MACHINE LEARNING Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
21 August 2019

REVISED
16 December 2019

ACCEPTED FOR PUBLICATION
9 January 2020

PUBLISHED
18 March 2020

PAPER

Machine learning the computational cost of quantum chemistry

Stefan Heinen , Max Schwilk , Guido Falk von Rudorff and O Anatole von Lilienfeld[1]

Institute of Physical Chemistry and National Center for Computational Design and Discovery of Novel Materials (MARVEL), Department of Chemistry, University of Basel, Klingelbergstrasse 80, CH-4056 Basel, Switzerland

[1] Author to whom any correspondence should be addressed.

E-mail: anatole.vonlilienfeld@unibas.ch

Keywords: high throughput computing (HTC), quantum chemistry, quantum machine learning (QML)

Supplementary material for this article is available online

Abstract

Computational quantum mechanics based molecular and materials design campaigns consume increasingly more high-performance computer resources, making improved job scheduling efficiency desirable in order to reduce carbon footprint or wasteful spending. We introduce quantum machine learning (QML) models of the computational cost of common quantum chemistry tasks. For 2D nonlinear toy systems, single point, geometry optimization, and transition state calculations the out of sample prediction error of QML models of wall times decays systematically with training set size. We present numerical evidence for a toy system containing two functions and three commonly used optimizer and for thousands of organic molecular systems including closed and open shell equilibrium structures, as well as transition states. Levels of electronic structure theory considered include B3LYP/def2-TZVP, MP2/6-311G(d), local CCSD(T)/VTZ-F12, CASSCF/VDZ-F12, and MRCISD+Q-F12/VDZ-F12. In comparison to conventional indiscriminate job treatment, QML based wall time predictions significantly improve job scheduling efficiency for all tasks after training on just thousands of molecules. Resulting reductions in CPU time overhead range from 10% to 90%.

1. Introduction

Solving Schrödinger's equation, arguably one of the most important computing tasks for chemistry and materials sciences, with arbitrary accuracy is an NP hard problem [1]. This leads to the ubiquitous limitation that accurate quantum chemistry calculations typically suffer from computational costs scaling steeply and nonlinearly with molecular size. Therefore, even if Moore's law was to stay approximately valid [2], scarcity in computer hardware would remain a critical factor for the foreseeable future. Correspondingly, chemistry and materials based computer projects have been consuming substantial CPU time at academic high-performance computer centers on national and local levels worldwide. For example, in 2017 research projects from chemistry and materials sciences used ~25% and ~35% of the total available resources at Argonne Leadership Computing Facility [3] and at the Swiss National Supercomputing Center (CSCS) [4], respectively. In 2018, ~30% of the resources at the National Energy Research Scientific Computing Center [5] were dedicated to chemistry and materials sciences and even ~50% of the resources of the ARCHER [6] super computing facility over the past month (May 2019). Assuming a global share of ~35% for the usage of the Top 500 super computers (illustrated in figure 1) over the last 25 years, this would currently correspond to ~0.5 exaFLOPS (floating point operations per seconds) per year. But also on most of the local medium to large size university or research center computer clusters, atomistic simulation consumes a large fraction of available resources. For example, at sciCORE, the University of Basel's computer cluster, this fraction typically exceeds 50%. Acquisition, usage, and maintenance of such infrastructures require substantial financial investments. Conversely, any improvements in the efficiency with which they are being used would result in immediate savings. Therefore a lot of work is done to constantly improve hardware and software of HPCs, e.g. at the International Supercomputing Conference NVIDIA announced the support of the Advanced RISC Machines (Arm) CPUs, which allows to build extremely energy

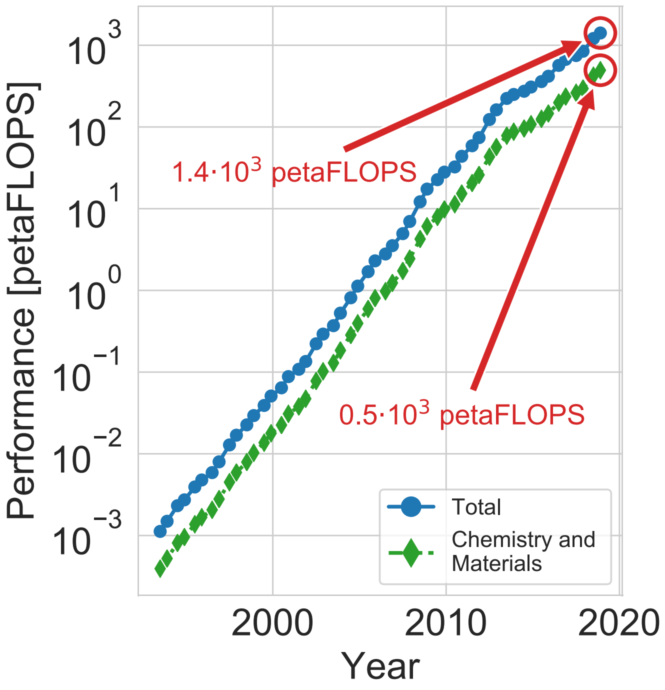

Figure 1. Computer resource growth of 500 fastest public supercomputers [10]. Estimated use by chemistry and materials sciences corresponds to 35%, corresponding to 2017 usage on Swiss National Supercomputing Center [4].

efficient exascale computers, by the end of the year [7]. Computer applications on such machines commonly rely on schedulers optimizing the simultaneous work load of thousands of calculations. While these schedulers are highly optimized to reduce overhead, there is still potential for application domain specific improvements, mostly due to indiscriminate and humanly biased run time estimates specified by users. The latter is particularly problematic when it comes to ensemble set-ups characteristic for molecular and materials design computer campaigns with very heterogeneous computer needs of individual instances. One could use the scaling behavior of methods to get sorted lists w.r.t wall times and improve scheduling by grouping the calculations by run time. For example the bottleneck of a multi-configuration self-consistent field calculation (MCSCF) is in general the transformation of the Coulomb and exchange operator matrices into the new orbital basis during the macro-iterations. This step scales as nm^4 with n the number of occupied orbitals and m the number of basis functions. All Configuration Interaction Singles Doubles (CISD) schemes that are based on the Davidson algorithm [8] scale formally as n^2m^4, where n the number of correlated occupied orbitals and m the number of basis functions [9]. As these methods (and basis sets) contain different scaling laws and geometry optimizations additionally depend on the initial geometry, a more sophisticated approach was applied: In this paper, we show how to use quantum machine learning (QML) to more accurately estimate run times in order to improve overall scheduling efficiency of quantum based ensemble computer campaigns.

Since the early 90s, an increasing number of research efforts from computer science has dealt with optimizing the execution of important standard classes of algorithms that occur in many scientific applications on HPC platforms [11–13], but also with predicting memory consumption [14], or, more generally, the computational cost itself (see [15, 16] for two recent reviews). Such predictive models may even comprise direct minimization of the estimated environmental impact of a calculation as the target quantity in the model [17]. ML has already successfully been applied, however, towards improving scheduling itself [18], or entire computer work flows [19, 20]. Furthermore, a potentially valuable application in the context of quantum chemistry may be the run time optimization of a given tensor contraction scheme on a specific hardware by predictive modeling techniques [21]. Another noteworthy effort has been the successful run time modeling and optimization of a self-consistent field (SCF) algorithm on various computer architectures in 2011 [22] using a simple linear model depending on the number of retired instructions and cache misses. Already in 1996, Papay *et al* contributed a least square fit of parameters in graph based component-wise run time estimates in parallelized self consistent field computations of atoms [23]. Other noteworthy work in the field of computational chemistry is the prediction of the run time of a molecular dynamics code [24], or the prediction of the success of density functional theory (DFT) optimizations of transition metal species as a classification problem by Kulik *et al* [25]. In the context of quantum chemistry and quantum mechanical solid state computations, very little literature on the topic is found. This may seem surprising, given the significant share of this domain on the overall HPC resource consumption (see figure 1). To the best of our knowledge, there is no (Q)ML study that predicts the computational cost (wall time, CPU time, FLOP count) of a given quantum chemical method across chemical space.

Today, a large number of QML models relevant to quantum chemistry applications throughout chemical space exists [26–28]. Common regressors include Kernel Ridge Regression [29–34] (KRR), Gaussian Process Regression [35] (GPR), or Artificial Neural Networks [34, 36–40] (ANN). For the purpose of estimating run times of new molecules, and contrary to pure computer science approaches, we use the same molecular representations (derived solely from molecular atomic configurations and compositions) in our QML models as for modeling quantum properties. As such, we view computational cost as a molecular 'quasi-property' that can be inferred for new, out-of-sample input molecules, in complete analogy to other quantum properties, such as the atomization energy or the dipole moment.

In general, a quantum chemistry SCF calculation optimizes the parameters of a molecular wave function with a clear minimum in the self-consistent system of nonlinear equations. I.e. the computational cost of a single point quantum chemistry calculation should be a reasonably smooth property over the chemical space. Pathological cases of SCF convergence failure are normally avoided by the careful choice of the quantum chemistry method for the single point (SP) calculation of a given chemical system. For geometry optimization (GO) and transition state (TS) searches on the other hand it is much harder to control the convergence, as a multitude of local minima and saddle points may exist on the potential energy surface defined by the degrees of freedom of the atomic coordinates in the molecule.

We therefore first investigated the performance of ML models to learn the number of discrete steps of common optimizers applied to the minimum search of nonlinear 2D functions that are known to cause convergence problems for many standard optimizers. In a second step, we investigated the capabilities of QML to learn the computational cost for a representative set of quantum chemistry tasks, including SP, GO, and TS calculations. To provide numerical evidence for hardware independence of the cost of quantum chemistry calculations, we trained a model on FLOPS as a 'clean' measurement.

2. Data

All QML approaches rely on large training data sets. Comprehensive subsets of the chemical space of closed shell organic molecules have been created in the past. The QM9 [41] data set of DFT optimized 3D molecular structures was derived from the GDB17 [42] data set of Simplified Molecular Input Line Entry System (SMILES) strings [43, 44]. This data set contains drug like molecules of broad scientific interest. GDB17 is an attempt to systematically generate molecules as mathematical graphs based on rules of medicinal chemistry, removing the bias of pre-existing building blocks in structure selection. QM9 itself is a well established benchmark data set for quantum machine learning where many different ML models were tested on [29, 32, 39, 45–53] and also contains many molecules which are commercially available and reported on many chemical data bases. Further relevant data sets in the literature include, among others, reaction networks [54], closed shell ground state organometallic compounds [55], or non-equilibrium structures of small closed shell organic molecules [56]. Yet, regions of chemical space that may involve more sophisticated and costly quantum chemistry methods, such as open shell and strongly correlated systems [57, 58] or chemical reaction paths, are still strongly underrepresented. For this study, we first generated two toy systems of nonlinear functions known to be difficult for many standard optimization methods. We used KRR to predict the number of optimization steps needed to find the functions' closest minimum for a systematically chosen set of starting points. The test case of optimizing analytical functions explores the fundamental question of learning computational cost of a nonlinear optimization problem outside the added complexity of quantum chemistry calculations. We then have generated measures of the computational cost associated to seven tasks which reflect variances of three common use cases: single point (SP), geometry optimization (GO) and transition state (TS) search calculations.

2.1. Toy system

To demonstrate that it is possible to learn the number of steps of an optimization algorithm, we apply our machine learning method to two cases from function optimization theory: quantifying the number of steps for an optimizer. The functions in question are the Rosenbrock function [59]

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{1}$$

and the Himmelblau function [60]

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2. \tag{2}$$

The functions are shown in the top row of figures 4(b) and (c). We applied three representative optimizers in their SciPy 1.3.1 [61] implementation on both functions: the 'NM' simplex algorithm (Nelder-Mead [62]), the gradient based 'BFGS' algorithm [63], and an algorithm using gradients and hessians (Conjugate Gradient with Newton search 'N-CG' [64]). For every function and optimizer we performed 10 200 optimizations from different starting points on a Cartesian grid over the domain $-5 \leqslant x, y \leqslant 5$ in steps of 0.1. The minimum of the Rosenbrock function and the four minima of the Himmelblau function lie within this domain. Figure 4(b) row two, three, and four show a heatmap of the number of optimization steps for NM, BFGS, and N-CG, respectively, for Rosenbrock (left column) and Himmelblau (right column). Generally, the minimum searches on the Himmelblau function required much fewer steps (mostly reached after a few tens of iterations). While the gradient based optimizer BFGS clearly outperforms NM for both functions, the N-CG optimization of the Rosenbrock function did not converge with a iteration limit of 400 for a set of points in the region of $x < -0.5$ and $y > 2.5$. A very small step size for the N-CG algorithm implementation in SciPy in the critical region is responsible for the slow convergence.

2.2. Quantum data sets

We have considered coordinates coming from three different data sets (QM9, QMspin, QMrxn) corresponding to five levels of theory (CCSD(T), MRCI, B3LYP, MP2, CASSCF) and four basis set sizes. Molecules in the three different data sets consist of the following:

(i) QM9 contains 134k small organic molecules in the ground state local minima with up to nine heavy atoms which are composed of H, C, N, O, and F. All coordinates were published in 2014 [41]. Here, we also report the relevant timings.

(ii) QMspin consists of carbenes derived from QM9 molecules containing calculations of the singlet and triplet state, respectively, with a state-averaged CASSCF(2e,2o) reference wave function (singlet and triplet ground states with equal weights). The entirety of this data set will be published elsewhere, here we only provide timings and QM9 labels.

(iii) QMrxn consists of reactants and S_N2 transition states of small organic molecules with a scaffold of C_2H_6 which was functionalized with the following substituents: $-NO_2$, $-CN$, $-CH_3$, $-NH_2$, $-F$, $-Cl$ and $-Br$. The entirety of this data set will be published elsewhere, here we only provide timings and geometries.

2.3. Quantum chemistry tasks

The three data sets were then divided into the seven following tasks for which timings were obtained (see also table 1):

$\mathbf{QM9^{SP}_{CC/DZ}}$ 5736 PNO-LCCSD(T)-F12/VDZ-F12 [65–67] single point energy timings. Details of the calculation results other than timings are subject of a separate publication [68].

$\mathbf{QM9^{SP}_{CC/TZ}}$ 3497 PNO-LCCSD(T)-F12/VTZ-F12 single point energy timings.

$\mathbf{QMspin^{SP}_{MRCI}}$ 2732 single point calculations using MRCISD+Q-F12/VDZ-F12 [69–72]. Details of the calculation results other than timings are subject of a separate publication [73].

$\mathbf{QM9^{GO}_{B3LYP}}$ 3724 geometry optimization timings with initial B3LYP/6-31G* [74, 75] geometries optimizing at the B3LYP/def2-TZVP level of theory.

$\mathbf{QMrxn^{GO}_{MP2}}$ 8148 geometry optimization timings on MP2/6-311G(d) level of theory.

$\mathbf{QMspin^{GO}_{CASSCF}}$ 1595 CASSCF(2e,2o)[Singlet]/VDZ-F12 [76, 77] geometry optimization timings.

$\mathbf{QMrxn^{TS}_{MP2}}$ 1561 timings of transition state searches on MP2 level of theory.

Further details on the data sets can be found in section 1 of the supplementary information (SI), which is available online at stacks.iop.org/MLST/1/025002/mmedia. A distribution of the properties (wall times) of the seven tasks is illustrated in figure 2. Single point calculations (the two $\mathbf{QM9^{SP}_{CC}}$ tasks) and the geometry optimization (task $\mathbf{QM9^{GO}_{B3LYP}}$) have wall times smaller than half an hour. In general, the smaller the variance in the data, the less complex the problem and the easier it is for the model to learn the wall times. For geometry optimizations and more exact (also more expensive) methods (task $\mathbf{QMspin^{SP}_{MRCI}}$ and $\mathbf{QMspin^{GO}_{CASSCF}}$) the average run time is ~9 h. With a larger variance in the data the problem is more complex (higher dimensional) and the learning is more difficult (higher off-set).

2.4. Timings, code, and hardware

The calculations were run on three computer clusters, namely our in-house computer cluster, the Basel University cluster (sciCORE) and the Swiss national supercomputer Piz Daint at CSCS. We used two electronic

Table 1. Seven tasks used in this work generated from three data sets (QM9, QMspin, QMrxn), using three use cases (SP, GO, TS) on different levels of theory and basis sets.

Task	**$QM9^{SP}_{CC/DZ}$**	**$QM9^{SP}_{CC/TZ}$**	**$QMspin^{SP}_{MRCI}$**	**$QM9^{GO}_{B3LYP}$**	**$QMrxn^{GO}_{MP2}$**	**$QMspin^{GO}_{CASSCF}$**	**$QMrxn^{TS}_{MP2}$**
Use case		SP			GO		TS
Data set	QM9		QMspin	QM9	QMrxn	QMspin	QMrxn
Level	CCSD(T)	CCSD(T)	MRCI	B3LYP	MP2	CASSCF	MP2
Basis set	VDZ-F12 [78]	VTZ-F12 [78]	VDZ-F12 [78]	def2-TZVP [79, 80]	6-311G(d) [81–83]	VDZ-F12 [78]	6-311G(d) [81–83]
Size	5736	3497	2732	3724	8148	1595	1561
Code	Molpro	Molpro	Molpro	Molpro	ORCA	Molpro	ORCA

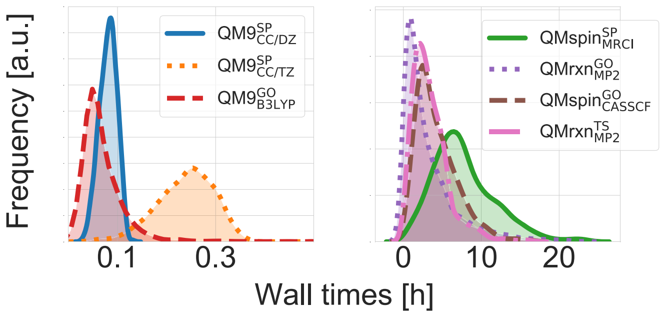

Figure 2. Wall time distribution of all tasks using kernel density estimation.

structure codes to generate timings. Molpro [84] was used to extract both CPU and wall times for data sets (i) and (ii), and ORCA [85] was used to extract wall times for data set iii). Further information of the data sets, the hardware, and the calculations can be found in sections 3 and 4 of the SI.

The retired floating point operations (FLOP) count of the local coupled cluster calculation task $\mathbf{QM9^{SP}_{CC/DZ}}$ was obtained as follows: the number of FLOPs have been computed with the *perf* Linux kernel profiling tool[2] for data set $\mathbf{QM9^{SP}_{CC/DZ}}$. *perf* allows profiling of the kernel and user code at run time with little CPU overhead and can give FLOP counts with reasonable accuracy. FLOP count is an adequate measure of the computational cost when the program execution is CPU bound by numerical operations, which is given in the PNO-LCCSD(T)-F12 implementation [65–67, 86] in Molpro.

3. Methods

3.1. Quantum machine learning

In this study, we used kernel based machine learning methods which were initially developed in the 1950s [87] and belong to the supervised learning techniques. In ridge regression, the input is mapped into a feature space and fitting is applied there. However, the best feature space is *a priori* unknown, and its construction is computationally hard. The 'kernel trick' offers a solution to this problem by applying a kernel k on a representation space $\mathcal{R}$ that yields inner products of an implicit high dimensional feature space: the Gram matrix elements $k(\mathbf{x}_i, \mathbf{x}_j)$ of two representations $\mathbf{x} \in \mathcal{R}$ between two input molecules i and j are the inner products $\langle i, j\rangle$ in the feature space. For example,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_1}{\sigma}\right) \tag{3}$$

or

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\sigma^2}\right) \tag{4}$$

with σ as the length scale hyperparameter, represent commonly made kernel choices, the Laplacian (equation (3)) or Gaussian kernel (equation (4)). Fitting coefficients $\boldsymbol{\alpha}$ can then be computed in input space via the inverse of the kernel matrix $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}, \tag{5}$$

where λ is the regularization strength, typically very small for calculated noise-free quantum chemistry data.

Hence, KRR learns a mapping function from the inputs $\mathbf{x}_i$, in this case the representation of the molecule, to a property $y_q^{\mathrm{est}}(\mathbf{x}_q)$, given a training set of N reference pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Learning in this context means interpolation between data points of reference data $\{(\mathbf{x}_i, y_i)\}$ and target data $\{(\mathbf{x}_q, y_q^{\mathrm{est}})\}$. A new property y_q^{est} can then be predicted via the fitting coefficients and the kernel:

$$y_q^{\mathrm{est}}(\mathbf{x}_q) = \sum_i^N \alpha_i \cdot k(\mathbf{x}_i, \mathbf{x}_q). \tag{6}$$

For the toy systems, a Laplacian kernel was used, the representation corresponding simply to the starting point ($\mathbf{x} = (x, y)$) of the optimization runs. For the purpose of learning of the run times, we used two widely used

[2] *Perf* of the Linux kernel version 3.10.0-327.el7.x86_64 tools was used. *Perf* measures the number of retired FLOP (as a certain amount of speculative executions may be negated, given that logical branches cannot be evaluated between instructions within a clock cycle).

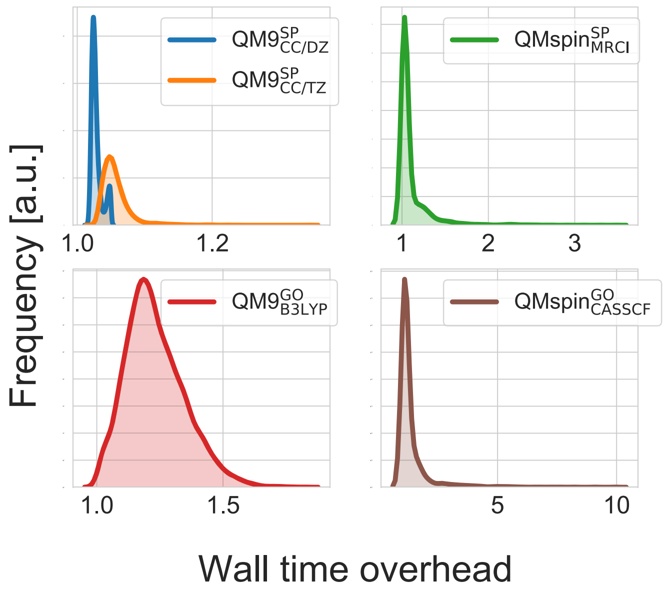

Figure 3. Wall to CPU time ratio (using kernel density estimation) for Molpro calculations to identify runs with high wall time overhead due to heavy I/O load on clusters.

representations, namely Bag of Bonds (BoB) [45] with a Laplacian kernel. BoB is a vectorized version of the Coulomb Matrix (CM) [29] that takes the Coulomb repulsion terms for all atom to atom distances and packs them into bins, scaled by the product of the nuclear charges of the corresponding atoms. This representation does not provide a strictly unique mapping [32, 88] which may deteriorate learning in some cases (*vide infra*). The second representation used was atomic FCHL [50] with a Gaussian kernel. FCHL accounts for one-, two-, and three-body terms (whereas BoB only contains two-body terms). The one-body term encodes group and period of the atom, the two-body term contains interatomic distances R, scaled by R^{-4}, and the three-body terms in addition contain angles between all atom triplets scaled by R^{-2}.

To determine the hyperparameters σ and λ, the reference data was split into two parts, the training and the test set. The hyperparameters were optimized only within the training set using random sub-sampling cross validation. To quantify the performance of our model, the test errors, measured as mean absolute errors (MAE), were calculated as a function of training set size. The leading error term is known to be inversely proportional to the amount of training points used [89]:

$$\mathrm{MAE} \approx a/N^{b}. \tag{7}$$

The learning curves should then result in a decreasing linear curve with slope b and offset $\log a$:

$$\log(\mathrm{MAE}) \approx \log(a) - b\log(N), \tag{8}$$

where a is the target similarity which gives an estimate of how well the mapping function describes the system [32] and b is the slope being an indicator for the effective dimensionality [90]. Therefore, good QML models are linearly decaying, have a low offset $\log(a)$ (achieved by using more adequate representations and/or base-line models [91]), and have steep slopes (large b).

For each task, QML models of wall times were trained and subsequently tested on out-of-sample test set which was not part of the training. As input for the representations the initial geometries of the calculations were used. To improve the predictions of geometry optimizations for the task **QMspin$^{GO}_{CASSCF}$**, we split the individual optimization steps into the first step (GO1) and the subsequent steps (GO2), because the first step takes on average ~20% more time than the following steps (for more details we refer to section 1.4 of the SI). For learning the timings of the geometry optimization task GO2, we took the geometries obtained after the first optimization step.

As input for the properties, wall times were normalized with respect to the number of electrons in the molecules. Figure 3 shows the wall time overhead (CPU time to wall time ratio) for calculations run with Molpro. To remove runs affected by heavy I/O, wall time overheads higher than 3%, 5%, 10%, 30%, and 50% were excluded from the tasks **QM9$^{SP}_{CC/DZ}$**, **QM9$^{SP}_{CC/TZ}$**, **QMspin$^{SP}_{MRCI}$**, **QMspin$^{GO}_{CASSCF}$**, and **QM9$^{GO}_{B3LYP}$**, respectively. In order to generate learning curves for all the seven tasks, all timings were normalized with respect to the median of the test set to get comparable normalized (MAE). The resulting wall time out-of-sample predictions were used as input for the scheduling algorithm. Whenever the QML model predicted negative wall times, the predictions were replaced by the median of all non-negative predictions.

All QML calculations have been carried out with QMLcode [92]. Wall times and CPU times (Molpro) and wall times (ORCA) for all the seven tasks, as well as QML scripts can be found in the SI.

3.2. Application: optimal scheduling

3.2.1. Job array and job steps

In many cases, efforts in computational chemistry or materials design require the evaluation of identical tasks on different molecules or materials. Distributing those tasks across a computer cluster is typically done in one of two ways. When using job arrays, the scheduler assigns computer resources to each calculation separately, such that the individual calculation is queued independently. This approach typically extends the total wall time, and has little overhead with the jobs themselves but leads to inefficiencies for the scheduler since the individual wall time estimate of each job needs to be (close to) the maximum job duration.

In the second approach, there are only few jobs submitted to the scheduler and tasks are executed in parallel as job steps. The first approach has little overhead with the jobs themselves but can lead to inefficiencies. The second approach yields inefficiencies due to lack of load balancing. These two common methods require no knowledge of the individual run time of each task, and usually rely on a conservative run time estimate in practice.

3.2.2. Scheduling simulator

Using the QML based estimated absolute timings turns the scheduling of the remaining calculations into a bin packing problem. For this problem we used the heuristic first fit decreasing (FFD) algorithm which takes all run time estimates for all tasks, sorts them in decreasing order and chooses the longest task that fits into the remaining time of a computer job (for more details on FFD, see section 2 in the SI). If there is no task left that is estimated to fit into a gap, then no task is chosen and resources are released early.

We implemented a job scheduling simulator assuming idempotent uninterruptible tasks for all three job schedulers: conventional job arrays, conventional job steps, and our new QML based job scheduler. Using a simulator is particularly useful because the duration of the job array and job step approaches depend on the (random) order of the jobs, and therefore requires averaging over multiple runs. We used this simulator in the context of two environments: our university cluster sciCORE (denoted *S*) where users are allowed to submit single-core jobs and the Swiss national supercomputer (CSCS, denoted *L*) where users are only allowed to allocate entire computer nodes of 12 cores. In all cases, we assumed that starting a new job via the scheduler takes 30 s and that every job queues for 1 h. These numbers have been observed for queuing statistics of sciCORE and CSCS.

4. Results and discussion

4.1. Toy system

From the total data set (10 200 optimizations) 3200 were chosen randomly for every combination of optimizer and function and the prediction error was computed for different training set sizes *N*. Figure 4(a) shows the learning curves for the Rosenbrock (‘Rosen’) and the Himmelblau (‘Him’) functions. Well behaved learning curves were obtained for both functions and all optimizers. The ML models for Him-BFGS and Him-N-CG have a lower offset because the variance of the data set is smaller (between 0 and 25 optimization steps) than for the others (∼50 to 120 steps). The offset of Rosen–Newton-CG can be explained by the truncated runs which caused a non smooth area in the function space ($x < -0.5$ and $y > 2.5$) which leads to higher errors.

In addition to the learning curves, we computed the relative prediction errors of the different optimization runs. These results are shown in figure 4(c). As expected, the errors get larger when the starting point is close to a saddle point: small changes in the starting point coordinates may lead to very different optimization paths. These discontinuities naturally occur for any optimizer based on the local information at the starting point and can be consistently observed in figure 4(b). Additional discontinuities can also be observed depending on the optimizer. For all these regions larger relative errors for KRR can be observed (shown in figure 4(c)) illustrating that small prediction errors rely on a reasonably smooth target function. In summary, we can show that KRR is capable of learning the discrete number of optimization steps which is a strong indication that the computational cost of quantum chemistry geometry optimization and transition state searches should be learnable in principle .

4.2. Quantum machine learning

4.2.1. Single point (SP) wall times

In the following, learning of the wall times for the different quantum chemistry tasks is discussed, the learning of the corresponding CPU times has also been investigated and results of the latter are given in the SI. Figure 5 (left) shows the performance of QML models of wall times using learning curves for the SP use case. For the two

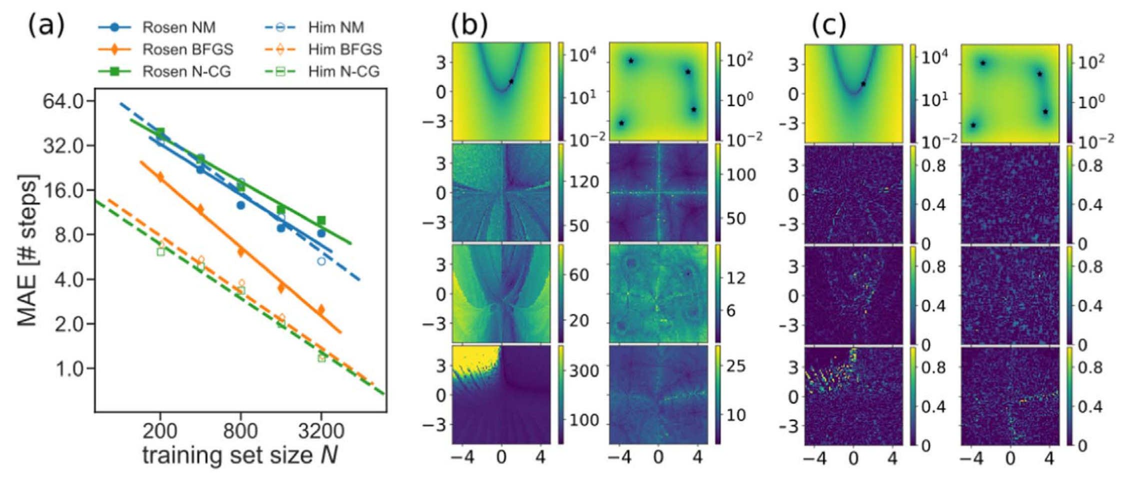

Figure 4. 2D nonlinear toy systems consisting of the Rosenbrock ('Rosen') and Himmelblau ('Him') functions and minimum search with three optimizers (Nelder-Mead (NM), BFGS, and Newton-CG (N-CG)). (a) Learning curves showing the prediction error of KRR for Rosen (solid lines) and Him (dashed lines) function using starting point (x, y) as representation input. (b) Top row shows the function values for Rosen (left) and Him (right). Row two, three, and four show the number of optimization steps (encoded in the heat map) for 10 200 starting points for NM, BFGS, and N-CG, respectively. (c) Row two, three, and four show the relative prediction error of the ML model trained on the largest training set size $N = 3200$ for NM, BFGS, and N-CG, respectively.

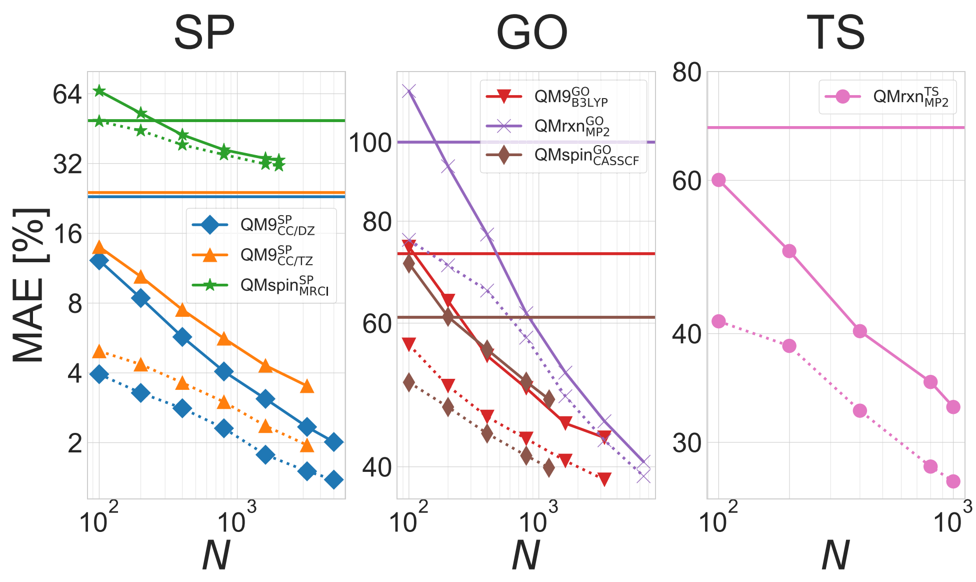

Figure 5. Learning curves showing normalized test errors (cross validated MAE divided by median of test set) for seven tasks using BoB (solid) and FCHL (dashed) representations. The model was trained on wall times normalized w.r.t. number of electrons. Horizontal lines correspond to the performance estimating all calculations have mean run time (standard deviation divided by mean wall time of the task).

similar tasks **QM9$^{SP}_{CC/DZ}$** and **QM9$^{SP}_{CC/TZ}$**, the timings of the smaller basis set was consistently easier to learn, i.e. smaller training set required to reach similar predictive accuracy. Similarly to physical observables [50], the use of the FCHL representation results in systematically improved learning curve off-set with respect to BoB. It is substantially more difficult to learn timings of multi-reference calculations (task **QMspin$^{SP}_{MRCI}$**), nevertheless, learning is achieved, and BoB initially also exhibits a larger off-set than FCHL, but the learning curves of the respective two representations converge for larger training set sizes. More specifically, for training set size $N = 1600$, BoB/FCHL based QML models reach an accuracy of 3.1/1.8, 4.3/2.4, and 33.7/31.8% for **QM9$^{SP}_{CC/DZ}$**, **QM9$^{SP}_{CC/TZ}$**, and **QMspin$^{SP}_{MRCI}$**, respectively. Corresponding respective average wall times in our data-sets, distributions shown in figure 2, average at ~6, 15, and 480 min. To the best of our knowledge, such predictive power in estimating computer timings has not yet been demonstrated for common quantum chemistry tasks.

The extraordinary accuracy that our model can reach in the prediction of the wall times for the **QM9$^{SP}_{CC/DZ}$** and **QM9$^{SP}_{CC/TZ}$** quantum chemistry tasks may be explained by the underlying quantum chemical algorithm.

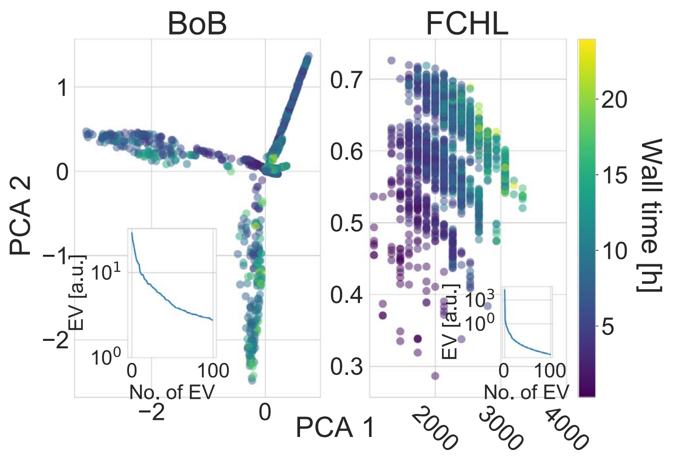

Figure 6. PCA plots of kernel elements for BoB (left) and FCHL (right) for data set **$\mathrm{QMspin}^{\mathrm{SP}}_{\mathrm{MRCI}}$**. The weights of the two first principal components for the molecules in the data sets are plotted against each other and corresponding wall times are encoded as a heat map. Insets show the first 100 eigenvalues on a log scale.

The tensor contractions in the local coupled cluster algorithm are sensitively linked to the chemically relevant many-body interactions expressed in the basis of localized orbitals. Therefore, the computational cost can be suitably encoded by atom-based machine learning representations.

In order to investigate the relative performance of BoB versus FCHL further, we have performed a principal component analysis (PCA) on the respective kernels (training set size $N = 2000$) for task **$\mathrm{QMspin}^{\mathrm{SP}}_{\mathrm{MRCI}}$**. The projection onto the first two components is shown in figure 6, color-coded by the training instance specific wall times, and with eigenvalue spectra as insets. For FCHL, the decay of the eigenvalues is very rapid (tenth eigenvalue already reaches 0.1). From the PCA projection, the number of heavy atoms emerges as a discrete spectrum of weights for the first principal component. The second principal component groups constitutional isomers. This reflects the importance of the one-body terms in the FCHL representation. The data covers well both components and the color various monotonically. All of this indicates a rather low dimensionality in the FCHL feature space which facilitates the learning. The kernel PCA plot of the FCHL representation shows that the learning problem is smooth in representation space and that there is a correlation between the property (computational cost) and the representation space. By contrast, the BoB's PCA projection onto the first two components displays a star-wise pattern with linear segments which indicate that more dimensions are required to turn the data into a monotonically varying hypersurface. The eigenvalue spectrum of BoB decays much more slowly with even the 100th eigenvalue still far above 1.0. All of this indicates that learning is more difficult, and thereby explains the comparatively higher off-set.

4.2.2. Geometry optimization (GO) wall times

Learning curves in figure 5 (middle) shows that it is, in general, possible to build QML models of GO timings for the tasks considered. We obtained accuracies for BoB/FCHL for $N = 800$ of 50.0/43.3, 61.7/57.6, and 50.7/41.2% for tasks **$\mathrm{QM9}^{\mathrm{GO}}_{\mathrm{B3LYP}}$**, **$\mathrm{QMrxn}^{\mathrm{GO}}_{\mathrm{MP2}}$**, and **$\mathrm{QMspin}^{\mathrm{GO}}_{\mathrm{CASSCF}}$**, respectively.

Interestingly, the comparatively larger off-set in the learning curves, however, indicates that it is more difficult to learn GO timings than SP timings. This is to be expected since GO timings involve not only SP calculations for various geometries but also geometry optimization steps. In other words, the QML model has to learn the quality of the initial guesses for subsequent GO optimizations. This cannot be expected to be a smooth function in chemical space. Furthermore, the mapping from an initial geometry (used in the representation for the QML model) to the target geometry can vary dramatically when the initial geometry happens to be close to a saddle point (or a second order saddle point in the case of TS searches, see next section): very slight changes in the initial geometry (or in the setup of the geometry optimization) may lead to convergence to very different stationary points on the potential energy surface. This makes the statistical learning problem much less well conditioned than for single point calculations, which also reflects in the larger variance of the geometry optimization timings compared to single point calculations. As such, GO timings represent a substantially more complex target function to learn than SP timings. Note that for any task (even for the toy system applications) we require a different QML model. The cost of the GO depends on the initial geometry and the convergence criteria. The latter varies only slightly within a data set. The former is part of the representation of the molecular structure and therefore captured by our model. The input structures for the task **$\mathrm{QMrxn}^{\mathrm{GO}}_{\mathrm{MP2}}$** are derived from the same molecular skeleton and are therefore very similar. The same holds for task **$\mathrm{QM9}^{\mathrm{GO}}_{\mathrm{B3LYP}}$** and **$\mathrm{QMspin}^{\mathrm{GO}}_{\mathrm{CASSCF}}$** which

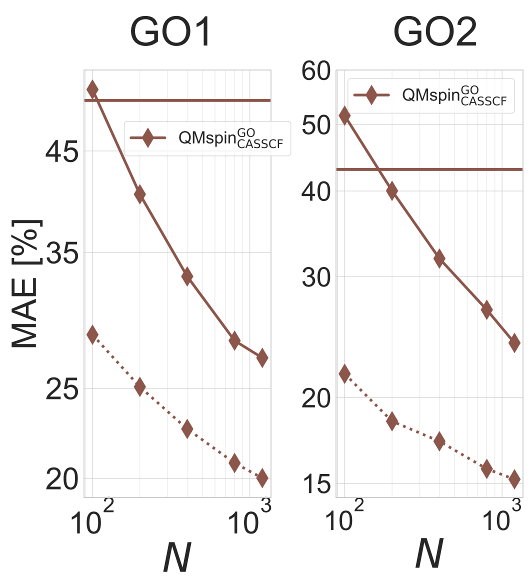

Figure 7. Learning curves showing normalized test errors (cross validated MAE divided by median of test set) for the first two geometry optimization steps on task $\mathbf{QMspin}^{\mathbf{GO}}_{\mathbf{CASSCF}}$ using BoB and FCHL as representations. The model was trained on CPU times divided by the number of electrons. Horizontal lines correspond to the performance estimating all calculations have mean run time (standard deviation divided by the mean wall time of the data set).

Table 2. QML results (normalized prediction errors) for seven task and both representations (BoB and FCHL) for largest training set size ($N_{\max}$).

Calculation	**SP**			**GO**			**TS**
Label	$\mathrm{QM9}^{\mathrm{SP}}_{\mathrm{CC/DZ}}$	$\mathrm{QM9}^{\mathrm{SP}}_{\mathrm{CC/TZ}}$	$\mathrm{QMspin}^{\mathrm{SP}}_{\mathrm{MRCI}}$	$\mathrm{QM9}^{\mathrm{GO}}_{\mathrm{B3LYP}}$	$\mathrm{QMrxn}^{\mathrm{GO}}_{\mathrm{MP2}}$	$\mathrm{QMspin}^{\mathrm{GO}}_{\mathrm{CASSCF}}$	$\mathrm{QMrxn}^{\mathrm{TS}}_{\mathrm{MP2}}$
$N_{\max}$	5000	3200	2000	3200	6400	1200	1000
BoB (%)	2.0	3.3	32.7	42.5	40.5	47.8	32.9
FCHL (%)	1.3	1.6	30.9	37.6	38.9	39.8	27.0

are derived from QM9 molecules. The convergence criteria also stay the same for all calculations within a data set and would only cause a more difficult learning task if a machine was trained over several different data sets. We also showed with the toy system that it is possible to learn the number of steps for different optimizer starting from different areas on the surface (see figure 4(b)). To further improve the performance of our model of task $\mathbf{QMspin}^{\mathbf{GO}}_{\mathbf{CASSCF}}$, we split the GO into the first GO step (GO1) and all subsequent steps (GO2). This choice has been motivated by our observation that most of the variance stemmed from the first GO step (requiring to build the wave-function from scratch), while the subsequent steps for themselves have a substantially smaller variance. The resulting learning curves are shown in figure 7 and justify this separation in leading to an improvement of the QML model to reach errors of less than 25% at $N = 800$ (rather than more than 40%), as well as further improved job scheduling optimization (shown below in figure 10).

4.2.3. Transition state (TS) wall times

Transition state search timings were slightly easier to learn than geometry optimization timings (see figure 5 (right)). Particularly for the larges training set size ($N_{\max} = 1000$) for BoB/FCHL we obtained MAEs of 32.9/27.0% and reduced the off-set by ~10% compared to learning curves for the GO use case. As already discussed in the previous section, the run time of GO and TS timings not only scales with the number of electrons but also depends on the initial structure. For the transition state search, the scaffold (which is close to a transition state) was functionalized with the different functional groups. Since the initial structures were closer to the final TS the offset of the learning curves is lower than for learning curves of the GO use case, where the initial geometries were generated with a semi empirical method (PM6) for task $\mathbf{QMrxn}^{\mathbf{GO}}_{\mathbf{MP2}}$, carbenes were derived from QM9 molecules for task $\mathbf{QMspin}^{\mathbf{GO}}_{\mathbf{CASSCF}}$, and geometries for task $\mathbf{QM9}^{\mathbf{GO}}_{\mathbf{B3LYP}}$ were obtained with a different basis set.

A summary of the results for all tasks for the largest training set size ($N_{\max}$) can be found in table 2.

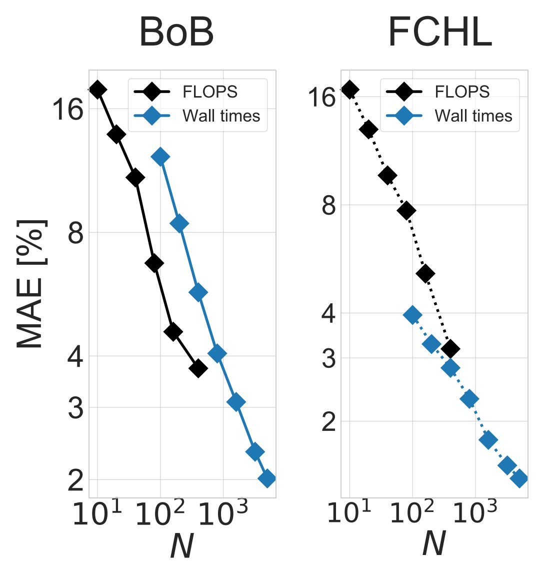

Figure 8. Learning curves showing normalized prediction errors (cross validated MAE divided by median of test set) for FLOP count and wall times on task $\mathbf{QM9^{SP}_{CC/DZ}}$ using BoB and FCHL representations.

4.2.4. Timings, code, hardware

Regarding hardware dependent models, within one data set we only used one electronic structure code which is also consistent with the general handling of the data set generation. The noise that is generated using different infrastructures affects the learning only in a negligible amount in our case, since the difference in hardware capabilities is minimal. When looking at the task $\mathbf{QMrxn^{TS}_{MP2}}$ where we used five different CPU types on two clusters (table 1 in the SI), we could not find any evidence that different hardware affects the learning compared to other GO tasks that ran on only one CPU type and cluster. However the hardware for these calculations is still very similar. When it differs to a greater extant, the noise level will rise. The noise does not only depend on the cluster itself but also on other calculations running on the cluster which is non-deterministic and will limit the transferability of the ML models. For this reason we removed some of the timings with large I/O overhead using figure 3. For the $\mathbf{QM9^{SP}_{CC}}$ tasks, the run time difference using the Intel MKL 2019 library [93] and OpenBlas 0.2.20 [94] were computed for a few cases and are found to be only within a few percents of the wall time. Furthermore, run times of a native build of the Molpro software package version 2018.3 with OpenMPI 3.0.1 [95], GCC 7.2.0 [96], and GlobalArrays 5.7 [97, 98] and the shipped executable were compared and yielded run times within a few percents of difference. The FLOP calculations on the $\mathbf{QM9^{SP}_{CC}}$ data set have been performed on a computer node with 24 processors (Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (Broadwell)). The significant part of the FLOP clock cycles constituted of vectorized double precision FLOP on the full 256 bit FLOP register, i.e. the essential numerical operations of the quantum chemistry algorithm were directly measured. Hence, FLOP count constitutes a valuable measure of the computation cost in our case[3]. We anticipate that Hardware specific QML models will be used in practice.

4.2.5. Single point (SP) FLOPs

To provide unequivocal numerical proof that it is justifiable to learn wall times we applied our models to FLOP counts for the task $\mathbf{QM9^{SP}_{CC/DZ}}$, shown in figure 8. FLOP count as a 'clean' measurement (almost no noise) for computational cost was slightly easier to learn than wall times and the learning curves show similar behavior: the model trained on the same task $\mathbf{QM9^{SP}_{CC/DZ}}$ reaches $\sim$4% MAE already with just 400 training samples, while $\sim$1000 training samples were required in the case of wall times using BoB. For FCHL, the performance is similar but the slope is steeper for the FLOP model which indicates a faster learning or less noise.

4.3. Application: optimal scheduling

4.3.1. Job array and job steps

For the scheduling optimization for all seven tasks ($\mathbf{QM9^{SP}_{CC/DZ}}$, $\mathbf{QM9^{SP}_{CC/DT}}$, $\mathbf{QMspin^{SP}_{MRCI}}$, $\mathbf{QM9^{GO}_{B3LYP}}$, $\mathbf{QMrxn^{GO}_{MP2}}$, $\mathbf{QMspin^{GO}_{CASSCF}}$, $\mathbf{QMrxn^{TS}_{MP2}}$), the QML model with the best representation (lowest MAE with maximum number of training points) was used which in all cases was FCHL. For the FFD algorithm absolute

[3] Due to non-deterministic run time behavior of the CPU, as well as measurement errors of *perf*, the FLOP count varies within a few tenth of percent for consecutive runs of the same calculation.

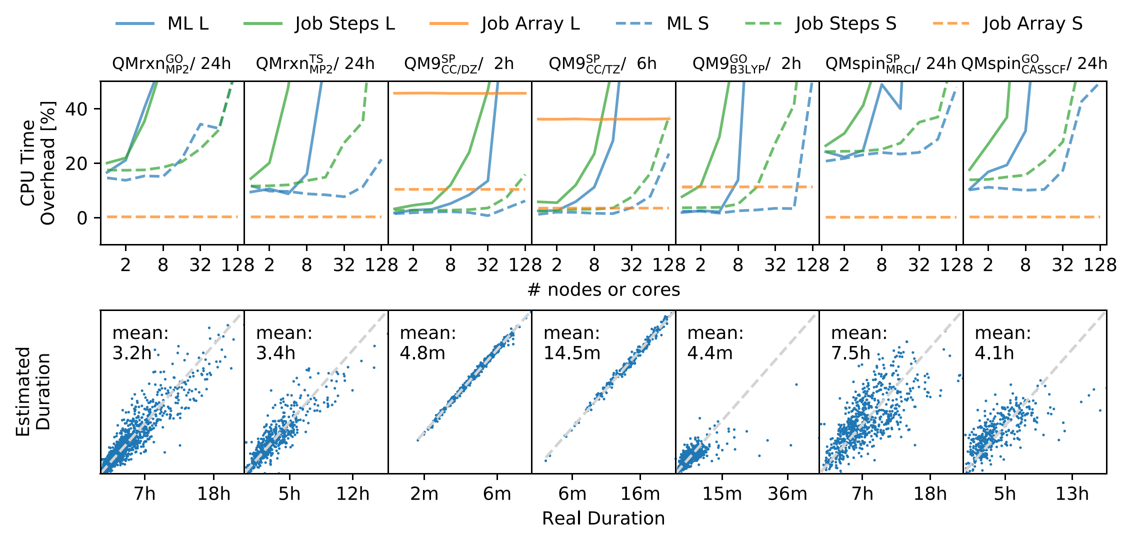

Figure 9. Scheduling efficiencies for the seven different tasks (columns) assuming a certain per-job wall time limit specified in column title. Infrastructure assumptions correspond to either a large (solid lines, L) computer center or a small (dashed lines, S) university computer center. Top row reports CPU time overhead reduction when using the QML based (blue) rather than the conventional (green, orange) packing. Results are given relative to the total CPU time needed for the calculations of each data set for established methods (job array and jobs steps, see text) and our suggested method (QML). Bottom row shows actual versus predicted times (using FCHL as representation) for all calculations in each data set using maximum training set size.

timing predictions are needed to make good decisions. The lower panel of figure 9 shows the accuracy of the QML predictions. While the individual predictions (absolute not relative) are in many cases not perfect and partially still exhibit a significant MAE (see figure 5), this level of accuracy is already sufficient to reduce the overhead of the job scheduling. The lower panel of figure 9 shows the accuracy of the QML predictions. While the individual predictions (absolute not relative) are in many cases not perfect and partially still exhibit a significant MAE (see figure 5), this level of accuracy is already sufficient to reduce the overhead or the wall time limits of the job scheduling. In particular, in the limit of a large number of cores working in parallel, our approach typically halved the computational overhead (data sets with closed shell systems and TS searches) while also reducing the time to solution by reducing the total wall time. This shows that for the scheduling efficiency problem, it is not required to obtain perfect estimates for the individual job durations, but rather reasonably accurate estimates. However, if there was the need for better accuracy, by virtue of the ML paradigm (prediction error decay systematically with training set size) this could easily be accomplished by decreasing the error simply through the addition of more training data.

When comparing the different methods in the upper panel of figure 9, we see that the job array approach had no overhead for cases where single-core jobs can be submitted separately. While this is true it means that every job needs to wait in the queue again, thus increasing the total time to solution. For large task durations, this effect is less pronounced but typically the job array approach doubles the wall time which renders this approach unfavorable.

Using job steps alone becomes inefficient if the task durations are long, since the assumption that all tasks are roughly of identical duration will mean that interruptions of unfinished calculations occur more often. Having a more precise estimate allows for more efficient packing. This becomes important on large computer clusters where only full nodes can be allocated: in this case, the imbalance of the durations of calculations running in parallel further increases the overhead. Our method typically gave a parallelization overhead of 10%–15% for a range of data sets. For example, in the task $\mathbf{QMrxn^{GO}_{MP2}}$, our approach allowed us to go to two orders of magnitude more computer resources and have the same overhead as job step parallelization. This is a strong case for using QML based timing estimates in a production environment—in particular, since the number of training data points required is very limited (see figure 5).

4.3.2. Geometry optimization steps

Given that the number of steps of a geometry optimization is difficult to learn (see lower panel of figure 9), the ability to accurately predict the duration of a single geometry optimization step allows to increase efficiency via another route. On hybrid computer clusters, the maximum duration of a single computer job is limited. We suggest to check during the course of a geometry optimization whether the remaining time of the current computer job is sufficient to complete another step. If not, it is more efficient to relinquish the computer resources immediately rather than committing them to the presumably futile undertaking of computing the

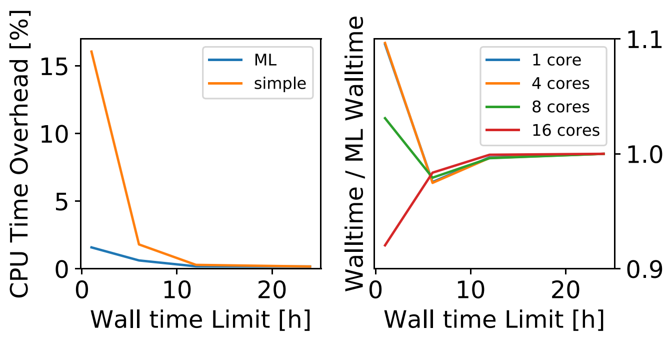

Figure 10. CPU time overhead and wall time for geometry optimizations compared between the simple approach and the QML approach. See text for details of the strategies. CPU time overhead given in percent relative to the bare minimum of CPU time needed. Wall time given relative to the wall time resulting from using the QML approach. All geometry optimizations come from task $\mathbf{QM9}^{\mathrm{GO}}_{\mathrm{CASSCF}}$.

next step. We refer to these strategies as the 'simple approach' (take all CPU time you can, give nothing back) and the 'QML approach' (give up resources early). Figure 10 shows the advantage of the QML approach: it allows to go towards shorter computer jobs and reduces the CPU time overhead by up to 90% for small wall time limits using the job array approach. This is more efficient for the scheduler and increases the likelihood of the job being selected by the backfiller, further shortening the wall time. Using the QML approach does not severely affect the wall time, i.e. the time-to-solution. This is largely independent of the extent of parallelization employed in the calculation (see right hand side plot in figure 10). We suggest to implement an optional stop criterion in quantum chemical codes where an external command can prematurely stop the progress of the geometry optimization to be resumed in the next computer job. This change can drastically improve computational efficiency on large scale projects. Estimating the current consumption to be on the order of at least 5×10^5 petaFLOPS (see discussion above in section 1) for computational chemistry and materials science this approach may lead to potentially large savings in economical cost.

5. Conclusion

We have shown that the computational complexity of quantum chemistry calculations can be predicted across chemical space by QML models. First we looked at a 2D nonlinear toy system consisting of example functions which are known to be difficult to optimize. Using these test functions and three optimizers, we build a first ML model and the learning curves show that it is possible to learn the number of optimization steps using only the starting position (x, y). Representations are designed to efficiently cover all relevant dimension in the given chemical space. Hence, if the computational cost is learnable by QML models, it is a reasonably smooth function in the variety of chemical spaces that we considered. This is a fundamental result.

Our approach succeeds in estimating realistic timings of a broad variety of representative calculations commonly used in quantum chemistry work-flows: single-point calculations, geometry optimizations, and transition state searches with very different levels of theory and basis sets. The machine learning performance depends on the quantum chemistry method and on the type of computational cost that is learned (FLOP, CPU, wall time). While the accuracy of the prediction is shown to be strongly dependent on the computational method, we could typically predict the total run time with an accuracy between 2% and 30%.

Exploiting QML out-of-sample predictions, we have demonstrably used computer clusters more efficiently by reordering jobs rather than blindly assuming all calculations of one kind to fit into the same time window. Without significant changes in the time-to-solution, we reduced the CPU time overhead by 10%–90% depending on the task. With the scheme presented in this work, computer resource usage can be significantly optimized for large scale chemical space computer campaigns. To support this case, all relevant code, data, and a simple-to-use interface is made available to the community online [99].

We believe that our findings are important since it is not obvious that established QML models, designed for estimating physical observables, are also applicable to more implicit quantities such as computational cost.

Acknowledgments

MS would like to acknowledge Dr Peter Zaspel for helpful discussions on the FLOP measurement and the setup of the native program builds. This work was supported by a grant from the Swiss National Supercomputing

Center (CSCS) under project ID s848. Some calculations were performed at sciCORE (http://scicore.unibas.ch/) scientific computing core facility at University of Basel. We acknowledge funding from the Swiss National Science foundation (No. 407540_167186 NFP 75 Big Data, 200021_175747) and from the European Research Council (ERC-CoG grant QML). This work was partly supported by the NCCR MARVEL, funded by the Swiss National Science Foundation.

Data availability statement

Any data (except the carbene data set) that support the findings of this study are included within the article. The carbene data set is available from the corresponding author upon reasonable request.

ORCID iDs

Stefan Heinen https://orcid.org/0000-0001-9382-2342
Max Schwilk https://orcid.org/0000-0001-6470-1779
Guido Falk von Rudorff https://orcid.org/0000-0001-7987-4330
O Anatole von Lilienfeld https://orcid.org/0000-0001-7419-0466

References

[1] Garey M R and Johnson D S 1990 *Computers and Intractability; A Guide to the Theory of NP-Completeness* (New York, NY: W. H. Freeman & Co.)
[2] Track E, Forbes N and Strawn G 2017 The end of Moore's law *Comput. Sci. Eng.* **19** 4–6
[3] Argone Leadership Computing Facility https://alcf.anl.gov/ (Accessed: 05 June 2019)
[4] Swiss National Supercomputing Center, Annual Report 2017 https://cscs.ch (Accessed: 26 April 2019)
[5] National Energy Research Scientigic Computing Center https://nersc.gov/ (Accessed: 02 June 2019)
[6] Archer http://archer.ac.uk/ (Accessed: 05 June 2019)
[7] NVIDIA enabling new path to exascale supercomputing, https://nvidianews.nvidia.com (Accessed: 24 June 2019)
[8] Davidson E R 1975 The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices *J. Comput. Phys.* **17** 87–94
[9] Sherrill C D 1996 Computational scaling of the configuration interaction method with system size http://vergil.chemistry.gatech.edu/notes/ciscale/ciscale.html (Accessed: 03 June 2019)
[10] List of top 500 super computers http://top500.org (Accessed: 24 April 2019)
[11] Singh K, Ipek E, McKee S A, de Supinski B R, Schulz M and Caruana R 2007 Predicting parallel application performance via machine learning approaches *Concurrency Comput.: Pract. Exp.* **19** 2219–35
[12] Malakar P, Balaprakash P, Vishwanath V, Morozov V and Kumaran K 2018 Benchmarking machine learning methods for performance modeling of scientific applications *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)* pp 33–44
[13] Wang Y, Qiao J, Lin S and Zhao T 2018 An approximate optimal solution to GPU workload scheduling *Comput. Sci. Eng.* **20** 63–76
[14] Rodrigues E R, Cunha R L, Netto M A and Spriggs M 2016 Helping HPC users specify job memory requirements via machine learning *2016 3rd Int. Workshop on HPC User Support Tools (HUST)* pp 6–13
[15] Witt C, Bux M, Gusew W and Leser U 2019 Predictive performance modeling for distributed batch processing using black box monitoring and machine learning *Inf. Syst.* **82** 33–52
[16] Nemirovsky D, Arkose T, Markovic N, Nemirovsky M, Unsal O, Cristal A and Valero M 2018 A general guide to applying machine learning to computer architecture *Supercomput. Frontiers Innov.* **5** 95–115
[17] Garg S K, Yeo C S, Anandasivam A and Buyya R 2011 Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers *J. Parallel Distrib. Comput.* **71** 732–49
[18] Nemirovsky D, Arkose T, Markovic N, Nemirovsky M, Unsal O, Cristal A and Valero M 2017 A deep learning mapper (DLM) for scheduling on heterogeneous systems *Latin American High Performance Computing Conf.* pp 3–20
[19] Kousalya G, Balakrishnan P and Raj C P 2017 *Automated Workflow Scheduling in Self-Adaptive Clouds* (Berlin: Springer) pp 119–35
[20] Sahni J and Vidyarthi D P 2018 A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment *IEEE Trans. Cloud Comput.* **6** 2–18
[21] Liu H, Zhao R and Nie K 2018 Using ensemble learning to improve automatic vectorization of tensor contraction program *IEEE Access* **6** 47112–24
[22] Antony J, Rendell A P, Yang R, Trucks G and Frisch M J 2011 Modelling the runtime of the Gaussian computational chemistry application and assessing the impacts of microarchitectural variations *Proc. Comput. Sci.* **4** 281–91
[23] Papay J, Atherton T J, Zemerly M J and Nudd G R 1996 Performance prediction of parallel self consistent field computation *Parallel Algorithms Appl.* **10** 127–43
[24] Mniszewski S M, Junghans C, Voter A F, Perez D and Eidenbenz S J 2015 TADSim: discrete event-based performance prediction for temperature-accelerated dynamics *ACM Trans. Modeling Comput. Simul. (TOMACS)* **25** 15
[25] Duan C, Janet J P, Liu F, Nandy A and Kulik H J 2019 Learning from failure: predicting electronic structure calculation outcomes with machine learning models *J. Chem. Theory Comput.* **15** 2331–45
[26] von Lilienfeld O A 2013 First principles view on chemical compound space: gaining rigorous atomistic control of molecular properties *Int. J. Quantum Chem.* **113** 1676–89
[27] von Lilienfeld O A 2018 Quantum machine learning in chemical compound space *Angew. Chem. Int. Ed.* **57** 4164
[28] Rupp M, von Lilienfeld O A and Burke K 2018 Guest editorial: special topic on data-enabled theoretical chemistry *J. Chem. Phys.* **148** 241401

[29] Rupp M, Tkatchenko A, Müller K-R and von Lilienfeld O A 2012 Fast and accurate modeling of molecular atomization energies with machine learning *Phys. Rev. Lett.* **108** 058301
[30] Hansen K, Montavon G, Biegler F, Fazli S, Rupp M, Scheffler M, von Lilienfeld O A, Tkatchenko A and Müller K-R 2013 Assessment and validation of machine learning methods for predicting molecular atomization energies *J. Chem. Theory Comput.* **9** 3404–19
[31] Ramakrishnan R and von Lilienfeld O A 2015 Many molecular properties from one kernel in chemical space *CHIMIA* **69** 182
[32] Huang B and von Lilienfeld O A 2016 Communication: understanding molecular representations in machine learning: the role of uniqueness and target similarity *J. Chem. Phys.* **145** 161102
[33] Ramakrishnan R and von Lilienfeld O A 2017 *Reviews in Computational Chemistry* vol 30 (New York: Wiley) pp 225–56
[34] Faber F A, Hutchison L, Huang B, Gilmer J, Schoenholz S S, Dahl G E, Vinyals O, Kearnes S, Riley P F and von Lilienfeld O A 2017 Prediction errors of molecular machine learning models lower than hybrid DFT error *J. Chem. Theory Comput.* **13** 5255–64
[35] Rasmussen C E and Williams C K I 2006 *Gaussian Processes for Machine Learning* (Cambridge, MA: MIT Press) http://gaussianprocess.org
[36] Montavon G, Rupp M, Gobre V, Vazquez-Mayagoitia A, Hansen K, Tkatchenko A, Müller K-R and von Lilienfeld O A 2013 Machine learning of molecular electronic properties in chemical compound space *New J. Phys.* **15** 095003
[37] Smith J S, Isayev O and Roitberg A E 2017 ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost *Chem. Sci.* **8** 3192–203
[38] Schütt K T, Arbabzadah F, Chmiela S, Müller K R and Tkatchenko A 2017 Quantum-chemical insights from deep tensor neural networks *Nat. Commun.* **8** 13890
[39] Schütt K T, Sauceda H E, Kindermans P-J, Tkatchenko A and Müller K-R 2018 SchNet—A deep learning architecture for molecules and materials *J. Chem. Phys.* **148** 241722
[40] Unke O T and Meuwly M 2018 A reactive, scalable, and transferable model for molecular energies from a neural network approach based on local information *J. Chem. Phys.* **148** 241708
[41] Ramakrishnan R, Dral P, Rupp M and von Lilienfeld O A 2014 Quantum chemistry structures and properties of 134 kilo molecules *Sci. Data* **1** 140022
[42] Ruddigkeit L, van Deursen R, Blum L and Reymond J-L 2012 Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17 *J. Chem. Inf. Model.* **52** 2684
[43] Weininger D 1988 SMILES, a chemical language and information system: I. Introduction to methodology and encoding rules *J. Chem. Inform. Comput. Sci.* **28** 31–6
[44] Weininger D, Weininger A and Weininger J 1989 SMILES. 2. Algorithm for generation of unique SMILES notation *J. Chem. Inf. Model.* **29** 97–101
[45] Hansen K, Biegler F, von Lilienfeld O A, Müller K-R and Tkatchenko A 2015 Machine learning predictions of molecular properties: accurate many-body potentials and nonlocality in chemical space *J. Phys. Chem. Lett.* **6** 2326
[46] Faber F A, Hutchison L, Huang B, Gilmer J, Schoenholz S S, Dahl G E, Vinyals O, Kearnes S, Riley P F and von Lilienfeld O A 2017 Prediction errors of molecular machine learning models lower than hybrid DFT error *J. Chem. Theory Comput.* **13** 5255–64
[47] Schütt K T, Arbabzadah F, Chmiela S, Müller K R and Tkatchenko A 2017 Quantum-chemical insights from deep tensor neural networks *Nat. Commun.* **8** 13890
[48] Gilmer J, Schoenholz S S, Riley P F, Vinyals O and Dahl G E 2017 Neural message passing for quantum chemistry arXiv:1704.01212
[49] Bartók A P, De S, Poelking C, Bernstein N, Kermode J R, Csányi G and Ceriotti M 2017 Machine learning unifies the modeling of materials and molecules *Sci. Adv.* **3** e1701816
[50] Faber F A, Christensen A S, Huang B and von Lilienfeld O A 2018 Alchemical and structural distribution based representation for universal quantum machine learning *J. Chem. Phys.* **148** 241717
[51] Unke O T and Meuwly M 2018 A reactive, scalable, and transferable model for molecular energies from a neural network approach based on local information *J. Chem. Phys.* **148** 241708
[52] Lubbers N, Smith J S and Barros K 2018 Hierarchical modeling of molecular energies using a deep neural network *J. Chem. Phys.* **148** 241715
[53] Eickenberg M, Exarchakis G, Hirn M, Mallat S and Thiry L 2018 Solid harmonic wavelet scattering for predictions of molecule properties *J. Chem. Phys.* **148** 241732
[54] Simm G N and Reiher M 2018 Error-controlled exploration of chemical reaction networks with Gaussian processes *J. Chem. Theory Comput.* **14** 5238–48
[55] Meyer B, Heinen S, von Lilienfeld O A, Sawatlon B and Corminboeuf C 2018 Machine learning meets volcano plots: computational discovery of cross-coupling catalysts *Chem. Sci.* **35** 7069–77
[56] Smith J S, Isayev O and Roitberg A E 2017 ANI-1, A data set of 20 million calculated off-equilibrium conformations for organic molecules *Sci. Data* **4** 170193
[57] Janet J P and Kulik H J 2017 Predicting electronic structure properties of transition metal complexes with neural networks *Chem. Sci.* **8** 5137–52
[58] Li Z, Omidvar N, Chin W S, Robb E, Morris A, Achenie L and Xin H 2018 Machine-learning energy gaps of porphyrins with molecular graph representations *J. Phys. Chem.* A **122** 4571–8
[59] Rosenbrock H H 1960 An automatic method for finding the greatest or least value of a function *Comput. J.* **3** 175–84
[60] Himmelblau D M 1972 *Applied Nonlinear Programming* (New York: McGraw-Hill)
[61] Jones E, Oliphant T and Peterson P 2001 SciPy: open source scientific tools for Python http://scipy.org/ [Version: 1.3.1]
[62] Nelder J A and Mead R 1965 A simplex method for function minimization *Comput. J.* **7** 308–13
[63] Byrd R H, Lu P, Nocedal J and Zhu C 1995 Limited memory algorithm for bound constrained optimization *SIAM J. Sci. Comput.* **16** 1190–208
[64] Nash S G 1984 Newton-type minimization via the Lanczos method *SIAM J. Numer. Anal.* **21** 770–88
[65] Schwilk M, Ma Q, Köppl C and Werner H-J 2017 Scalable electron correlation methods. 3. Efficient and accurate parallel local coupled cluster with pair natural orbitals (PNO-LCCSD) *J. Chem. Theory Comput.* **13** 3650–75
[66] Ma Q, Schwilk M, Köppl C and Werner H-J 2017 Scalable electron correlation methods. 4. Parallel explicitly correlated local coupled cluster with pair natural orbitals (PNO-LCCSD-F12) *J. Chem. Theory Comput.* **13** 4871–96
[67] Ma Q and Werner H-J 2018 Scalable electron correlation methods. 5. Parallel perturbative triples correction for explicitly correlated local coupled cluster with pair natural orbitals *J. Chem. Theory Comput.* **14** 198–215
[68] Schwilk M, Zaspel P, von Lilienfeld O A and Harbrecht H 2020 to be published
[69] Knowles P J and Werner H-J 1988 An efficient method for the evaluation of coupling coefficients in configuration interaction calculations *Chem. Phys. Lett.* **145** 514–22

[70] Werner H and Knowles P J 1988 An efficient internally contracted multiconfiguration-reference configuration interaction method *J. Chem. Phys.* **89** 5803–14
[71] Shiozaki T, Knizia G and Werner H-J 2011 Explicitly correlated multireference configuration interaction: MRCI-F12 *J. Chem. Phys.* **134** 034113
[72] Shiozaki T and Werner H-J 2013 Multireference explicitly correlated F12 theories *Mol. Phys.* **111** 607–30
[73] Tahchieva D N, Schwilk M and von Lilienfeld O A 2020 to be published
[74] Becke A D 1993 Density-functional thermochemistry: III. The role of exact exchange *J. Chem. Phys.* **98** 5648
[75] Lee C, Yang W and Parr R G 1988 Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density *Phys. Rev.* B **37** 785–9
[76] Werner H and Knowles P J 1985 A second order multiconfiguration SCF procedure with optimum convergence *J. Chem. Phys.* **82** 5053–63
[77] Busch T, Esposti A D and Werner H 1991 Analytical energy gradients for multiconfiguration self-consistent field wave functions with frozen core orbitals *J. Chem. Phys.* **94** 6708–15
[78] Peterson K A, Adler T B and Werner H-J 2008 Systematically convergent basis sets for explicitly correlated wavefunctions: the atoms H, He, B-Ne, a and Al-Ar *J. Chem. Phys.* **128** 084102
[79] Weigend F and Ahlrichs R 2005 Balanced basis sets of split valence, triple zeta valence and quadruple zeta valence quality for H to Rn: design and assessment of accuracy *Phys. Chem. Chem. Phys.* **7** 3297
[80] Weigend F 2006 Accurate Coulomb-fitting basis sets for H to Rn *Phys. Chem. Chem. Phys.* **8** 1057
[81] Binkley J S, Pople J A and Hehre W J 1980 Self-consistent molecular orbital methods. 21. Small split-valence basis sets for first-row elements *J. Am. Chem. Soc.* **102** 939–47
[82] Petersson G A, Bennett A, Tensfeldt T G, Al-Laham M A, Shirley W A and Mantzaris J 1988 A complete basis set model chemistry: I. The total energies of closed-shell atoms and hydrides of the first-row elements *J. Chem. Phys.* **89** 2193–218
[83] Petersson G A and Al-Laham M A 1991 A complete basis set model chemistry: II. Open-shell systems and the total energies of the first-row atoms *J. Chem. Phys.* **94** 6081–90
[84] Werner H-J *et al* MOLPRO, a package of ab initio programs http://molpro.net
[85] Neese F 2006 ORCA 2.8 *An ab initio, Density Functional and Semiempirical Program Package* (Germany: University of Bonn)
[86] Ma Q and Werner H-J 2018 Explicitly correlated local coupled-cluster methods using pair natural orbitals *Wires Comput. Mol. Sci.* **8** e1371
[87] Krige D G 1951 A statistical approaches to some basic mine valuation problems on the witwatersrand *J. Chem., Metall. Mining Soc. South Afr.* **52** 119–39
[88] von Lilienfeld O A, Ramakrishnan R, Rupp M and Knoll A 2015 Fourier series of atomic radial distribution functions: a molecular fingerprint for machine learning models of quantum chemical properties *Int. J. Quantum Chem.* **115** 1084
[89] Müller K R, Finke M, Murata N, Schulten K and Amari S 1996 A numerical study on learning curves in stochastic multilayer feedforward networks *Neural Comput.* **8** 1085
[90] Huang B and von Lilienfeld O A 2017 The 'DNA' of chemistry: scalable quantum machine learning with 'amons' arXiv:1707.04146
[91] Ramakrishnan R, Dral P, Rupp M and von Lilienfeld O A 2015 Big data meets quantum chemistry approximations: the Δ-machine learning approach *J. Chem. Theory Comput.* **11** 2087
[92] Christensen A S, Faber F A, Huang B, Bratholm L A, Tkatchenko A, Müller K-R and von Lilienfeld O A 2017 QML: a python toolkit for quantum machine learning 10.5281/zenodo.817332
[93] Intel Corporation, Intel Math Kernel Library. 2018 https://software.intel.com/en-us/mkl (Accessed: 20 November 2018)
[94] Xianyi Z, Qian W and Saar W 2017 OpenBLAS, an optimized BLAS library http://openblas.net/ (Accessed: 16 November 2018)
[95] Open MPI: Open Source High Performance Computing. 2018 https://open-mpi.org/ (Accessed: 15 November 2018)
[96] Free Software Foundation, Inc., GCC, the GNU Compiler Collection. 2017 https://gcc.gnu.org/ (Accessed: 14 November 2018)
[97] Global Arrays Programming Models. 2018 http://hpc.pnl.gov/globalarrays/ (Accessed: 16 November 2018)
[98] Nieplocha J, Palmer B, Tipparaju V, Krishnan M, Trease H and Apra E 2006 Advances, applications and performance of the global arrays shared memory programming toolkit *Int. J. High Perf. Comp. Appl.* **20** 203–31
[99] Raw data for this work and sample implementations for easy use can be found on https://github.com/ferchault/mlscheduling