Tomasz Rybotycki
Piotr Gawron

AQMLATOR – AN AUTO QUANTUM MACHINE-LEARNING E-PLATFORM

Abstract *The successful implementation of a machine-learning (ML) model requires three main components: a training data set, a suitable model architecture, and a suitable training procedure. Given the data set and task, finding an appropriate model might be challenging. AutoML, a branch of ML, focuses on an automatic architecture search – a meta method that aims to remove the need for human interaction with the ML system-design process. The success of ML and the development of quantum computing (QC) in recent years has led to the birth of a new fascinating field called quantum machine learning (QML), which incorporates quantum computers into ML models (among other things). In this paper, we present AQMLator, an auto quantum machine-learning platform that aims to automatically propose and train the quantum layers of an ML model with minimal input from the user. In this way, data scientists can bypass the entry barrier for QC and use QML. AQMLator uses standard ML libraries, making it easy to introduce into existing ML pipelines.*

Keywords auto machine learning, quantum computing, quantum machine learning

Citation Computer Science 26(SI) 2025: 29–43

1. Introduction

Machine learning (ML) is one of the fastest-progressing research directions in applied computer science. This field investigates the development of algorithms that can learn from data by fitting a collection of model parameters to the data via the iterative optimization of an objective function. The selection of a model structure (be it a neural network or kernel function) is a problem-dependent task that is often performed by hand; however, auto-ML systems [14] exist that can choose a model automatically depending solely on the input data and the task at hand.

Quantum computing (QC) studies how difficult computational problems can be efficiently solved by using quantum mechanics. A large-scale error-corrected quantum computer can solve computational problems that do not have a classical solution. A prime example of this is Shor's algorithm [30] for integer factorization. The "holy grail" of applied QC is the so-called quantum supremacy or quantum advantage. This is the name for the technological milestone that marks the moment when quantum machines will solve a specific task faster than the most advanced supercomputer. Although there have already been several quantum supremacy claims in recent years [4], there are no practical problems that are solvable by only using quantum computing as of yet.

The search for such practical problems focuses on applications in the soft computing areas that are less susceptible to the current quantum hardware imperfections; one of the possible applications of QC is quantum machine learning (QML) [7]. This field of science investigates how quantum computers can be employed to build ML models that can be fit to data and then be used during the inference process. In one of the QML scenarios, a variational quantum circuit that forms a quantum neural network (QNN) constitutes only one part of the ML data-processing pipeline. Since designing such a pipeline with a quantum component is challenging for non-experts in QC, we propose an auto-ML solution that suggests ready-to-use QML models.

This paper is organized as follows. In the next section, we present a short overview of the state of the art. We point out the challenges that have been laid before auto(mated) (quantum) machine learning, the most recent techniques that tackle these problems, and the related software. In Section 3 (the main part of this work), we present AQMLator – an auto quantum machine-learning (AQML) e-platform that was designed to fill the gap in the AQML literature. We overview its construction, design principles, and key features. The paper is concluded with some insights about future works.

2. State of the art

As machine learning is becoming progressively more complex, the need for methods of its automation arises. Complicated ML tasks like neural architecture search (NAS), hyperparameter optimization (HPO), or even model selection (MS) are often beyond

the abilities of scientists from outside the field. Automated machine learning was introduced as a way to address these shortcomings [15].

With the emergence of a new interdisciplinary field like quantum machine learning, a revision of AutoML is required. QML models are often vastly different from their classical counterparts. While several standard techniques of tackling typical ML model-creation problems still apply, automated quantum machine learning may require some adjustments due to the quantum nature of its models. For instance, when discussing QML, one no longer talk about NAS but rather QAS (quantum architecture search). A recent extensive summary of the differences between quantum and classical ML can be found in [9]. Adding quantum computing to the list of skills that are required to properly introduce a QML model to one's research, it is clear why non-experts may be discouraged from using such models. This is exactly the reason why auto(mated) quantum machine learning is required.

As pointed out in [9], one of the crucial challenges in QML is the development of efficient quantum architecture search techniques. Given that parametrized (variational) quantum circuits can also be viewed as quantum machine-learning models [6], QAS has a significant impact on the whole field of quantum computing. This is especially true if we also consider the low-circuit-depth requirement of the current NISQ-era[1] machines.

There have been many propositions for quantum architecture search automation; a vast area of this research has been occupied by reinforcement learning-based techniques. These techniques use an ML agent to autonomously explore the search spaces of possible circuits in order to find the architecture optimal for a given task [19, 25]. Recent works on QAS have also used Kolmogorov-Arnold networks, which have gained some recognition in recent months [20]. Although autonomous to some degree, these techniques and a plethora of others (see [22]) use classical ML to find optimal quantum circuit architectures. In order to use and (especially) tweak these, specialized knowledge is required.

In a typical scenario, one first must consider which class of (Q)ML models that they should use before they even try to solve N/QAS. With a development as rapid as in the case of ML, it comes as no surprise that this itself is a daunting task. Even if we restrict the their selection to hybrid-QML models, the number is still substantial [8]. Considering how vital model selection is for research [27], how expensive the training of a model can be [12], and how the access to quantum resources is currently limited, the need for guidance in this matter becomes clear. Although [11] addressed this issue to some degree, the work is very technical and requires QC know-how.

When it comes to hyperparameter optimization, the situation is significantly better. Their importance [23, 24] and optimization [13] is a topic of multiple studies – especially in the context of quantum neural networks. Although insightful, these studies are often technical and do not provide any software. On the other hand, classical

[1]Noisy intermediate scale quantum.

approaches have been shown to work well for QML models. In [21], for instance, the authors used the Optuna hyperparameter-optimization framework [1] to find hyperparameters for their hybrid models. What is especially important in the context of this work is that Optuna did not require any knowledge about its underlying algorithms and, thus, could be considered to be an automated hyperparameter optimizer.

None of the above solutions explicitly claimed to be such a system even though they addressed the issues that were expected from an automated machine-learning system. There are, however, a number of works where the authors regarded their solutions as such. It is vital to start with the recent whitepaper [17]; its authors reviewed current classical AutoML solutions and their possible extensions for quantum models. In their other work [16], some of the authors presented AutoQML – an actual Python auto quantum machine-learning framework. In [3], the authors proposed another so-called AutoQML method that used genetic algorithms for the automatic generation of trained quantum-inspired classifiers. The term "quantum-inspired" was meant to underline the fact that the resultant model was supposed to run on fully classical hardware despite its use of unitary gates – a common building block in quantum computing. Given the nature of this approach, this seems to be applicable for QML models. In [31], the authors proposed a multi-locality search algorithm for initial points, circuit parameters, and data-preprocessing options; however, this paper was highly technical. Finally, there were also signs of the practical use of AQML for a Wi-Fi sensing task [18] where the authors claimed to use their so-called AutoAnsatz approach for a quantum architecture search and hyperparameter optimization. We found that the code was provided only for genetic-oriented AutoQML [2] even though the first AutoQML team had a GitHub repository [5].

3. AQMLator

We call our solution an **A**uto **Q**uantum **M**achine-**L**earning platform – AQMLator [29]. This aims to enable data scientists (especially those without knowledge of QC) to use QML models. Given a task and data, AQMLator will seek a model that fits it best. As a result, the platform will propose a quantum model, its circuit structure, and its weights. The user can then use the suggested model as-is or encompass it in a hybrid quantum – a classical model; e.g., by extracting the proposed model as a torch layer [26].

In Figure 1, we present a data-flow diagram of the AQMLator platform. It shows that the only input that is required from the user is the source of the data and the task that the user wants the model to perform; no specialized knowledge is required.

AQMLator can be used with simulators or physical quantum devices. In Figure 2, we present a sequence diagram of the model-training and inference process using our platform. The diagram shows that the device that one uses during the training may differ from the device that is used for inference. We can also see that, throughout the training, AQMLator strives to minimize the reservation time and the the number of costly quantum device calls; in this sense, it is quantum-resource-aware (QRA).

As the goal of AQMLator was to provide a complex auto quantum machine-learning solution, it needed to address all of the challenges that were mentioned in the section above. Not only does it do this, it also provides the source code thus, abiding by open science practices). In Table 1, we summarize the features that are offered by the current auto quantum machine-learning systems.

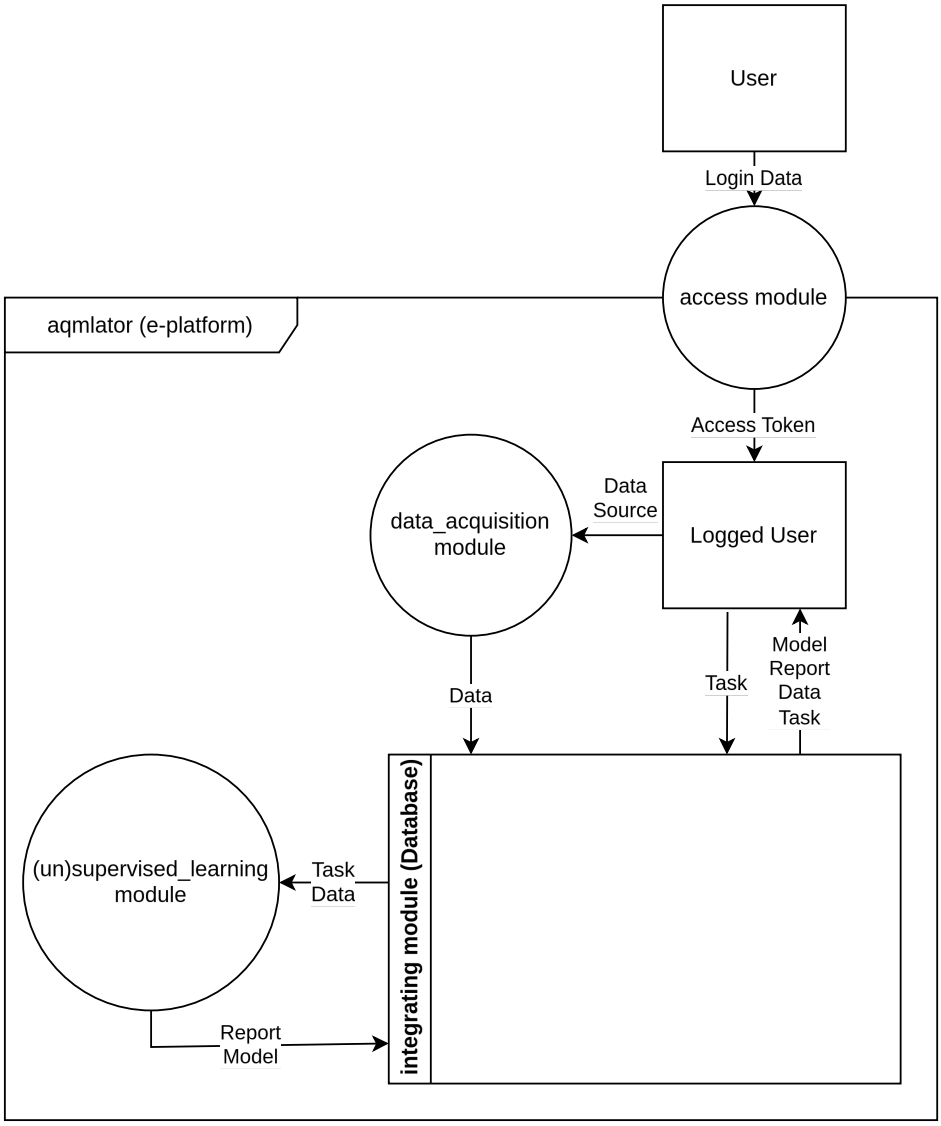

Figure 1. Data-flow diagram of AQMLator platform; notice that user's input is reduced only to task and data-source specification

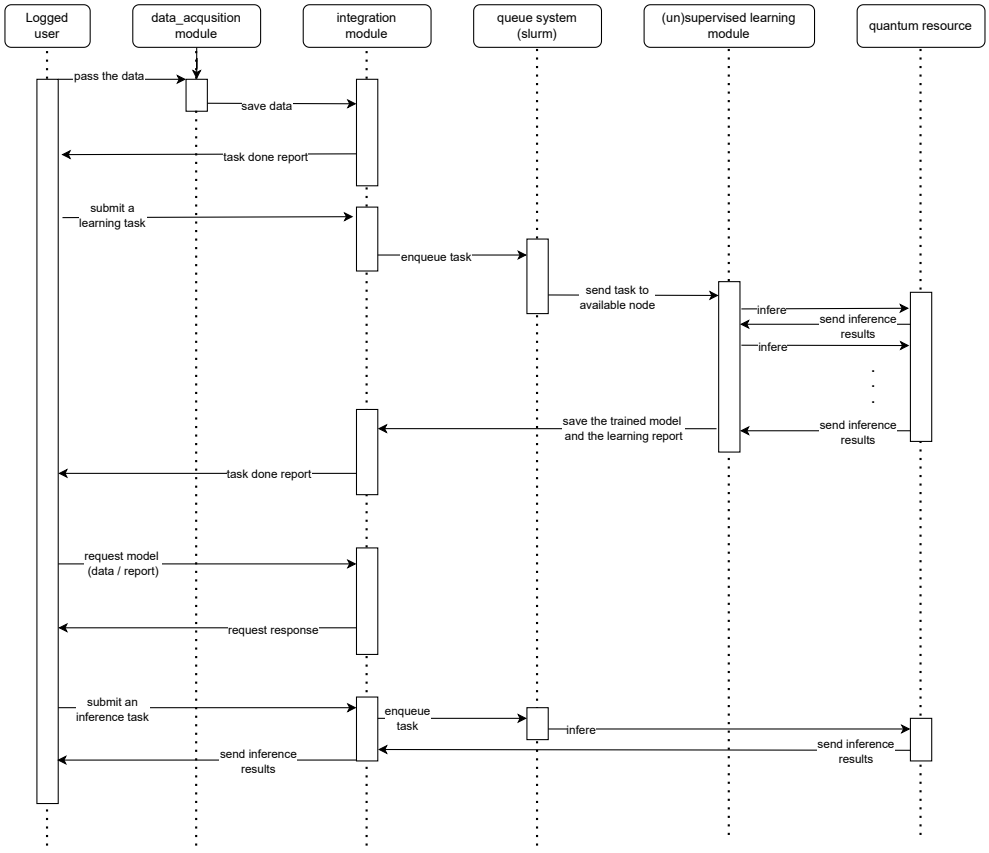

Figure 2. Sequence diagram of quantum model-training and inference process in AQMLator; quantum resource we used can be classical (simulator) or quantum and, in general, can vary between runs

Table 1

Summary of key features that are offered by state-of-the-art AQML solutions; note that only AQMLator satisfies all of these (features that were not mentioned in respective works are marked with ×)

Solution	MS	QAS	HPO	QRA	Open source
AutoQML [16]	✓	✓	✓	✓	×
AutoQML [3]	×	✓	✓	×	✓
AutoAnsatz	×	✓	✓	×	×
AQMLator	✓	✓	✓	✓	✓

3.1. Design principles

There were several key ideas behind the development of AQMLator. First and foremost, the platform is meant to be used by the non-QC and even non-ML practitioners. Ideally, the user should only provide the data and a task for which the model should be proposed after installation. AQMLator will then search for and ultimately suggest a quantum machine-learning model. We present the intended use in the following code snippet.

```
# Initialize the model finder.
classifier_finder: ModelFinder = ModelFinder(
    task_type=MLTaskType.CLASSIFICATION,
    features=cls_x,
    classes=cls_y,
    n_cores=1,  # Can be left blank.
    n_trials=n_trials,  # Can be left blank.
    n_seeds=n_seeds,  # Can be left blank.
    n_epochs=n_epochs,  # Can be left blank.
    device=dev,  # Will use default simulator if left blank.
)
# Find the model.
model_finder: QMLModel = classifier_finder.find_model()
```

The ease of access was further improved by the detailed documentation [28], the well-commented code, and the numerous examples of its usage that could be found in the unit tests that were distributed with the code.

The simplicity of its use does not impair our solution in any way. AQMLator covers a range of standard ML tasks (classification, linear regression, clustering) and a number of models (QNN, Quantum Kernels, Restricted Boltzmann Machines). It was also designed with extendability and customization in mind; for example, the default distribution of AQMLator suggests two data-embedding methods: `AngleEmbedding`, or `AmplitudeEmbedding`. One can see this in the `dicts` configuration of the `tuner.py` module.

```
data_embeddings: Dict[str, Dict[str, Any]] = {
    "ANGLE": {"constructor": AngleEmbedding, "kwargs": {}, "fixed_kwargs
        ": {}},
    "AMPLITUDE": {
        "constructor": AmplitudeEmbedding,
        "kwargs": {},
        "fixed_kwargs": {"pad_with": 0, "normalize": True},
    },
}
```

Extending the `data_embeddings dict` with different `pennylane`-compatible embedding methods will automatically increase the available search-space of the `ModelFinder`. Additional the out-of-the-box QML model layers and data-embedding methods are described in the Pennylane documentation [32]. Similar extensibility also applies to the other parts of the framework (models, layers, optimizers, ML tasks), yet it may require minimal competencies in either ML or QC.

AQMLator implements the notion of a limited-quantum-resources budget and aims to find such a QML model that will require the least number of quantum device calls during the training on the actual device while at least maintaining the evaluation metric threshold that is specified by the user. By default, AQMLator uses mean accuracy and silhouette scores for supervised and unsupervised models, respectively; both are standard ML models evaluation methods. The user can easily introduce their own evaluation functions either globally (by modifying the `QMLModel.score` method) or for each model individually.

AQMLator is a framework that was developed in `Python` 3.11 using standard (Q)ML libraries such as `PennyLane`, `sklearn` (`SciKit-Learn`), `PyTorch`, and `Optuna` with a `PostgreSQL` database. `PennyLane` provided the basic building blocks for our supervised-learning models. We used `sklearn mixins` to provide a standard interface for our models. `PennyLane` allows one to import its models as `PyTorch` layers; this makes the introductions of any AQMLator-found models into the current ML pipelines seamless. `Optuna` is the heart of our framework, as it is used for model selection, quantum architecture search, and hyperparameter optimization. Finally, we are also able to interactively review any experimental results that are stored in the `PostreSQL` database with `optuna-dashboard`. Our unsupervised-learning module is an extension of the implementation that was provided by the QBM4EO project [10]. AQMLator is fully-documented, unit-tested, and open-source software that is available via PyPi [29].

3.2. Quantum Architecture Search

AQMLator treats quantum architecture search as a hyperparameter optimization task. By doing so, we can use `Optuna` to find both a model and its architecture. During the `Optuna`-optimization process, we first sample the number of layers that the model will have.

```
kwargs: Dict[str, Any] = {
    "wires": len(self._x[0]),
    "n_layers": trial.suggest_int(
        "n_layers" + self._optuna_postfix,
        self._models_dict[model_type]["n_layers"][0],
        self._models_dict[model_type]["n_layers"][1],
    ),
    "n_epochs": self._n_epochs,
    "accuracy_threshold": self._minimal_accuracy,
}
```

The layer number's bounds (both maximal and minimal) can be customized for each model in one of the tuner's customization `dicts`. For instance, we can see that our `QNNBinaryClassifier` has 1–3 layers in the default AQMLator distribution, while our `QuantumKernelBinaryClassifier` can have 3–5 layers.

```
binary_classifiers: Dict[str, Dict[str, Any]] = {
    "QNN": {
```

```
        "constructor": QNNBinaryClassifier,
        "kwargs": {
            "batch_size": (15, 25),  # Might need to be data size-
                dependent instead.
        },
        "fixed_kwargs": {},
        "n_layers": (1, 3),
    },
    "QEK": {
        "constructor": QuantumKernelBinaryClassifier,
        "kwargs": {},
        "fixed_kwargs": {},
        "n_layers": (3, 5),
    },
}
```

With the number of layers set, AQMLator uses `Optuna` to propose the data-embedding method and the following layers of the model.

```
def _suggest_supervised_model_kwargs(
    self, trial: optuna.trial.Trial, model_type: str
) -> Dict[str, Any]:

    <...>

    self._suggest_embedding(trial, kwargs)
    self._suggest_layers(trial, kwargs)

    <...>
```

AQMLator samples the `Pennylane` data-embedding methods and the QML model layers (or, rather, their constructors) as categorical parameters. If present, additional parameters, can be specified in one of the configuration `dicts` in `tuner.py`.

```
def _suggest_layers(
    self, trial: optuna.trial.Trial, kwargs: Dict[str, Any]
) -> None:

    <redacted docstrings>

    layers: List[Type[qml.operation.Operation]] = []

    for i in range(kwargs["n_layers"]):
        layer_type: str = trial.suggest_categorical(
            f"layer_{i}" + self._optuna_postfix, list(layer_types)
        )
        layers.append(layer_types[layer_type]["constructor"])

    kwargs["layers"] = layers
    kwargs.pop("n_layers")  # No longer needed.
```

After the layer sampling is done, the model creation is possible; this requires calling the constructors in the same order that they were sampled. We repeat this process

a user-specified number of times. The model that scores the best (that is, meets the quality metric threshold and minimizes the number of quantum device calls) is returned by the end of the search.

3.3. Hyperparameter optimization

Hyperparameter optimization with AQMLator is a two-part process. The main part of the optimization takes place during the model search. Analogously to the data-embedding method and model layers, a number of additional parameters are suggested by `Optuna`. Consider the following method of `ModelFinder`.

```
def _suggest_unsupervised_model_kwargs(
    self, trial: optuna.trial.Trial, model_type: str
) -> Dict[str, Any]:
    lbae_input_shape: Tuple[int] = (1,) + np.array(self._x[0]).shape
    lbae_input_size: int = prod(lbae_input_shape)

    kwargs: Dict[str, Any] = {
        "lbae_input_shape": lbae_input_shape,
        "lbae_out_channels": trial.suggest_int(
            name="lbae_out_channels",
            low=floor(sqrt(lbae_input_size)),
            high=ceil(0.75 * lbae_input_size),
        ),
    }

    kwargs["rbm_n_visible_neurons"] = kwargs["lbae_out_channels"]

    kwargs["rbm_n_hidden_neurons"] = trial.suggest_int(
        name="rbm_n_hidden_neurons",
        low=floor(sqrt(kwargs["lbae_out_channels"])),
        high=ceil(0.75 * kwargs["lbae_out_channels"]),
    )

    kwargs_data: Dict[str, Any] = self._models_dict[model_type]["kwargs"
        ]

    kwargs["lbae_n_layers"] = trial.suggest_int(
        "lbae_n_layers" + self._optuna_postfix,
        kwargs_data["lbae_n_layers"][0],
        kwargs_data["lbae_n_layers"][1],
    )

    kwargs["fireing_threshold"] = trial.suggest_float(
        "fireing_threshold" + self._optuna_postfix,
        kwargs_data["fireing_threshold"][0],
        kwargs_data["fireing_threshold"][1],
    )

    kwargs["n_epochs"] = self._n_epochs

    return kwargs
```

One can see that `lbae_out_channels`, `rbm_n_hidden_neurons`, `lbae_n_layers`, `n_epochs`, and all of the hyperparameters of the unsupervised model are selected with `Optuna`. In the cases of supervised-learning models, the number of layers is the most important hyperparameter.

The second part of HPO using AQMLator concerns the model training. If a user would like to ensure that the default training method –`PennyLane GradientDescentOptimizer` – is the optimal one, they can use our `HyperparameterTuner` to compare the training performance by using different optimizers. This is done by instantiating the tuner and calling its only public method (as is presented in the code snippet below).

```
tuner: HyperparameterTuner = HyperparameterTuner(
    x,
    y,
    classifier,  # Or any other QMLModel.
    n_seeds=n_seeds,
    n_trials=n_trials,
)
tuner.find_hyperparameters()
```

3.4. Experiment revision with optuna-dashboard

Once the model-finding or hyperparameter-tuning experiments are done, their results can be interactively reviewed using `optuna-dashboard`.

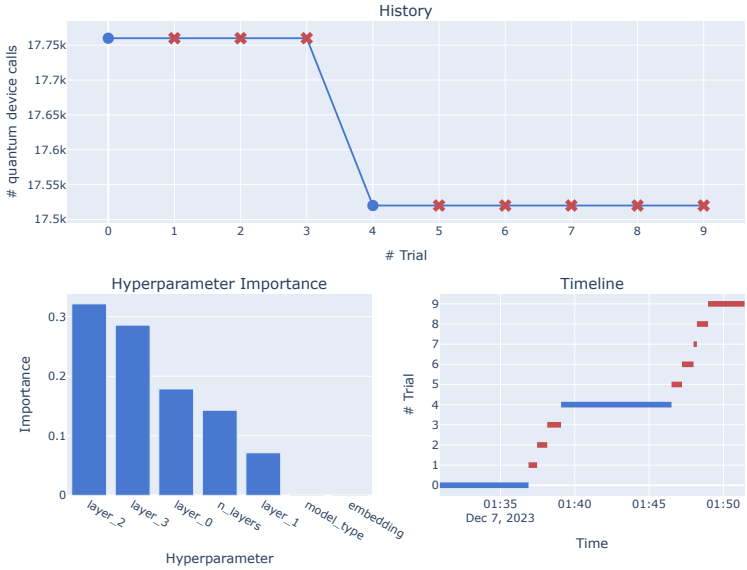

Figure 3. Example results of model-finding using AQMLator; note that additional information (specifically, values of hyperparameters for given trial) are accessible via interaction with plot in `optuna-dashboard`

For this to happen, however, the user must specify one of the available types of databases as the `Optuna` storage. We present an example of how `optuna-dashboard` presents the experiment results in Figure 3. Notice that `optuna-dashboard` also allows one to access additional info about each data point over the plot while mousing over. This feature is not shown in the figure.

4. Conclusions and future work

In this paper, we present AQMLator – an auto quantum machine-learning platform that was designed for users without knowledge of quantum computing and very limited experience in machine learning. We argue that this will allow a broad audience to include QML techniques in their ML pipelines, thus filling the gap in the field of accessible auto quantum machine-learning platforms.

Our approach is open source and can be run on both simulators and real quantum devices via `qiskit`. It fully automatically suggests a QML model given only data and a task from a user. The platform was built using standard (Q)ML libraries, allowing for a seamless introduction of the resultant models to the existing (Q)ML pipelines. It is also fully documented, making it easy to use, extend, and customize to one's needs.

Although complete in its current form, there are many ways that AQMLator could be improved. A natural course of action would be to introduce additional layers, data-embedding methods, and models to the default version of the framework so that users will not have to specify them themselves. Another approach would be to introduce advanced reinforced learning-based techniques of quantum architecture search to our framework so that an inspection of the whole available circuit space would be done in a more complete fashion. We could also further simplify the process of adding additional evaluation metrics to AQMLator and enable mutli-objective optimization to be native to `Optuna`. This way, the user could directly specify the desired cost-to-quality ratio.

Acknowledgments

The authors would like to thank Etos sp. z o.o. and the QBM4EO team for providing the QBM4EO code. We acknowledge the financial support by the EuroHPC PL project, which was co-financed by the EU within the Smart Growth Operational Programme (Contract No. POIR.04.02.00-00-D014/20-00). We gratefully acknowledge the funding support by the "Excellence Initiative – Research University" program for AGH University of Krakow as well as the ARTIQ/0004/2021 project.

References

[1] Akiba T., Sano S., Yanase T., Ohta T., Koyama M.: Optuna: A Next-generation Hyperparameter Optimization Framework, 2019. https://arxiv.org/abs/1907.10902.

[2] Altares-López S., García-Ripoll J.J., Ribeiro A.: AutoQML: Quantum Inspired Kernels by Using Genetic Algorithms for Grayscale Images [Source Code], https://codeocean.com/capsule/5670396/tree, 2023. doi: 10.24433/CO.3955535.v1.

[3] Altares-López S., García-Ripoll J.J., Ribeiro A.: AutoQML: Automatic generation and training of robust quantum-inspired classifiers by using evolutionary algorithms on grayscale images, *Expert Systems with Applications*, vol. 244, 122984, 2024. doi: 10.1016/j.eswa.2023.122984.

[4] Arute F., Arya K., Babbush R., Bacon D., Bardin J.C., Barends R., Biswas R., *et al.*: Quantum supremacy using a programmable superconducting processor, *Nature*, vol. 574(7779), pp. 505–510, 2019.

[5] AutoQML Team: AutoQML Repository, https://github.com/AutoQML, 2023. Accessed: 2024-09-18.

[6] Benedetti M., Lloyd E., Sack S., Fiorentini M.: Parameterized quantum circuits as machine learning models, *Quantum Science and Technology*, vol. 4(4), 043001, 2019. doi: 10.1088/2058-9565/ab4eb5.

[7] Biamonte J., Wittek P., Pancotti N., Rebentrost P., Wiebe N., Lloyd S.: Quantum machine learning, *Nature*, vol. 549(7671), pp. 195–202, 2017.

[8] Bowles J., Ahmed S., Schuld M.: Better than classical? The subtle art of benchmarking quantum machine learning models, 2024. https://arxiv.org/abs/2403.07059.

[9] Cerezo M., Verdon G., Huang H.Y., Cincio L., Coles P.J.: Challenges and opportunities in quantum machine learning, *Nature Computational Science*, vol. 2, pp. 567–576, 2022. doi: 10.1038/s43588-022-00311-3.

[10] Feral Qubits: QBM4EO: Supervised quantum machine learning system for Earth land cover understanding., https://feralqubits.github.io/qbm4eo-lp/. Accessed: 2024-09-18.

[11] Gentile A., Flynn B., Knauer S., Wiebe N., Paesani S., Granade C.E., Rarity J.G., *et al.*: Learning models of quantum systems from experiments, *Nature Physics*, vol. 17, pp. 837–843, 2021. doi: 10.1038/s41567-021-01201-7.

[12] Guerra E., Wilhelmi F., Miozzo M., Dini P.: The Cost of Training Machine Learning Models Over Distributed Data Sources, *IEEE Open Journal of the Communications Society*, vol. 4, pp. 1111–1126, 2023. doi: 10.1109/OJCOMS.2023.3274394.

[13] Herbst S., Maio V.D., Brandic I.: On Optimizing Hyperparameters for Quantum Neural Networks, 2024. https://arxiv.org/abs/2403.18579.

[14] Hutter F., Kotthoff L., Vanschoren J. (eds.): *Automated Machine Learning*, Springer International Publishing, 2019.

[15] Hutter F., Kotthoff L., Vanschoren J. (eds.): *Automated Machine Learning: Methods, Systems, Challenges*, Springer, Cham, 2019. doi: 10.1007/978-3-030-05318-5.

[16] Klau D., Krause H., Kreplin D., Roth M., Tutschku C., Zöller M.: AutoQML – A Framework for Automated Quantum Machine Learning, Technical Report, 2023. https://www.digital.iao.fraunhofer.de/content/dam/iao/ikt/de/documents/AutoQML_Framework.pdf.

[17] Klau D., Zöller M., Tutschku C.: Bringing Quantum Algorithms to Automated Machine Learning: A Systematic Review of AutoML Frameworks Regarding Extensibility for QML Algorithms, 2023. https://arxiv.org/abs/2310.04238.

[18] Koike-Akino T., Wang P., Wang Y.: AutoQML: Automated Quantum Machine Learning for Wi-Fi Integrated Sensing and Communications, 2022. https://arxiv.org/abs/2205.09115.

[19] Kundu A.: Reinforcement learning-assisted quantum architecture search for variational quantum algorithms, 2024. https://arxiv.org/abs/2402.13754.

[20] Kundu A., Sarkar A., Sadhu A.: KANQAS: Kolmogorov-Arnold Network for Quantum Architecture Search, 2024. https://arxiv.org/abs/2406.17630.

[21] Lusnig L., Sagingalieva A., Surmach M., Protasevich T., Michiu O., McLoughlin J., Mansell C., *et al.*: Hybrid Quantum Image Classification and Federated Learning for Hepatic Steatosis Diagnosis, *Diagnostics*, vol. 14(5), 2024. doi: 10.3390/diagnostics14050558.

[22] Martyniuk D., Jung J., Paschke A.: Quantum Architecture Search: A Survey, 2024. https://arxiv.org/abs/2406.06210.

[23] Moussa C., Patel Y.J., Dunjko V., Bäck T., van Rijn J.N.: Hyperparameter importance and optimization of quantum neural networks across small datasets, *Machine Learning*, vol. 113, pp. 1941–1966, 2024. doi: 10.1007/s10994-023-06389-8.

[24] Moussa C., van Rijn J.N., Bäck T., Dunjko V.: Hyperparameter Importance of Quantum Neural Networks Across Small Datasets. In: P. Pascal, D. Ienco (eds.), *Discovery Science*, Lecture Notes in Computer Science, vol. 13601, pp. 32–46, Springer, Cham, 2022. doi: 10.1007/978-3-031-18840-4_3.

[25] Ostaszewski M., Trenkwalder L.M., Masarczyk W., Scerri E., Dunjko V.: Reinforcement learning for optimization of variational quantum circuit architectures, 2021. https://arxiv.org/abs/2103.16089.

[26] Paszke A., Gross S., Chintala S., Chanan G., Yang E., DeVito Z., Lin Z., *et al.*: Automatic differentiation in PyTorch. In: *NIPS-W*, 2017.

[27] Raschka S.: Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning, 2020. https://arxiv.org/abs/1811.12808.

[28] Rybotycki T., Gawron P.: AQMLator Documentation. https://aqmlator.readthedocs.io/en/latest/index.html. Accessed: 2024-09-18.

[29] Rybotycki T., Gawron P.: AQMLator PyPi page, https://pypi.org/project/AQMLator/. Accessed: 2024-09-18.

[30] Shor P.: Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[31] Subasi O.: Toward Automated Quantum Variational Machine Learning, 2023. https://arxiv.org/abs/2312.01567.

[32] Xanadu Quantum Technologies: PennyLane Documentation, https://docs.pennylane.ai/en/stable/index.html. Accessed: 2024-09-18.

Affiliations

Tomasz Rybotycki
Systems Research Institute of the Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland
Nicolaus Copernicus Astronomical Center of the Polish Academy of Sciences, Bartycka 18, 00-716 Warsaw, Poland
Center of Excellence in Artificial Intelligence of AGH University, al. Mickiewicza 30, 30-059 Krakow, Poland, rybotycki.tomasz+aqmlator@gmail.com

Piotr Gawron
Nicolaus Copernicus Astronomical Center of the Polish Academy of Sciences, Bartycka 18, 00-716 Warsaw, Poland
Center of Excellence in Artificial Intelligence of AGH University, al. Mickiewicza 30, 30-059 Krakow, Poland, pgawron@agh.edu.pl

Received: 6.03.2025
Revised: 12.03.2025
Accepted: 12.03.2025