Prediction of Time-to-Solution in Material Science Simulations Using Deep Learning

Prediction of time-to-solution in material science simulations using Deep Learning

Federico Pittino*
federico.pittino@unibo.it
Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna
Bologna, Italy

Pietro Bonfà*
pietro.bonfa@fis.unipr.it
Department of Mathematical, Physical and Computer Sciences, University of Parma
Parma, Italy

Andrea Bartolini
a.bartolini@unibo.it
Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna
Bologna, Italy

Fabio Affinito
f.affinito@cineca.it
CINECA
Casalecchio di Reno (Bologna), Italy

Luca Benini
luca.benini@unibo.it
Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna
Bologna, Italy
Integrated Systems Laboratory, ETH Zurich
Zurich, Switzerland

Carlo Cavazzoni
c.cavazzoni@cineca.it
CINECA
Casalecchio di Reno (Bologna), Italy

ABSTRACT

Predicting the time to solution for massively parallel scientific codes is a complex task. The reason for this is the presence of multiple, strongly interconnected algorithms that possibly react differently to the changes in compute power, vectorization length, memory and network bandwidth and latency and I/O throughput. A reliable prediction of execution time is however of great importance to the user who wants to plan on large scale simulations or virtual screening procedures characteristic of high throughput computing. In this article we present a practical approach based on machine learning techniques to achieve very accurate predictions of the time to solution for a DFT-based material science code. We compare our results with the predictions provided by a parametrized analytical performance model showing that deep learning solutions allow for a greater accuracy without the need of domain knowledge to introduce an explicit description of the algorithms implemented in the code.

ACM Reference Format:
Federico Pittino, Pietro Bonfà, Andrea Bartolini, Fabio Affinito, Luca Benini, and Carlo Cavazzoni. 2020. Prediction of time-to-solution in material science simulations using Deep Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Both authors contributed equally to this research.

1 INTRODUCTION

One of the consequence of the end of Moore's era and Dennard's scaling has been the push towards architecture specialization to keep the pace in performance growth of HPC systems [1]. On one side we see the rise of hybrid systems, where the standard central processing unit (CPU) is accompanied by one or more accelerators. Other approaches still adopt the classical CPU-based solution, but with a drastic increase of the number of cores per node and of the set of vectorized instructions. At the same time, scientific software targeting HPC systems faces this growing diversity by branching out algorithms and enriching the parallel or accelerated execution options. As a consequence, the (parallel) execution of these applications becomes more complex and their performance on different machines becomes harder to predict. This may impact the users of high throughput computing (HTC) and HPC systems significantly. On the one hand, it may become difficult to identify execution problems as the expected execution time on a given machine becomes more unpredictable. On the other hand, when planning for large scale simulations or HTC approaches, reliable estimates of the time to solution for a given set of simulation are very hard to obtain.

Finally, it may also become rather complex to estimate the reduction of the time to solution for different parallel execution schemes as the number of possible approaches to solve the same problem grows.

All these problems derive from the difficulty of finding reliable strategies to estimate the computational efficiency of scientific codes given a computational method, a given scientific case and a set of compute node(s). In fact, having an accurate estimate of the execution time for any scientific application with a given input file in an HPC environment would lead to more precise resources allocation, therefore saving money and improving utilisation of computational resources, which is particularly relevant for high-throughput computations.

Miu and Missier [2] showed promising results in the estimation of the execution time of chemical engineering workflows using decision tree methods to perform machine learning on the base of a set of input features. A different approach based on radial basis function neural network (RBF-NN) was proposed by Nadeem *et al.* [3] for the prediction workflows' execution time for grid based computing. Notably, one of the workflows considered by the authors is based on the *ab initio* code Wien2K [4]. A thorough review of other approaches to solve this problem is provided by [5], mostly employing Machine Learning techniques. The common trait to all these approaches is the assumption that the models should work by monitoring the online status of the machine and the workload characteristics. These approaches are thereby not exploiting any domain knowledge on how the application is designed, and neither of its current input.

A completely different approach is pursued in [6], where the authors model the applications performance using their source code as input. The advantage of such an approach is that the model becomes aware of the operation of each application, and therefore its predictions more independent of the particular architecture on which it was derived. On the other hand, such a model is very complex to obtain.

In our work we take instead a different approach, by focusing on the application level and using as input to the prediction models a set of features derived from the application input files. We also specialize the goal of finding the execution time by considering a selected set of subroutines of the QuantumESPRESSOsuite [7]. We show that accurate predictions can be obtained with machine learning (ML) techniques using a modest amount of training data and algorithms. As expected, the best performance is obtained by using advanced deep learning techniques, i.e., training multi-layered artificial neural networks [8], which are rapidly becoming the state-of-the-art in multiple different fields. In particular, a deep artificial neural network is able to even beat the accuracy of a full-custom analytical model specifically derived using expert knowledge of the application, improving the prediction error by up to 5%.

The article is organized as follows. In section 2 we describe how the data used for the machine leaning was generated, collected and prepared. Section 3 discusses the various ML approaches considered and describes a parametrized analytical performance model that we developed to compare the quality of its predictions against the ML results, that appear in section 4. Conclusions and perspectives are drawn in section 5.

2 DATA AND CODE DESCRIPTION

In this section we present the code used to perform the runs of the application considered in this work. Some relevant characteristics of the data collection are also discussed in relation to their composition and diversity. Both these points are in fact of importance for the sake of performing machine learning based approaches.

A set of about 10k runs is publicly available in the "2D structures and layered materials" collection [9] stored in the MaterialsCloud repository [10]. These runs were mainly produced with the codes of QuantumESPRESSO (QE) [7, 11], a suite for performing material science simulations at the nanoscale in massively parallel environment.

2.1 Codes

The QuantumESPRESSO suite is a collection of applications, mostly written in modern Fortran, for performing simulations aiming at obtaining electronic structure details and estimating materials' properties from first principles. The term "first principles" is used to indicate that this kind of simulations does not depend on ad-hoc parametrizations but only requires the derivation from the fundamental laws of (quantum) physics of a (many-body) interaction problem to be solved numerically. In the field of materials modelling, the definition of the problem only depends on the position of the elements in a compound. As a matter of fact, a number of approximations enters the theory and the discretization of the mathematical problem implied by numerical approaches increases the number of initial information that these simulations require. As a consequence, the type of input data that one requires to devise a simulation can be divided into two classes: the physical description of the material under study and the set of information that is required to represent its quantum-physical description into a discretized computational approach.

There is a third class of control variables that is specific to the code and algorithm implementations used to solve the problem. These variables are defined during the execution phase, i.e. at runtime. Examples may include parallelization options, input/output policies, optimization levels etc.

To describe how these data enter the evaluation of the total time to solution, let us start describing briefly the code of the QE suite we focused in this work: PWscf.

2.2 Algorithms

PWscf, solves the Khon-Sham (KS) equations [12] iteratively on a plane wave basis and using various approaches for treating core electrons, namely, norm conserving, Ultrasoft [13] or PAW pseudopotentials [14]. It features a hybrid MPI+OpenMP parallelism with hierarchical levels for data and computational intensity distribution. These levels are mapped onto MPI Communicators and are all user tunable to optimize the code performance on different architectures.

Tab. 1 summarizes the list of runtime options that govern the parallel execution strategies. Following a top-down description, `NPool` portions out the world communicator and defines the number of groups of MPI processes that will solve the KS Hamiltonian at a particular point $\mathbf{k}$ in reciprocal space. In each group there will be `NBgrp` MPI processes sharing distributed real and reciprocal space grids. A subset (or all) `NBgrp` processes can be used for the solution of the eigenvalue problem of each pool. This value is set by the `NDiag` metric. Finally, `NCores` is the total number of MPI and OpenMP processes running the code. The following relation holds:

$$N_{Cores} = N_{Pool} \times N_{Bgrp} \times N_{OpenMP} \tag{1}$$

where N_{OpenMP} is the number of OpenMP threads spawned by each MPI process.

From the algorithmic point of view, the computational intensive tasks mainly consist of as small sized Fourier transforms, linear algebra operations (with BLAS level 3 operations dominating the workload) and the solution of dense eigenvalue problems originating from the iterative diagonalization (see below). These three

classes, generally performed by specialized and optimized libraries, account for a sizable part of the (per MPI process) execution time, possibly close to 50% as the size of the simulation grows. Another sizable component in the time to simulation is MPI communication. As shown by Cesarini *et al.* [15], as the number of MPI processes grows, the synchronization and communication time may reach up to half of the total time to solution.

The eigenvalue problem in plane wave based DFT codes is generally tackled with iterative algorithms. The reader is referred to one of the many articles and reviews on this topic [7, 16–19] for the details. In PWscf, two algorithms are currently implemented, Davidson and Conjugate Gradient. These two approaches are used to solve the generalized eigenvalue problem

$$\mathcal{H}\psi_i = \varepsilon_i \mathcal{S}\psi_i \tag{2}$$

with $\mathcal{H}$ and $\mathcal{S}$ being the Hamiltonian and the overlap operator, ε_i and ψ_i the i-th eigenvalues and eigenstates. In general, the lowest N eigen-pairs corresponding to the occupied states are sought.

In both cases one has to compute the application of the Hamiltonian and overlap operators on a pre-existing trial solution and, in a number of steps dependent on the chosen algorithm, the converged eigen-pairs are obtained. These inner iterations, performed every time the solution of the KS equation is needed, require either the solution of eigenvalue problems defined on a (at least) $2N$ dimensional subspace (see Appendix A.2.1 in Ref [7]) that provide the correction to the trial solution or from a sequence of (constrained) minimization problems.

As it is evident, the two methods have substantially different parallel performance and this is accounted for by the `CG` category in Tab. 1.

The physical description of the material under study is encoded in the variables `NKS` and `pseudo` that represent the number of KS states in the system and the elements present in the lattice respectively.

A second set of parameters is connected with the discretization and the approximations entering the numerical method. The characteristics of the pseudopotential (for what concerns its non-local part) are embedded in the `Nl` value, which contains the number of β functions and their angular momentum[1]. In addition, the system may be described with or without spin degrees of freedom. The absence of spin degrees of freedom speeds up the simulation by about a factor 2. This is accounted for by the `nspin` metric.

The truncation of the plane wave basis set leads to the definition of the FFT grids where, in reciprocal space, these are contained in a sphere. These details are stored in `densegrid_G` and `densegrid_real`.

2.3 Data

In this section we describe the data collected in [9] without entering too much into the scientific case discussed in [21]. The database contains a large set of simulations that are part of a screening procedure that, starting from the bulk structures available in the Crystallography Open Database [22], aims at identifying good candidates for 2D materials that could be easily exfoliated from their parent compounds.

This database collects a number of simulations that include ground state energy and electron density estimations, atomic structure relaxations and lattice vibration (phonon) dispersion curves. The majority of the ground state energy simulations was performed on periodic systems with small to medium sized unit cell with 90% of the volumes in the range $10 \div 500$ Å^3.

The simulations available in the published database [9] were performed using a number of different compute nodes and with different versions of the codes. We focused on a selected subset by collecting the results obtained with the PWscf executable and re-grouping the various outputs according to the code version and the machine name that was used to generate them.

The data span a vast number of structures and include 72 elements of the periodic table. As already stated, the size of the simulation is peaked at small sized structures. As a consequence, a limited number of combinations of parallelization options was used. On the other hand, pseudopotential methods span both Ultrasoft and PAW methods while we find a limited use of Norm Conserving pseudopotentials. The time to solution taken by the simulations spans over four orders of magnitude and the time per iteration, later used in our analysis, ranges from tenths to tens of seconds. The distributions of time per iteration for each compute node is shown in Fig. 1.

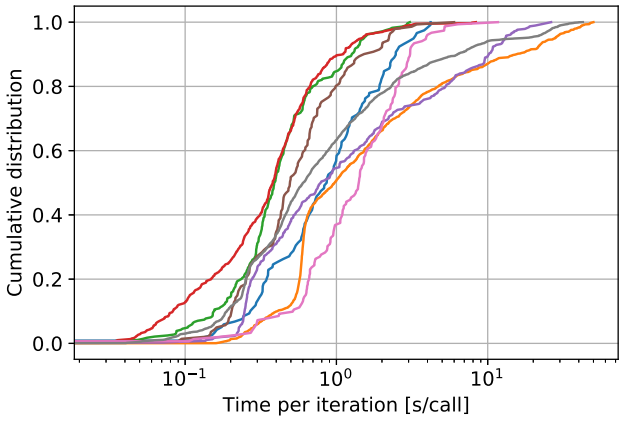


Figure 1: Distribution of the time per iteration for each of the 8 compute nodes used in our dataset (the different curves).

3 METHODS

The goal of this investigation is the prediction of the execution time for each iteration of the SCF loop. In this section we then describe the models we developed for this purpose and how relevant features for such models are extracted from the input files.

[1]For a description of the pseudopotential formalism see for example page 220 of Ref. [20]

3.1 Iteration execution time

To assess the duration of each iteration in the loop, we have used the timer "cbands" from the output files. The value of the timer is then divided by the number of calls to the function to obtain the average duration per iteration.
The prediction of execution time is computed with a diverse set of Machine Learning (ML) algorithms based on a set of features of the simulation extracted from the input files. All features, except for the categorical ones, are normalized to zero mean and unit variance. The features we have chosen for this task are summarised in Tab. 1, and they aim at providing information about various aspects of the application:

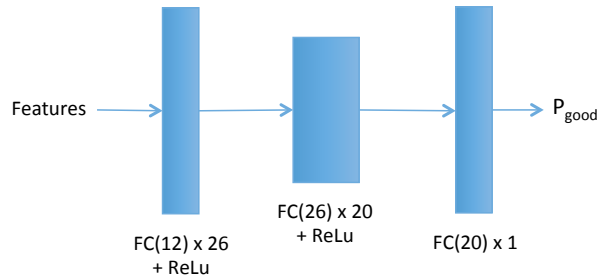
- Algorithm used in the application: `nspin`, `CG`;
- Resources used by the machine: `NDiag`, `NCores`, `NBgrp`, `NPool`;
- Choices for the physical representation of the simulated system: `pseudo`, `NKS`, `Nl`;
- Parameters of the discretization: `densegrid_real`, `densegrid_G`, `Nk`.

Table 1: Features chosen for the ML algorithms

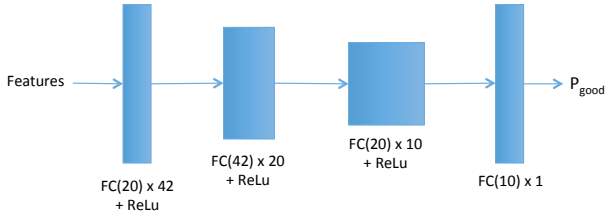
Metric name	Description	Values
`nspin`	Whether spin polarisation is present or not	$[0, 1]$
`CG`	If CG algorithm for eigenvalue problem is used	$[0, 1]$
`NDiag`	Number of MPI processes used in the parallel diagonalization subroutine	$[1, \infty]$
`pseudo`	Number of different element types	$[1, \infty]$
`NCores`	Number of cores used for parallel calculations	$[1, \infty]$
`NBgrp`	Number of MPI processes in the BandGroup communicator	$[1, \infty]$
`NPool`	Number of MPI processes in the POOL communicator	$[1, \infty]$
`Nk`	Number of k points used to sample the reciprocal space	$[1, \infty]$
`NKS`	Number of Kohn-Sham states	$[1, \infty]$
`Nl`	Sum of the contributions from the l components for each β-function	$[1, \infty]$
`densegrid_G`	Number of G vectors	$[1, \infty]$
`densegrid_real`	Dimension in real space of the FFT grid	$[1, \infty]$

Moreover, we have decided to predict not directly the execution time, but its logarithm, both in order to obtain an always-positive number (which has to represent a time) and since the variations between execution times of different runs can be very large.

A last feature we explored is the information about the machine (compute node) on which the simulation was run. Since simulations run on different machines can behave very differently due to hardware diversity (and we do not use specific hardware information in this work), we have chosen two approaches to deal with this information:



(a) separate per-machine models



(b) joint model

Figure 2: Architecture of the FCNN models.

- for each machine we train a completely different model (we call this "separate model");
- we train a joint model with data from all machines, using the information about the machine as an additional categorical feature (we call this "joint model"); since 8 machines have been used in our dataset, this information translates into 8 additional boolean features.

The percentage of data in the test sets is always 20% of the available data, and this is derived separately for each machine in the case of separate models. The training data is then used for a 10-fold cross-validation.

ML algorithms. As far as the algorithms are concerned, we have explored three different choices of ML algorithms:

(1) Linear Regression (LR) with L2 weight decay [23];
(2) Kernel Ridge Regression (KRR) with radial basis functions [23];
(3) Fully Connected Neural Network (FCNN) with L2 weight decay [8].

The architectures of the FCNNs for the joint and separate models are shown in Fig. 2. For training and testing the models we have used the libraries Scipy [24], for the LR and KRR, and PyTorch using the Adam algorithm [25], for the training of the FCNN.

3.2 Performance model

In order to verify how machine learning approaches compare with analytical performance modeling results, we also defined a strategy to reproduce the time required by a single iteration of the KS procedure. There are multiple approaches to do so, here we briefly introduce the most common ones. A possibility is to fit a number of parameters describing the underlying hardware in terms of

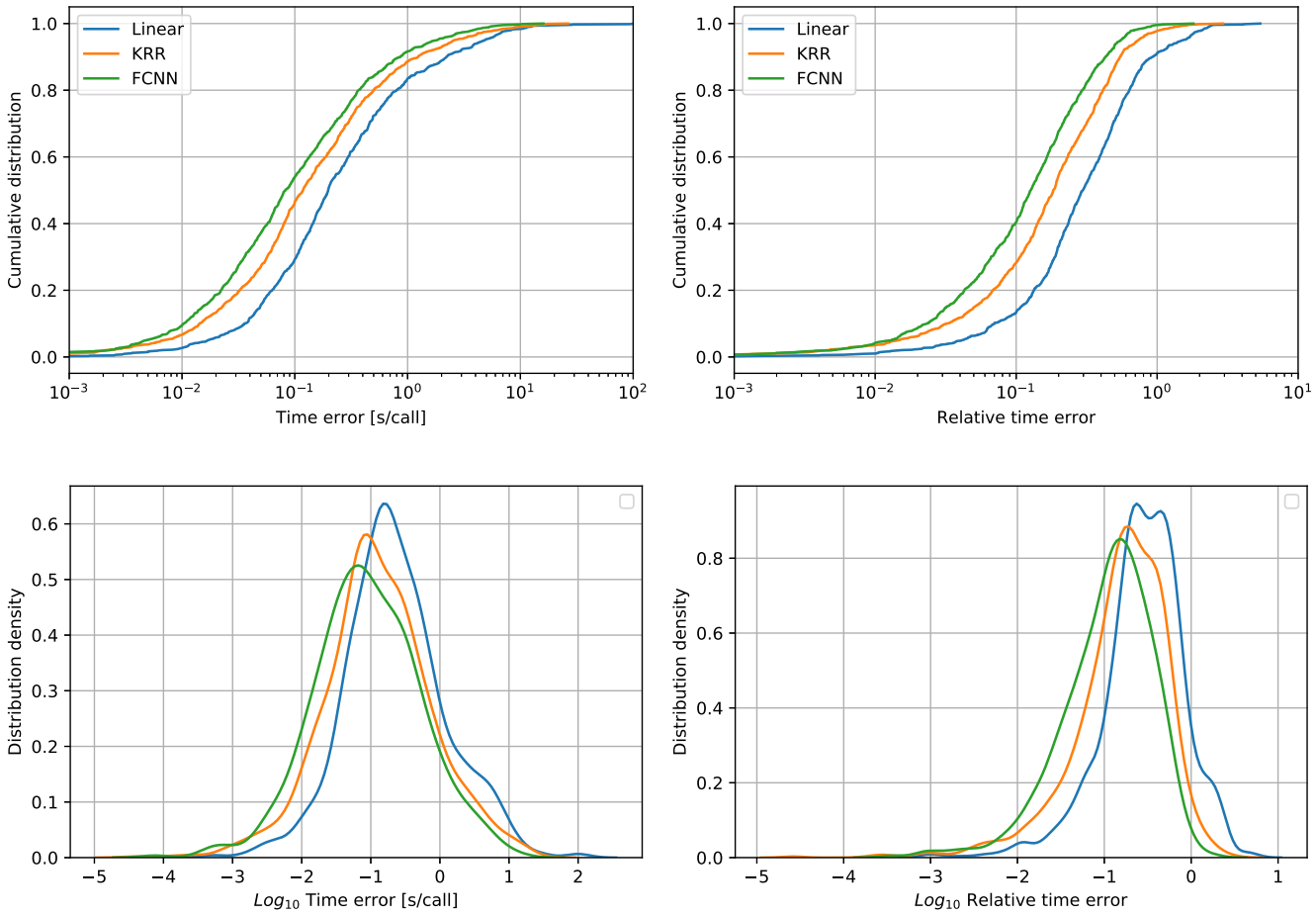


Figure 3: Distribution of the timing errors (absolute and relative), both in terms of cumulative distribution and distribution density. Models obtained separately for each machine.

FLOP/s, memory bandwidth, network bandwidth, I/O bandwidth and possibly cache sizes against repeated measurements of the time to solution obtained as the above parameters are (sometime artificially) varied.

A second approach is based on the identification of the most time consuming kernels constituting the application and a subsequent (possibly approximate) estimation of the dependence of these algorithms on the most relevant compute node characteristics [26].

We followed this second path, by dividing the most relevant algorithms of QE into memory bound, compute bound, network bandwidth/latency bound problems, and identifying for each of them a single parameter that allows to reproduce its performance given the underlying hardware characteristics. As an example, let us discuss the case of xGEMM (with x being either D or Z for the double-precision real or complex general matrix multiplication subroutines respectively). For this compute bound operation a good indicator is the number of FLOP/s performed by the CPU under consideration. However, while for large matrix size xGEMM implementations may almost reach theoretical peak performance, there are conditions dependant on the matrix size or shape that may lead to sizable deviations from the peak value. For this reason, we do not fix the value of the hardware parameters of the model to their theoretical values. These may instead be estimated with simple mini-benchmark targeting a specific kernel on the machine under consideration or by least square fitting against a set of results, as explained in section 4.

As already mentioned in section 2.2, a few well defined operations dominate the walltime:

(i) Linear algebra and Fourier transform operations are used to move from/to real and reciprocal space during the evaluation of the components of the Hamiltonian, inside the iterative diagonalization procedure, and to apply projection operators. For the particular case of PWscf, the input parameters define the size and the number of calls of this kind of operations, that generally appear in the same fashion at each iteration. Therefore, since the number of floating point operations performed by linear algebra subroutines and 1D FFT calls is easily evaluated, given a set of input parameters

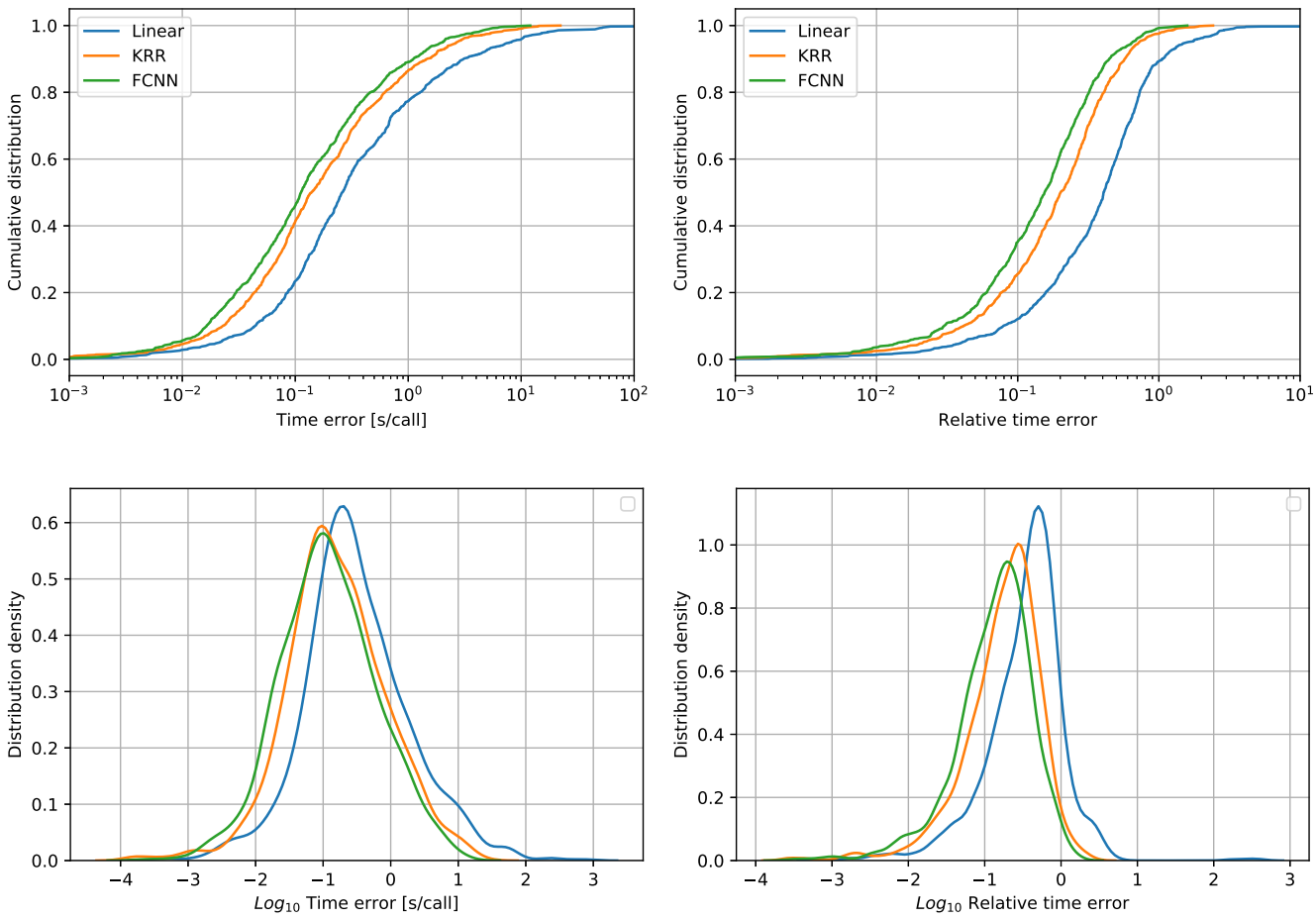


Figure 4: Same as Fig. 3, featuring models jointly obtained using data from all machines.

and the computational power of the underlying hardware, it is straightforward to calculate the time taken by these functions.

(ii) The eigenvalue problem solved as a part of the Davidson method (the only one we actually considered) mainly consists of the extraction of a subset k of the n eigenvalues in a generalized eigenvalue problem. When parallel diagonalization algorithms are not used (a rather common situation in our case since the dimension of the eigenvalue problem is of the order of few hundreds of elements in our data set) this is done through the `xHEGVX` LAPACK subroutine [27]. The computational intensity of this task as implemented in the MKL subroutine used as a benchmark, to the best of our knowledge, is still of the order of $O(k * n^2)$.

(iii) The QE's custom distributed FFT algorithm, a separate domain specific library called FFTXlib [28], provides optimized workloads and communication patterns for the Fourier transform of 3D distributed data defined only inside spheres in reciprocal space. This is one of the most used kernels in the PWscf code and thus requires special attention. Its functionality has been modeled by considering not only the time taken by the standard 1D FFT calls (already discussed above), but also memory access patterns and communication time in alltoall MPI calls. All these operations enter the scattering of the data owned by different MPI processes. This was modeled by considering, as a function of the problem size, the time taken by the initialization of auxiliary variables that are zeroed out at each call, the time taken by intra-process data scattering and preparation for the all-to-all communication, and the time for the all-to-all MPI communication .

The final time to solution estimated with this approach is given by the sum of the individual components, repeated as per input parameters that in turn define the various subroutines' execution trees and loops. The number of iterations performed by the Davidson iterative diagonalization approach represents the only unknown in this procedure. This number is hard to predict and it has been approximated with an empirical trend of the form $NC_{i+1} = (NC_i)^k - 1$ where NC is the number of unconverged eigenvalues at each iteration i and k was set to 0.7.

In conclusion, 8 parameters define the model. For the basic linear algebra operations we parametrize the performance of the implementations of the DGEMM and the ZGEMM subroutines. These two values should match but the size of the matrices considered

may lead to small differences. Diagonalization is described by two effective parameters that account for the performance of ZHEGV and ZHEGVX lapack subroutines. Finally, the FFTXlib is described by four parameters that describe memory bandwidth as obtained from a setting operation, memory latency that enters data scattering, 1D FFT performance, and communication time in all-to-all MPI calls.

Of course, a significant part of the wall-time taken by the code is not considered with this approach. However, if one assumes that the set of operations considered, for which the number of calls is fixed by the input parameters, is representative of the remaining part of the application that evades the model, one may extrapolate the time for each iteration by multiplying the result with an effective factor. When setting the values of the various parameters through ad-hoc micro-benchmarks, we have found that the real time per iteration is consistently underestimated by about 15%.

4 RESULTS

The first step in our experiments has been to clean and normalize the data. In particular, in order to clean the data and to remove outliers, we have excluded all simulations with an average time per iteration greater than 50s, which would be unreasonable for the kind of systems considered in this study. Then, the normalization has been performed according to the procedure described in Sec. 3.

Figs. 3 - 4 report the timing errors on the test set using all ML algorithms, either training a separate model for each machine or training a single joint model, respectively. We note that, as expected, the FCNN is always performing better than the KRR, which in turn is always better than the LR. Moreover, we note relatively small differences when using FCNN between the separate and the joint models, with the former being the most accurate.

Focusing on the relative error, in our best scenario we can achieve for 99% of the points an error below 100%, with a distribution peak and median at about 10% relative error. Looking also at the density distributions, the FCNN is consistently highly monomodal, while for the other algorithms this is approximately true only in the joint training case. Moreover, inspecting in detail the cases where the relative error is above 100%, it turns out that these are generally the cases where the absolute timer is very small, much below 1 s/call, so these are also the quickest simulations whose impact on the total batch is generally small. On the other hand, the points where the absolute error is greatest are also the ones where the relative error is small, so also these cases are not so critical.

When choosing an architecture and an optimization procedure for a FCNN, there are multiple degrees of freedom and the criteria for choosing them are based only on empirical rules [8]. For these reasons, it is important to evaluate the loss function behaviour during the training phase, and the 10-fold cross-validation is particular useful for this purpose. Fig. 5 shows then the behaviour of the loss function for the FCNN as a function of the training epoch. The blue and red lines are the average losses on the training and validation sets, respectively, while the vertical bars are the losses standard deviations over the 10 training/validation sets combination (that arise from the 10-fold cross-validation). The green dots are, instead, the losses on the test set. The fact that the test and validation losses are very similar and that they are not far from the training loss, and not decreasing anymore with the training epoch beyond a certain

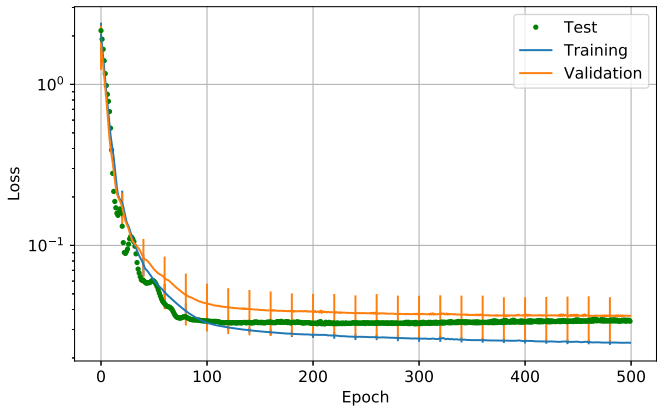


Figure 5: Example of the behaviour of the loss function for the FCNN as a function of the training epoch estimated with 10-fold cross-validation.

threshold, assures that the algorithm has converged with neither overfitting nor underfitting, which are common problems of ML algorithms [8].

Finally, we compared the results of the ML models to the analytical model described in Sec. 3.2. The derivation of the latter assumed the set of 8 parameters described in Sec.3.2 to be known. In our case however such parameters were not available from the dataset, so we had to estimate them. The estimation was done via a least-squares procedure, minimizing the prediction error of the model on a training set which is exactly the one used for training the ML models. With the estimated parameters, the model is then applied on the test set, and Fig. 6 shows the comparison of the absolute and relative error distributions with respect to the ML models. In this case we have used only the ML models separate by machine and not the joint one. Note also that the results are slightly different than the ones from Fig. 3, since in this case we had to exclude all simulations with enabled Conjugate Gradient calculation, a case not supported by the analytical model. Also in this case the FCNN shows the best performance, being the only machine learning based approach to overtake the analytical model both in terms of absolute and relative errors. More in detail, the FCNN improves the prediction error by up to 5% with respect to the model.

The main reason for the superior performance of the FCNN is a better description of the iterative diagonalization approach, that is modeled with a phenomenological convergence trend, and the more comprehensive description of the code that the FCNN approach may provide. The analytical model only considers a limited number of time consuming kernels. The missing components are generally small, but of the same order of the improvement provided by the neural network approach, thus probably justifying its improved results.

It is noteworthy that Deep Learning models have obtained such a good performance despite the relatively small dimensions of the training dataset. We therefore expect to dramatically increase their performance and applicability by drastically enlarging the datasets

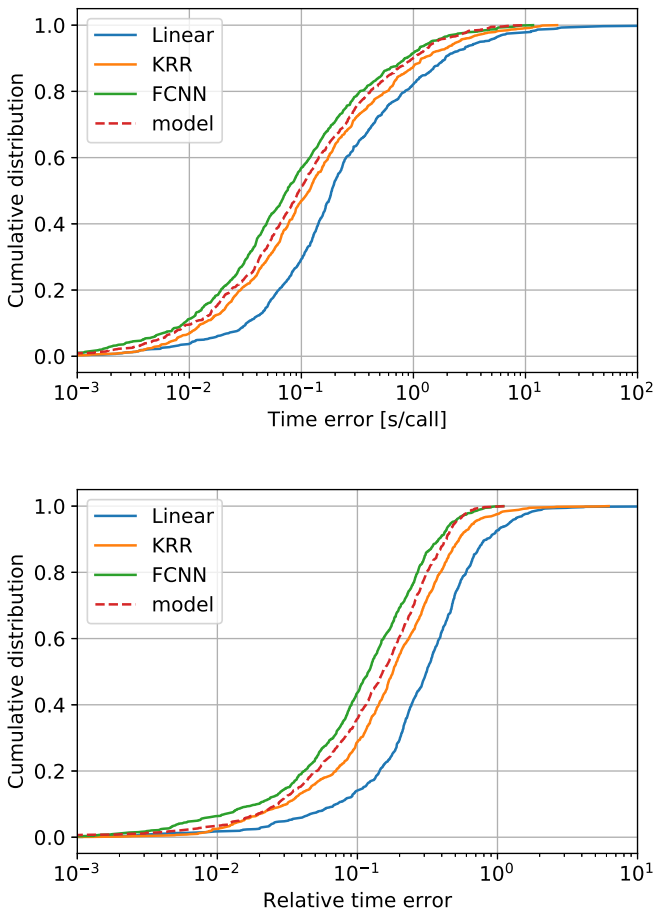


Figure 6: Distribution of the timing errors (absolute and relative), compared to the analytical model.

both in terms of number of simulations and in terms of diversity of codes executions and versions.

From this study we can then conclude that Deep Learning models are a very promising technology capable of outperforming analytical models with domain expertise for performance estimation on complex codes in an HPC environment.

5 CONCLUSIONS

In this work we have proven that out-of-the-box Machine Learning models can predict surprisingly accurately the time-to-solution of complex scientific applications, like QuantumESPRESSO. In particular, we have revealed that Deep Learning algorithms, in our case a Fully Connected Neural Network, achieve the best performance, which corresponds to a relative error lower than 100% for 99% of the simulations, with a distribution peak and median at about 10% relative error.

On the other hand we have also shown that a full-custom semi-analytical model specifically tailored to solve this task, whose few free parameters have been optimized on this dataset, exhibits a lower performance than that of the Neural Network.

It should be noted, however, that all models described in this paper have been trained using very similar versions of one scientific application, all in the same major release cycle.. Once a new major version of the code is released, it is therefore highly probably that the models will need some retraining to retain their accuracy. The investigation on how to generalise the models to multiple codes and multiple versions of the same code, together with a thorough cross-validation of the architecture and hyperparameters, will be the subject of our future work.

Our work paves the way to the development of very accurate models for predicting in advance the properties of scientific applications. It can then be extended to predict not only the time per iteration, but also the number of iterations or other properties of the applications execution. It serves as a valuable tool for an accurate scheduling of the applications, but it can also be used to provide an a-posteriori evidence to the user of an issue on the execution when the predicted execution time is very different from the actual one.

6 ACKNOWLEDGEMENTS

This work was supported by the EU project H2020-INFRAEDI-2018-1 MaX "Materials Design at the Exascale. European Centre of Excellence in materials modelling, simulations, and design" (Grant No. 824143), by the European H2020 FET project OPRECOMP (g.a. 732631) and by the CINECA research grant on Energy-Efficient HPC systems.

REFERENCES

[1] J. Hennessy, D. Patterson, and K. Asanović, *Computer Architecture: A Quantitative Approach*, ser. Computer Architecture: A Quantitative Approach. Elsevier Science, 2012. [Online]. Available: https://books.google.it/books?id=v3-1hVwHnHwC

[2] T. Miu and P. Missier, "Predicting the execution time of workflow activities based on their input features," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 64–72.

[3] F. Nadeem, D. Alghazzawi, A. Mashat, K. Fakeeh, A. Almalaise, and H. Hagras, "Modeling and predicting execution time of scientific workflows in the grid using radial basis function neural network," *Cluster Computing*, vol. 20, no. 3, pp. 2805–2819, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10586-017-1018-x

[4] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz, "Wien2k: An augmented plane wave plus local orbitals program for calculating crystal properties (technical university of vienna, vienna, 2001)," *Google Scholar*.

[5] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *Journal of Network and Computer Applications*, vol. 82, pp. 93–113, 2017.

[6] S. Lee, J. S. Meredith, and J. S. Vetter, "Compass: A framework for automated performance modeling and prediction," in *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015, pp. 405–414.

[7] P. Giannozzi, S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. D. Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch, "Quantum espresso: a modular and open-source software project for quantum simulations of materials," *Journal of Physics: Condensed Matter*, vol. 21, no. 39, p. 395502, 2009. [Online]. Available: http://stacks.iop.org/0953-8984/21/i=39/a=395502

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] N. Mounet, M. Gibertini, P. Schwaller, D. Campi, A. Merkys, A. Marrazzo, T. Sohier, I. E. Castelli, A. Cepellotti, G. Pizzi *et al.*, "Two-dimensional materials from high-throughput computational exfoliation of experimentally known compounds," *Nature nanotechnology*, vol. 13, no. 3, p. 246, 2018.

[10] "MaterialCloud," https://www.materialscloud.org/, 2018, [Online; accessed 19-July-2018].

[11] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. B. Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. D. Corso, S. de Gironcoli, P. Delugas, R. A. D. Jr, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. KÃijÃğÃijkbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. O. de-la Roza, L. Paulatto, S. PoncÃĪ, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, "Advanced capabilities for materials modelling with q uantum espresso," *Journal of Physics: Condensed Matter*, vol. 29, no. 46, p. 465901, 2017. [Online]. Available: http://stacks.iop.org/0953-8984/29/i=46/a=465901

[12] W. Kohn and L. J. Sham, "Self-consistent equations including exchange and correlation effects," *Phys. Rev.*, vol. 140, pp. A1133–A1138, Nov 1965. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.140.A1133

[13] D. Vanderbilt, "Soft self-consistent pseudopotentials in a generalized eigenvalue formalism," *Phys. Rev. B*, vol. 41, pp. 7892–7895, Apr 1990. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.41.7892

[14] G. Kresse and D. Joubert, "From ultrasoft pseudopotentials to the projector augmented-wave method," *Phys. Rev. B*, vol. 59, pp. 1758–1775, Jan 1999. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.59.1758

[15] D. Cesarini, A. Bartolini, P. Bonfà, C. Cavazzoni, and L. Benini, "COUNTDOWN - three, two, one, low power! A Run-time Library for Energy Saving in MPI Communication Primitives," *arXiv e-prints*, p. arXiv:1806.07258, Jun. 2018.

[16] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos, "Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients," *Rev. Mod. Phys.*, vol. 64, pp. 1045–1097, Oct 1992. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.64.1045

[17] E. R. Davidson, "The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices," *Journal of Computational Physics*, vol. 17, no. 1, pp. 87 – 94, 1975. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0021999175900650

[18] G. Kresse and J. Furthmüller, "Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set," *Phys. Rev. B*, vol. 54, pp. 11 169–11 186, Oct 1996. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.54.11169

[19] M. P. Teter, M. C. Payne, and D. C. Allan, "Solution of schrödinger's equation for large systems," *Phys. Rev. B*, vol. 40, pp. 12 255–12 263, Dec 1989. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.40.12255

[20] R. Martin, R. Martin, and C. U. Press, *Electronic Structure: Basic Theory and Practical Methods*. Cambridge University Press, 2004. [Online]. Available: https://books.google.it/books?id=dmRTFLpSGNsC

[21] N. Mounet, M. Gibertini, P. Schwaller, D. Campi, A. Merkys, A. Marrazzo, T. Sohier, I. E. Castelli, A. Cepellotti, G. Pizzi, and N. Marzari, "Two-dimensional materials from high-throughput computational exfoliation of experimentally known compounds," *Nature Nanotechnology*, vol. 13, no. 3, pp. 246–252, 2018. [Online]. Available: https://doi.org/10.1038/s41565-017-0035-5

[22] "Crystallography Open Database," https://www.crystallography.net, 2019, [Online; accessed 1-Jan-2019].

[23] H. Trevor, T. Robert, and F. JH, "The elements of statistical learning: data mining, inference, and prediction," 2009.

[24] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed <today>]. [Online]. Available: http://www.scipy.org/

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[26] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama, "Warpp: A toolkit for simulating high-performance parallel scientific codes," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 19:1–19:10. [Online]. Available: https://doi.org/10.4108/ICST.SIMUTOOLS2009.5753

[27] E. Anderson, Z. Bai, C. Bischof, L. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Society for Industrial and Applied Mathematics, 1999. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898719604

[28] "FFTXlib repository," https://github.com/QEF/q-e/tree/master/FFTXlib, 2017, [Online; accessed 19-July-2018].