The Journal of Chemical Physics ARTICLE pubs.aip.org/aip/jcp

Gaussian approximation potentials: Theory, software implementation and application examples

Cite as: J. Chem. Phys. 159, 174108 (2023); doi: 10.1063/5.0160898
Submitted: 6 June 2023 • Accepted: 12 September 2023 •
Published Online: 6 November 2023

Sascha Klawohn,[1] James P. Darby,[1] James R. Kermode,[1] Gábor Csányi,[2] Miguel A. Caro,[3] and Albert P. Bartók[4,a)]

AFFILIATIONS

[1] Warwick Centre for Predictive Modelling, School of Engineering, University of Warwick, Coventry CV4 7AL, United Kingdom
[2] Engineering Laboratory, University of Cambridge, Cambridge CB2 1PZ, United Kingdom
[3] Department of Chemistry and Materials Science, Aalto University, 02150 Espoo, Finland
[4] Department of Physics, University of Warwick, Coventry CV4 7AL, United Kingdom and Warwick Centre for Predictive Modelling, School of Engineering, University of Warwick, Coventry CV4 7AL, United Kingdom

Note: This paper is part of the JCP Special Topic on Software for Atomistic Machine Learning.
[a)] **Author to whom correspondence should be addressed:** apbartok@gmail.com

ABSTRACT

Gaussian Approximation Potentials (GAPs) are a class of Machine Learned Interatomic Potentials routinely used to model materials and molecular systems on the atomic scale. The software implementation provides the means for both fitting models using *ab initio* data and using the resulting potentials in atomic simulations. Details of the GAP theory, algorithms and software are presented, together with detailed usage examples to help new and existing users. We review some recent developments to the GAP framework, including Message Passing Interface parallelisation of the fitting code enabling its use on thousands of central processing unit cores and compression of descriptors to eliminate the poor scaling with the number of different chemical elements.

I. INTRODUCTION

Machine Learned Interatomic Potentials (MLIPs) have revolutionised atomic simulations by offering predictive and computationally inexpensive force fields.[1–4] While their implementations differ, these models approximate the Potential Energy Surface (PES) of atomic systems based on a database of atomic configurations with corresponding high-accuracy properties calculated with *ab initio* quantum mechanical methods.

Among the numerous related methods and their software implementations,[1,2,5–9] the Gaussian Approximation Potential (GAP) approach follows a Bayesian approach, which allows the formulation of prior knowledge about the atomic system and interactions as hyperparameters, as well as uncertainty estimation. A further advantage of using Gaussian Process Regression (GPR) is that fitting the model is a convex optimisation problem[10] equivalent to the solution of a linear system, therefore many problems associated with minimising the loss function of neural networks[11] do not occur.

The GAP framework, originally proposed by Bartók *et al.*,[3] uses GPR to infer local atomic properties via a set of descriptors that map Cartesian atomic coordinates to invariant representations. GAP models have been used successfully to model silicon,[3,12] carbon,[13] tungsten,[14] phosphorus,[15] water,[16] iron,[17] gold and platinum,[18,19] hafnia[20] and gallia,[21] among others.

Similarly to other MLIP frameworks, the GAP package can utilise reference atomic databases produced with arbitrary *ab initio* methods and software packages. Total energies and derivative quantities (forces and stresses) are used to fit the PES, although models for local atomic properties, such as nuclear magnetic resonance (NMR) shieldings or Hirshfeld volumes may also be generated using our software. Recently, GAP was used to accelerate *ab initio* molecu-

lar dynamics simulations within the CASTEP[22] package, utilising an adaptive scheme that produces an evolving and improving GAP model during the dynamics.[23]

In this paper, we present the current status of the GAP framework, discussing the particular adaptation of the sparse GPR that enables a performant implementation suitable to fit energetic properties of atomic systems. GAP, as implemented within the Quantum and Interatomic Potentials (QUIP)[24] software package, is introduced, emphasising the flexibility and extensibility of the code. We also discuss recent developments, such as parallelisation and compression of descriptors, making connections between the theory and practical usage. Finally, we present some examples, which are intended to illustrate a selection of features enabled by the GAP package.

By documenting implementation details for the available options, this paper is not only intended for practitioners fitting GAP models, but also for those developing other MLIP frameworks.

II. THEORY

The GAP framework utilises sparse GPR which is customised to fit PESs as well as local properties of atomic systems. A detailed introduction to GPR can be found elsewhere[10,25] and background on the GAP framework is presented in the review article by Deringer *et al.*[26] Here we revise only the formulae necessary for discussing the software implementation.

A central assumption in fitting the PES of atomic systems is that the total quantum mechanical energy may be decomposed into local contributions ε which depend on descriptors $\mathbf{x}$:

$$E = \sum_{d}^{\text{descriptors}} \sum_{i=1}^{N_d} \varepsilon_d(\mathbf{x}_i), \tag{1}$$

where N_d is the number of descriptors of type d. Descriptors may be the arguments to two-body energy terms, based on the interatomic distance, optionally augmented by the symmetrised atomic coordinations, or three-body terms, based on the bond angle and the symmetrised bond distances, optionally including the coordination of the central atom. The greatest advantage of the non-linear regression techniques enabled by machine learning methods is the ability to parameterise the highly complex many-body energy terms. The descriptors forming the arguments to these functions may be n-body terms, based on the interatomic distances within a cluster of n atoms, or flexible many-body terms, based on the Smooth Overlap of Atomic Positions (SOAP)[27] descriptor, the bispectrum[3,28] or the Behler–Parrinello symmetry functions.[1] Finally, GAP implements customised descriptors, to represent molecules, dimers and trimers.

In GAP, each energy term ε_d is written as an independent sparse Gaussian Process, in the form

$$\varepsilon_d(\mathbf{x}) = \sum_{m=1}^{M_d} c_m k_d(\mathbf{x}, \mathbf{x}_m), \tag{2}$$

where M_d is the number of sparse or representative points of descriptor d, k_d is the kernel, covariance or similarity function and c_m are the fitting coefficients.

The coefficients in (2) are fitted using a database of atomic configurations, where corresponding total energies and derivative quantities, such as forces and virial stresses, have been determined using *ab initio* quantum mechanical calculations. The target properties, denoted by $\mathbf{y}$, of the fitting procedure are therefore the sum of local energy contributions in the form of total energies, or the sum of the partial derivatives of local energy terms in the form of forces or virial stresses. The differentiation operator with respect to a Cartesian coordinate $r_{i\alpha}$ is propagated through the kernel function, resulting in partial derivatives

$$\frac{\partial \varepsilon}{\partial r_{i\alpha}} = \sum_{m=1}^{M_d} c_m \nabla_{\mathbf{x}} k(\mathbf{x}, \mathbf{x}_m) \frac{\partial \mathbf{x}}{\partial r_{i\alpha}}. \tag{3}$$

The sparse GPR adapted to our case[26] becomes

$$\mathbf{c} = \left[\mathbf{K}_{MM} + (\hat{L}\mathbf{K}_{NM})^{\top}\boldsymbol{\Sigma}^{-1}\hat{L}\mathbf{K}_{NM}\right]^{-1}(\hat{L}\mathbf{K}_{NM})^{\top}\boldsymbol{\Sigma}^{-1}\mathbf{y}. \tag{4}$$

The kernel or covariance matrices $\mathbf{K}_{MM}$ and $\mathbf{K}_{NM}$ contain the function values $k(\mathbf{x}_m, \mathbf{x}_{m'})$ and $k(\mathbf{x}_n, \mathbf{x}_m)$ respectively, where m and m' denote sparse points and n denote descriptors of the database configurations. In case of $\mathbf{K}_{NM}$, kernel functions may be the derivative values, $-\nabla_{\mathbf{x}} k(\mathbf{x}_n, \mathbf{x}_m)\frac{\partial \mathbf{x}_n}{\partial r_{i\alpha}}$, if the corresponding target observation in $\mathbf{y}$ is a force quantity. The diagonal $\boldsymbol{\Sigma}$ matrix contains the regularisation strength parameters (σ_{energy}, σ_{force} and σ_{virial}, which may be specified individually), encoding the prior assumption regarding the accuracy of target values. Finally, the operator $\hat{L}$ is a shorthand for the summation that accumulates the local terms composing each target value in $\mathbf{y}$.

Foster *et al.* have shown[29] that solving Eq. (4) directly can lead to numerically unstable results, in which uncertainties in the input lead to disproportionate errors in the output. Instead, we first define

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{\Sigma}^{-1/2}\hat{L}\mathbf{K}_{NM} \\ \mathbf{U}_{MM} \end{bmatrix}, \tag{5}$$

where the Cholesky decomposition of $\mathbf{K}_{MM}$ results in the upper triangular matrix $\mathbf{U}_{MM}$ such that $\mathbf{K}_{MM} = \mathbf{U}_{MM}^{T}\mathbf{U}_{MM}$. While $\mathbf{K}_{MM}$ is positive semidefinite, depending on the database configurations and descriptor types, the sparse points may be highly correlated, leading to an ill-conditioned $\mathbf{K}_{MM}$ matrix, preventing the Cholesky decomposition we use to obtain $\mathbf{U}_{MM}$. To regularise the sparse covariance matrix $\mathbf{K}_{MM}$, we add a small constant to the diagonal, informally called the *jitter*, which is typically 8–10 orders of magnitude less than the elements of $\mathbf{K}_{MM}$. The *jitter* has a similar effect on the resulting sparse model as the noise hyperparameter in a full GPR. As both the aleatoric and epistemic uncertainty is controlled by $\boldsymbol{\Sigma}$, the error in the local energy term introduced by the use of *jitter* is a small broadening, which is expected to be on the order of the square root of the *jitter*.

Padding the vector of target properties $\mathbf{y}$ by an M-long vector of zeros,

$$\mathbf{b} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}, \tag{6}$$

we rewrite Eq. (4) as the solution of the least-squares problem

$$\min_{\mathbf{c}} (\mathbf{A}\mathbf{c} - \mathbf{b})^{T}(\mathbf{A}\mathbf{c} - \mathbf{b}), \tag{7}$$

leading to the solution in the form of

$$\mathbf{c} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{8}$$

A numerically stable solution can be obtained by first carrying out a QR factorisation of $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is orthogonal, namely, it is formed by *orthonormal* column vectors, while $\mathbf{R}$ is an upper triangular matrix. Substituting the factorised form of $\mathbf{A}$ into Eq. (8) results in

$$\mathbf{c} = (\mathbf{R}^T\mathbf{Q}^T\mathbf{QR})^{-1}\mathbf{R}^T\mathbf{Q}^T\mathbf{b} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{b}, \tag{9}$$

as $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.

III. IMPLEMENTATION

We develop and maintain a collection of software tools called QUIP[30] to carry out molecular dynamics simulations. Part of this suite is an implementation of GAP, including the `gap_fit` program, implementing the sparse GPR. The majority of QUIP is written in modern Fortran, utilising many object-oriented features, although not fully exploiting the most recent Fortran standards due to lack of compiler support at the time of the original development of the code, which started in 2005. QUIP features a Python interface, `quippy`,[31] allowing access to various functionalities and casting all atomic potentials as Atomic Simulation Environment (ASE) calculators.[32] There also exist generic C and LAMMPS-specific C++ interfaces, that allow GAP models to be used in external simulation packages. The source code can be found on GitHub.[24]

A. The GAP submodule

Similarly to other MLIPs, the main components of GAP are the calculation of descriptors, mapping Cartesian coordinates into invariant representations, and a regression method, in this case GPR.

1. Representations

The `descriptors.f95` file implements the mapping from the Cartesian coordinates of the atoms to invariant representations. In the package we provide a number of descriptors, but it is straightforward to implement new ones. While the user fitting GAP models does not have to interact with the source code, in the following we give an overview of what is necessary to implement or adapt a descriptor.

A set of standardised interfaces are used for each descriptor, which are overloaded, therefore adding new descriptors is straightforward and does not require any further modifications elsewhere. The `initialise` interface interprets the parameters of the descriptor, which are provided by the user as key-value pairs, and stores these in a `descriptor` object. The query function `cutoff` returns the spatial cutoff of a descriptor, whereas `finalise` resets the descriptor object and deallocates all storage. The `descriptor_sizes` function takes an `Atoms` object and determines how many descriptors and partial derivatives will be calculated based on the cutoff and the connectivity of the particles. Finally, the `calc` function returns descriptors calculated based on an `Atoms` and a `descriptor` object, and optionally, their partial derivatives with respect to atomic coordinates in a generic container object. All of this functionality is exposed in `quippy`, ensuring interoperability with ASE.

2. Regression

GPR is implemented in the file `gp_predict.f95`, with some fitting-specific features in `gap_fit_module.f95` and sparse point selection in `clustering.f95`. For a pair of descriptors $\mathbf{x}$ and $\mathbf{x}'$ of dimensions D, whose distance r is defined as

$$r = \sqrt{\sum_{i=1}^{D} \frac{(x_i - x_i')^2}{2\theta_i^2}}, \tag{10}$$

we have implemented the squared exponential kernel

$$k_{\mathrm{SE}}(\mathbf{x}, \mathbf{x}') \equiv k_{\mathrm{SE}}(r) = \exp(-r^2), \tag{11}$$

and the piece-wise polynomial kernel with compact support

$$k_{\mathrm{PP}}(\mathbf{x}, \mathbf{x}') \equiv k_{\mathrm{PP}}(r) = (1 - |\mathbf{x} - \mathbf{x}'|)^{j+1}[(j+1)|\mathbf{x} - \mathbf{x}'| + 1], \tag{12}$$

where $j = \lfloor \frac{D}{2} \rfloor + 2$. The SOAP descriptors should be used with the dot-product or, more generally, polynomial kernel

$$k_{\mathrm{DP}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^{\zeta}. \tag{13}$$

The hyperparameters, such as $\boldsymbol{\theta}$ or ζ, and the coefficients $\mathbf{c}$ are stored in a Fortran object which is used by the function `gp_predict` to evaluate (2) as well as the partial derivatives with respect to descriptor components and variances predicted from GPR.

During the training procedure, all descriptors $\mathbf{x}$ and their partial derivatives are calculated and stored. Pointers are used to denote which descriptors and derivatives contribute to target properties, thereby avoiding the need to store repetitive information. The kernel matrices $\mathbf{K}_{NM}$ used in the fitting procedure in (4) are not calculated explicitly, only the accumulated terms in $\hat{L}\mathbf{K}_{NM}$ corresponding to quantum mechanical observable quantities.

B. GPR using `gap_fit`

Finding the coefficients used in GAP models can be accomplished using the `gap_fit` command line program, where parameters are set as arguments or a configuration file. The input to the fitting procedure consists of the fitting data, model definitions, and additional options. The database of atomic configurations, together with the quantum mechanical properties are read in from an extended XYZ[33] file. The extended XYZ contains information of the lattice, atomic numbers and Cartesian coordinates, and may provide the total energy, forces and virial stress, or any combination of these for each individual configuration.

Each individual atomic configuration may optionally have a type specification, given within the extended XYZ file using the `config_type` keyword. The configuration type may be used for fine-grain control, such as selecting a specific number of sparse points from those configurations.

The particulars of the model are provided using the `gap` argument as a form of colon-separated list of descriptor definitions. These include hyperparameters, such as the spatial cutoff or the desired number of sparse points per descriptor. Other hyperparameters, such as the regularisation, can be provided either in the command line, or specifically for each individual frame within the extended XYZ file, allowing fine-grained control and the use of inhomogeneous accuracy of target quantities. Additional options can be

used to adjust the processing, tune technical parameters, or enable additional features like more verbose output. Details about the currently available arguments can be found later in Sec. V, or running `gap_fit-help` for up-to-date information.

1. Program structure

After initialisation and reading of the input, the extended XYZ frames, where each frame consists of an atomic structure, are parsed for the number of target properties and descriptors. Storage for the descriptor arrays are allocated accordingly, and descriptors are calculated during a second pass over the atomic configurations.

For each set of N descriptors, $M \ll N$ points are chosen as a representative, or sparse, set. Options include random selection, k-means clustering, a uniform grid spanning the range of descriptors and CUR decomposition.[34] It is also possible to provide the sparse points via text files. As the sparse points need to form a linearly independent covariance matrix, duplicates within a given tolerance are removed and only considered once. This may result in fewer sparse points used than specified by the user, particularly if the atomic environments in the database are highly correlated.

With the specified sparse points, the covariance matrices $\mathbf{K}_{MN}$ and $\mathbf{K}_{MM}$ are calculated, and matrix $\mathbf{A}$ constructed. The coefficients are determined via QR decomposition using (Sca)LAPACK[35,36] routines.

The memory requirement for the `gap_fit` program depends on the atomic structures, the number of target properties and sparse points, and descriptor definitions. In particular, the two main data components with significant memory requirements are the descriptors and their partial derivatives and the kernel matrix $\hat{L}\mathbf{K}_{NM}$. The memory associated with storing descriptor derivatives scales linearly with the number of atomic environments and the dimensionality of the descriptor, as well as the number of neighbours within the spatial cutoff. Efforts directed at developing compact descriptors, using compression techniques, therefore significantly reduce the memory requirements of the fitting procedure, as well as the computational complexity of evaluating the descriptors. The size of the kernel matrix scales linearly with the number of target values and the total number of sparse points.

Recently, we have implemented domain decomposition in `gap_fit` that aims to utilise massively parallel computer architectures.[37] The implementation relies on Message Passing Interface (MPI) and is illustrated on Fig. 1.

After determining the number of atoms and, consequently, the number of target data values in each configuration of the database, configurations are assigned to individual MPI tasks. Descriptors are computed locally, and the covariance matrix $\mathbf{K}_{MN}$ is constructed in a distributed fashion. This approach allows the memory requirements of the program to be distributed over many computational nodes, therefore larger databases can be easily fitted without the need of specialised, high-memory servers. The linear algebra step makes use of the ScaLAPACK library, carrying out the QR decomposition and subsequent back-substitution steps in parallel, thereby reducing the computational time. We demonstrate the benefits of the parallel fitting approach by studying two fitting problems that would require significant amounts of wall-time and memory using a single node. One of the training databases consists of the high-entropy alloy configurations by Byggmästar *et al.*,[38] and the other is a data set of silicon carbide configurations from Ref. 37. The dependence

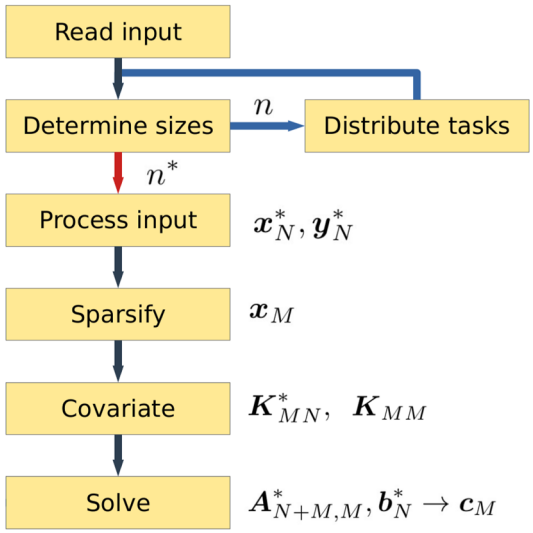

FIG. 1. MPI `gap_fit` workflow. If run in serial, the tasks are not distributed and n is used instead of n^*.

of computational speedup and total memory use is presented on Fig. 2, showing that we have achieved excellent parallel performance, and we can utilise the aggregate memory of multiple computational nodes, thereby largely eliminating any limitations. We note that the dependence of the memory usage on the number of cores is due to overheads associated with repeated data allocations that are private to a process. Given the typical amount of memory, on the order of hundred GBs, available on commodity computing nodes, this is not likely to be a significant barrier to large-scale parallel fitting calculations. For more information, we refer the user to our prior work,[37] where we explored hybrid OpenMP-MPI parallel strategies that optimise overall memory use and runtime.

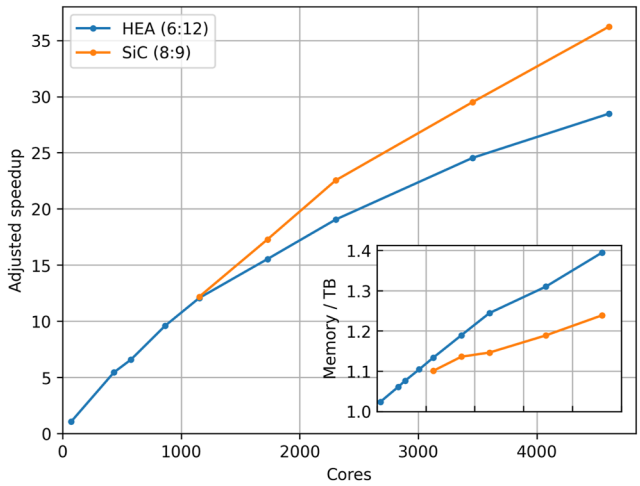

FIG. 2. Speedup (main panel) and memory (inset) requirements for fitting a high-entropy alloy (HEA) and silicon carbide (SiC) training set using a varying number of nodes.

2. Sparse point selection

The program `gap_fit` implements several methods for the sparse point selection, controlled by the `sparse_method` command line argument. Within the definition of each descriptor, the `n_sparse` argument controls the number of sparse points. Alternatively, `config_type_n_sparse` allows the user to specify a given number of sparse points from labelled configurations, to ensure adequate representation. Given `none`, no sparsification is applied, apart from the removal of duplicate descriptors, and all points are selected. The points may be chosen directly with either the `file` or `index_file` option, in conjunction with the `sparse_file` argument to specify the filename containing the descriptors or the indices of descriptors. The indices are 1-based and refer to descriptors as calculated in the database file.

Recommended choices are `uniform` for a `distance_2b` descriptor and `cur_points` for a `soap` descriptor. For completeness, we list all currently implemented options in the Appendix.

C. Descriptors

The choice of invariant representation of atomic environments has a profound effect on the quality of the resulting interatomic potential. QUIP, being a test bed for methodological developments of MLIPs, implements numerous descriptors, of which some frequently used ones are presented in this section.

In the `gap_fit` program, descriptors are specified in the `gap` command line argument, using the syntax `gap={descriptor1 key=value ...: descriptor2 key=value ...}`. The descriptor definitions are, by default, treated as templates by `gap_fit`, and after parsing the database configurations, each descriptor is expanded with element (chemical species) information. If `add_species=F` is added to the descriptor definition, the descriptor is not modified in this step.

Most descriptors implement the `cutoff` keyword, specifying the spatial cutoff within atomic connectivities are considered. The `cutoff_transition_width` keyword provides a smooth transition ensuring a numerically well behaved characteristic when the descriptor is used to build an interatomic potential.

1. Descriptors based on interatomic distances

Based on the idea of the cluster expansion of the total energy

$$E = \sum_i E_i^{(1)} + \sum_{i<j} E_{ij}^{(2)} + \sum_{i<j<k} E_{ijk}^{(3)} + \cdots + E^{(N)}, \tag{14}$$

the n-body terms may be fitted using GPR or other regression methods. The cluster of n atoms, representing the input variable of each term, is well defined by a monotonic function, which could be just the identity, of interatomic distances $\mathbf{r} = [r_{12}, r_{13}, \ldots]$.

In case of the two-body descriptor, GAP implements a polynomial transformation that generates a descriptor from the pair distance r in the form of $[r^{p_1}, r^{p_2}, \ldots]$, where $\{p_i\}_{i=1}^n$ are a set of exponents. When using this descriptor in conjunction with a dot-product kernel, the generalised form of a pair potential

$$V(r) = \sum_{i=1}^{n} c_i r^{p_i} \tag{15}$$

may be recovered.

However, for three- or higher body energy terms the descriptor formed as a list of interatomic distances is not invariant with respect to the permutation of indices of the same elements within the cluster, therefore cannot be directly used for regression. Permutational invariance is achieved by symmetrising, then normalising, the kernel:

$$k'(\mathbf{r}, \mathbf{r}') = \sum_{\hat{P}} k(\mathbf{r}, \hat{P}\mathbf{r}'), \tag{16}$$

$$k''(\mathbf{r}, \mathbf{r}') = \frac{k'(\mathbf{r}, \mathbf{r}')}{\sqrt{k'(\mathbf{r}, \mathbf{r}) k'(\mathbf{r}', \mathbf{r}')}}, \tag{17}$$

where $\hat{P}$ represents the permutation of the order of atoms, and k'' is used in the GPR.

In GAP, we implemented the `distance_nb` descriptor, where the body order is defined using the `order` keyword. The `compact_cluster` keyword specifies the topology of the cluster. If `compact_cluster=T`, each atom in an atomic configuration is considered as a central atom, and clusters are formed with its $n-1$ neighbours that are within the spatial cutoff. With the `compact_cluster=F`, all possible graphs where the graph edge has a distance less than the cutoff are formed, allowing, for example, linear chains.

The special cases corresponding to two- and three-body terms are implemented as `distance_2b` and `angle_3b`, respectively. In case of `angle_3b`, the trimer of atoms formed by the central atom i and its two neighbours j, k is represented by the invariant descriptors $[r_{ij} + r_{ik}, (r_{ij} - r_{ik})^2, r_{jk}]$.

As the interatomic distances are used in the kernel directly in these descriptor classes, the implementation of a smoothly varying spatial cutoff must be implemented in the kernel function. We modify the kernel by multiplying it by a cutoff function which smoothly interpolates between zero, where any of the interatomic distances are greater than the spatial cutoff, and one. The elementary cutoff function

$$f_{\mathrm{cut}}(r) = \begin{cases} 1 & \text{if } r < r_{\mathrm{cut}} - d \\ 0 & \text{if } r \geq r_{\mathrm{cut}} \\ \frac{1}{2}\left[\cos\left(\pi \frac{r - r_{\mathrm{cut}} + d}{d}\right) + 1\right] & \text{otherwise} \end{cases}, \tag{18}$$

where r_{cut} is the spatial cutoff and d is a transition width, is evaluated for each pairwise distance. The final cutoff function is obtained as a product of elementary cutoff functions, ensuring that each energy term vanishes smoothly.

GAP also allows further adjustment of the tail behaviour of the two-body descriptor, in order to approximate the polynomial decay of some interaction types. This is achieved by further multiplying the cutoff function by $\left(\frac{\mathrm{erf}(\alpha r)}{r}\right)^q$, where α is a range parameter and q is an exponent.

2. SOAP descriptors

The SOAP descriptors were proposed a decade ago[28] as invariant descriptors of atomic environments, and have been used successfully to develop interatomic potentials,[12,13,38,39] clustering[40] as well as other machine learning tasks, such as the ShiftML model used to predict Nuclear Magnetic Resonance chemical shifts.[41] For details

on the theory, the review by Musil *et al.*[27] may be consulted. Here we only repeat what is necessary to explain the implementation options.

To construct the SOAP descriptor, the atomic environment is first written as a neighbourhood density function, where atoms of element α are represented by Gaussians:

$$\rho^{\alpha}(\mathbf{r}) = \sum_i \delta_{\alpha z_i} \exp\left[\frac{-|\mathbf{r}-\mathbf{r}_i|^2}{2\sigma^2}\right] f_{\text{cut}}(|\mathbf{r}_i|), \tag{19}$$

where the sum over i includes all neighbouring atoms with position vector $\mathbf{r}_i$, z_i is the corresponding atomic number and σ is a length scale hyperparameter. The element density ρ^{α} is expanded in a basis set consisting of the products of orthonormal radial basis functions g_n and the spherical harmonics Y_{lm},

$$\rho^{\alpha}(\mathbf{r}) = \sum_{nlm} c^{\alpha}_{nlm} Y_{lm}(\hat{\mathbf{r}}) g_n(r), \tag{20}$$

resulting in the coefficients c^{α}_{nlm}. In the following, we often refer to grouping of certain indices in the basis expansions as *channels*, a term borrowed from signal processing. Therefore, the element indices α are the *element channels* and the radial basis indices n are the *radial channels*. The spherical harmonics indices l, together with the corresponding m indices, form the *angular channels*.

An invariant kernel or similarity function of two atomic environments is obtained by calculating the overlap of the densities, which has to be integrated over all rotations:

$$k(\rho,\rho') = \int \mathrm{d}\hat{R} \left| \sum_{\alpha} \int \mathrm{d}\mathbf{r} \rho^{\alpha}(\mathbf{r}) \rho'^{\alpha}(\hat{R}\mathbf{r}) \right|^{\nu}. \tag{21}$$

For the choice of $\nu = 2$, the SOAP kernel can be analytically evaluated in the form of a dot-product kernel

$$k(\rho,\rho') = \sum_{\alpha\beta} \sum_{nn'l} p^{\alpha\beta}_{nn'l} p'^{\alpha\beta}_{nn'l} = \mathbf{p}\cdot\mathbf{p}', \tag{22}$$

$$p^{\alpha\beta}_{nn'l} = \sum_m c^{\alpha *}_{nlm} c^{\beta}_{n'lm} = \mathbf{c}^{\alpha *}_{nl} \cdot \mathbf{c}^{\beta}_{n'l}, \tag{23}$$

due to the properties of the Wigner D-matrices representing the rotational transformation of the coefficients. To ensure that $\bar{k}(\rho,\rho) = 1$ for any ρ, we normalise the kernel as

$$\bar{k}(\rho,\rho') = \frac{k(\rho,\rho')}{\sqrt{k(\rho,\rho)}\sqrt{k(\rho',\rho')}}. \tag{24}$$

The dimension of $\mathbf{p}$ scales as $\mathcal{O}(n^2_{\max} l_{\max} S^2)$ where $n_{\max}$, $l_{\max}$ and S are the number of radial basis functions, spherical harmonics and elements, respectively. Apart from the original implementation of SOAP, a more efficient variant introduced by Caro[42] is available as `soap_turbo`. This descriptor is further described in Sec. III C 4. and a comparison with regular SOAP is provided in Sec. IV B.

Increasing $n_{\max}$ and $l_{\max}$ improves the resolution of the basis set expansion, and are therefore convergence parameters of the SOAP kernel. Optimal values are strongly dependent of the typical number of neighbours and the Gaussian broadening parameter σ. In many applications, the user has the choice to adjust $n_{\max}$ and $l_{\max}$ to achieve the desired balance of accuracy and computational speed. However, the length of $\mathbf{p}$ has a quadratic dependence on the number of elements, thereby the computational cost of both the components of $\mathbf{p}$ and the $k(\rho,\rho')$ as a dot product are impacted. Various strategies to reduce this scaling have been proposed, which are discussed below.

3. SOAP compression

The $\mathcal{O}(l_{\max} n^2_{\max} S^2)$ scaling of the number of descriptor components in SOAP is often limiting as it makes studying chemically diverse systems, such as multi-component alloys or proteins, very computationally demanding. A widely used approach[2,43–46] to reduce this scaling is to embed the elements (and optionally radial) channels into a fixed K-dimensional space as $\mathfrak{c}^k_{nlm} = \sum_{\alpha} w^k_{\alpha} c^{\alpha}_{nlm}$ (or $\mathfrak{c}^k_{lm} = \sum_{n\alpha} w^k_{n\alpha} c^{\alpha}_{nlm}$) and then form a compressed descriptor as

$$p^{kk'}_{nn'l} = \sum_m \mathfrak{c}^k_{nlm} \mathfrak{c}^{k'}_{n'lm}. \tag{25}$$

(or $p^{kk'}_{l} = \sum_m \mathfrak{c}^k_{lm} \mathfrak{c}^{k'}_{lm}$) which reduces the scaling to $\mathcal{O}(l_{\max} n^2_{\max} K^2)$ [or $\mathcal{O}(l_{\max} K^2)$]. To achieve good performance for $K < S$[47] the embedding weights are typically optimised during fitting and, following Willatt *et al.*, they are interpretable as encoding similarity between different elements via the alchemical kernel[43] $\kappa_{\alpha\beta} = \sum_k w^k_{\alpha} w^k_{\beta} = \mathbf{w}^{\alpha}\cdot\mathbf{w}^{\beta}$.

This idea was extended by Darby *et al.*,[48] where it was shown that it is sufficient to couple the embedding channels to themselves only, rather than taking a full tensor product across the embedded index k, thus making the scaling linear in K, rather than quadratic. Two flavours of these tensor-reduced descriptors were proposed. The first is motivated by considering fitting a linear model as

$$\varphi = \sum_{\alpha\beta nn'l} a^{l}_{(\alpha,n),(\beta,n')} p^{\alpha\beta}_{nn'l}, \tag{26}$$

where the a are the model coefficients and the element and radial indices have been grouped together. For each value of l the matrix of coefficients a^l can be approximated using symmetric eigen-decomposition as

$$a^{l}_{(\alpha n),(\beta n')} = \sum_{k=1}^{K} \lambda^l_k w^k_{(\alpha,n)} w^k_{(\beta,n')} \tag{27}$$

This decomposition is exact for $K = n_{\max} S$ with w the eigenvectors of a^l and is systematically improvable with random w. Substituting this approximation into Eq. (26) results in

$$\varphi = \sum_{\alpha\beta nn'l} \sum_k \lambda^l_k w^k_{(\alpha,n)} w^k_{(\beta,n')} p^{\alpha\beta}_{nn'l} \tag{28}$$

$$= \sum_{k,l} \lambda^l_k \left(\sum_{\alpha n} w^k_{(\alpha,n)} \mathbf{c}^{\alpha}_{nl} \right) \cdot \left(\sum_{\beta n'} w^k_{(\beta,n')} \mathbf{c}^{\beta}_{n'l} \right) \tag{29}$$

$$= \sum_{kl} \lambda^l_k \mathbf{c}^k_l \cdot \mathbf{c}^k_l = \sum_{kl} \lambda^l_k \tilde{p}^k_l, \tag{30}$$

where $\tilde{p}^k_l$ are the new features. As the approximation in Eq. (27) is systematic, any function that can be fit as a linear function of $p^{\alpha\beta}_{nn'l}$ can also be fit using a linear function of $\tilde{p}^k_l$.

An alternative and complementary view motivated by using random mixing weights $w^k_{(\alpha n)}$ is to "sketch" the power spectrum as

$$\hat{p}^k_l = \left(\sum_{\alpha n} w^k_{\alpha n} \boldsymbol{c}^\alpha_{nl}\right) \cdot \left(\sum_{\beta n'} u^k_{\beta n'} \boldsymbol{c}^\beta_{n'l}\right) \tag{31}$$

so that

$$\mathbb{E}\left[\hat{\mathbf{p}} \cdot \hat{\mathbf{p}}'\right] = \sum_{kl} \sum_{\substack{\alpha\beta nn' \\ \delta\gamma qq'}} \underbrace{\mathbb{E}\left[w^k_{\alpha n} u^k_{\beta n'} w^k_{\delta q} u^k_{\gamma q'}\right]}_{\sigma^4 \delta_{(\alpha,n),(\delta,q)} \delta_{(\beta,n'),(\gamma,q')}} \boldsymbol{c}^\alpha_{nl} \boldsymbol{c}^\beta_{n'l} \boldsymbol{c}'^\delta_{ql} \boldsymbol{c}'^\gamma_{q'l} \tag{32}$$

$$= \sigma^4 \sum_k \sum_{\alpha\beta nn'l} \left(\boldsymbol{c}^\alpha_{nl} \cdot \boldsymbol{c}^\beta_{n'l}\right)\left(\boldsymbol{c}'^\alpha_{nl} \cdot \boldsymbol{c}'^\beta_{n'l}\right) \tag{33}$$

$$= K\sigma^4 \mathbf{p} \cdot \mathbf{p}', \tag{34}$$

where we have used the fact that $\mathbb{E}[w^k_i w^k_j] = \sigma^2 \delta_{ij}$ if the w^k_i are symmetric random variables with zero mean and that u and w are independent. This is a form of tensor-sketching[49] and is also systematic with the expected error in approximating the kernel decreasing as $K^{-\frac{1}{2}}$.

The different flavours of element-embedding and tensor-reduction listed above are all accessible via specifying various combinations of `R_mix`, `Z_mix`, `sym_mix` and `K`, which specify how the initial channels should be mixed, and the `coupling` keyword which specifies how the resulting channels should be coupled together. Note that optimisation of the embedding weights w is not available in `gap_fit` with normallly distributed random weights used instead. Please see the keyword glossary for more details.

An alternative compression strategy proposed by Darby *et al.*[50] simply involves summing over one (or more) of the α, β, n or n' indices in Eq. (23). It is efficient to perform this summation at the level of the density expansion coefficients c^α_{nl} where it can also be most easily understood; summing over a radial index n is equivalent to projecting the 3D density onto the surface of the unit sphere whilst summing over the element index α corresponds to forming the total, element-agnostic density. The power-spectrum is a generalised 3-body descriptor where each term in the following sum

$$p^{\alpha\beta}_{nn'l} = \sum_{ij} \mathbf{c}^\alpha_{i,nl} \cdot \mathbf{c}^\beta_{j,n'l} \tag{35}$$

corresponds to a triangle formed by the central atom and the neighbour atoms i and j. As such, the various possible effects of this compression scheme on an individual 3-body (correlation order 2) term in this summation can be visualised as in Fig. 3. The different options are labelled according to the element-sensitive ν_S and radially-sensitive ν_R correlation orders where each summation over an element (or radial) index lowers the respective correlation order by one, e.g., $\nu_S = 1$, $\nu_R = 1$ specifies $p^\alpha_{nl} = \sum_{\beta n'} p^{\alpha\beta}_{nn'l}$.

Finally, it is also possible to achieve compression through the experimental `Z_map` keyword, which allows the user to group different elements together; equivalent to element embedding with $w^k_\alpha = 1$ if element α is in group k or 0 if it is not. As two densities are coupled, two distinct sets of groups may be specified if desired. Please see the keyword glossary for more details.

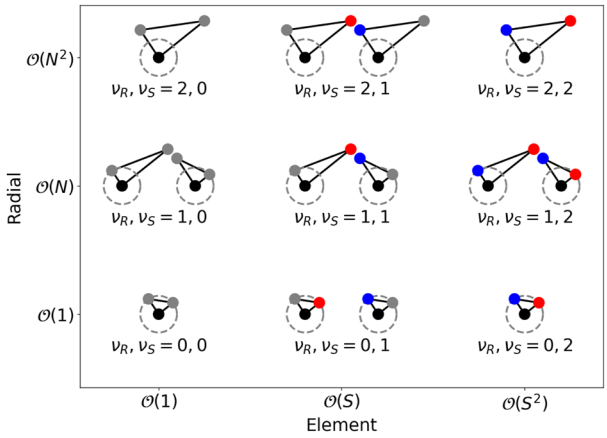

FIG. 3. Different SOAP compression strategies. Neighbour atoms around the central atom (black) may be represented as element-agnostic (grey) or element-specific (red or blue). To eliminate the radial dependence, neighbours may be projected on the unit sphere (dashed circle) around the central atom. Reprinted with permission from, npj Comput. Mater. **8**, 166 (2022). Copyright 2015 Author(s), licensed under a Creative Commons Attribution 4.0 License.

4. `soap_turbo` descriptors

The `soap_turbo` descriptor is a variant of SOAP optimised for computational efficiency. A detailed account of this descriptor has been given in Ref. 42. Here, we briefly describe its main features while giving a more in-depth description of the features that have been introduced since the publication of the original paper, namely multispecies support and compression. A comparison between the `soap` and `soap_turbo` implementations of SOAP is given in Sec. IV B.

The representation of the atomic density field in the local neighborhood, that is, within a cutoff sphere of radius r_{cut} of atom i, is carried out in an explicitly separable form of radial and angular channels. Therefore the expansion coefficients can also be split into components that depend exclusively on the radial index n or angular indices l, m:

$$\rho_i(\mathbf{r}) = \sum_{j \in S_i(r_{\text{cut}})} \sum_{nlm} c^{i,j}_{nlm} g_n(r) Y_{lm}(\theta, \phi), \tag{36}$$

$$c^{i,j}_{nlm} = b^{i,j}_n a^{i,j}_{lm}, \tag{37}$$

where $b^{i,j}_n$ are the radial expansion coefficients, $a^{i,j}_{lm}$ are the angular expansion coefficients and j runs over all neighbours of i within the cutoff sphere. A number of smoothing and scaling functions are introduced, to make the width of the atom-centered smooth functions depend on the distance from the center of the SOAP sphere. The main implementation differences between `soap` and `soap_turbo` are the use of smoother polynomial radial basis functions and several numerical tricks that allow to express the radial and angular expansion coefficients as recursive series. There are also differences in how multispecies support and compression are handled, described below.

Support in `soap_turbo` for multiple chemical elements is provided by augmenting the radial basis set via a direct sum:

$$\{g(r)\} = \bigoplus_{s=1}^{N_s} \{g(r)\}_s, \tag{38}$$

where s runs over the number of elements. The only advantage of this approach compared to the regular SOAP multielement support is that each element can be represented with a different radial basis set, including a different number of radial basis functions. One instance where this feature may be useful is when one of the elements can be represented with fewer radial basis functions than another, reducing the dimensionality of the descriptor and thus its computational cost. In principle, this approach also allows for using different cutoff radii for different elements within the same descriptor although, in practice, the GAP interface currently restricts the cutoff to be the same for all elements. The angular basis, on the other hand, is the same for all elements.

Compression is also supported by `soap_turbo` through three different approaches. The first compression scheme is a heuristic recipe, that we refer to as "trivial," which retains only the SOAP elements that run over $n = 1$ for single element and the first radial component of the element-specific basis for multiple elements:

$$\{\tilde{p}\} \equiv \bigoplus_{nn'l} \{p_{nn'l}\}, \tag{39}$$

$$\text{with} \quad n = 1, N_r^l + 1, \ldots, \sum_{i=1}^{N_s-1} N_r^i + 1,$$

$$n' = 1, \ldots, \sum_{i=1}^{N_s} N_r^i, \quad \text{and} \quad l = 0, \ldots, l_{\max},$$

where the tilde indicates the compressed descriptor and N_r^1 is the number of radial basis functions for the first element, and the direct sum continues until the last element in the descriptor has been considered. Thus, only components with $n = 1$, $n = N_r^1 + 1$, etc., are retained. Trivial compression affords of the order of a factor of 5 in dimensionality reduction without significant loss in accuracy for most production GAP models that we have fitted so far.

The second compression scheme in `soap_turbo` provides a quasiequivalence with the radial- and element-sensitive correlation orders offered in regular SOAP compression introduced in Sec. III C 3. Numerical comparisons of these compression recipes are given in the local property example in Sec. IV.

The third compression scheme has no predefined recipe. Instead, the user can provide an arbitrary linear transformation (via an input text file) that projects the SOAP vector from its original N-dimensional space to a reduced M-dimensional space, where $M < N$:

$$\tilde{\mathbf{p}} = P\mathbf{p}, \qquad P \in \mathcal{M}_{M\times N}. \tag{40}$$

In all cases, the descriptor is renormalized after compression.

Finally, we remark that due to the overlap properties of the polynomial radial basis sets and related instabilities in the numerical approach employed to construct the orthonormal radial basis used to construct `soap_turbo` descriptors, there is a practical limitation of $n_{\max} \approx 10$. For most practical purposes (e.g., in constructing accurate GAP force fields), there is no need to increase the size of the radial basis set beyond ≈8 basis functions.

IV. PRACTICAL EXAMPLES

A. Si interatomic potential

Among the first successful applications of GAP was a general-purpose interatomic potential for silicon.[39] We have used the database of atomic configurations to train a series of GAP models to demonstrate the effect of the most crucial descriptor and kernel hyperparameter choices on the performance and computational cost of the resulting potential. The extended XYZ file, containing the database and shared in the supplementary material of Ref. 39, was randomly split into a training and a test set, containing 80% and 20% of the original configurations, respectively. We list the parameters used in the `gap_fit` command line in Table I with a detailed explanation for each.

While keeping all other parameters constant, we individually varied the SOAP parameters $n_{\max}$, $l_{\max}$, r_{cutoff} and σ, as well as the polynomial kernel exponent ζ and the number of sparse points M. Based on the original Si GAP model, the parameters, $n_{\max} = 6$, $l_{\max} = 6$, $r_{\text{cutoff}} = 5$ Å, $\sigma = 0.5$ Å, $\zeta = 4$ and $M = 8000$ were used.

We have evaluated the interaction energies, forces and virial stresses of all atomic configurations in the test set with the resulting models, and calculated the RMSE with respect to the *ab initio* energies.

To illustrate how the choice of the parameters affects the computational cost of each model, we have also determined the average calculation time per atom, using a desktop computer utilising a single core of an Intel® Core™ i5-9600K central processing unit (CPU) at 3.70 GHz.

Trends are presented in Fig. 4, generally showing that more complex models, i.e., those with higher $l_{\max}$, $n_{\max}$, ζ and M values, are more accurate. Thanks to the regularisation term in the GPR, higher complexity does not result in overfitting. However, the computational cost of models with higher $l_{\max}$, $n_{\max}$ and sparse points is increased due to the larger number of terms. Even though the polynomial kernel at higher orders results in more terms, these are not calculated explicitly, therefore the computational cost remains approximately constant at different ζ values.

Increasing the spatial cutoff of the atomic neighbourhood environment results in more accurate models up to 5 Å, as further neighbours may influence the local energy function. However, at higher cutoff values the quality of the model deteriorates, which may be regarded as a sign of underfitting, when the available data is not sufficient to determine the dependence of the local energy terms on further neighbours.

The cutoff radius of SOAP or other descriptors may also be chosen by considering the force constant matrix of the atomic system, using a criterion on the spatial decay of the elements.[12]

The smoothness parameter σ has a strong influence on the accuracy of the model, which is related to how the neighbouring atoms are represented. Narrow Gaussians lead to fewer similar kernel values between two atomic environments, resulting in overfitting, whereas with wide Gaussians the resolution of the representation is lower.

TABLE I. Command line parameters of `gap_fit` used to fit a GAP model for silicon.

Key	Value	Comments
`atoms_filename`	`train.xyz`	Extended XYZ file of training configurations
`gap_file`	`gp_c5.0_n6_l6_s0.5_z4_p8000.xml`	Filename of output XML of GAP model
`energy_parameter_name`	`dft_energy`	Target energy key in the extended XYZ file. Default: `energy`
`force_parameter_name`	`dft_force`	Target force key in the extended XYZ file. Default: `force`
`virial_parameter_name`	`dft_virial`	Target virial stress key in the extended XYZ file. Default: `virial`
`e0_offset`	`2.0`	Shifts the baseline energy which is determined from the isolated atom energy.
`sparse_jitter`	`1e-8`	Regularisation of $\mathbf{K}_{MM}$
`default_kernel_regularisation`	`{0.001 0.1 0.05 0.0}`	Kernel regularisation for target values. Format: `{energy force virial hessian}`
`config_type_kernel_regularisation`	`{ liq:0.003:0.15:0.2:0.0: amorph:0.01:0.2:0.4:0.0: sp:0.01:0.2:0.4:0.0: }`	Override factors for tagged XYZ frames format: `{config_type:energies:forces:virials:hessians}`
`gap`	`{ soap`	SOAP descriptor, used as a template.
	`n_max=6`	One per element is generated, based
	`l_max=6`	on the configurations in the database.
	`atom_gaussian_width=0.5`	Broadening of the atoms in the neighbour density (σ)
	`soap_exponent=4`	Exponent of the polynomial soap kernel
	`n_sparse=8000`	Many sparse points are needed due to
	`cutoff=5.0`	the high dimensionality of the descriptor.
	`cutoff_transition_width=1.0`	Length scale of radial cutoff in Å
	`sparse_method=cur_points`	Sparse points are chosen using the CUR method.
	`covariance_type=dot_product`	Form of the kernel
	`energy_scale=3.0`	Prefactor of the kernel in eV
	`}`	

The kernel regularisation hyperparameters provide control on the accuracy and smoothness of the resulting model. Lower values bias the potential to fit the training data more accurately, but may result in overfitting. For a given spatial cutoff in the descriptors, the kernel regularisation on the forces may be derived from the decay of the force constant matrix or by quantifying the force uncertainty from *ab initio* calculations.[12] Finding appropriate figures for the kernel regularisation of energy and virial stress values may require cross-validation, but a typical target energy error in condensed systems is 1 meV/atom, therefore this is often a suitable starting figure. The choices of kernel regularisation hyperparameters is discussed extensively in Ref. 12.

To illustrate the effect of varying the kernel regularisation hyperparameters, we fitted a series of silicon GAP models using the same parameters, listed in Table I, and same training database as previously, but varied the hyperparameters corresponding to energy (σ_{energy}), force (σ_{force}) and virial stress (σ_{virial}) independently. We have utilised the test set to predict energy, force and virial stress values using the resulting models, and evaluated RMSE figures for each model and quantity.

These results are shown in Fig. 5, highlighting how different choices of regularisation hyperparameters affect the quality of the fit. For the RMSE of the predicted energies, there is an optimal value of σ_{energy}, while the RMSE of the predicted forces and virial stresses decrease monotonically when increasing σ_{energy}. Similar opposing changes in the errors of predicted quantities may be observed when σ_{force} and σ_{virial} are varied. We attribute these trends to the epistemic uncertainty of our models which is due to our assumptions, such as the locality of the descriptor or the body-order representa-

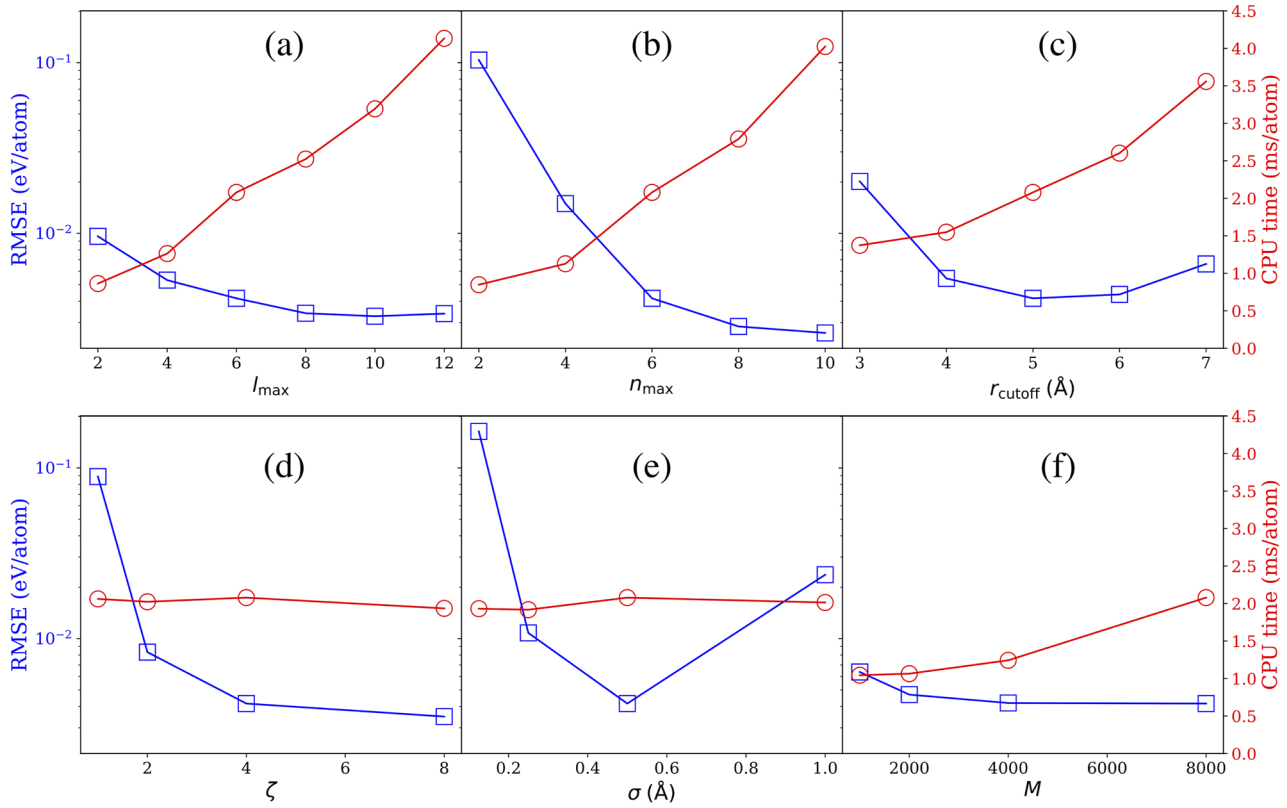

FIG. 4. Performance (blue squares) and computational cost (red circles) of different GAP models. The performance is quantified by the RMSE of the predicted energy, evaluated on a test set. Comparisons with respect to changes in (a) the radial resolution n_{max} of SOAP; (b) the angular resolution l_{max} of SOAP; (c) the spatial cutoff r_{cutoff} of SOAP; (d) the power of the polynomial kernel ζ; (e) the width σ of Gaussians representing the atoms in SOAP; (f) the number of sparse points M.

tion. These assumptions limit the simultaneous accuracy the model may achieve in energies, forces and virial stresses, and biasing the fit towards reproducing a particular quantity causes a deterioration in others.

These results are intended to provide a guide to fitting GAP models and their adaptation is almost certainly necessary when fitting potentials representing different atomic systems. While an exhaustive hyperparameter search is not always feasible, the recently parallelised `gap_fit` allows rapid creation and subsequent evaluation of models.

B. Local property

While the local energies predicted by GAPs in cohesive energy models are not physical observables, there are different local atomic properties with physical significance amenable to direct learning within the GAP framework. SOAP descriptors are particularly suited for this task. Examples of such models that we have trained in the past include adsorption energies,[51] effective Hirshfeld volumes[52] and core-electron binding energies (CEBEs).[53] Here, we revisit the CEBE

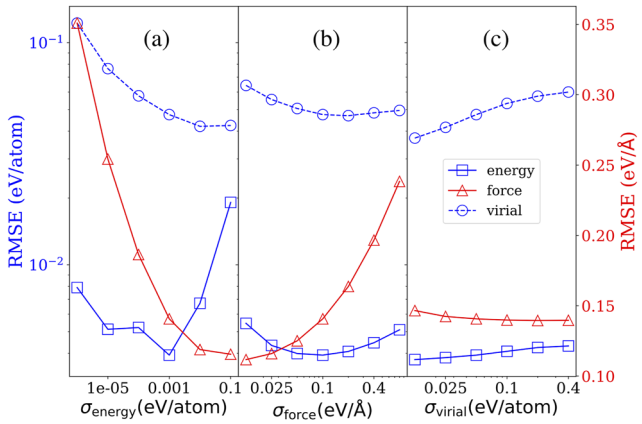

FIG. 5. Performance of GAP models as the function of regularisation hyperparameters. The RMSE of the predicted energies (blue squares), forces (red triangles) and virial stresses (blue circles) are shown as the energy (panel a), force (panel b) and virial (panel c) regularisation hyperparameters are varied independently.

database of Ref. 53 and use it as a test bed for the performance of SOAP-based local property models as a function of different convergence parameters.

First, let us briefly provide the context for the usefulness of CEBEs in materials science. X-ray photoelectron spectroscopy (XPS) uses monochromatic (fixed energy) X-ray light to excite the deep-lying core electrons in materials and molecules. In oxygen- and carbon-containing compounds these are the 1s states. When an X-ray photon with sufficient energy is absorbed by one such core electron the latter becomes photoejected, in such a way that its kinetic energy can be measured by a detector. Since the energy of the incident photons is fixed, the difference between the measured kinetic energy and the incident energy equals the CEBE. This CEBE is characteristic of the chemical environment around the atom whose core electron was excited, and so an XPS experiment provides a spectrum whose characteristic peaks give insight into the atomic structure of the material or molecule being probed. Because the core electron is strongly localized around the nucleus and only feels the influence of the immediate surrounding medium, XPS is particularly well suited to learning with local atomic descriptors such as SOAP, as we showed in Ref. 53. One of the results presented in that paper is a database of *GW*-level CEBEs for a subset of the CHO-containing molecules in the QM9 dataset,[54] a large dataset of small stable organic molecules, containing up to nine non-hydrogen atoms.

Reference 53 presented learning curves for C1s and O1s CEBE models trained from the QM9-*GW* data using `soap_turbo` descriptors without sparsification. Here we take a more detailed look at the effect of different technical parameters on the quality of the fit: cutoff radius for SOAP neighbours, sparsification scheme, `soap` vs `soap_turbo` and the effect of different compression recipes on the results. We start out by splitting the QM9-*GW* CEBE database of Ref. 53 into a training set (80% of the structures) and a test set (20% of the structures).

The `gap_fit` local property feature relies on the user providing a per-atom local property array. In this case, an ASE-format XYZ file is provided with a list of per-atom CEBEs. The following is an example for a formaldehyde molecule:

```
4
Lattice="10.89392123 0.0 0.0 0.0 10.81307256 0.0 0.0 0.0 9.01510639" \
Properties=species:S:1:pos:R:3:GW_CEBE:R:1:local_property_mask:L:1 pbc="T T T"
C        5.45500662        5.70744485        4.50565078      294.52870000 T
O        5.47139525        4.50000000        4.50000000      538.69210000 T
H        6.39392123        6.31307256        4.50174003        0.00000000 F
H        4.50000000        6.28730276        4.51510639        0.00000000 F
```

The fifth column contains the 1s CEBE for the C and O atoms, given in eV in this example. Since H atoms do not have a core, CEBEs for these atoms are not available, and we pad the array with zeros. A mask is provided to let `gap_fit` know these are to be ignored during the fit.

In addition to the database of atomic structures and observables (CEBEs here), one needs to provide the name of the local property as specified in the database (`local_property_parameter_name="GW_CEBE"` here), the default regularization parameter (`default_local_property_sigma = 0.01` here, in the same units as the local property), and possibly offsets for the properties to be learned (`local_property0={C:290.816456:O:537.946208:H:0}` here). In our case, the 0 offset is computed as the average CEBE of C1s and O1s core electrons separately. It is important to provide these offsets so that the ML model only needs to fit the (smooth) differences in the local property values, and not the absolute numbers, which are significantly harder to learn. Otherwise, the specification of the atomic descriptors is done in the same way as for a regular GAP model.

All the results for this example are summarized in Fig. 6. In panel (a) we show a comparison of models for C1s CEBEs fitted using a `soap_turbo` descriptor with varying cutoff radii and varying number of sparse points; the cutoff radius is the single most important hyperparameter in SOAP-based models. Clearly, the accuracy of the models can be systematically increased by increasing the cutoff. However, the number of sparse points limits the expressivity of the model, and models with less sparse points will not benefit from further increasing the cutoff beyond a certain point. E.g., with 50 sparse points a 3 Å model performs equal to a 7 Å model. We observe that the statistical variation in the model performance increases with the cutoff, due to the corresponding increase in the size of configuration space covered by the descriptor (the error bars are given as the standard deviation computed over ten different models obtained from ten different randomly chosen sparse sets).

In Fig. 6(b) we show a comparison of C1s and O1s models fitted using `soap` and `soap_turbo` descriptors of the same dimensionality (CHO-sensitive descriptors with 8 radial basis functions per element and up to 8th degree spherical harmonics, the same basis set as used for all the calculations in Fig. 6). `soap_turbo` models perform slightly better than `soap` models, except for the C1s models with maximum number (1000) of sparse points, where they perform equally.

If Fig. 6(c) we assess the effect of using random sparse point selection vs using CUR decomposition to select the sparse set descriptors, as well as the possible effect of using

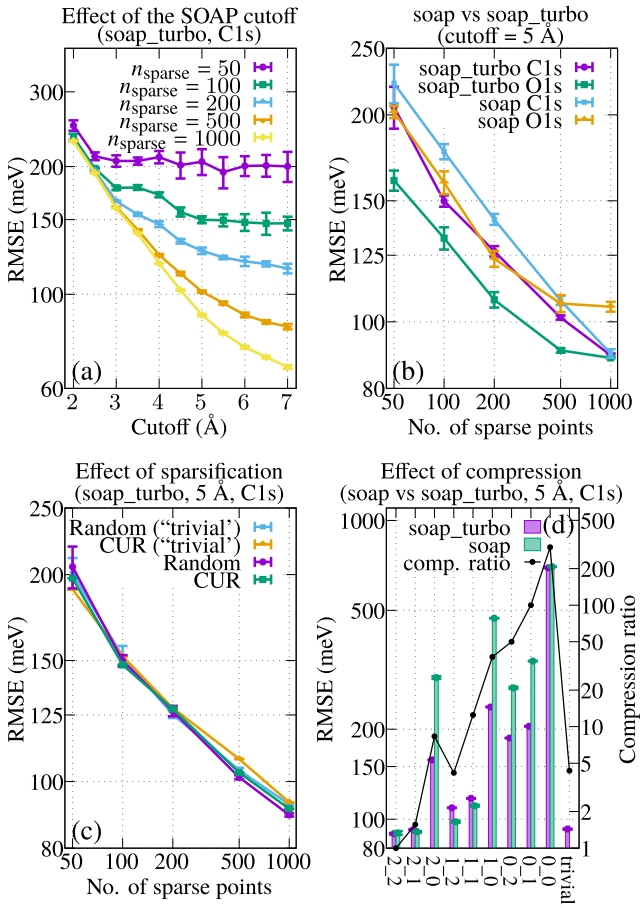

FIG. 6. (a) RMSEs for a `soap_turbo` C1s model as a function of SOAP cutoff and sparse set size with random selection. (b) Comparison between `soap` and `soap_turbo` C1s and O1s models with random sparse set size (fixed cutoff of 5 Å). (c) Effect of random and CUR sparsification strategies, for a `soap_turbo`-based C1s model (fixed cutoff of 5 Å); the effect of adding compression (`soap_turbo`'s "trivial" recipe) is also tested. (d) Effect of different compression recipes on `soap` and `soap_turbo` C1s model performance (fixed cutoff of 5 Å); the right axis gives the compression ratio of the descriptor as the number of dimensions of the full descriptor (2700) divided by the number of dimensions of the compressed descriptor. In all cases (a)–(d), error bars are estimated from the standard deviation of the RMSEs calculated among ten different models with random sparse set selection.

descriptor compression (`soap_turbo`'s "trivial" compression recipe) on the results. For this numerical test, the performance of all models is essentially the same for most practical purposes.

Finally, in Fig. 6(d) we perform a thorough numerical test of different compression recipes on the accuracy of the QM9-*GW* models. The *i_j* labeling convention refers to the $\nu_R \equiv i$ and $\nu_S \equiv j$ `soap` sensitivity parameters discussed above, and their quasiequivalent recipes for `soap_turbo`. Additionally, `soap_turbo` can use the "trivial" compression scheme as detailed above. In addition to the root-mean-squared error (RMSE), the graph shows the compression ratio computed as the number of dimensions of the compressed descriptor divided by those of the full descriptor. Unsurprisingly, the errors increase with the compression ratio, with most compression recipes providing better performance for `soap_turbo`, except for 2_1, 1_2 and 1_1, where `soap` performs slightly better. That is, the advantage of using `soap_turbo` increases with the degree of compression, whereas `soap` performs equally or slightly better than `soap_turbo` at low compression ratios. The "trivial" compression recipe is only available for `soap_turbo` and provides arguably the best compromise between accuracy and compression ratio (a factor of 4.3 vs uncompressed SOAP) among the tested schemes, at least for this particular test with the QM9-*GW* database.

Although not shown here, the relative advantage of `soap_turbo` vs `soap` increases when looking at the O1s models, likely because the QM9-*GW* training database we used contains significantly more C1s data (11.5k entries) than O1s data (1.5k entries). This indicates better generalization and data efficiency for `soap_turbo`, although it is important to note that the performance of each descriptor is dataset specific.

C. High-entropy alloy

The Mo–Nb–Ta–V–W quinary high-entropy alloy studied by Byggmästar *et al.*[38] is a complex, multicomponent system. To illustrate the generation of a GAP model, we provide the parameters, complete with explanations and comments, that were used to test and benchmark the MPI-ScaLAPACK implementation of the `gap_fit` program.[37] Here we highlight a recent feature which conveniently allows the fitting parameters to be stored in a file, rather than provided as command line arguments. The fitting parameters are entered in the file supplementary material in a `key=value` format, with commentary on each provided in Table II.

The fitting database is openly available as the supplementary material of the article by Byggmästar *et al.*,[38] which may be downloaded from the Fairdata repository.[55] The fitting procedure can then be carried out by running the command `gap_fit config_file=config` with the database file `db_HEA_reduced.xyz` and the configuration file `config` in the same directory. For the parallel implementation, the command line `mpirun -np 2 gap_fit config_file=config` executes the process on two computational cores. Hybrid OpenMP-MPI execution is possible, for which the number of threads may be adjusted by setting the environment variable as `export OMP_NUM_THREADS=4`. For the specific queuing system available to the user, the documentation should be consulted.

The resulting GAP model is stored in the `gp_HEA.xml` file and a set of text files according to the naming pattern

```
from quippy.potential import Potential
p = Potential(param_filename="gp_HEA.xml")
...
a.calc = p
```

TABLE II. Contents of file `config` used to fit a GAP model for the Mo–Nb–Ta–V–W quinary high-entropy alloy.

Key	Value	Comments
`atoms_filename`	`db_HEA_reduced.xyz`	Extended XYZ file of training configurations
`do_copy_at_file`	`F`	Do not copy XYZ data to output XML
`gap_file`	`gp_HEA.xml`	Filename of output XML of GAP model
`sparse_jitter`	`1e-8`	Regularisation of $\mathbf{K}_{MM}$
`default_kernel_regularisation`	`{0.002 0.1 0.5 0.0}`	Kernel regularisation for target values format: `{energy force virial hessian}`
`config_type_kernel_regularisation`	`{`	Override factors for tagged XYZ frames format: `{config_type:energies:forces:virials:hessians}`
	`dimer:0.1:1.0:1.0:0.0:`	
	`hea_short_range:0.05:0.8:2.0:0.0:`	
	`hea_surface:0.01:0.4:1.0:0.0:`	
	`isolated_atom:0.0001:0.04:0.01:0.0:`	
	`liquid_composition:0.01:0.5:2.0:0.0:`	
	`liquid_hea:0.01:0.5:2.0:0.0:`	
	`short_range:0.05:0.8:0.8:0.0:`	
	`surf_liquid:0.01:0.4:0.2:0.0`	
	`}`	
`gap`	`{`	Two-body descriptor, used as a template.
	`distance_2b`	One per element pair is generated.
	`n_sparse=20`	In one dimension few sparse points
	`sparse_method=uniform`	(uniformly spaced) are enough.
	`covariance_type=ard_se`	Kernel is squared exponential (se).
	`cutoff=5.0`	
	`cutoff_transition_width=1.0`	
	`energy_scale=10.0`	
	`lengthscale_uniform=1.0`	
	`:`	
	`soap`	SOAP descriptor, used as a template.
	`n_sparse=4000`	One per element is generated, based
	`sparse_method=cur_points`	on the configurations in the database.
	`covariance_type=dot_product`	Many sparse points are needed due to
	`n_max=8`	the high dimensionality of the descriptor.
	`l_max=4`	Sparse points are chosen using the CUR method.
	`soap_exponent=2.0`	
	`atom_gaussian_width=0.5`	
	`cutoff=5.0`	
	`cutoff_transition_width=1.0`	
	`energy_scale=1.0`	
	`}`	

`gp_HEA.xml.sparseX.GAP_*`. The interatomic potential may be accessed as a `Calculator` in ASE as

Where the Python variable `a` indicates an ASE `Atoms` object.

Massively parallel simulations with GAP models are possible with LAMMPS. To use the high-entropy alloy potential, the following lines should be added to the LAMMPS input file:

```
pair_style quip
pair_coeff**gp_HEA.xml "" 42 41 73 23 74
```

Where the LAMMPS atom types 1, 2, 3, 4, and 5 are mapped to Mo, Nb, Ta, V and W, respectively.

V. CONCLUSION AND OUTLOOK

We have reviewed the GAP framework from an implementation point of view, highlighting how a generic sparse GPR formalism is adapted for the prediction of interatomic potentials and

related quantities. An overview of the software package, coding practices and recent developments was provided, together with usage examples and detailed explanations of adjustable parameters, with references to the theory. The QUIP and GAP suite remains under maintenance and in active development by the authors, and will serve as a test bed for exploring ideas in the field of MLIPs. With interfaces to Python and major simulation packages, GAP serves as a useful tool for computational modelling.

Future developments will include the inclusion of more descriptors, such as Atomic Cluster Expansion (ACE),[2] rigorous means for uncertainty quantification, utilising modern computing architectures such as GPUs, and the implementation of more robust and efficient solvers for the fitting procedure. With the availability and popularity of more modern programming languages such as Python or Julia, Fortran may appear as an outdated choice. However, its interoperability with MPI and related libraries such as ScaLAPACK has proved to be an advantage in utilising the more traditional, but still highly prevalent, high performance computing facilities consisting of networked servers. As supercomputers and programming skills change, the current framework might prove to be too restrictive, but the practical insight documented here and the software will remain valuable for future endeavours.

SUPPLEMENTARY MATERIAL

Please see the supplementary material for the fitting parameters in a `key=value` format.

ACKNOWLEDGMENTS

This work was financially supported by the NOMAD Centre of Excellence (European Commission Grant Agreement No. 951786) and the Leverhulme Trust Research Project (Grant No. RPG-2017-191). A.P.B. acknowledges support from the CASTEP-USER project, funded by the Engineering and Physical Sciences Research Council under the Grant Agreement No. EP/W030438/1. M.A.C. acknowledges personal funding from the Academy of Finland under Grant No. 330488. We acknowledge computational resources provided by the Max Planck Computing and Data Facility provided through the NOMAD CoE, the Scientific Computing Research Technology Platform of the University of Warwick, the EPSRC-funded HPC Midlands + consortium (Grant No. EP/T022108/1), ARCHER2 (https://www.archer2.ac.uk/) via the UK Car-Parinello consortium (Grant No. EP/P022065/1), CSC-IT Center for Science, and the Aalto University Science-IT project. We thank the technical staff at each of these HPC centres for their support.

AUTHOR DECLARATIONS

Conflict of Interest

A.P.B. and G.C. are listed as inventors on a patent filed by Cambridge Enterprise, Ltd. related to SOAP and GAP (US patent 8843509, filed on 5 June 2009 and published on 23 September 2014). A.P.B., M.A.C., and G.C. benefit from licensing the GAP software to industrial users. Not-for-profit use for academic and educational purposes is granted under the Academic Software License for no cost. The other authors have no conflicts to disclose.

Author Contributions

Sascha Klawohn: Data curation (lead); Formal analysis (equal); Methodology (supporting); Validation (supporting); Visualization (equal); Writing – original draft (supporting). **James P. Darby**: Data curation (lead); Formal analysis (supporting); Methodology (supporting); Validation (supporting); Writing – original draft (supporting). **James R. Kermode**: Funding acquisition (equal); Methodology (supporting); Supervision (equal); Writing – original draft (supporting). **Gábor Csányi**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Supervision (equal); Writing – original draft (supporting); Writing – review & editing (supporting). **Miguel A. Caro**: Conceptualization (supporting); Formal analysis (supporting); Methodology (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (supporting). **Albert P. Bartók**: Conceptualization (equal); Formal analysis (equal); Funding acquisition (equal); Methodology (equal); Supervision (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

All databases referred to this work are available in public data repositories.

ARGUMENTS OF `GAP_FIT`

We provide a snapshot of the currently available command line arguments of the `GAP_FIT` program for completeness. As the QUIP and `GAP` packages are under constant development, this list may change. It should also be noted that most of the parameters need little adjustment, whereas those that pertain to the particular fitting problem are mandatory, requiring the user to specify a value. Some keywords have aliases, often less descriptive, but both of which are acceptable. These are indicated as `option (alias)`.

Common arguments

`config_file` File as alternative input to command line arguments. Newlines are converted to spaces.

`atoms_filename (at_file)` extended XYZ file containing database configurations in a concatenated form

`gap` Initialisation string for GAPs

`e0` Atomic energy value to be subtracted from energies before fitting, and added back on after prediction. Possible options are: a single number, used for all species; or by species, e.g.: `{Ti:-150.0:O:-320.1}`.

`local_property0` Local property value to be subtracted from the local property before fitting, and added back on after prediction. Possible options are: a single number, used for all species; or by species: e.g. `{H:20.0:Cl:35.0}`.

`e0_offset` Offset of baseline. If zero, the offset is the average atomic energy of the input data or the e0 specified manually.

`e0_method` Method to determine the constant energy baseline `e0`, if not explicitly specified. Possible options: `isolated` (default, each atom present in the XYZ needs to have an isolated representative, with a valid energy); `average` (e0 is the average of all total energies across the XYZ).

default_kernel_regularisation (default_sigma) Prior assumption of error in (energies forces virials hessians)

default_kernel_regularisation_local_property (default_local_property_sigma) Prior assumption of error in local_property.

sparse_jitter Extra regulariser used to regularise the sparse covariance matrix before it is passed to the linear solver. Use something small, it really shouldn't affect your results, if it does, your sparse basis is still very ill-conditioned.

hessian_displacement (hessian_delta) Finite displacement to use in numerical differentiation when obtaining second derivative for the Hessian covariance.

baseline_param_filename (core_param_file) QUIP XML file which contains a potential to subtract from data, and added back after prediction.

baseline_ip_args (core_ip_args) QUIP initialisation string for a potential to subtract from data, and added back after prediction.

energy_parameter_name Name of energy property in the input extended XYZ file that describes the data

local_property_parameter_name Name of local_property as a column in the input XYZ file that describes the data.

local_property_mask_parameter_name Used to exclude local properties on specific atoms from the fit. In the XYZ, it must be a logical column.

force_parameter_name Name of force property, as three columns, in the input XYZ file that describes the data.

virial_parameter_name Name of virial property in the input XYZ file that describes the data.

stress_parameter_name Name of stress property (6-vector or 9-vector) in the input XYZ file that describes the data - stress values only used if virials are not available. Note the opposite sign and standard Voigt order.

hessian_parameter_name Name of hessian property (column) in the input XYZ file that describes the data

config_type_parameter_name Allows grouping on configurations into. This option is the name of the key that indicates the configuration type in the input XYZ file. With the default, the key-value pair config_type=bcc would place that configuration into the group bcc.

kernel_regularisation_parameter_name (sigma_parameter_name) Kernel regularisation parameters for a given configuration in the database. Overrides the command line values for both defaults and config-type-specific values. In the input XYZ file, it must be prepended by energy_, force_, virial_ or hessian_ keywords.

force_mask_parameter_name To exclude forces on specific atoms from the fit. In the XYZ, it must be a logical column.

parameter_name_prefix Prefix that gets uniformly appended in front of {energy,local_property,force,virial,...}_parameter_name

config_type_kernel_regularisation (config_type_sigma) The kernel regularisation values to choose for each type of data, when the configurations are grouped into config_types. Format: {configtype1:energy:force:virial:hessian: config_type2: energy:force:virial:hessian}

kernel_regularisation_is_per_atom (sigma_per_atom) Interpretation of the energy and virial regularisation parameters specified in default_kernel_regularisation and config_type_kernel_regularisation. If T, these are interpreted as per-atom errors, and the variance will be scaled according to the number of atoms in the configuration. If F, they are treated as absolute errors and no scaling is performed. NOTE: values specified on a per-configuration basis (see kernel_regularisation_parameter_name) are always absolute, not per-atom.

do_copy_atoms_file (do_copy_at_file) Copy the input XYZ file into the GAP XML file (should be set to False for NetCDF input).

sparse_separate_file Save sparse point data in separate file, recommended for large number of sparse points.

sparse_use_actual_gpcov Use actual GP covariance for sparsification methods.

gap_file (gp_file) Name of output XML file that will contain the fitted potential

verbosity Verbosity control. Options: NORMAL, VERBOSE, NERD, ANALYSIS.

rnd_seed Random seed.

openmp_chunk_size Chunk size in OpenMP scheduling.

do_ip_timing To enable or not the timing of the interatomic potential.

template_file Template XYZ file for initialising object.

sparsify_only_no_fit If true, sparsification is done, but no fitting is performed. The sparse index is printed by adding print_sparse_index=file.dat to the descriptor specification string under the gap option.

dryrun If true, exits after memory estimate, before major allocations.

condition_number_norm Norm for condition number of matrix A of the linear system; O: 1-norm, I: inf-norm, <empty>: skip calculation (default)

linear_system_dump_file Basename prefix of linear system dump files. Skipped if <empty> (default).

mpi_blocksize_rows Blocksize of MPI distributed matrix rows. Affects efficiency and memory usage slightly. Maximum if specified as 0 (default).

mpi_blocksize_cols Blocksize of MPI distributed matrix columns. Affects efficiency and memory usage considerably. Maximum if 0. Default: 100.

mpi_print_all If true, each MPI processes will print its output. Otherwise, only the first process does (default).

Arguments of the GAP string

The following keywords are to be specified for each descriptor within the gap command line argument.

energy_scale (delta) Set the typical scale of the function being fitted, or the specific energy term if using multiple descriptors. It is equivalent to the standard deviation of the Gaussian Process in the probabilistic view, and typically this would be set to the standard deviation (i.e. root mean square) of the function that is approximated with the Gaussian Process.

`f0` Set the mean of the Gaussian Process. Defaults to 0.
`n_sparse` Number of sparse points to use in the sparsification of the Gaussian Process
`config_type_n_sparse` Number of sparse points in each configuration type. Format: `type1:50:type2:100`
`sparse_method` Sparsification method. Possible options: `RANDOM`(default), `PIVOT`, `CLUSTER`, `UNIFORM`, `KMEANS`, `COVARIANCE`, `NONE`, `FUZZY`, `FILE`, `INDEX_FILE`, `CUR_COVARIANCE`, `CUR_POINTS`. For explanations, see below.
`lengthscale_factor (theta_fac)` Set the width of Gaussians for the Gaussian and piecewise polynomial kernel by multiplying the range of each descriptor by `lengthscale_factor`. Can be a single number or different for each dimension. For multiple numbers in `lengthscale_factor`, separate each value by whitespaces.
`lengthscale_uniform (theta_uniform)` Set the width of Gaussians for the Gaussian and piecewise polynomial kernel, same in each dimension.
`lengthscale_file (theta_file)` Set the width of Gaussians for the Gaussian kernel from a file. There should be as many real numbers as the number of dimensions, in a single line.
`sparse_file` Sparse points from a file. If `sparse_method=FILE`, descriptor values as real numbers listed in a text file, one element per line. If `sparse_method=INDEX_FILE`, 1-based index of sparse points, one per line.
`mark_sparse_atoms` If true, reprints the original extended XYZ file after sparsification process, with a `sparse_property` column added, which is true for atoms associated with a sparse point.
`add_species` If true (default), create species-specific descriptors, using the descriptor string as a template.
`covariance_type` Type of covariance function to use. Available: `GAUSSIAN`, `DOT_PRODUCT`, `BOND_REAL_SPACE`, `PP` (piecewise polynomial).
`soap_exponent (zeta)` Exponent of `soap` type dot product covariance kernel
`print_sparse_index` If given, after determining the sparse points, their 1-based indices are appended to this file.
`unique_hash_tolerance` Hash tolerance when filtering out duplicate data points.
`unique_descriptor_tolerance` Descriptor tolerance when filtering out duplicate data points.

Options for sparse point selection

`none` No sparsification, selects all datapoints.
`index_file` Reads indices of sparse points from the file given by `sparse_file` and selects those from the de-duplicated data.
`file` Reads sparse points from the file given by `sparse_file`.
`random` Selects `n_sparse` random descriptors with the same probability.
`uniform` Computes a histogram of the data with `n_sparse` bins and returns a data point from each bin. This option is only suitable for low-dimensional descriptors.
`kmeans` The *k*-means clustering algorithm is performed on all descriptors to generate `n_sparse` clusters, of which the descriptors closest to the cluster means are selected as sparse points.
`fuzzy` A fuzzy version of *k*-means clustering[56] is used to generate `n_sparse` clusters.
`cluster` A *k*-medoid clustering based on the full covariance matrix of descriptors is performed, resulting in `n_sparse` clusters. The medoid points are selected as sparse points.
`pivot` The `n_sparse` "pivot" indices of the full covariance matrix are found, and used as the sparse points.
`covariance` Greedy data point selection based on the sparse covariance matrix, to minimise the GPR variance of all datapoints.
`cur_points` A CUR decomposition, based on the datapoints, is carried out to find the most representative `n_sparse` points.
`cur_covariance` A CUR decomposition, based on the full covariance matrix, is carried out to find the most representative `n_sparse` points.

Descriptors

The GAP module implements over 30 descriptors, most of which being experimental or unsupported. In the following, we include those that are commonly used by practitioners and supported by the GAP developers.

distance_2b arguments

`cutoff` Cutoff for `distance_2b`-type descriptors.
`cutoff_transition_width` Transition width of cutoff for `distance_2b`-type descriptors.
`Z1` Atom type #1 in bond. Any atom type if missing.
`Z2` Atom type #2 in bond. Any atom type if missing.
`resid_name` Name of an integer property in the atoms object giving the residue identifier of the molecule to which the atom belongs.
`only_intra` Only calculate bonds, i.e. *intra*molecular pairs with equal residue identifiers
`only_inter` Only apply to non-bonded atom pairs, i.e. *inter*molecular pairs with different residue identifiers.
`n_exponents` Number of exponents.
`exponents` Exponents in a list format, for example: `{-12 -6}`
`tail_range` Tail range.
`tail_exponent` Tail exponent.

soap arguments

`cutoff` Cutoff distance.
`cutoff_transition_width` Transition width of cutoff function.
`cutoff_dexp` Cutoff decay exponent.
`cutoff_scale` Cutoff decay scale.
`cutoff_rate` Inverse cutoff decay rate.
`l_max` $l_{\max}$ (spherical harmonics basis band limit) for soap-type descriptors.
`n_max` $n_{\max}$ (number of radial basis functions) for soap-type descriptors.
`atom_gaussian_width (atom_sigma)` Width of atomic Gaussian functions for soap-type descriptors.
`central_weight` Weight of central atom in environment.
`central_reference_all_species` Place a Gaussian reference for all atom species densities. By default (`F`) only consider when neighbour is the same species as centre.

`average` Whether to calculate averaged SOAP - one descriptor per atoms object. If false (default), atomic SOAP is returned.
`diagonal_radial` Only return the $n_1 = n_2$ elements of the power spectrum.
`covariance_sigma0` σ_0 parameter in polynomial covariance function.
`normalise (normalize)` Normalise descriptor, so magnitude is 1. In this case the kernel of two equivalent environments is 1.
`basis_error_exponent` $10^{-\texttt{basis\_error\_exponent}}$ is the max difference between the target and the expanded function.
`n_Z` How many different types of central atoms to consider.
`n_species` Number of species for the descriptor.
`species_Z` Atomic number of species.
`xml_version` Version of GAP the XML potential file was created.
`Z` Atomic number of central atom, 0 is the wild-card *or* Atomic numbers to be considered for central atom, must be a list.

soap compression arguments

`Z_mix` Mix the element channels together if present.
`R_mix` Mix the radial channels together if present.
`sym_mix` Specifies whether a single set of mixing weights is used *or* whether two sets are used. If `sym_mix=T`, tensor-decomposition is enabled. If `sym_mix=F` tensor-sketching is used.
`K` Integer specifying how many mixed channels to create. For example, `R_mix=T Z_mix=T K=5` will create 5 mixed channels whereas `R_mix=F n_max=6 Z_mix=T K=5` will result in `K*n_max` = 30 channels.
`coupling` If `coupling=T` full tensor-product coupling is applied across the resulting channels after mixing, whereas if `coupling=F` element-wise coupling is applied instead. The **only** exception to this rule occurs for `Z_mix=T R_mix=F` (or similarly for `Z_mix=F R_mix=T`) with `coupling=F`. Here, element-wise coupling is applied across the mixed-element channels but tensor-product coupling is applied across the unmixed radial channels, resulting in $p^k_{nn'l}$ (or similarly $p^{\alpha\beta}_{kl}$).
`mix_shift` Integer specifying the shift to the default seed that is used for the random number generator used to generate the mixing weights.
`nu_R` radially sensitive correlation order. Allowed values are 0, 1 and 2 (default).
`nu_S` species sensitive correlation order. Allowed values are 0, 1 and 2 (default).
`Z_map` Commas separate groups within a density. A colon separates the two densities if present. Otherwise the groups are taken to be equal. `Z_map = {1, 3, 22 23 24 }` has a separate channel for H and Li but treats Ti, V and Cr as identical `Z_map = {1, 3, 22, 23, 24 : 1, 3, 22 23 24}` has a separate channel for each element in the first density. In the second density there is a separate channel for H and Li but Ti, V and Cr are identical

soap_turbo arguments

`rcut_hard` Hard cutoff distance.
`rcut_soft` Soft cutoff distance.
`n_species` Number of species for the descriptor.
`l_max` $l_{\max}$ (spherical harmonics basis band limit) for soap-type descriptors.
`nf` Sets the rate of decay of the atomic density in the region between soft and hard cutoffs.
`radial_enhancement` Integer index (0, 1 or 2) that simulates the effect of modulating the radial overlap integral with the radial distance raised to this number.
`basis` Options: `poly3` or `poly3gauss` chooses a 3rd and higher degree polynomial radial basis set and augments it with a Gaussian at the origin, respectively.
`compress_file` Optional user-provided file specifying the compression recipe.
`compress_mode` Optionally provides a predefined compression recipe.
`central_index` 1-based index of central atom `species_Z` in the species array.
`alpha_max` Radial basis resolution for each species.
`atom_sigma_r` Width of atomic Gaussian functions for soap-type descriptors in the radial direction.
`atom_sigma_r_scaling` Scaling rate of radial sigma: scaled as a function of neighbour distance.
`atom_sigma_t` Width of atomic Gaussian functions for soap-type descriptors in the angular direction.
`atom_sigma_t_scaling` Scaling rate of angular sigma: scaled as a function of neighbour distance.
`amplitude_scaling` Scaling rate of amplitude: scaled as an inverse function of neighbour distance.
`central_weight` Weight of central atom in environment.
`species_Z` Atomic number of species, including the central atom.

REFERENCES

[1] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).
[2] R. Drautz, Phys. Rev. B **99**, 014104 (2019).
[3] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, Phys. Rev. Lett. **104**, 136403 (2010).
[4] K. T. Schütt, H. E. Sauceda, P. J. Kindermans, A. Tkatchenko, and K. R. Müller, J. Chem. Phys. **148**, 241722 (2018).
[5] K. Gubaev, E. V. Podryabinkin, and A. V. Shapeev, J. Chem. Phys. **148**, 241727 (2018).
[6] L. Zhang, J. Han, H. Wang, R. Car, and W. E, Phys. Rev. Lett. **120**, 143001 (2018).
[7] R. Jinnouchi, J. Lahnsteiner, F. Karsai, G. Kresse, and M. Bokdam, Phys. Rev. Lett. **122**, 225701 (2019).
[8] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky, Nat. Commun. **13**, 2453 (2022); arXiv:2101.03164.
[9] C. Chen and S. P. Ong, Nat. Comput. Sci. **2**, 718 (2022).
[10] C. E. Williams and C. K. I. Williams, *Gaussian Processes for Machine Learning* (MIT Press, 2007).
[11] K. Kawaguchi, in *Advances in Neural Information Processing Systems*, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc., 2016), Vol. 29.
[12] V. L. Deringer, N. Bernstein, G. Csányi, C. B. Mahmoud, M. Ceriotti, M. Wilson, D. A. Drabold, and S. R. Elliott, Nature **589**, 59 (2020).
[13] V. L. Deringer and G. Csányi, Phys. Rev. B **95**, 094203 (2017).
[14] W. J. Szlachta, A. P. Bartók, and G. Csányi, Phys. Rev. B **90**, 104108 (2014).
[15] V. L. Deringer, M. A. Caro, and G. Csányi, Nat. Commun. **11**, 5461 (2020).
[16] A. P. Bartók, M. J. Gillan, F. R. Manby, and G. Csányi, Phys. Rev. B **88**, 054104 (2013).

[17]D. Dragoni, T. D. Daff, G. Csányi, and N. Marzari, Phys. Rev. Mater. **2**, 013808 (2018); arXiv:1706.10229.
[18]J. Kloppenburg, A. Pedersen, K. Laasonen, M. A. Caro, and H. Jónsson, Nanoscale **14**, 9053 (2022).
[19]J. Kloppenburg, L. B. Pártay, H. Jónsson, and M. A. Caro, J. Chem. Phys. **158**, 134704 (2023).
[20]G. Sivaraman, L. Gallington, A. N. Krishnamoorthy, M. Stan, G. Csányi, A. Vazquez-Mayagoitia, and C. J. Benmore, Phys. Rev. Lett. **126**, 156002 (2021).
[21]Y.-B. Liu, J.-Y. Yang, G.-M. Xin, L.-H. Liu, G. Csányi, and B.-Y. Cao, J. Chem. Phys. **153**, 144501 (2020).
[22]S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. J. Probert, K. Refson, and M. Payne, Z. Kristallogr. - Cryst. Mater. **220**, 567 (2005).
[23]T. K. Stenczel, Z. El-Machachi, G. Liepuoniute, J. D. Morrow, A. P. Bartók, M. I. J. Probert, G. Csányi, and V. L. Deringer, J. Chem. Phys. **159**, 044803 (2023).
[24]See https://github.com/libAtoms/QUIP for The QUIP repository.
[25]D. Mackay, *Information Theory, Inference, and Learning Algorithms* (Cambridge University Press, 2003).
[26]V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti, and G. Csányi, Chem. Rev. **121**, 10073 (2021).
[27]F. Musil, A. Grisafi, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti, Chem. Rev. **121**, 9759 (2021).
[28]A. P. Bartók, R. Kondor, and G. Csányi, Phys. Rev. B **87**, 184115 (2013).
[29]L. Foster, A. Waagen, N. Aijaz, M. Hurley, A. Luis, J. Rinsky, C. Satyavolu, M. J. Way, P. Gazis, and A. Srivastava, J. Mach. Learn. Res. **10**, 857 (2009).
[30]Quantum Mechanics and Interatomic Potentials.
[31]See https://pypi.org/project/quippy-ase/ for The quippy-ase Python package.
[32]A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, J. Phys.: Condens. Matter **29**, 273002 (2017).
[33]See https://github.com/libAtoms/extxyz for The extended XYZ format.
[34]M. W. Mahoney and P. Drineas, Proc. Natl. Acad. Sci. U. S. A. **106**, 697 (2009).
[35]E. Anderson, Z. Bai, C. Bischof, S. 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, Philadelphia, PA, 1999).
[36]L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997).
[37]S. Klawohn, J. R. Kermode, and A. P. Bartók, Mach. Learn.: Sci. Technol. **4**, 015020 (2023).
[38]J. Byggmästar, K. Nordlund, and F. Djurabekova, Phys. Rev. B **104**, 104101 (2021).
[39]A. P. Bartók, J. Kermode, N. Bernstein, and G. Csányi, Phys. Rev. X **8**, 041048 (2018).
[40]S. De, A. P. Bartók, G. Csányi, and M. Ceriotti, Phys. Chem. Chem. Phys. **18**, 13754 (2016).
[41]F. M. Paruzzo, A. Hofstetter, F. Musil, S. De, M. Ceriotti, and L. Emsley, Nat. Commun. **9**, 4501 (2018).
[42]M. A. Caro, Phys. Rev. B **100**, 024112 (2019).
[43]M. J. Willatt, F. Musil, and M. Ceriotti, Phys. Chem. Chem. Phys. **20**, 29661 (2018).
[44]A. Goscinski, F. Musil, S. Pozdnyakov, J. Nigam, and M. Ceriotti, J. Chem. Phys. **155**, 104106 (2021).
[45]K. Gubaev, E. V. Podryabinkin, G. L. Hart, and A. V. Shapeev, Comput. Mater. Sci. **156**, 148 (2019).
[46]A. Bochkarev, Y. Lysogorskiy, S. Menon, M. Qamar, M. Mrovec, and R. Drautz, Phys. Rev. Mater. **6**, 013804 (2022).
[47]N. Lopanitsyna, G. Fraux, M. A. Springer, S. De, and M. Ceriotti, Phys. Rev. Mater. **7**, 045802 (2023).
[48]J. P. Darby, D. P. Kovács, I. Batatia, M. A. Caro, G. L. W. Hart, C. Ortner, and G. Csányi, Phys. Rev. Lett. **131**, 028001 (2023); arXiv:2210.01705.
[49]D. P. Woodruff *et al.*, Found. Trends® Theor. Comput. Sci. **10**, 1 (2014).
[50]J. P. Darby, J. R. Kermode, and G. Csányi, npj Comput. Mater. **8**, 166 (2022).
[51]M. A. Caro, A. Aarva, V. L. Deringer, G. Csányi, and T. Laurila, Chem. Mater. **30**, 7446 (2018).
[52]H. Muhli, X. Chen, A. P. Bartók, P. Hernández-León, G. Csányi, T. Ala-Nissila, and M. A. Caro, Phys. Rev. B **104**, 054106 (2021).
[53]D. Golze, M. Hirvensalo, P. Hernández-León, A. Aarva, J. Etula, T. Susi, P. Rinke, T. Laurila, and M. A. Caro, Chem. Mater. **34**, 6240 (2022).
[54]R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, Sci. Data **1**, 140022 (2014).
[55]See https://doi.org/10.23729/1b845398-5291-4447-b417-1345acdd2eae for The Mo–Nb–Ta–V–W database.
[56]C. Döring, M.-J. Lesot, and R. Kruse, Comput. Stat. Data Anal. **51**, 192 (2006).