Quantum Machine Intelligence (2024) 6:59
https://doi.org/10.1007/s42484-024-00198-5

RESEARCH ARTICLE

Distributed hybrid quantum-classical performance prediction for hyperparameter optimization

Eric Wulff[1] · Juan Pablo Garcia Amboage[1] · Marcel Aach[2,3] · Thorsteinn Eli Gislason[3] · Thorsteinn Kristinn Ingolfsson[3] · Tomas Kristinn Ingolfsson[3] · Edoardo Pasetto[2,4] · Amer Delilbasic[2,3] · Morris Riedel[2,3] · Rakesh Sarma[2] · Maria Girone[1] · Andreas Lintermann[2]

Received: 15 April 2024 / Accepted: 3 September 2024

Abstract

Hyperparameter optimization (HPO) of neural networks is a computationally expensive procedure, which requires a large number of different model configurations to be trained. To reduce such costs, this work presents a distributed, hybrid workflow, that runs the training of the neural networks on multiple graphics processing units (GPUs) on a classical supercomputer, while predicting the configurations' performance with quantum-trained support vector regression (QT-SVR) on a quantum annealer (QA). The workflow is shown to run on up to 50 GPUs and a QA at the same time, completely automating the communication between the classical and the quantum systems. The approach is evaluated extensively on several benchmarking datasets from the computer vision (CV), high-energy physics (HEP), and natural language processing (NLP) domains. Empirical results show that resource costs for performing HPO can be reduced by up to 9% when using the hybrid workflow with performance prediction, compared to using a plain HPO algorithm without performance prediction. Additionally, the workflow obtains similar and in some cases even better accuracy of the final hyperparameter configuration, when combining multiple heuristically obtained predictions from the QA, compared to using just a single classically obtained prediction. The results highlight the potential of hybrid quantum-classical machine learning algorithms. The workflow code is made available open-source to foster adoption in the community.

Keywords Hyperparameter optimization · Quantum annealing · Hyperband · Distributed computing

1 Introduction

The performance of neural networks with respect to their accuracy is highly sensitive to the choice of hyperparameters (HPs). To optimize HPs efficiently, current popular hyperparameter optimization (HPO) algorithms, such as Hyperband (Li et al. 2017), the Asynchronous Successive Halving Algorithm (ASHA) (Li et al. 2018), and Bayesian Optimization Hyperband (BOHB) (Falkner et al. 2018), rely on early termination. In this method, underperforming trials are automatically terminated to free up compute resources for more promising trials. Choosing, e.g., the validation accuracy or the validation loss as a metric to relatively rank the trials may lead to a suboptimal selection of trials to terminate due to the non-linearity of the training process. That is, the ranking of trials is unpredictably dynamic over the number of epochs. In practice, this problem is often mitigated by training such a large number of total model configurations that the impact of wrongfully early stopping a few promising configurations is minimized.

A potential extension of the early termination approach is to use a non-linear stopping criterion, e.g., using a model performance predictor that predicts future model performance

Eric Wulff, Juan Pablo Garcia Amboage, and Marcel Aach contributed equally to this work.

✉ Eric Wulff
eric.wulff@cern.ch

[1] CERN, Espl. de Particules 1, 1211 Meyrin, Geneva, Switzerland

[2] Jülich Supercomputing Centre, Forschungszentrum Jülich, Wilhelm-Johnen-Straße, 52428 Jülich, Germany

[3] School of Engineering and Natural Sciences, University of Iceland, 107 Reykjavík, Iceland

[4] RWTH Aachen University, 52056 Aachen, Germany

Published online: 14 September 2024

improvements from a partially trained model. Such predictions can be used to either rank configurations in a more informed manner or to make the evaluation process more aggressive, i.e., by replacing actual evaluations of some configurations that have been trained up to a certain epoch with predictions of their performance. This was first suggested by Baker et al. (2017), where support vector regression (SVR) was used as a performance predictor. In conclusion, the training of the most promising configurations can be prioritized based on the predicted performance. This way, it is avoided to fully train configurations predicted to perform poorly. Consequently, this approach holds great potential for reducing the time and computational resources required for HPO.

In this work, a novel HPO algorithm called Swift-Hyperband (Amboage et al. 2023) (an extension of the original Hyperband method) is incorporated into a hybrid high-performance computing (HPC) environment where it distributes the training of the target models onto multiple graphics processing units (GPUs) and trains the performance predictors on a quantum annealer (QA) in an autonomous fashion. Results show that Swift-Hyperband accelerates the HPO process of a high-energy physics (HEP)-based algorithm, machine-learned particle flow (MLPF) (Pata et al. 2021a), and other machine learning models that run on HPC systems. The goal of this paper is to show empirically how such a way of integrating a quantum system in a machine learning (ML) workflow can lead to resource savings in comparison to the plain Hyperband algorithm without any performance prediction and how leveraging a QA for the performance prediction can match and sometimes outperform classical performance prediction. In specific, the workflow can be applied to ML models of any size, as only a small part of the computation needs to be done on the QA, and the main training of the models is performed on classical GPUs. This makes it different from pure quantum ML workflows, which can currently only handle small problems. It should be noted that currently, there is not a clear, theoretical advantage of integrating QAs.

The presented hybrid workflow serves as a proof of concept for the integration of HPC and quantum computing technologies in a large-scale distributed fashion. By demonstrating the feasibility of such an integration, the way for future endeavors in harnessing the combined power of these computing paradigms, such as applying classic HPO techniques to quantum models among other future use cases, is paved.

The paper is structured as follows: The technical background and related work are explained in Section 2. Section 3 presents the experimental setup as well as the distributed Swift-Hyperband algorithm. The empirical results are detailed in Section 4, while Section 5 provides a conclusion and directions for future work.

2 Related work and theoretical background

In this section, the relevant literature regarding HPO in general, the process of performance prediction to speed it up, and the foundations of classical and quantum SVR methods are summarized.

2.1 Quantum annealing and QUBO problems

Quantum annealing (Apolloni et al. 1989; Kadowaki and Nishimori 1998) is a heuristic for optimization based on quantum computation that is often used to solve quadratic unconstrained binary optimization (QUBO) problems, i.e., discrete unconstrained optimization problems in which the problem variables can take values over a binary set (Date et al. 2021). The general cost function $E(v_1, \ldots, v_M)$ of a QUBO problem with M binary variables $v_i,\ i = 1, \ldots M$ is given by the following:

$$E(v_1, \ldots, v_M) := \sum_{i \leq j} Q_{ij} v_i v_j, \tag{1}$$

where Q is the QUBO weight matrix that stores the coefficients of the problem. In quantum annealing, the quantum system is set to the ground state of an initial Hamiltonian H_i, whose ground state is known and easy to prepare. The system is then slowly evolved for a total annealing time T_a by adding the contribution of a target Hamiltonian H_p, whose ground state encodes the solution of the optimization problem to be solved, and by reducing the contribution of the initial Hamiltonian H_i. The resulting Hamiltonian is then given by the following:

$$H(t) = A(t)H_i + B(t)H_p \tag{2}$$

where $A(t)$ is a monotonically decreasing function such that $A(t = 1) = 1$ and $A(t = T_a) = 0$, and $B(t)$ is a monotonically increasing function such that $B(t = 0) = 0$ and $B(t = T_a) = 1$ (McGeoch 2014). QAs from the company D-Wave implement a specific default annealing schedule, for which the corresponding values of the functions $A(t)$ and $B(t)$ can be found in the documentation.[1] On D-Wave systems, a default annealing time of $20\mu s$ is set. Both the

[1] D-Wave annealing schedule: https://docs.dwavesys.com/docs/latest/doc_physical_properties.html.

annealing schedule and annealing time are set to their default values in this study.

2.2 Hyperparameter optimization algorithms

Early termination algorithms are nowadays one of the main tools for HPO of deep neural networks. They have been shown to save considerable amounts of resources in the HPO process without losing the ability to find good configurations for the target model (Li et al. 2018; Falkner et al. 2018; Yu and Zhu 2020). The successive halving algorithm (Jamieson and Talwalkar 2016) is the foundation of most of the relevant early termination algorithms. Successive halving trains a set of randomly generated configurations for a certain number of epochs, discards the worst-performing half, and repeats this process until only one configuration remains. The Hyperband algorithm (Li et al. 2017) runs multiple instances of successive halving sequentially with different numbers of initial configurations and different locations of the decision points. BOHB (Falkner et al. 2018) is an algorithm that extends Hyperband by following a Bayesian optimization approach for selecting the configurations instead of generating them randomly. Finally, ASHA (Li et al. 2018) stands for asynchronous successive halving and extends successive halving to efficiently benefit from large-scale distributed resources.

2.3 Performance prediction

Performance prediction aims at predicting the future performance of a model under a given set of hyperparameters based on the early results of the training, e.g., using a partial learning curve (see Fig. 1). The problem can formally be defined as predicting the validation loss or validation accuracy $l(\lambda)_R$ at training epoch $R \in \mathbb{N}$ that will be obtained by a machine learning model with the hyperparameter configuration $\lambda \in \Lambda \subset \mathbb{R}^h$, where h is the number of hyperparameters of the network (see Baker et al. 2017 and Liu et al. 2022). The performance of the target model at previous training epochs can be used for this prediction. If the auxiliary model chosen to be used as a performance predictor is fast to train relative to the target model, performance prediction can be used to accelerate the HPO process. In this regard, it has been shown that computationally cheap regression methods, such as SVR, are good choices to be used as performance predictors (Baker et al. 2017).

A simple strategy to integrate performance prediction in an HPO process can be described as follows. First, a series of random HP configurations from a given search space is generated, a few of them are trained for R epochs, and the generated learning curves are used to train a performance predictor model such as an SVR. Next, the remaining configurations are initially trained for $\tau < R$ epochs (e.g., $\tau = \frac{R}{2}$), whereafter their future performances are predicted. Finally, only those configurations that are considered promising, according to the predicted performance, are allowed to continue training until epoch R. Note that the initial full and partial trainings can be done in parallel.

2.4 Support vector regression methods

Support vector machines (Boser et al. 1992; Drucker et al. 1996) are popular supervised learning algorithms that can be applied to classification and regression tasks. One of the reasons for their popularity is found in the fact that the determination of the model parameters amounts to a convex optimization problem; therefore, any local solution corresponds to a global one (Bishop 2006). However, it is important to point out that the global minimum of the training cost function may not be optimal in terms of generalization to the test set (Willsch et al. 2020). Moreover, they can approximate non-linear functions by applying the so-called *kernel trick* (Burges 1998) thus increasing the algorithm's expressive potential.

The mathematical formulation of the SVR is subsequently briefly outlined. Given a training dataset $\{(\mathbf{x}_n, y_n),\ n = 1, \ldots, N\}$, where $\mathbf{x}_n \in \mathbb{R}^z$ is the input vector, N is the number of training samples, and y_n is its corresponding target value. The objective is to approximate a regression function

$$g(\mathbf{x}) = \sum_{n=1}^{N} (\alpha_n - \hat{\alpha}_n)\kappa(\mathbf{x}_n, \mathbf{x}) + b, \tag{3}$$

which maps from $\mathbb{R}^z$ to $\mathbb{R}$. The parameters $\alpha_n, \hat{\alpha}_n$ are determined in the optimization process. The term $\kappa(\mathbf{x}_n, \mathbf{x})$ denotes the kernel function, which is in this study the RBF kernel with formula $e^{(-\gamma||\mathbf{x}_n - \mathbf{x}_m||^2)}$. More generally, a RBF kernel is a kernel whose value depends only on the distance of the input vectors, i.e., $\kappa(\mathbf{x}_m, \mathbf{x}_n) = \kappa(||\mathbf{x}_m - \mathbf{x}_n||)$. RBF kernels are one of the most popular choices for SVR kernels along with polynomial and sigmoid kernels (Bishop 2006). It can be shown that the training phase amounts to solving the following constrained optimization problem (also referred to as the cost function):

$$\begin{aligned} L(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = &\frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} (\alpha_n - \hat{\alpha}_n)(\alpha_m - \hat{\alpha}_m)\kappa(\mathbf{x}_n, \mathbf{x}_m) + \\ &- \epsilon \sum_{n=0}^{N-1} (\alpha_n + \hat{\alpha}_n) + \sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha}_n) y_n, \end{aligned} \tag{4}$$

satisfying the constraints:

$$\sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha}_n) = 0, \tag{5a}$$

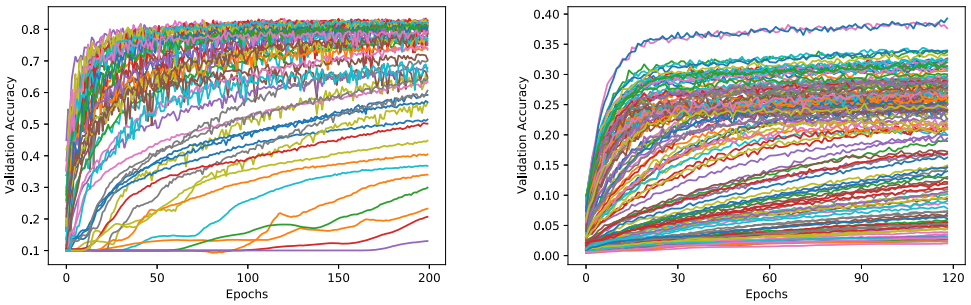


Fig. 1 Example learning curves of different convolutional neural networks (CNNs) on the CIFAR-10 and TinyImageNet dataset

$$0 \leq \alpha_n \leq C, \tag{5b}$$

$$0 \leq \hat{\alpha}_n \leq C, \tag{5c}$$

where the terms C and ϵ are hyperparameters that control the overfitting and the error sensitivity. The vectors $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\alpha}}$ are defined as $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_N\}$ and $\hat{\boldsymbol{\alpha}} = \{\hat{\alpha}_1, \ldots, \hat{\alpha}_N\}$, respectively. The value of b can be obtained from any point for which $0 < \alpha_n < C$ by

$$b = y_n - \epsilon - \sum_{m=1}^{N} (\alpha_m - \hat{\alpha}_m) \kappa(\mathbf{x}_n, \mathbf{x}_m). \tag{6}$$

Due to the constraints, it must satisfy $\epsilon + g(x_n) - y_n = 0$. It is, however, preferable to average over different estimates of b to yield a stable solution, i.e.,

$$b = \frac{1}{|\mathcal{S}|} \sum_{n \in \mathcal{S}} \left(y_n - \epsilon - \sum_{m=1}^{N} (\alpha_m - \hat{\alpha}_m) \kappa(\mathbf{x}_n, \mathbf{x}_m) \right). \tag{7}$$

In this equation, $\mathcal{S}$ corresponds to the set of *support vectors*, i.e., those vectors that contribute to the prediction of the target value (c.f. Eq. 3) (Bishop 2006).

2.5 Quantum support vector regression

To optimize the training phase of a quantum-trained support vector regression (QT-SVR) (Pasetto et al. 2022), it is necessary to reformulate the optimization problem as either an Ising or QUBO problem. In the current study, the problem is restructured as a QUBO problem by carrying out a three-step problem conversion procedure, which consists of (i) encoding the problem variables, (ii) adding penalty terms to encode the constraints, and (iii) defining the QUBO matrix. These steps are subsequently explained in more detail.

2.5.1 Problem variable encoding

The first step towards the construction of the QUBO problem consists of turning the problem variables into binary ones. Specifically, each of the original problem variables is encoded using K qubits according to

$$\alpha_n = \sum_{k=0}^{K-1} B^{k-P} a_{Kn+k}, \tag{8}$$

$$\hat{\alpha}_n = \sum_{k=0}^{K-1} B^{k-P} a_{K(N+n)+k}, \tag{9}$$

where B is an encoding basis and a is the value of the qubits. The parameter P is used to allow the usage of negative exponents in the encoding procedure.

This results in $2KN$ QUBO variables, where the first KN variables are used to encode $\boldsymbol{\alpha}$, whereas the last KN variables are used to represent $\hat{\boldsymbol{\alpha}}$. The $\boldsymbol{\alpha}$ and $\hat{\boldsymbol{\alpha}}$ are then substituted into (4).

2.5.2 Penalty term addition

The QUBO problem must be unconstrained. It is, therefore, necessary to add penalty terms, whose influence is regulated by hyperparameters, to implicitly enforce the constraints. To enforce (5a), a square penalty regulated by the hyperparameter ξ is added to the cost function in Eq. 4:

$$\xi \left(\sum_{n=1}^{N} (\alpha_n - \hat{\alpha})_n \right)^2. \tag{10}$$

The constraints defined by Eqs. 5b and 5c, which are also referred to as box constraints, are implicitly satisfied by the encoding equations. As the qubit values collapse to either 0 or 1, the maximum value that each α_n and $\hat{\alpha}_n$ can take is

$$\sum_{i=0}^{K} B^{K-1-P}, \tag{11}$$

This is obtained if all qubit values a_{Kn+k} or $a_{K(N+n)+k}$ collapse to 1 for $k = 0, \ldots, K-1$.

2.5.3 QUBO matrix definition

After the addition of the penalty terms (see Eq. 10) to the cost function (see Eq. 4) and the subsequent encoding of the problem variables (see Eq. 8), the final QUBO cost function takes the form:

$$\sum_{n,m=0}^{N-1}\sum_{i,j=0}^{K-1}\sum_{s,t=0}^{1} a_{K(sN+n)+i}\,\tilde{Q}_{K(sN+n)+i,K(tN+m)+j}\,a_{K(tN+m)+j}, \tag{12}$$

where $\tilde{Q}$ is an $2KN \times 2KN$ matrix that encodes the problem and whose elements are given by

$$\tilde{Q}_{K(sN+n)+i,K(tN+m)+j} = (-1)^{(1-\delta_{st})} B^{i+j-2P}\left(\frac{1}{2}\kappa(\mathbf{x}_n,\mathbf{x}_m)+\xi\right) \tag{13}$$

$$+\,\delta_{nm}\delta_{ij}B^{i-P}\delta_{st}\left(\epsilon+(-1)^{(1-s)(1-t)}y_n\right) \tag{14}$$

To obtain a problem formulation similar to Eq. 1, it is necessary to construct the upper-triangular $2KN \times 2KN$ QUBO matrix Q from $\tilde{Q}$ by using

$$Q_{i,j} = \begin{cases} \tilde{Q}_{i,j}+\tilde{Q}_{j,i}, & \text{if } i<j; \\ \tilde{Q}_{i,j}, & \text{if } i=j; \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

The minimization problem can then be written as

$$\min_{\underline{\boldsymbol{a}}\in\{0,1\}^{2KN}} \underline{\boldsymbol{a}}^T\, Q\, \underline{\boldsymbol{a}}, \tag{16}$$

where $\underline{\boldsymbol{a}}$ is the $2KN$ vector obtained by the concatenation of $\boldsymbol{a}$ and $\hat{\boldsymbol{a}}$.

The final step to run a problem instance on the QA is creating a *minor embedding* (Choi 2011). This arises from the fact that the graph structure of the quantum processing unit (QPU) is not fully connected. It is, therefore, necessary to represent a logical qubit with a group of connected qubits that are constrained to have the same values. For a given QUBO problem, it is possible to construct the corresponding problem graph $G(V,\mathcal{E})$ by considering as the set of nodes $V=\{a_i,\ldots,a_n\}$, where each node corresponds to a problem variable and the set of edges $\mathcal{E}=\{(a_i,a_j),\ \forall\ (i,j)$ such that $Q_{i,j}\neq 0\}$. The D-Wave Advantage system uses a *Pegasus* (Boothby et al. 2020) topology graph. For the experiments conducted in this study, the problem graph always had the same structure for each test run since it is a fully connected graph with a number of nodes equal to $2KN$. The values of K and N were always the same for each experimental run. Therefore, the embedding was calculated only once using the functions provided by the official D-Wave Ocean tools ,[2] which provide an interface to the QA machine.

2.5.4 Advantages of Q-SVMs over classical counterparts

So far, established work combining support vector machines (SVMs) and general quantum computing (QC) has focused on performing the computation of the kernel on the quantum hardware (Rebentrost et al. 2014). This approach has a theoretical advantage in runtime, as kernel computation requires only logarithmic runtime in the quantum setting but at least quadratic runtime in the classical setting. However, computing not the kernel but the training process (as performed in this study) on a QA has two advantages:

- **Time complexity:** Usually, the whole training procedure of (classical) SVMs and SVRs in specific has **cubic** time complexity (Bottou et al. 2007; Abdiansah Abdiansah 2015), which can lead to long runtimes for large datasets. To the contrary, a QUBO solved on QAs returns a set of low-energy solutions after a predefined amount of time, **independent of the input problem size** (Date et al. 2021). This can lead to potential speed-ups for large training datasets in the future.
- **Solution combinations:** While classical SVMs return a single optimal solution, QAs provide multiple low-energy solutions to a given problem. These different solutions can be combined, analogous to ensembling. This has empirically been shown to improve generalization performance in classical learning algorithms (Dietterich 2000), and a similar effect can be observed in quantum learning algorithms (Willsch et al. 2020; Cavallaro et al. 2020).

It should be noted that the set of QAs solutions is not guaranteed to be theoretically optimal (as in the case of classical SVMs) and that limitations in terms of the number of available qubits on current QAs also limit the problem size that can be solved in a quantum setting. For D-Wave QAs, the number of available qubits has however grown steadily from 2000 to more than 5000 during the past years .[3]

[2] D-Wave Ocean SDK version 6.9.0: https://www.dwavesys.com/solutions-and-products/ocean/.

[3] D-Wave QAs: https://www.dwavesys.com/solutions-and-products/systems/.

3 Experimental setup

3.1 Machines

Two different types of machines are used for the hybrid setup of the workflow. The classical calculations are executed on the Extreme Scale Booster partition of the DEEP-EST supercomputer.[4] It features a total of 75 nodes, where each node is equipped with an Intel Xeon Central Processing Unit (CPU) with 8 cores and a NVIDIA V100 GPU. The quantum calculations take place on the D-Wave Advantage system JUPSI ,[5] as of 04/2024 the largest QA in Europe with 5614 qubits. Both machines are located at the Jülich Supercomputing Centre.

3.2 Performance prediction algorithm

The Fast-Hyperband (Baker et al. 2017) algorithm is a modified version of Hyperband that adds an additional, performance prediction-based decision point at the end of every epoch. This allows the algorithm to estimate which configurations are less likely to be promoted to the next round and terminate them earlier than the original Hyperband. Naturally, this saves computational resources compared to the classical Hyperband algorithm.

The Swift-Hyperband algorithm is similar to Fast-Hyperband but adds only one extra decision point based on performance prediction inside each Hyperband bracket round (instead of one extra decision point every epoch) (see Fig. 2). Therefore, Swift-Hyperband requires training considerably fewer performance predictors than Fast-Hyperband. This enables Swift-Hyperband to run in a hybrid quantum-classical workflow, where the performance predictors are trained using a QA. Furthermore, each round of Swift-Hyperband can be parallelized as the trial trainings can be performed in parallel in contrast to Fast-Hyperband, which is a sequential approach.

Algorithms 1 and 2 in Appendix A present the pseudocode for Swift-Hyperband and its auxiliary routine `run_then_return_val_loss`, where the performance predictors are both trained and used to make predictions. For simplicity, only the pseudocode of the sequential version of Swift-Hyperband is presented. In the distributed implementation, the parallelization is performed at a round level in the same way that is described for Hyperband in Li et al. (2018). This is, the outer *for* loop in the routine `run_then_return_val_loss` is adapted to run its multiple iterations in parallel. The only disadvantage of this type of parallelization is the same as the one discussed for the classical version of Hyperband in Li et al. (2018). That is, the number of iterations in the parallelized *for* loop is not constant throughout the whole algorithm but instead decreases from the start to the end of each bracket. For this reason, using a high number of parallel computing nodes may result in some of them being idle towards the end of each bracket.

3.3 Hybrid quantum-classical workflow

Swift-Hyperband is suited for hybrid quantum-classical workflows. A QT-SVR trained on a QA can be used as the performance predictor, while the trainings of the target model can be performed in parallel across several nodes in an HPC center. However, this does not hold for Fast-Hyperband. Note that, apart from being a sequential algorithm, Fast-Hyperband has the disadvantage that a high number of performance predictors are necessary. This makes it infeasible to run with QT-SVRs due to the time needed to communicate with the QA.

For this work, a hybrid quantum-classical implementation of Swift-Hyperband that uses the Message Passing Interface (MPI) standard to communicate across multiple nodes in an HPC cluster and relies on the D-Wave Ocean SDK to connect to a D-Wave Advantage System to train the QT-SVRs has been developed. In this implementation, one node of the HPC cluster acts as a head node, communicating with the QA as well as coordinating up to 50 GPU-equipped worker nodes, where the different target model configurations are trained. This workflow is illustrated graphically in Fig. 3.

On current quantum hardware, the size of problems that can be computed is still limited. Considering the cubic time complexity associated with the training of classical SVR predictors and the capability of a quantum system to run calculations in a matter of milliseconds, a speed-up can, however, be expected for large datasets on future quantum hardware (Date et al. 2021).

3.3.1 QT-SVR solution combination

As mentioned in Sect. 2.5.4, the QA returns a set of low-energy solutions after a fixed amount of time, instead of just a single one. These multiple solutions can then be combined to create a more robust prediction, which has already been performed in earlier literature (Willsch et al. 2020; Pasetto et al. 2022). Given the true training target values y^{train} and the predicted target values $\tilde{y}_i^{train}$, where $i = 1, \ldots, s$ in the range of the total number of solutions s, each of the s candidate solutions (i.e., the different set of values for α_n and $\hat{\alpha}_n$) is assigned a weight w_i, $i = 1, \ldots, s$. To do so, for each set of α_n and $\hat{\alpha_n}$, the mean squared error loss L_i is

[4] DEEP-EST: https://www.fz-juelich.de/en/ias/jsc/systems/prototype-systems/deep_system

[5] JUPSI: https://www.fz-juelich.de/en/ias/jsc/systems/quantum-computing/juniq-facility/juniq/d-wave-advantagetm-system-jupsi

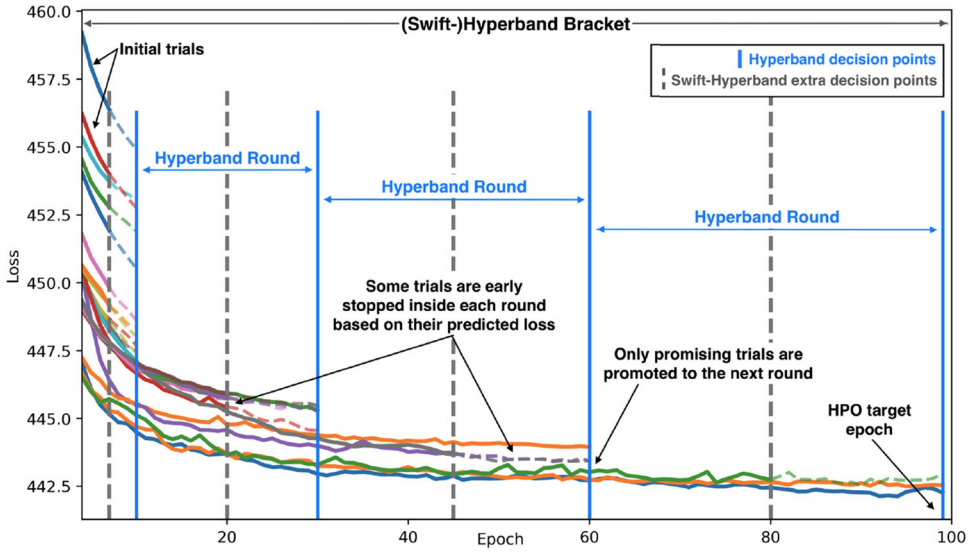


Fig. 2 Graphic representation of the Swift-Hyperband algorithm, taken from Amboage et al. (2023)

computed on the training dataset:

$$L_i = \frac{1}{N}\sum_{j=1}^{N}(y^{train(j)} - \tilde{y}^{train(j)})^2 \tag{17}$$

In order to give more credit to the solutions which achieved better performance (on the training data) and to reduce the contributions of the solutions that performed worse, for each $i = 1, \ldots, s$, a coefficient is defined $\hat{w}_i$ as $\hat{w}_i := \frac{1}{L_i}$. Finally, the solution combination coefficients are obtained as

$$w_i := \frac{\hat{w}_i}{\sum_{l=1}^{s} w_l} \tag{18}$$

These weights are then used to obtain the final solution $\boldsymbol{\alpha}^{final} := \{\alpha_1^{final}, \ldots, \alpha_N^{final}\}$ as a weighted average over the solutions returned by the QA:

$$\boldsymbol{\alpha}^{final} := \sum_{i=1}^{s} w_i \boldsymbol{\alpha}_i, \tag{19}$$

with $\boldsymbol{\alpha}_i := \{\alpha_i, \ldots, \alpha_s\}$. Essentially, this resembles a set of weak classifiers, that combined yield a strong classifier (Willsch et al. 2020; Cavallaro et al. 2020). In this study, the number of solutions combined is set to $s = 4$.

3.4 Datasets and models

For this study, the capabilities of the hybrid workflow are evaluated on multiple datasets and neural network models. Fully connected neural networks (FCNN) are trained on small regression datasets from the OpenML project (Vanschoren et al. 2014), CNNs are trained on two well-known

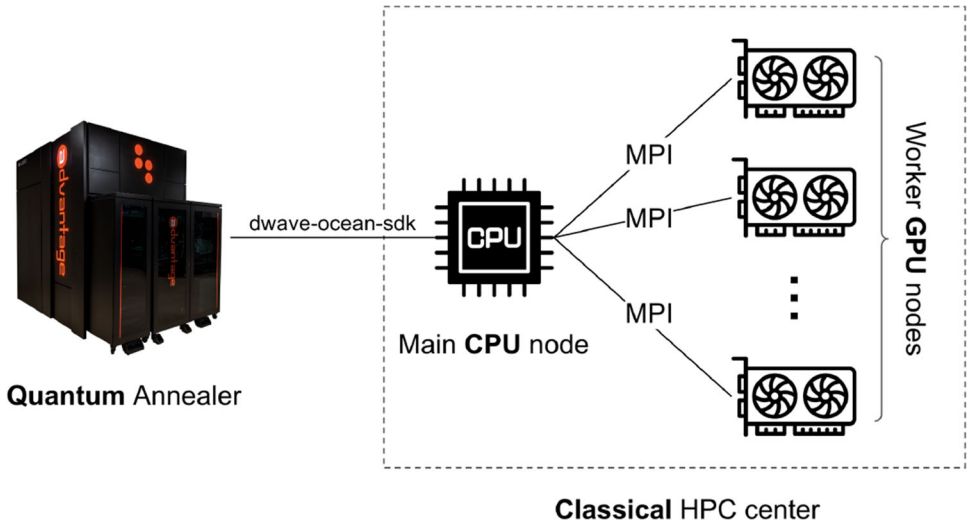


Fig. 3 Components along with their communication protocols used for the Swift-Hyperband distributed hybrid quantum-classical implementation

Table 1 Hyperparamters from the HPOBench search space used on the OpenML datasets

Hyperparameter	Type	Range
Alpha	Float	Log[1e-8, 1]
Batch size	Int	Log[4,256]
Depth	Int	[1,3]
Learning rate	Float	Log[1e-5, 1]
Width	Int	[16,1024]

Depth and width determine the general shape of the network while batch size, initial learning rate, and weight decay (alpha) influence the Adam optimizer

Table 2 Hyperparameter search space used on CIFAR-10 and TinyImageNet

Hyperparameter	Type	Range
Layers	Int	[2,3,4]
Filters	Int	[16,32,48,64]
Batch size	Int	[64,128,256,512]
Learning rate	Float	Log[1e-4, 1]
Momentum	Float	Log[1e-4,0.9]

Layers and filters determine the general shape of the CNN while batch size, learning rate, and momentum influence the optimizer

medium-sized datasets from the computer vision (CV) domain, a long short-term memory (LSTM) is trained on a natural language processing (NLP) dataset, and the MLPF algorithm is trained on a HEP dataset. In the following, the datasets as well are explained. To enable the reproducibility of the empirical results, also, the HP search spaces that the HPs are sampled from a provided in detail.

3.4.1 OpenML datasets

The OpenML platform[6] features a wide selection of easily accessible datasets, algorithms, and experiments. To make sure the results are reproducible, the method is tested on three datasets from the OpenML Curated Tabular Regression benchmark (Fischer et al. 2023). The focus of this benchmark is on tabular, high-quality datasets from different domains with 500 to 100,000 observations that are not trivially solvable by linear methods. For this study, the Grid Stability (id 44973), Video Transcoding (id 44974), and Naval Propulsion Plant (id 44969) datasets are selected.

However, to generate reproducible learning curves, not only the dataset and evaluation procedure are important, but also the hyperparameter search space that the different model configurations are sampled from has to be consistent. Therefore, the chosen OpenML datasets are trained on the hyperparameter search space detailed in the HPOBenchmark library (Eggensperger et al. 2021), a general benchmark for HPO. The search space consists of two architectural parameters of FCNNs (depth and width) and three optimizer-related parameters (batch size, initial learning rate, and weight decay). The range the hyperparameters are sampled from is shown in Table 1. Each model is trained for a maximum of 50 epochs.

3.4.2 Computer vision datasets

The two datasets of choice from the CV domain are the CIFAR-10 (Krizhevsky 2009) and TinyImageNet (Mnmoustafa 2017) datasets. The CIFAR-10 dataset consists of 60,000 images in ten different classes. The training is performed on 50,000 images while the remaining 10,000 images are used to compute the validation accuracy. TinyImageNet contains 100,000 images, split into 200 classes. For each class, there are 500 training images and 50 validation images. For both datasets, the image pre-processing steps involved a `RandomCrop`, `Resize`, `RandomHorizontalFlip`, and `Normalization` operation from the torchvision library.[7] Similarly to Baker et al. (2017), small architectures with varying numbers of convolutional layers, convolutional filters, batch size, learning rate, and momentum are sampled from the hyperparameter search space (see Table 2) for the CIFAR-10 models, which are trained for a maximum of 100 epochs. For the TinyImageNet case, a fixed ResNet18 architecture (He et al. 2016) is trained for a maximum of 35 epochs and only optimizer-related parameters (batch size, learning rate, and momentum) are tuned. Performance prediction in both cases is performed on the validation set learning curves.

3.4.3 HEP model and dataset

In response to the considerable surge in data generation anticipated in large HEP experiments in the forthcoming decades, there are ongoing efforts towards substituting conventional CPU-based algorithms with neural network-powered ones that can be efficiently and readily executed on GPUs, field-programmable gate arrays (FGPAs), or other hardware accelerators. A prime illustration of such novel algorithms is the so-called MLPF (Pata et al. 2021a) algorithm, designed to perform particle-flow reconstruction (Sirunyan et al. 2017)

[6] OpenML: https://www.openml.org/

[7] Torchvision library: https://pytorch.org/vision/stable/index.html

Table 3 Hyperparameter search space used on MLPF

Hyperparameter	Type	Range
Learning rate	Float	Log[1e-6, 3e-2]
Dropout	Float	[0,0.5]
Weight decay	Float	Log[1e-6,1e-1]
Num graph layers id	Int	[0,4]
Num graph layers reg	Int	[0,4]
Bin size	Int	[8,16,32,64,128]
Output dim	Int	[8,16,32,64,128,256]

through a data-driven methodology. Advantages of MLPF include extensibility, portability, and scalability. Extensibility because the algorithm can easily be adapted to new detector geometries or conditions by retraining, portability, because the model can be executed on a wide variety of different hardware accelerators, and scalability, because runtime and memory consumption scale approximately linearly with the input collision event size.

The dataset used for HPO studies in this work is the open and publicly available DELPHES dataset (Pata et al. 2021b) first presented in Pata et al. (2021a). HPO is performed on a seven-dimensional search space (see Table 3), and each trial is trained for a maximum of 100 epochs.

3.4.4 NLP model and dataset

The dataset of choice from the NLP domain is the bAbI tasks dataset (Weston et al. 2015), in particular task 17. The bAbI tasks dataset consists of 20 elementary reasoning tasks, where task 17 is related to spatial and positional reasoning. Each particular instance of task 17 is conformed by a group of sentences related to the relative position of multiple colored blocks followed by a "yes or no" question about the location of one of the blocks. Therefore, the goal of the model is to infer the correct "yes/no" answer to the question given its precedent sentences. The training is performed on 1000 questions, and another 10,000 questions are used to compute the validation accuracy. A fixed LSTM architecture with varying training hyperparameters (see Table 4), where each model can be trained up to 300 epochs, is used .[8]

4 Results and evaluation

For performance evaluation, the distributed quantum-classical implementation of Swift-Hyperband is compared to the original Hyperband algorithm and to Swift-Hyperband using classical SVRs. Several target models from different domains, i.e., from CV, NLP, and HEP, are optimized. For each target model, the best accuracy or loss achieved by each algorithm is reported along with the average resources consumed (in terms of the total number of training epochs), using an average of three runs, each one initialized with a different random seed (0, 1, and 2). Detailed information on the target models and the HPO processes used to test the algorithms are shown in Table 5. The results, computed on 51 nodes of the DEEP-EST supercomputer in combination with the JUPSI QA, are shown in Figs. 4 and 5.

Table 4 Hyperparameter search space used on the LSTM

Hyperparameter	Type	Range
Learning rate	Float	Log[1e-10, 1]
Dropout	Float	[0,1]
Rho	Float	[0,1]
Weight decay	Float	Log[1e-5, 0.1]

From Figs. 4 and 5, it can be seen that Swift-Hyperband, both in the case of using SVRs and QT-SVRs, achieves results similar to classical Hyperband in terms of the target model performance while consuming less computational resources in all cases. The largest savings of $\sim 9.4\%$ are observed for the CNN training on CIFAR-10 (Fig. 4a), where Hyperband takes 3237 epochs, Swift-Hyperband SVR takes 3014, and Swift-Hyperband QT-SVR only 2960 epochs on average. For the Tiny ImageNet cases (Fig. 4b) Hyperband requires 872 training iterations, while Swift-Hyperband QT-SVR finishes in 834, resulting in a resource saving of $\sim 4.6\%$ with only a small difference in validation loss (0.012 vs. 0.014).

For all other cases, $\sim$ 2–5% in savings can be seen. While plain Hyperband only bases the future performance of a trial based on the current validation loss or accuracy, Swift-Hyperband can make use of performance prediction methods and thus terminate some trials earlier, resulting in fewer total training epochs. When comparing the Swift-Hyperband SVR and QT-SVR versions, the quantum-based regression method is able to match the validation set performance of the classical method in almost all cases, and in the majority of cases even outperforms it. As explained in Section 2.5.4, the main difference between the quantum and classical SVR is that the QT-SVR makes use of multiple, heuristically obtained predictions, which are weighted and combined into a single prediction. On the contrary, the classical SVR only uses a single deterministic prediction. The empirical results of these experiments prove that for these benchmarking cases, the QT-SVR produces more robust predictions of the future performance of trials when used inside a distributed version

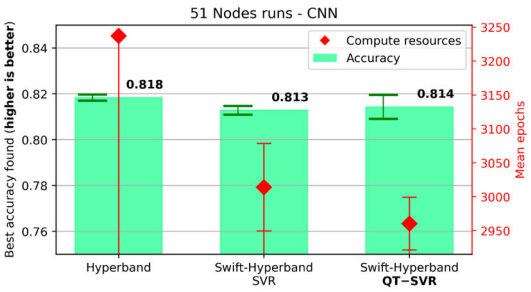
[8] LSTM Training: https://docs.ray.io/en/latest/tune/examples/includes/pbt_memnn_example.html

Table 5 Summary of the benchmarking cases used to test the HPO algorithms

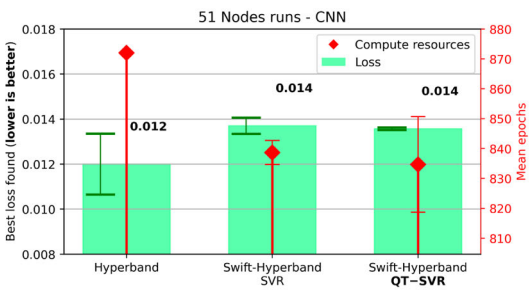
NN architecture	Dataset	Domain	Evaluation metric	# HPs for HPO	Target epoch for HPO	# GPU Nodes for HPO
CNN	CIFAR-10 (Krizhevsky 2009)	CV	Accuracy	5	100	50
CNN	TinyImageNet (Le and Yang 2015)	CV	Cross entropy loss	3	35	50
LSTM	bABI (Weston et al. 2015), task 17	NLP	Accuracy	4	300	50
MLPF (Pata et al. 2021a)	Delphes (Pata et al. 2021b)	HEP	Focal loss + huber loss	7	100	Simulated
FCNN	OpenML	Tabular	Mean squared error loss	5	50	50

of Swift-Hyperband, which then leads to higher validation scores in the majority of cases.

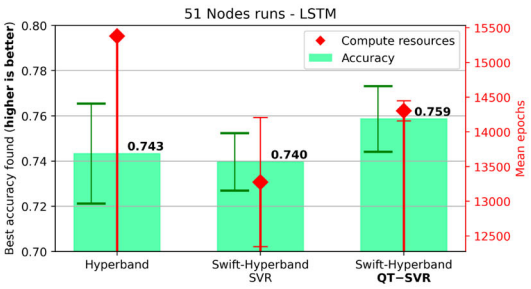
In terms of computing resources, the QT-SVR version requires fewer epochs than the SVR for the CNN cases (see



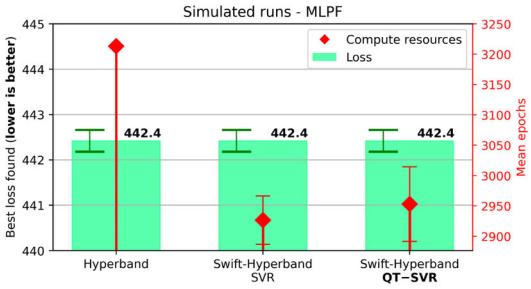
(a) CNN for cifar-10 dataset.



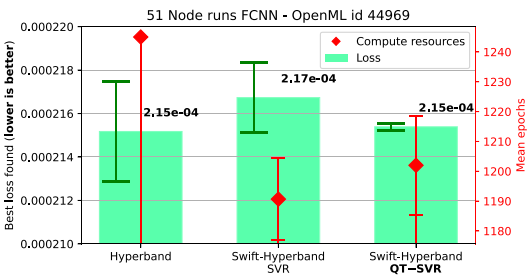
(b) CNN for Tiny ImageNet.



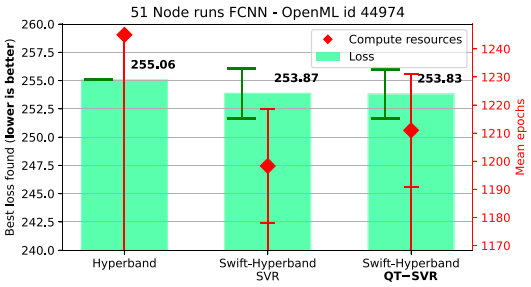
(c) LSTM for bAbI tasks dataset (task 17).



(d) MLPF for Delphes dataset.



(e) FCNN on the Naval Propulsion dataset.



(f) FCNN on the Video Transcoding dataset.

Fig. 4 Average resource consumption and performance of the best configuration found for each HPO algorithm applied to different target models. Results are averaged over three different random seeds, and error bars are shown. Note that the number of epochs for Hyperband is deterministic, and therefore, no error bar is shown for the algorithm

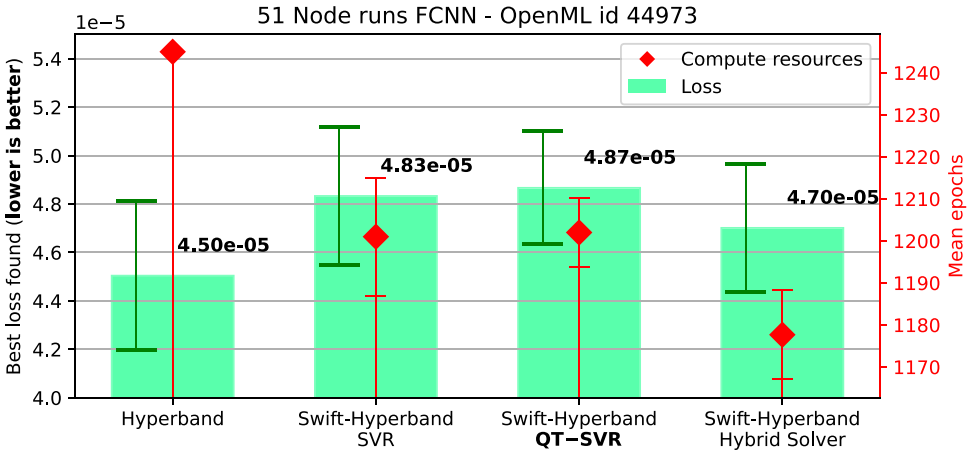


Fig. 5 Average resource consumption and performance of the best configuration found for each HPO algorithm on the grid stability dataset, including D-Waves internal hybrid solver. Results are averaged over three different random seeds, and error bars are shown. Note that the number of epochs for plain Hyperband is deterministic, and therefore, no error bar is shown for the algorithm

Figs. 4a and b), but more for all other instances. This indicates that on the one hand, the QT-SVR version tends to overestimate the performance of the target models, hence early stopping fewer configurations in the prediction-based decision points (and use more compute resources) for the LSTM, MLPF, and FCNN models. For the CNN models, on the other hand, it tends to underestimate the performance, thus early stopping more configurations and saving resources, but still achieves comparable target model performance. This is because the shapes of the learning curves (to which the regression method is applied) are highly dependent on the nature of the target model and the application case. It is interesting to observe that the dimension of the hyperparameter search space (e.g., a seven-dimensional search space for Fig. 4d and only a three-dimensional one for Fig. 4b) does not influence the performance of the algorithms.

As an alternative to solving the QUBO problem purely on the QA, D-Wave offers the option to use a cloud-based hybrid solver.[9] This solver internally solves a part of the problem with state-of-the-art classical algorithms, while sending only those parts to the QPU that primarily benefit from it. In this case, also, the combination of the solutions is performed internally and is not public. While this takes notably longer than pure GPU calculations (a few seconds vs. a few hundred milliseconds), it can also handle larger problems. A comparison on the Grid Stability dataset (see Fig. 5) shows the hybrid solver to outperform both SVR and QT-SVR-based Swift-Hyperband in terms of best-found model and the number of epochs used. The plain Hyperband algorithm achieves the lowest loss overall, but at the cost of a much higher compute resource consumption. In total, the results still indicate that the hybrid solver is able to estimate the performance of the target model with high accuracy.

5 Summary, conclusion, and outlook

This study presented a workflow for performing distributed, quantum-classical performance prediction for HPO. Compared to the established Hyperband algorithm, the proposed workflow saves resources with minimal sacrifices in terms of validation set performance of the best-found model. Especially in a distributed setting of 50 GPU running neural network training at the same time, resource savings of more than 9% are substantial. It was also evident that choosing a QT-SVR or a hybrid-solver method results empirically in better-performing models for a wide range of application cases compared to classical SVR, due to combining multiple heuristically obtained solutions. This stresses the potential of using quantum machine learning methods.

This work presents an important first step in the direction of automated integration of quantum devices in the supercomputing environment. This is of great importance, as current quantum machines are still too small to solve meaningful problems on their own. By combining them with a powerful supercomputer, it becomes possible to tackle relevant, real-world problems from a diverse set of scientific domains. The proposed workflow is agnostic to the underlying machine learning model and can be applied to any problem. As the code of the workflow is open-source,[10] the research community can benefit directly from this work.

[9] D-Wave Hybrid Solver: https://docs.ocean.dwavesys.com/en/latest/overview/hybrid.html

[10] GitHub link: https://github.com/JP-Amboage/qtml-hybrid-workflow

The most promising direction of future work is the usage of larger and more advanced quantum hardware. For this study, a QA was chosen, as it is able to handle a much larger problem size than current, gate-based machines. With increases in the number of qubits of gate-based machines and advances in algorithm development, it might be possible to run other, more advanced quantum optimization algorithms that have the potential to not only save compute resources in a quantum-classical setting but also find more accurate solutions.

Appendix: Swfit-Hyperband pseudocode

Algorithm 1 Swift-Hyperband.

Input: $R, \eta, d, \phi, known\_curve$
Output: Configuration with lowest loss seen during the algorithm
Initialize: $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max}+1)R$, $D = \text{Dict}()$, $M = \text{Dict}()$
for $s \in \{s_{\max}, s_{\max-1}, ..., 0\}$ **do**
  $n = \lceil \frac{B\eta}{R(s+1)} \rceil$, $r = R\eta^{-s}$
  `/* Begin Succesive Halving with n different configurations */`
  for $i \in \{0, ..., s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    if $r_i \notin D.keys$: $D[r_i] = \text{List}()$
    if $r_i \notin M.keys$: $M[r_i] = \text{Dict}()$
    $n_{next} = \lfloor \frac{n_i}{\eta} \rfloor$ **if** $i \neq s$, **else** 1
    `/* Performance prediction and parallelization (if applied) take part inside the following routine */`
    $L = \text{run\_then\_return\_val\_loss}(T, r_{prev}, r_i, n_{next}, d, \phi, D, M, known\_curve)$
    $T = \text{top\_k}(T, L, \lfloor n_i/\eta \rfloor)$
    $r_{prev} = r_i$
  end for
end for

Algorithm 2 Routine run_then_return_val_loss for Swift-Hyperband.

Input: $T, r_{prev}, r_i, n_{next}, D, M, d, \phi, known\_curve$
Output: L (List with the final loss for each configuration in T)
Initialize:
  $L = \text{List}()$, $fully\_trained = 0, dp = r_{prev} + \lceil (r_i - r_{prev}) known\_curve \rceil$, $thres = 0$
for $t \in$ T **do**
  $l = \text{List}()$
  for $i \in \{0, .., dp-1\}$ **do**
    $l_i = \text{loss\_of\_t\_at\_epoch\_i}(t, i)$
    $l.\text{append}(l_i)$
  end for
  if $fully\_trained < \phi \cdot n_{next}$ **or** $D[r_i].\text{length} < d$ **then**
    for $i \in \{dp, .., r_i\}$ **do**
      $l_i = \text{loss\_of\_t\_at\_epoch\_i}(t, i)$
      $l.\text{append}(l_i)$
    end for
    $L.\text{append}(l_i)$
    $fully\_trained = fully\_trained + 1$
    $D[r_i].\text{append}(l)$
    $thres = L.\text{get\_quantile}(0.25)$
  else
    if $dp \notin M[r_i].keys$ **then**
    `/* model to predict performance at r_i using the learning curve until dp */`
      $M[r_i][dp] = \text{PerformancePredictor}()$
      $M[r_i][dp].\text{train}(D[r_i][:, 0:dp], D[r_i][:, r_i])$
    end if
    $pred = M[r_i][dp].\text{predict}(l)$
    if $pred < thres$ **then**
      for $i \in \{dp, .., r_i\}$ **do**
        $l_i = \text{loss\_of\_t\_at\_epoch\_i}(t, i)$
        $l.\text{append}(l_i)$
      end for
      $L.\text{append}(l_i)$
      $fully\_trained = fully\_trained + 1$
      $D[r_i].\text{append}(l)$
    else
      $L.\text{append}(\infty)$
    end if
  end if
end for

In this appendix, the pseudocode of the sequential version of Swift-Hypeband is presented. As it can be seen in Algorithm 1, Swift-Hyperband has five parameters: R, η, d, ϕ, and *known_curve*. The parameters η and R have the same function as the parameters with the same respective name in the original Hyperband algorithm. That is, η controls the trial discarding ratio at the end of every round. For $\eta = 2$, only the best-performing half of all configurations at the end of a given round are promoted to the next round; for $\eta = 3$, only the best third is promoted, etc. Therefore, increasing the value of η makes the algorithm more aggressive. The least aggressive setting $\eta = 2$ is used by default in the experiments of this study. The quantity R defines the target epoch for the HPO process, i.e., no configuration is trained for more than R epochs. It is a problem-dependent parameter that is chosen by the user depending on the model or architecture to optimize. The remaining parameters d, ϕ, and *known_curve* are specific for Swift-Hyperband. The quantity d represents the minimum number of learning curves required to train each performance predictor, ϕ is the minimum fraction of trials that is trained until the end of each round independently of their predicted performance, and *known_curve* controls the position of the extra decision point. The natural choice $known\_curve = 0.5$ places the new decision points exactly in the middle of each Hyperband round.

Acknowledgements The authors gratefully acknowledge the computing time granted through JARA on the supercomputer JURECA (Jülich Supercomputing Centre, 2021) at Forschungszentrum Jülich. The authors gratefully acknowledge the Jülich Supercomputing Centre for funding this project by providing computing time through the

Jülich UNified Infrastructure for Quantum computing (JUNIQ) on the D-Wave Advantage^{TM} System JUPSI.

Author Contributions E.W., M.A. and JP. GA. wrote the main manuscript text. All authors reviewed the manuscript. M.A. prepared Figs. 4 and 5. JP.GA. prepared Figs. 3 and 4. E. W. was in charge of the datasets and models in Section 3.4.3 and their respective implementation. M. A. was in charge of the datasets and models in Sections 3.4.1 and 3.4.2 and their respective implementation. JP.GA. was in charge of the datasets and models Section 3.4.4. and their respective implementation. JP.GA. implemented the final hybrid distributed workflow algorithm.

E.W.: conceptualization, running experiments, writing, supervision, review. JP.GA.: conceptualization, running experiments, contribution to distributed hybrid workflow code on the HPC side, writing, supervision, review. M.A.: conceptualization, running experiments, contribution to distributed hybrid workflow code on the HPC side, writing, review. T.E.G, T.K.I, T.K.I, A. D.: contribution to quantum workflow code. E. P.: contribution to quantum workflow code and theoretical background, review. M. R., R.S.: supervision, contribution to analysis, review. A. L.: supervision, funding, contribution to analysis, review. M.G.: supervision, funding.

Funding Open access funding provided by CERN (European Organization for Nuclear Research). E. Wulff, J.P. García Amboage, M. Aach, R. Sarma, M. Riedel, and A. Lintermann were supported by CoE RAISE. The CoE RAISE project has received funding from the European Union's Horizon 2020 - Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

Abdiansah Abdiansah RW (2015) Time complexity analysis of support vector machines (SVM) in LIBSVM. Int J Comput Appl 128(3):28–34. https://doi.org/10.5120/ijca2015906480. https://ijcaonline.org/archives/volume128/number3/22854-2015906480/

Amboage JG, Wulff E, Girone M et al (2023) Optimizing AI-based HEP algorithms using HPC and quantum computing. https://indico.jlab.org/event/459/contributions/11847/attachments/9508/13784/CHEP2023___RAISE_Poster_FINAL.pdf

Apolloni B, Carvalho C, de Falco D (1989) Quantum stochastic optimization. Stoch Process Appl 33(2):233–244. https://doi.org/10.1016/0304-4149(89)90040-9. https://www.sciencedirect.com/science/article/pii/0304414989900409

Baker B, Gupta O, Raskar R et al (2017) Accelerating neural architecture search using performance prediction. https://doi.org/10.48550/ARXIV.1705.10823

Bishop CM (2006) Pattern Recognit Mach Learn (Inf Sci Stat). Springer-Verlag, Berlin, Heidelberg

Boothby K, Bunyk P, Raymond J et al (2020) Next-generation topology of D-Wave quantum processors. https://doi.org/10.48550/ARXIV.2003.00133

Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth annual workshop on computational learning theory. Association for Computing Machinery, New York, NY, USA, COLT '92, p 144–152. https://doi.org/10.1145/130385.130401

Bottou L, Chapelle O, DeCoste D et al (2007) Support vector machine solvers, pp 1–27

Burges CJ (1998) Data Min Knowl Disc 2(2):121–167. https://doi.org/10.1023/a:1009715923555

Cavallaro G, Willsch D, Willsch M et al (2020) Approaching remote sensing image classification with ensembles of support vector machines on the D-Wave quantum annealer. In: IGARSS 2020 - 2020 IEEE international geoscience and remote sensing symposium, pp 1973–1976. https://doi.org/10.1109/IGARSS39084.2020.9323544

Choi V (2011) Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design. Quantum Inf Process 10(3):343–353. https://doi.org/10.1007/s11128-010-0200-3

Date P, Arthur D, Pusey-Nazzaro L (2021) QUBO formulations for training machine learning models. Sci Rep 11(1):10029. https://doi.org/10.1038/s41598-021-89461-4

Dietterich TG (2000) Ensemble methods in machine learning. Mult Classifier Syst. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 1–15

Drucker H, Burges CJC, Kaufman L et al (1996) Support vector regression machines. In: Mozer M, Jordan M, Petsche T (eds) Advances in neural information processing systems, vol 9. MIT Press. https://proceedings.neurips.cc/paper_files/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf

Eggensperger K, Müller P, Mallik N et al (2021) HPOBench: a collection of reproducible multi-fidelity benchmark problems for HPO. In: 35th Conference on neural information processing systems datasets and benchmarks track (round 2). https://openreview.net/forum?id=1k4rJYEwda-

Falkner S, Klein A, Hutter F (2018) BOHB: robust and efficient hyperparameter optimization at scale. In: Proceedings of the 35th international conference on machine learning, pp 1436–1445

Fischer SF, Feurer M, Bischl B (2023) OpenML-CTR23 – a curated tabular regression benchmarking suite. In: AutoML conference 2023 (workshop). https://openreview.net/forum?id=HebAOoMm94

He K, Zhang X, Ren S et al (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on computer vision and pattern recognition (CVPR), pp 770–778. https://doi.org/10.1109/CVPR.2016.90

Jamieson K, Talwalkar A (2016) Non-stochastic best arm identification and hyperparameter optimization. In: Gretton A, Robert CC (eds) Proceedings of the 19th international conference on artificial intelligence and statistics, proceedings of machine learning research, vol 51. PMLR, Cadiz, Spain, pp 240–248. https://proceedings.mlr.press/v51/jamieson16.html

Kadowaki T, Nishimori H (1998) Quantum annealing in the transverse Ising model. Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Top 58:5355–5363. https://doi.org/10.1103/PHYSREVE.58.5355

Krizhevsky A (2009) Learning multiple layers of features from tiny images

Le Y, Yang XS (2015) Tiny imagenet visual recognition challenge

Li L, Jamieson K, DeSalvo G et al (2017) Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res 18(1):6765–6816. https://dl.acm.org/doi/abs/10.5555/3122009.3242042

Li L, Jamieson KG, Rostamizadeh A et al (2018) Massively parallel hyperparameter tuning. CoRR abs/1810.05934. https://arxiv.org/abs/1810.059341810.05934

Liu S, Zhang H, Jin Y (2022) A survey on computationally efficient neural architecture search. J Autom Intell 1(1):100002. https://doi.org/10.1016/j.jai.2022.100002, https://www.sciencedirect.com/science/article/pii/S2949855422000028

McGeoch CC (2014) Adiabatic quantum computation and quantum annealing: theory and practice, vol 5. Morgan & Claypool Publishers. https://doi.org/10.2200/S00585ED1V01Y201407QMC008

Mnmoustafa MA (2017) Tiny imagenet. https://kaggle.com/competitions/tiny-imagenet

Pasetto E, Riedel M, Melgani F et al (2022) Quantum SVR for chlorophyll concentration estimation in water with remote sensing. IEEE Geosci Remote Sens Lett 19:1–5. https://doi.org/10.1109/LGRS.2022.3200325

Pata J, Duarte J, Vlimant J et al (2021a) MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. Eur Phys J C 81(5). https://doi.org/10.1140/epjc/s10052-021-09158-w

Pata J et al (2021b) Simulated particle-level events of $t\bar{t}$ and QCD with PU200 using PYTHIA8+DELPHES3 for machine learned particle flow (MLPF). https://zenodo.org/record/4559324

Rebentrost P, Mohseni M, Lloyd S (2014) Quantum support vector machine for big data classification. Phys Rev Lett 113(13). https://doi.org/10.1103/physrevlett.113.130503. http://dx.doi.org/10.1103/PhysRevLett.113.130503

Sirunyan AM et al (2017) Particle-flow reconstruction and global even description with the CMS detector. J Instrum 12(10):P10003–P10003. https://doi.org/10.1088/1748-0221/12/10/p10003, https://arxiv.org/abs/1706.049651706.04965

Vanschoren J, van Rijn JN, Bischl B et al (2014) OpenML: networked science in machine learning. SIGKDD Explor Newsl 15(2):49–60. https://doi.org/10.1145/2641190.2641198

Weston J, Bordes A, Chopra S et al (2015) Towards AI-complete question answering: a set of prerequisite toy tasks. 1502.05698

Willsch D, Willsch M, Raedt HD et al (2020) Support vector machines on the D-Wave quantum annealer. Comput Phys Commun 248. https://doi.org/10.1016/j.cpc.2019.107006

Yu T, Zhu H (2020) Hyper-parameter optimization: a review of algorithms and applications. arXiv:abs/2003.05689. https://api.semanticscholar.org/CorpusID:212675087