [13] IBM Quantum, <https://quantum-computing.ibm.com> (2023).
- [14] Amazon Web Services, *Amazon Braket* (2020).
- [15] 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.

- [16] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, [Tune: A research platform for distributed model selection and training](#) (2018), [arXiv:1807.05118 \[cs.LG\]](#).
- [17] R. Berganza Gómez, C. O’Meara, G. Cortiana, C. B. Mendl, and J. Bernabé-Moreno, Towards autoqml: A cloud-based automated circuit architecture search framework, in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)* (2022) pp. 129–136.
- [18] S. Altares-López, J. J. García-Ripoll, and A. Ribeiro, Autoqml: Automatic generation and training of robust quantum-inspired classifiers by using evolutionary algorithms on grayscale images, *Expert Systems with Applications* **244**, 122984 (2024).
- [19] T. Koike-Akino, P. Wang, and Y. Wang, Autoqml: Automated quantum machine learning for wi-fi integrated sensing and communications, in *2022 IEEE 12th Sensor Array and Multichannel Signal Processing Workshop (SAM)* (2022) pp. 360–364.
- [20] S. Altares-López, A. Ribeiro, and J. J. García-Ripoll, Automatic design of quantum feature maps, *Quantum Science and Technology* **6**, 045015 (2021).
- [21] M. Incudini, D. L. Bosco, F. Martini, M. Grossi, G. Serra, and A. D. Pierro, Automatic and effective discovery of quantum kernels, *IEEE Transactions on Emerging Topics in Computational Intelligence*, **1** (2024).
- [22] X. Dai, T.-C. Wei, S. Yoo, and S. Y.-C. Chen, Quantum machine learning architecture search via deep reinforcement learning, in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Vol. 01 (2024) pp. 1525–1534.
- [23] F. Rapp, D. A. Kreplin, M. F. Huber, and M. Roth, Reinforcement learning-based architecture search for quantum machine learning, *Machine Learning: Science and Technology* **6**, 015041 (2025).
- [24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, Pytorch: an imperative style, high-performance deep learning library, in *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (Curran Associates Inc., Red Hook, NY, USA, 2019).
- [25] M.-A. Zöllner, T.-D. Nguyen, and M. F. Huber, [Incremental search space construction for machine learning pipeline synthesis](#) (2021), [arXiv:2101.10951 \[cs.LG\]](#).
- [26] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning - Methods, Systems, Challenges* (Springer, 2019).
- [27] M. Feurer, A. Klein, K. Eggenberger, J. T. Springenberg, 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.
- [28] 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) pp. 108–122.
- [29] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, Algorithms for hyper-parameter optimization, in *Advances in Neural Information Processing Systems*, Vol. 24, edited by J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger (Curran Associates, Inc., 2011).
- [30] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Quantum circuit learning, *Phys. Rev. A* **98**, 032309 (2018).
- [31] L. Innocenti, S. Lorenzo, I. Palmisano, A. Ferraro, M. Paterostro, and G. M. Palma, Potential and limitations of quantum extreme learning machines, *Communications Physics* **6**, 118 (2023).
- [32] J. Schnabel and M. Roth, [Quantum kernel methods under scrutiny: A benchmarking study](#) (2024), [arXiv:2409.04406 \[quant-ph\]](#).
- [33] 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 (2019).
- [34] F. Rapp and M. Roth, Quantum gaussian process regression for bayesian optimization, *Quantum Machine Intelligence* **6**, 5 (2024).
- [35] F. Struckmeier and F. P. León, Nesting in the sheet metal industry: dealing with constraints of flatbed laser-cutting machines, *Procedia Manufacturing* **29**, 575 (2019), “18th International Conference on Sheet Metal, SHEMET 2019” “New Trends and Developments in Sheet Metal Processing”.
- [36] D. Basilewitsch, J. F. Bravo, C. Tutschku, and F. Struckmeier, [Quantum neural networks in practice: A comparative study with classical models from standard data sets to industrial images](#) (2024), [arXiv:2411.19276 \[quant-ph\]](#).
- [37] H. Stühler, D. Pranjic, and C. Tutschku, Evaluating quantum support vector regression methods for price forecasting applications., in *ICAART (3)* (2024) pp. 376–384.
- [38] T. Chen and C. Guestrin, Xgboost: A scalable tree boosting system, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’16* (Association for Computing Machinery, New York, NY, USA, 2016) p. 785–794.
- [39] sQUlearn Team, [sQUlearn Documentation](#) (2025).
- [40] T. Hubregtsen, D. Wierichs, E. Gil-Fuster, P.-J. H. S. Derks, P. K. Faehrmann, and J. J. Meyer, [Training quantum embedding kernels on near-term quantum computers](#) (2021).
- [41] T. Haug, C. N. Self, and M. S. Kim, Quantum machine learning of large datasets using randomized measurements, *Machine Learning: Science and Technology* **4**, 015005 (2023).
- [42] D. Aharonov, X. Gao, Z. Landau, Y. Liu, and U. Vazirani, A polynomial-time classical algorithm for noisy random circuit sampling, in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023* (Association for Computing Machinery, New York, NY, USA, 2023) p. 945–957.
- [43] A. Angrisani, A. Schmidhuber, M. S. Rudolph, M. Cerezo, Z. Holmes, and H.-Y. Huang, [Classically estimating observables of noiseless quantum circuits](#) (2024), [arXiv:2409.01706 \[quant-ph\]](#).
- [44] J. Kübler, S. Buchholz, and B. Schölkopf, The inductive bias of quantum kernels, in *Advances in Neural Information Processing Systems*, Vol. 34, edited by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan (Curran Associates, Inc., 2021) pp. 12661–12673.
- [45] M. Cerezo, G. Verdon, H.-Y. Huang, L. Cincio, and P. J. Coles, Challenges and opportunities in quantum machine learning, *Nature Computational Science* **2**, 567 (2022).
- [46] D. A. Kreplin and M. Roth, Reduction of finite sampling noise in quantum neural networks, *Quantum* **8**, 1385 (2024).