MACHINE
LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
8 December 2020

REVISED
26 January 2021

ACCEPTED FOR PUBLICATION
15 February 2021

PUBLISHED
22 April 2021

TECHNICAL NOTE

# A bin and hash method for analyzing reference data and descriptors in machine learning potentials

**Martín Leandro Paleico and Jörg Behler**

Universität Göttingen, Institut für Physikalische Chemie, Theoretische Chemie, Tammannstraße 6, 37077 Göttingen, Germany

**E-mail:** martin.paleico@uni-goettingen.de

**Keywords:** machine learning, dataset, algorithm, atomic fingerprint, neural network potential

Supplementary material for this article is available online

## Abstract

In recent years the development of machine learning potentials (MLPs) has become a very active field of research. Numerous approaches have been proposed, which allow one to perform extended simulations of large systems at a small fraction of the computational costs of electronic structure calculations. The key to the success of modern MLPs is the close-to first principles quality description of the atomic interactions. This accuracy is reached by using very flexible functional forms in combination with high-level reference data from electronic structure calculations. These data sets can include up to hundreds of thousands of structures covering millions of atomic environments to ensure that all relevant features of the potential energy surface are well represented. The handling of such large data sets is nowadays becoming one of the main challenges in the construction of MLPs. In this paper we present a method, the bin-and-hash (BAH) algorithm, to overcome this problem by enabling the efficient identification and comparison of large numbers of multidimensional vectors. Such vectors emerge in multiple contexts in the construction of MLPs. Examples are the comparison of local atomic environments to identify and avoid unnecessary redundant information in the reference data sets that is costly in terms of both the electronic structure calculations as well as the training process, the assessment of the quality of the descriptors used as structural fingerprints in many types of MLPs, and the detection of possibly unreliable data points. The BAH algorithm is illustrated for the example of high-dimensional neural network potentials using atom-centered symmetry functions for the geometrical description of the atomic environments, but the method is general and can be combined with any current type of MLP.

## 1. Introduction

Machine-learning (ML) has become an important tool for the development of atomistic potentials, with a wide variety of applications in chemistry, physics and materials science [1–3]. Machine learning potentials (MLPs), like many other applications of ML algorithms, aim at approximating unknown functions, which in the present case is the multidimensional potential energy surface (PES) of the system of interest as a function of the atomic positions. The required information is obtained from sampling the PES at discrete points, i.e. particular atomic configurations, utilizing comparably demanding electronic structure methods such as density functional theory (DFT) [4, 5]. Once constructed, the MLP can then be used to perform cheap simulations with first principles accuracy for systems of significantly increased size and for extended time scales, to address problems which are inaccessible, e.g. to *ab initio* molecular dynamics simulations.

Many types of MLPs have been developed in recent years, including different flavors of artificial neural-network based potentials [6–14], Gaussian approximation potentials [15, 16], moment tensor potentials [17], spectral neighbor analysis potentials [18], and many others [19, 20]. Apart from reproducing atomic interactions, ML methods have also seen increasing applications that attempt to predict derived properties instead of those directly associated with the PES, such as dipole moments [21–23], charges

[14, 24–27], electronegativities [28], band gaps [29, 30], spins [31], and atomization energies [32]. All these applications of ML algorithms rely on the availability of large reference data sets that are used to train the respective ML method to reliably reproduce the property of interest. Generating these data sets is computationally very demanding, and thus the amount of data should be kept as small as possible, which is a very challenging task. In the present work we address this by introducing the bin and hash (BAH) algorithm, enabling a computationally very efficient analysis of large data sets. This analysis is possible before training of the ML algorithm of choice has been performed, and even before the electronic structure calculations are carried out, which allows one to guide the selection of the most important structures.

Data set maintenance and analysis as well as atomic fingerprint selection, i.e. finding suitable representations of atomic geometric environments, have been active areas of research accompanying the rise in popularity of ML methods. The use of large and increasingly automatically generated data sets and algorithms to programatically explore PESs [33–35] has led to the need for tools that can deal with the amount and complexity of data. One such method is the dimensionality reduction algorithm SketchMap [36, 37], which can be utilized to group structures together into similarity clusters. More direct tools measuring distances in configuration space [38] and structural similarities of solids [39] are also useful for analyzing collections of structures. Previous attempts based on ML descriptors such as SOAPs [40] have also been successful at establishing a similarity measurement algorithm, and recently a more generalized study has been published, looking at the most common ML descriptors [41] and their relative behavior in describing atomic environments as well as the relationships between property space (in this case energy) and distances in descriptor space.

As an inherent part of most MLP approaches, atomic fingerprint selection, has also attracted a lot of attention. In the wider field of ML this is done with meta-analysis methods, such as hyperparameter optimization [42–44]. Unfortunately these methods are usually rather complex and expensive, requiring multiple training and fitting iterations, which precludes their use for large MLP data sets. Methods specifically designed for MLP also exist, that attempt to refine the contents of these atomic fingerprints. Among them we find attempts at utilizing genetic algorithm optimization [45, 46] to select the best fingerprint sets through evolution, or CUR decomposition [47] to select fingerprints through dimensionality reduction.

In this work we use high-dimensional neural network potentials (HDNNPs) as proposed by Behler and Parrinello in 2007 [7, 48] to illustrate our algorithm, but the algorithm is very general and can be used in combination with many other types of MLPs and atomic environment descriptors. The main idea of the HDNNP approach, which is also used in most other classes of high-dimensional MLPs, is the construction of the total potential energy $E$ of the system as a sum of atomic energy contributions $E_i$ from all $N_{\mathrm{atom}}$ atoms in the system as

$$E = \sum_{i=1}^{N_{\mathrm{atoms}}} E_i. \tag{1}$$

These atomic energies depend on the local chemical environments up to a cutoff radius $R_{\mathrm{c}}$, which has to be chosen large enough to capture all energetically relevant atomic interactions. Typically cutoff values of 6–10 Å are used. The positions of all neighboring atoms in the resulting cutoff sphere must be provided to individual element-dependent atomic neural networks yielding the atomic energies. Many types of descriptors are available in the literature [19, 45, 49–53], and the most frequently used type in the context of HDNNPs are atom-centered symmetry functions (ACSFs) [54], which form a vector $\mathbf{G}_i$ of input coordinates for each atomic neural network that is invariant with respect to rotation, translation and permutation, i.e. the order of the atoms in the system. A detailed discussion of the functional forms of ACSFs and their properties can be found in [54], and here we just use them as placeholders for any ordered set of descriptor values that provides a meaningful structural fingerprint of the local atomic environments.

The atomic neural networks represent the analytic functional form of the HDNNP and contain a large number of fitting parameters, the neural network weights, which are optimized in an iterative training process to reproduce a given reference data set of energies and forces for representative systems obtained from electronic structure calculations. Once the HDNNP has been trained using this data, the energies and forces of a large number of configurations can be computed at a small fraction of the computational costs of the underlying electronic structure method, which enables extended molecular dynamics and Monte Carlo simulations of large systems with close-to first-principles quality. For all details about the method, the training process and the validation strategies for HDNNPs we refer the interested reader to a series of recent reviews [48, 55, 56].

The construction of HDNNPs involves the use of large amounts of data, and the generation of the reference electronic structure data often represents the computationally most demanding step. It is therefore desirable to reduce the amount of data as much as possible by only including those structures—or more specifically atomic environments—which are different enough from the data already included in the reference set to justify the effort of an electronic structure calculation. In addition, also the training process of the HDNNP becomes more time consuming with increasing amount of data. In recent years, active learning [57] has become a standard procedure to identify the most relevant structures [58–61]. Still, the inclusion of a wide range of structurally different atomic environments in the training process is essential for the construction of a reliable HDNNP, as the underlying functional form is non-physical, and the correct physical shape of the potential-energy surface can only be learned if all of its relevant features are included in the training set. Consequently, for each system a compromise between the effort of constructing large data sets and the accuracy and range of applicability of the HDNNP has to be found.

The use of large amounts of data poses several challenges. First, a set of ACSF descriptors has to be defined for each element in the system to construct structural fingerprints that can be used by the atomic neural networks to construct the energy expression of the HDNNP. These ACSFs can be used for the quantification of the similarity of different atomic environments. Typically, a set of 20–100 ACSFs is used for this purpose, which depend on parameters defining their spatial shape [54]. Second, to keep the data sets small, the inclusion of redundant information has to be avoided, which requires an efficient analysis and comparison of the local chemical environments of the atoms given by the ACSF vectors. As we will see below, naive pairwise comparisons are not a viable option for the typical data sets consisting of tens of thousands of structures, each containing up to a few hundred atoms. Third, the costs of the reference electronic structure calculations should be kept as low as possible, but numerical noise that can arise, e.g. from loose but time-saving settings of the electronic structure codes must be avoided. Substantial noise in the data represents contradictory information, which prevents a smooth convergence of the fitting process to low root-mean squared errors for the energies and forces.

In this paper, we propose a simple, fast and efficient algorithm for detecting similar atomic environments as processed by common ML descriptors. The algorithm is based on the well known hash table [62] data structure, and is described in section 2. We use the vector of ACSF values belonging to an atomic environment, the same vector that an atomic neural network in HDNNPs would receive as an input, but we first pre-process it by a BAH approach. Binning is described in section 2.2.3, and the procedure of hashing and the workings of hash tables in section 2.2.4. This creates a numerically unique representation of each environment, where searches for repeated representations are fast and scale well with the number of environments under consideration. In addition, this procedure does not depend on the availability of a trained HDNNP, which is an advantage compared to active learning strategies. The procedure is very fast, and we benchmark it in relation to a naive direct comparison approach in section 2.2.2, with big $O$ notation [62] scaling discussed in section 2.3.

In section 3, we show results from the application of the algorithm. Concrete timings are presented in section 3.1, confirming the scaling expected based on theoretical considerations. Section 3.2 demonstrates how the BAH algorithm reproduces distances in ACSF vector space, while section 3.3 shows the behavior of the algorithm when changing the number of binning subdivisions and the ACSF set description of the data set, and how this can be utilized to qualitatively evaluate the suitability of a given ACSF set, without requiring the lengthy process of previously fitting a potential. Finally, section 3.4 shows how the method can be easily utilized to find similar atomic environments and contradicting information in a data set.

Overall these applications are examples for the well known and complex problem of efficiently finding distances and nearest neighbors in points belonging to multi-dimensional data. Previous approaches include making use of complex binary tree data structures such as kDtrees [62, 63], that can efficiently store data points according to their mutual distance in multi-dimensional space and rapidly reduce a search space due to their binary structure; and dimensionality reduction algorithms such as principal component analysis (PCA) [64, 65] and SketchMap [36] that instead reduce the size of the space under consideration. All of these algorithms are very powerful and suited for their particular applications, but are often too complex and slow for the current goal. Our BAH approach is fast and simple, and works in principle for any dimensionality. It simplifies the process of dimensionality reduction by performing a reduction evenly across the coordinate space instead of centering on the most important directions like PCA and SketchMap.

## 2. The BAH algorithm

### 2.1. Description of the algorithm

Here, we will first give a general overview about the BAH algorithm summarized as pseudocode in code block 1. The details of each of its components will be discussed in the following sections.

As example system we choose zinc oxide. A typical distribution of ACSF values is presented in figure 1 in the form of a stacked histogram plot, for the first 10 ACSFs of a small data set containing 1192 configurations

```
divs = number of subdivisions in ACSF space
for atom_env_i in data set
    for acsf_j in acsf_set
        calculate symmetry function vector Gi={Gj}
find Gjmax and Gjmin across each acsf component Gj
initialize empty hash table Ht
for each Gi vector
    bin Gi vector Bi={Bj},
        Bj=divs*(Gjmax-Gj)/(Gjmax-Gjmin)
    calculate hash Hi=hash(Bi)
    if Hi not in Ht
        store it Ht[Hi]=j index
    else
        count as collision ncolls+=1
        add to existing record in hash table
        Ht[Hi] append(j index)
```

**Code Block 1.** Pseudocode for the bin and hash algorithm.

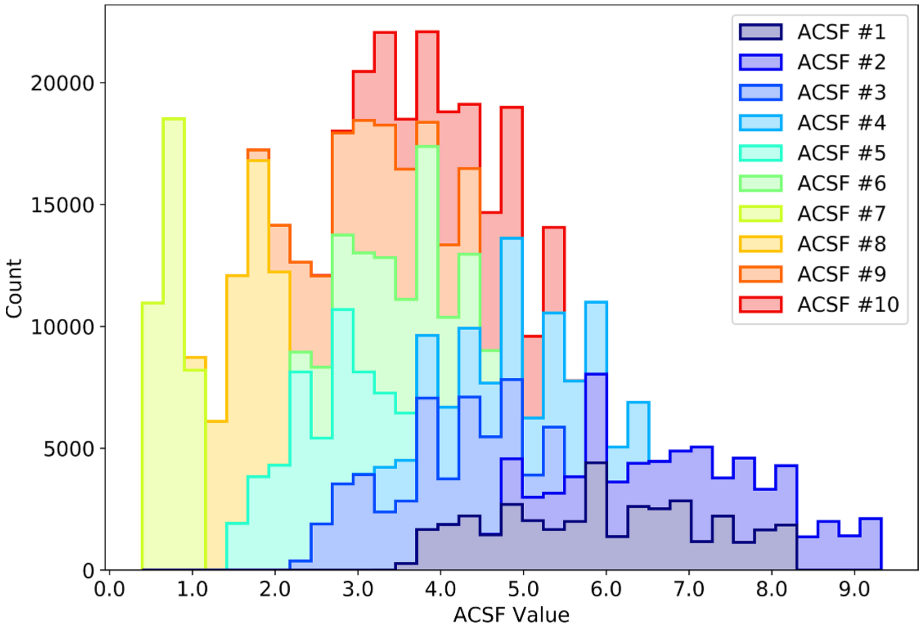
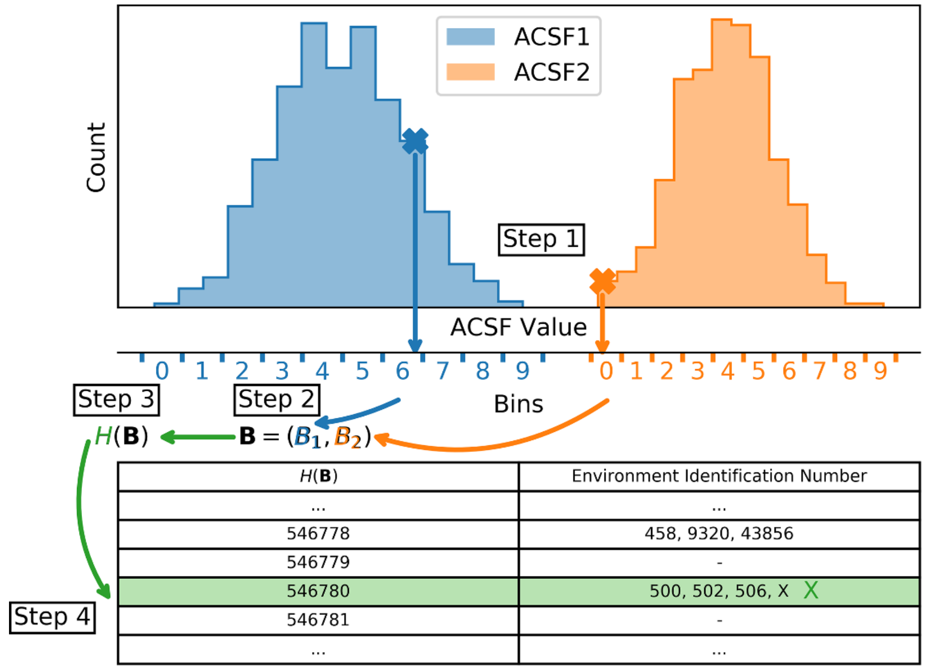


**Figure 1.** Stacked histogram plot of the values of the first 10 radial ACSFs in the ZnO data set describing the atomic environments of the oxygen atoms.

of a ZnO($10\bar{1}0$) surface slab with in total 75 360 atoms, which corresponds to an equal number of atomic environments. The structures included in the data set consist of bulk cut slabs, relaxed slabs, and configurations extracted from MDs, with different number of layers. This means that the atomic environments present across the structures are a mixture of repeated, similar, and unique configurations. Overall, 58 distinct ACSFs are used per chemical element (that is, the same number and type for Zn and O) to describe the atomic environments, and the parameters defining the ACSFs are given in the supporting



| $H(\mathbf{B})$ | Environment Identification Number |
|---|---|
| ... | ... |
| 546778 | 458, 9320, 43856 |
| 546779 | - |
| 546780 | 500, 502, 506, X |
| 546781 | - |
| ... | ... |

**Figure 2.** Illustration of the BAH approach: Each atomic environment in this example is characterized by a two-dimensional ACSF vector $\mathbf{G} = (G_1, G_2)$. In step 1, the histograms corresponding to the ACSFs are generated as a visualization aid. The values of $G_1$ and $G_2$ are highlighted by the crosses for one particular example environment. This ACSF vector is then binned to a pair of integer values, forming the binned vector $\mathbf{B} = (B_1, B_2)$ in step 2. In step 3 the hash $H(\mathbf{B})$ of this binned vector is calculated (since the specific hash function utilized depends on the implementation of the hash table in a given programming language, it is not provided explicitly here). Finally, in step 4 this hash is used (directly or indirectly) to index into the hash table, and add the atomic environment to a counter for similar environments.

information (available online at stacks.iop.org/MLST/2/037001/mmedia). We can see that even for such a relatively small data set the distribution of data already has a rather complex form.

The individual steps forming the BAH algorithm are illustrated in figure 2. Starting from the histogram of ACSF values (shown schematically at the top of the figure), in a first step the range of each ACSF is split into a predefined number of subdivisions, typically between $10^1$ and $10^7$ bins, taking into account the maximum and minimum values present in the data set. This transforms the ACSF vector $\mathbf{G}_i$ for a given atomic environment $i$ from a float-based continuous representation to an integer-valued binned vector $\mathbf{B}_i$ of the same dimensionality (step 2). This binned vector is then hashed generating the one-dimensional hash key $H_i$ (step 3), which is then used for constructing a hash table $(H_t)$ (step 4). The binning achieves two goals at once: getting rid of the floating point representation, which does not allow for an accurate transformation to a hash, since the hash would be numerically very sensitive to the round-off errors of the floating point values, and binning similar ACSF vectors to the same $\mathbf{B}_i$ vector, finally yielding the same hash key. The step of hashing the integer vectors into hash buckets enables a fast and efficient storage and lookup for large data sets. Both parts of the algorithm—binning and hashing—are thus vital for its performance.

Any $\mathbf{G}_i$ vectors that result in a hash collision, i.e. they end up in the same hash table bucket, are deemed to be similar, and—depending on the number of subdivisions—usually exactly the same apart from floating point round-off errors (see section 3.2). The algorithm keeps track of the total number of collisions recorded for a data set and the maximum number of collisions for all the buckets. Additionally, every time a collision is detected the ID of the colliding atomic environment is stored in the hash table in the corresponding bucket, which enables to retrieve the colliding environments afterwards for analysis.

With the introduction of the concepts of bin and bucket, we want to add some remarks to help differentiate them. unfortunately both words have a similar meaning in common speak, which might result in confusion, and the concepts are also deeply related in BAH. A ‘bucket’ is a common name for a slot in a hash table. A ‘bin’ is the name given to the integer associated with an ACSF value that was originally a floating point number, and can also be conceptualized as a subdivision of ACSF space as in figure 2. Since ACSF vectors that bin the same way will also end in the same bucket in a hash table, the two concepts are in the end somewhat equivalent.

An obvious problem of this algorithm is that environments might be very close to the border between two bins. Given two very similar environments, both could be assigned to different bins resulting in completely different hash values, although the atomic configurations are essentially identical. In this case, two environments that should lead to a collision, do not. A straightforward solution to this problem is to use the algorithm with multiple different divisions of the ACSF domain, and to compare the obtained binning. In this way it can be excluded that very similar environments are converted to different hash keys. Still, even when using multiple binnings, the algorithm remains computationally very efficient.

## 2.2. Analysis of the algorithm

Next, we analyze the scaling of the algorithm. This scaling is of particular relevance given the sheer size of the typical data sets used in the construction of MLPs. Many other more sophisticated algorithms work perfectly well when tested on small example cases, but scale very inefficiently for realistic data sets containing tens or even hundreds of thousands of structures, each consisting of many atomic environments. Initially, we comment on the possibility of utilizing neighbor lists. Then, we describe the naive approach of a brute force comparison as a reference, before discussing the behavior of the binning and hashing operations. Finally, we derive the scaling in big $O$ notation [62].

### *2.2.1. Cell-based neighbor lists*

Efficient distance calculation is a common problem in molecular dynamics simulations, since most force fields depend on interatomic distances in one way or another. A simple and common approach is to utilize cell lists [66], where the system is divided into smaller cubic cells, and atoms are assigned to these cells according to their coordinates. If the size of the cells is chosen properly with respect to the cutoff radius of the potential, checking for neighbors becomes simple: for each atom only atoms within the same cell and the directly neighboring cells need to be considered.

It is possible to envision taking this approach to further dimensions, where we would now create cells not in coordinate space but in the higher-dimensional ACSF space. Unfortunately, this simple but robust approach in unfeasible as the computational costs increase rapidly with dimensionality: in a one-dimensional system we need to check the central bin plus two neighbor cells, in two dimensions it is the central cell plus eight cells organized in a square, and so on with the total number of cells to be checked scaling as $3^D$ with $D$ the dimensionality of the space. This is clearly unfeasible for an ACSF set whose dimensionality starts at 20 but can contain as many as 100 ACSFs per atomic environment, and even cases with many hundred functions have been reported [14].

In conclusion, cell-based neighbor lists efficiently reduce the degrees of freedom of the problem by creating cells, which we essentially also use for the binning step in the BAH algorithm. However, it rapidly fails when used in higher dimensions, which we avoid in our BAH algorithm by only finding points in ACSF space that are in the same bin/cell, and by utilizing hash tables to perform this check very efficiently using only a one-dimensional property for the comparison. The cost for this efficiency increase is that the BAH algorithm might miss some neighbors (for example in situations where values fall right at the border between bins, as described at the end of section 2.1), while the cell list algorithm is always robust and finds all the neighbors that are present.

### *2.2.2. The naive approach*

The naive approach to comparing atomic environments is to compare ACSF vectors for each pair of atoms directly. The only obvious simplification is that only atoms of the same element need to be compared. The performance of this procedure is very poor, since it scales linearly with the number of ACSFs, and quadratically with the number of environments in the data set, as for environment number $N$, we need to compare it with all the previous $N-1$ environments already processed.

Hashing and using hash tables solves this scaling problem, since lookup in a hash table is—in principle—a constant time operation [62] that does not depend on the amount of data already stored in the table. Binning is needed before reaching this point, since similar floating point numbers would have very different hash values without a preparatory discretization step.

### *2.2.3. Binning*

Consequently, binning is the first step in the algorithm. The maximum and minimum values of each ACSF depend on the available data set and are known beforehand. For each ACSF, the resulting range is divided into an arbitrary number of intervals and the binning is done according to

$$B_j = \mathrm{nint}\left(\mathrm{divs} \times \frac{\mathrm{ACSF}_{\mathrm{max}} - \mathrm{ACSF}_{\mathrm{val}}}{\mathrm{ACSF}_{\mathrm{max}} - \mathrm{ACSF}_{\mathrm{min}}}\right) \tag{2}$$

where $B_j$ is the bin value for the $j$th ACSF, nint is the nearest integer function, i.e. a round-off to the closest integer; and $\mathrm{ACSF}_{\mathrm{max}}$, $\mathrm{ACSF}_{\mathrm{min}}$, and $\mathrm{ACSF}_{\mathrm{val}}$ are the maximum, minimum, and current value of the ACSF under consideration, respectively. The number of intervals is kept the same for all the ACSF types, although some of them might have larger or smaller ranges (see for example figure 1). A possible improvement to the binning procedure would thus be to aim for a certain density of ACSF values in each division, by tailoring the length and number of divisions to each ACSF.

This binning achieves multiple goals. In the first place, it transforms floating point numbers, which are imprecise and hard to hash, into integers. Floats should not be hashed directly because small changes in the accuracy of the floating point number representation, such as the limited precision when reading it from a file or small deviations resulting from rounding errors, give rise to very different hash values. Integers, on the other hand are easy to convert to a hash.

Additionally, binning provides a sense of 'distance' in the data set. Calculating distances directly from the difference between ACSF vectors suffers from the same scaling problems as the naive approach, and the usefulness of an Euclidean distance decreases with the size of the vector, as it becomes less unique and loses meaning [67] as dimensionality increases. As the bins get smaller, fewer ACSF vectors will coincide, making the algorithm more sensitive only leaving those environments that are more and more similar in the same bucket.

Binning on its own does not solve the problem of the naive approach, since we would still need to do an all-against-all comparison of the individual bin vectors, with integers instead of floats. To solve this, a hash table is required, as described in the following section.

### *2.2.4. Hashing and hash tables*

Hash functions [62] are a family of functions that can map data of arbitrary size to data of fixed size. In effect, a hash is a one-way function, that can assign an integer to any data type. This assignment is not unique as two objects that are different can result in the same hash value, i.e. a hash collision. This conversion is usually non-reversible such that if the hash is known, it is not possible to reconstruct the original object unless by brute force trial and error and comparing the resulting hashes. If two objects share the same hash (a 'hash collision'), they will usually be either exactly equal or very different, which is a desired property in some applications. Small changes to the input object will result in very different hash values, so the hash value in principle cannot be used directly as a measure of distance in input space. Hash functions are used in a variety of fields, such as in cryptography, where passwords as usually stored pre-hashed instead of in plaintext; or in the realm of data-validation and proofing such as in checksums, credit card numbers, bank routing numbers, ISBN book numbers, or blockchains. Hash functions make heavy use of the modulo function and byte-shifting operations.

The properties of a hash function allow us to create a hash table. A hash table resembles an array, but instead of assigning positions sequentially as in a normal array, positions to the hash table's 'buckets' are assigned using the hash function. In effect, the hash value is used to index the hash table array using

$$\text{index} = \text{hash}\%\text{array\_size}, \tag{3}$$

where 'index' is the index to be used when accessing the hash table array, 'hash' is the hash function value of the object of interest, 'array_size' is the size of the array holding the hash table, and % is the modulo operator. The hash will always index an array position, no matter the size of the array.

One apparent problem arises here: The number of bins can reach up to $10^7$ subdivisions per ACSF. For the usual dozens to hundreds of symmetry functions required for a HDNNP data set, this amounts to a large amount of possible bin vectors that grows in a combinatorial fashion. How then is it possible to map all the possible bin vectors into a hash table of restricted size? As mentioned above, hash functions map larger spaces into smaller ones, so collisions are unavoidable [68]. Various solutions exist for solving this problem [62], which are implementation dependent. One possibility, known as separate chaining, is to store all the collided keys in the same bucket as a list. Assignment to the hash table then consists of rapidly finding the correct bucket as in equation (3), followed by a slower (but short) search through the list of key in this bucket. Another possibility, known as open addressing, is to assign keys to the first open bucket address if the current one is already occupied. Assignment of a new key then consists of using equation (3) to find an initial bucket (a fast operation), and then continuing through the bucket addresses until an unoccupied address is found (slower but a short process). Whatever the implementation utilized for collision resolution, it inflicts a computation overhead to all hash table operations, but if the number of collisions is kept low, this is not a problem. In normal operation many of the possible single bin vectors will not be encountered since the data utilized to construct a HDNNP is not completely random, so this is not expected to involve much overhead.

An interesting feature of hashes is that this ansatz results in a constant (when the number of hash collisions is not too high) search, assignment and insertion time of data into the table. In a normal array, if we want to check whether a new object is already present in the array, we need to traverse the array and compare element by element until it is either found and we stop the search early, or we reach the end of the

**Table 1.** Big $O$ notation scaling of the different algorithms under consideration. $N$ is the number of atoms corresponding to the number of atomic environments in the data set. $M$ is the number of functions in the atom-centered symmetry function vector.

| Algorithm | Scaling |
|---|---|
| Naive comparison | $O(M * N^2)$ |
| Binning | $O(M * N)$ |
| Hashing | $O(M * N)$ |
| Hash table lookup | $O(N)$ |

array. In a hash table, we instead calculate the hash of the object and immediately check the corresponding position in the table.

This efficiency comes at the cost of some overhead: requiring more memory for storing the hash table since many buckets might be empty if the hash table is constructed with sequential memory positions, the need to precompute the hash for objects going into the table although hash calculations are usually fast, and dealing with hash collisions when they happen if we want to maintain unique buckets. Due to their properties, hash tables are a basic data structure in computer science [62], often utilized for efficient storage and retrieval of data.

A final advantage of hash tables for the use in this work is that they can be easily stored into a text file for future use. This way, a data set can be preprocessed into a hash table, and future structures can easily be compared against this record to detect repeated configurations. To store the hash table all that is needed is to write the unique binned integer vectors to the file (in an arbitrary order), optionally with a numeric ID associated to the structures in the data set that fall into that bucket of the table for an easy identification. To reconstruct the table, these binned arrays are read and used as members of a new table.

### 2.3. Scaling

Next, we look at the scaling of the different parts of the algorithm in the big $O$ notation [62]. This is important to realize why the naive approach soon becomes unfeasible and how the BAH algorithm improves on it. The results are summarized in table 1. We will consider the case of searching once through a complete data set, and attempting to find repeated atomic environments.

In the following discussion, $N$ is the number of environments in the data set, i.e. the total number of atoms in all structures. $M$ is the number of functions in each ACSF vector corresponding to the dimensionality of our problem. We note that atoms of the same element always have the same ACSF sets, but this is not necessarily true for different elements. The scaling with respect to $N$ can be more important than regarding $M$, since the number of ACSF in a HDNNP is usually less than 100 per element for most systems, while the number of atomic environments can reach millions and has no upper bound.

The following scaling is observed:

- Naive comparison and lookup: Comparison scales at worst as $O(M)$, since we need to compare each element in one ACSF vector to the corresponding element in another ACSF vector, but we might end early if a mismatch is detected. We then need to compare environment 1 with the next $N-1$ environments, environment 2 with the next $N-2$ environments and so on until environment $N-1$ for the last single comparison with environment $N$. This is a mathematical series that in the end scales as $O(N^2)$. Both parts of the algorithm together scale as $O(M * N^2)$.
- Binning: Binning scales with both the number of elements in each ACSF vector—since we need to bin each element individually—as $O(M)$. Additionally, it has to be done for each of the $N$ atomic environments ($O(N)$). Combined it scales as $O(N * M)$. This operation is usually very fast.
- Hashing: Hashing scales weakly with the size of the object being hashed ($O(M)$). There is some dependence on the specific implementation of the hash function (see section 3.1) and the hashing needs to be repeated for each ACSF to be compared ($O(N)$). It is a comparably slow operation compared with a straight division in binning.
- Hash tables: Addition of data to a hash table and lookup are constant with respect to the size of the stored data set (which would be proportional to N), except for hash collisions $O(1)$. This is where the main time saving comes from. We have to repeat this $N$ times, once per hashed array, resulting in a scaling of $O(N)$.

Now we can estimate the total processing times. The naive case is simple, we need to perform $M * N^2$ operations to process the whole data set. For the BAH algorithm, we need to first bin the whole data set, then hash the resulting binned arrays, and finally store the result in the hash table detecting a collision if present. All of these times are additive since they are independent sequential operations. Putting this all together, we obtain

$$\begin{aligned} t_{\text{naive}} &= k_{\text{comp and lookup}} * O(M * N^2) \\ t_{\text{bah}} &= k_{\text{binning}} * O(M * N) + \\ &\quad + k_{\text{hashing}} * O(M * N) + k_{\text{hash lookup}} * O(N), \end{aligned} \tag{4}$$

where each $k$ is the timing constant to perform that operation once, which depends on the actual implementation of each algorithm, the programming language of choice, and the CPU architecture. Notice that the naive approach shows the worst scaling, since it scales as $N^2$, with typical values of $N$ in the order of $10^4$–$10^6$. The BAH algorithm, on the other hand, consists of three linearly scaling additive components. This is tested in section 3.1 for an illustrative example, and the different timing constants estimated, for a Python implementation.

### 2.4. Implementation

The algorithm has been implemented in Python 3.5, using the dict [69] data structure, which is a hash table with the possibility to associate arbitrary data to each hash bucket. The set [69] data structure is similar and can also be used, but can only store the hashed object and no other associated data. It can also be implemented easily in many other languages, since hash tables are a widely used data structure, and only pointers or allocatable arrays are needed to implement them from scratch. The dict object in Python already incorporates the step of hashing the data, so no explicit hash function is required in this case, and the actual implementation of the hash function is not relevant to the result as long as it avoids as many spurious collisions as possible.

The algorithms is straightforward to parallelize if this is required for larger data sets, or for non-synchronous processing, e.g. using a compute cluster associated with a database. This is due to the fact that hash tables can be easily combined. A central master process can hold the copy of the hash table, and dispatch binning and hashing operations to the slave processes; or each slave process can hold its own hash table and report back to a central process, which combines the slave sub-tables into a master hash table.

## 3. Results

### 3.1. Performance and timings

For illustrative purposes, we present the timings and scalings of the naive and BAH algorithms on randomly generated values, as obtained from Python3.5 on a Intel Core i5-5300U CPU 2.30 GHz. Figure 3 plots the behavior of the different algorithms for increasing data sets.

As can be seen in figure 3(a), the naive algorithm for the comparison of the atomic environments scales with the square of the data set size, while the BAH algorithm in figure 3(b) scales linearly. In the logarithmic scale of figure 3(c) combining the data of panels (a) and (b), it can be clearly seen that the costs of the naive algorithm increase much faster than those of the BAH algorithm. Figure 3(d) shows the speedup (the relative time gain, $t_{\text{algo}}/t_{\text{naive}}$ for any of the sub-algorithms involved in BAH) between the BAH and the naive algorithms. Notice that this speedup *increases* as the data set size increases, since the naive approach scales as the square of the data set size but the BAH scales linearly. Consequently, the larger the data set becomes, the faster the BAH approach becomes with respect to the naive approach. Figure 3(e) shows that the hashing algorithms scales linearly with the size of the ACSF vector under consideration, but is extremely fast for typical vector dimensionalities. Finally, figure 3(f) confirms that, as expected, operations regarding the hash table object—assignment to the hash table, and looking up if an object belongs to the hash table—remain constant in time with data set size.

From these analyses and data we can estimate the different proportionality constants of equation (4), they are compiled in table 2. Notice that the Naive and BAH halves of the table have different units. The fastest part of the BAH algorithm is the hash calculation ($k_{\text{hashing}}$), while the bottleneck in the current implementation seems to be the binning ($k_{\text{binning}}$). This is probably due to the division and rounding nearest integer operations involved in binning, and it could probably be improved with some vectorization or better numerical libraries. Not considered here is the required I/O to read ACSF data from a file, which might become a more serious bottleneck for larger data sets, but is however common to both algorithms. The values obtained here represent only an approximate order of magnitude since this will change significantly for different implementations and computing architectures.

### 3.2. Analysis of the distance in symmetry function space

An interesting question is how the algorithm reflects distances in ACSF space, since some information is lost in the process of binning and hashing of the atomic environment vectors. Hashes themselves are not a useful measure of distance since the resulting hash is not smoothly continuous with respect its inputs, but we would expect similar ACSF vectors to end in the same bucket. A reliable binning of only similar structures is an

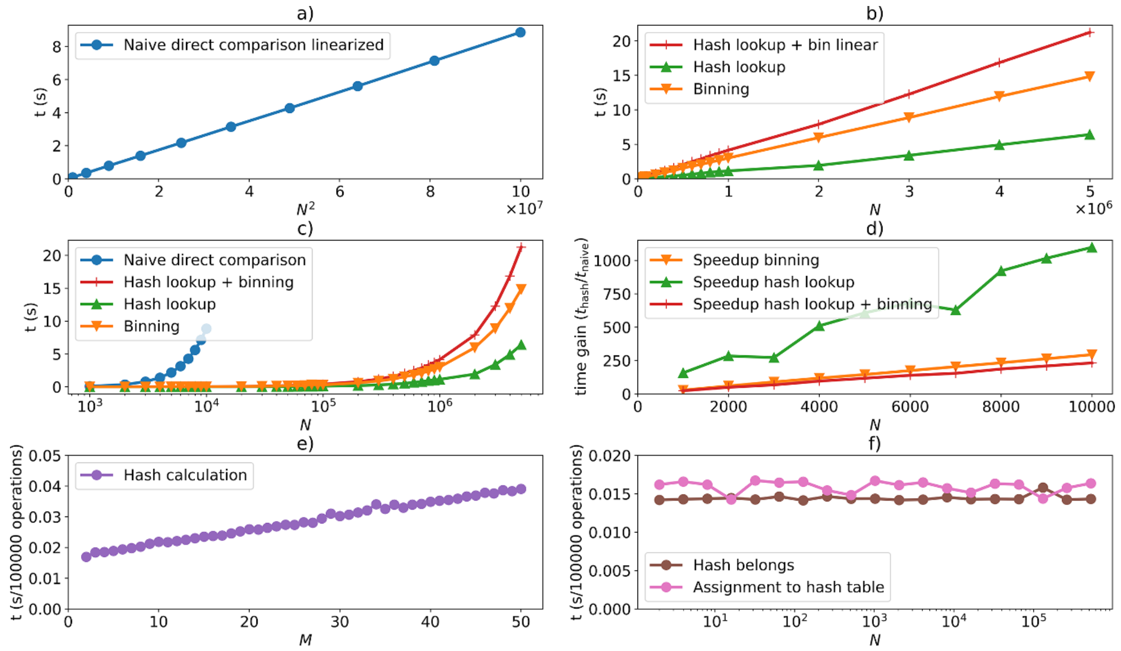


**Figure 3.** Plots of the timing of the different algorithms with increasing system size. (a) Naive lookup vs. squared size of data set. (b) Different parts of the BAH algorithm vs. size of data set, with $10^1$ subdivisions. Different numbers of subdivisions do not appear to have a relevant effect on the speed of the algorithm. (c) All algorithms together in log scale for comparison. (d) Relative speedup or time gain of the different parts of the BAH algorithms compared to the naive approach, calculated as $t_{\text{algo}}/t_{\text{naive}}$, with $t_{\text{algo}}$ the timings of the different parts of the algorithm from (b). (e) Scaling of the hash calculation with ACSF vector size, per 100 000 operations. (f) Behavior of hash table operations with data set size, per 100 000 operations.

**Table 2.** Estimated scaling constants for the different parts of the naive and BAH algorithms, at a constant $M = 10$ (scaling is assumed linear for other $M$ values, in the cases where relevant). Units are in seconds required per operation (s op$^{-1}$). The inverse constant is also given providing the number of operations per second (op s$^{-1}$). Note that the naive algorithm only seems 'faster' because it is expressed in terms of op$^2$. The different scaling constant units are a necessary consequence of some algorithms scaling linearly while the naive algorithm does so quadratically.

| | Naive | |
|---|---|---|
| Constant | Value (s op$^{-2}$) | op$^2$ s$^{-1}$ |
| $k_{\text{comp and lookup}}$ | $8.8 \times 10^{-8}$ | 11.000.000 |
| | BAH | |
| Constant | Value (s op$^{-1}$) | op s$^{-1}$ |
| $k_{\text{binning}}$ | $3.0 \times 10^{-6}$ | 336.000 |
| $k_{\text{hashing}}$ | $1.8 \times 10^{-7}$ | 5.500.000 |
| $k_{\text{hash lookup}}$ | $2.9 \times 10^{-7}$ | 3.400.000 |
| $k_{\text{BAH global}}$ | $4.2 \times 10^{-6}$ | 238.000 |

important condition for the BAH method to be useful. For this purpose, we now investigate all the ACSF vector distances obtained for atomic environments that fall in the same bucket using different subdivisions of the ACSF space. We define a relative distance in ACSF space, $\delta_{ij}$ between atoms $i$ and $j$ of the same element, as

$$\delta_{ij} = \frac{|\mathbf{G}_i - \mathbf{G}_j|}{0.5(|\mathbf{G}_i| + |\mathbf{G}_j|)} \tag{5}$$

where $\mathbf{G}_i$ and $\mathbf{G}_j$ are a pair of symmetry function vectors corresponding to atomic environments that ended up in the same bucket, and which are thus similar for the BAH algorithm. We plot a series of histograms of the calculated distances in figure 4 for different subdivision numbers. Most of the distances in the histograms are close to zero as expected. Notice that as we increase the number of subdivisions, the maximum intra-bucket distance drops quickly due to the more stringent criterion for structural similarity in the binning process, becoming close to the floating point noise (either due to the limited precision of floating point numbers in a computer representation a.k.a. the 'machine epsilon', or the limited precision of data such as coordinates and ACSF values held in text format) for the maximum number of subdivisions such that the

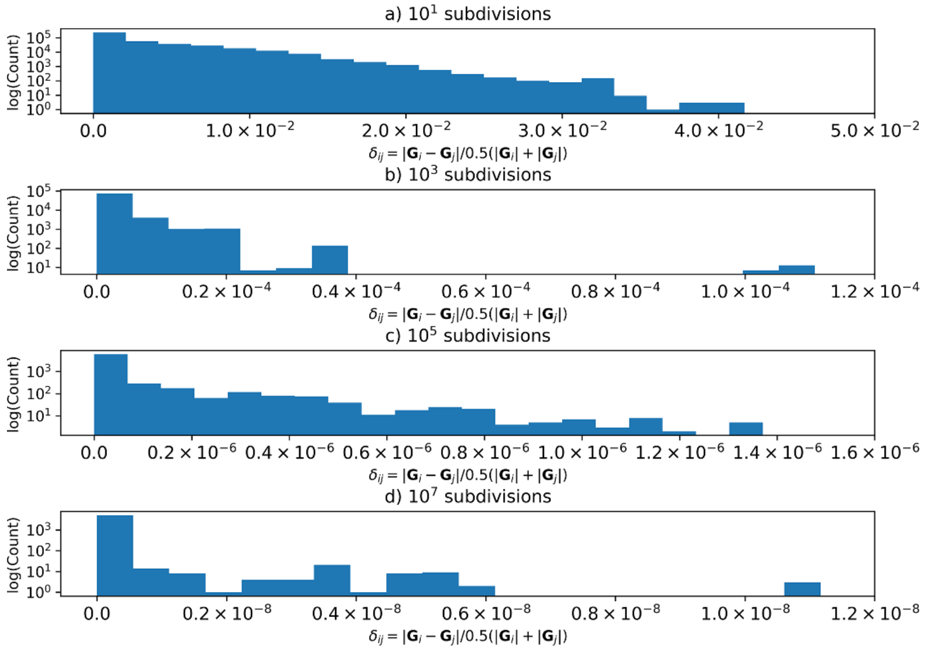


**Figure 4.** (a)–(d) Histograms for the typical intra-bucket ACSF relative distance ($\delta$) values for different subdivisions ($10^1, 10^3, 10^5, 10^7$) in the ZnO slab data set. Other intermediate subdivisions ($10^{\ 2}, 10^4, 10^6$) exhibit similar behaviors. The counts axis is logarithmic for better visualization.

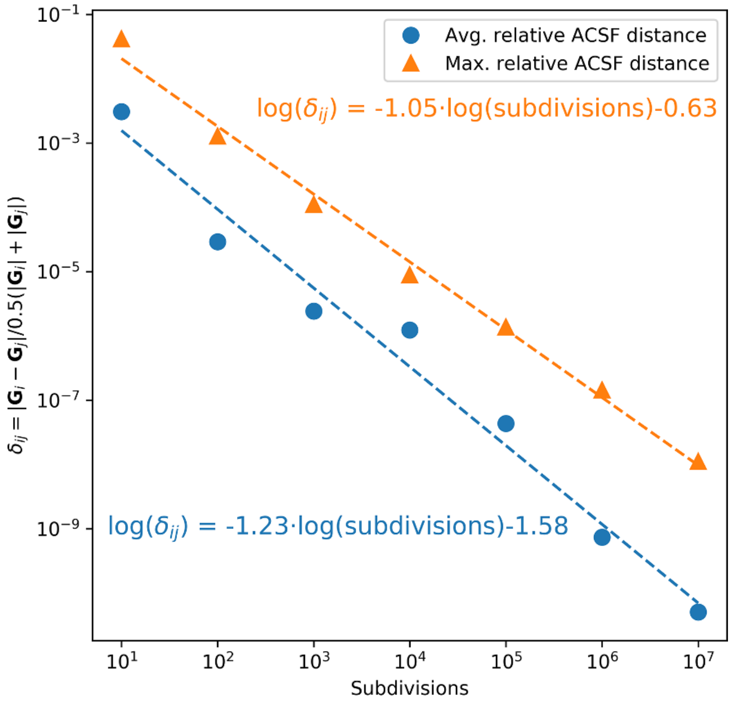


**Figure 5.** Maximum and average intra-bucket relative distances for the histograms in figure 4 versus number of subdivisions, in log scales. Notice that they follow approximately linear relationships, and trendlines with corresponding fitting equations are included.

differences for many subdivisions are probably due to round-off errors and float-to-string conversions rather than significant distances in ACSF space. Consequently, the histograms show that the BAH algorithm is indeed closely correlated to distances in ACSF space, up to a given maximum distance depending on how the multi-dimensional space is subdivided for the binning step.

Interestingly, as shown in figure 5, the maximum and average $\delta$ obtained from these histograms follow a linear relationship with the number of subdivisions, on a double logarithmic scale. Therefore, changing the subdivisions parameter allows us to fine-tune the maximum detected atomic environment distance in a predictable way.

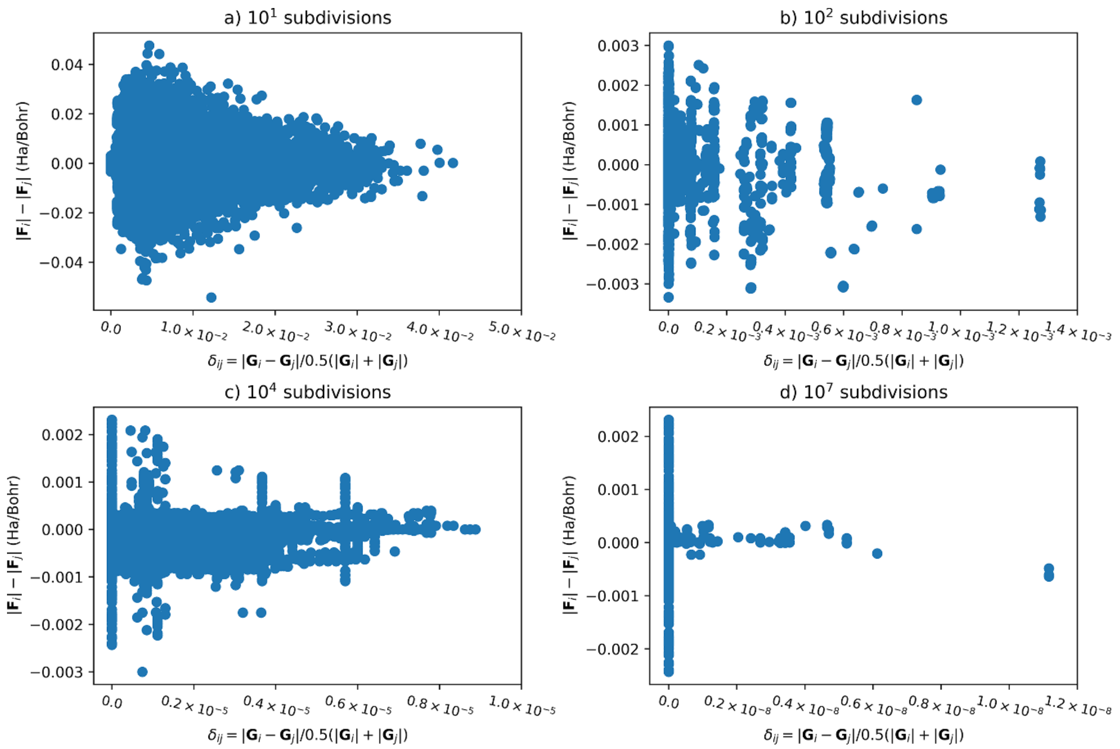


**Figure 6.** Difference in force magnitude vs. the ACSF relative distance, $\delta$, for different subdivisions of the BAH algorithm applied to the ZnO slab data set. The points present in each subplot are not always the same, since the plots are generated from environments that collided for a given number of subdivisions. Notice the difference in the scale of the $X$ and particularly the $Y$-axis for (a) when compared to (b)–(d); the force spread for structures with $\delta_{ij} \approx 0$ is due to remaining numerical noise in the DFT data.

Given this behavior of the distances in ACSF space, it is also of interest to study the corresponding behavior of the properties associated to each atomic environment such as the atomic forces. In figure 6 we plot the difference in force magnitude [70] vs. the ACSF relative distance, $\delta$, for different subdivisions. As shown in (a), there is a relationship between the two quantities, since one would expect that atoms whose environments/ACSF vectors are similar should also present similar forces. Despite this, the relationship is not strong, since distances in 'force space' do not necessarily transfer linearly into ACSF space [41]. As the number of divisions increases and the force vectors considered correspond to closer environments, the force distance quickly falls. In the end (d), this force distance corresponds to the numerical noise present in the reference DFT data, since he environments detected are actually identical (up to numerical noise).

### 3.3. Results for different divisions and symmetry functions

An interesting question is how the resolution power of the algorithm, i.e. the ability to differentiate ACSF vectors, changes as we increase the number of binning subdivisions, and as we change the ACSF descriptor set itself. For this purpose, we have analyzed the ZnO $(10\bar{1}0)$ slab data set.

A count of collisions was performed on this data set, which as described before occur when two environments end up in the same hash table bucket, due to their binned vectors being the same, which implies their original ACSF vectors were at least similar. We keep track of the total number of collisions, and the maximum number of collisions in a single bin, for different divisions and an increasing ACSF set.

We would expect both total and maximum number of collisions to go down as both divisions and numbers of ACSFs increase, since more divisions means that environments need to be more similar in ACSF space to collide (see section 3.2) and more ACSFs lead to a more granular description of each environment. Eventually, this count converges as we are left with only the environments that are exactly the same, which can happen in a data set due to repeated parts of a configuration for example, if parts of a slab far away from a chemically modified region remain essentially constant. This is in fact found in figure 7. Here we have performed the BAH analysis on an increasing number of ACSFs, in the order presented in the supporting information.

In this figure we note that in (a), collisions go down extremely quickly as we increase the ACSF descriptor set, and then plateau with a slight downward trend that is hard to observe due to the scale of the plot. The line with $10^5$ divisions seems to offer the most granularity, showing changes across the whole ACSF set under

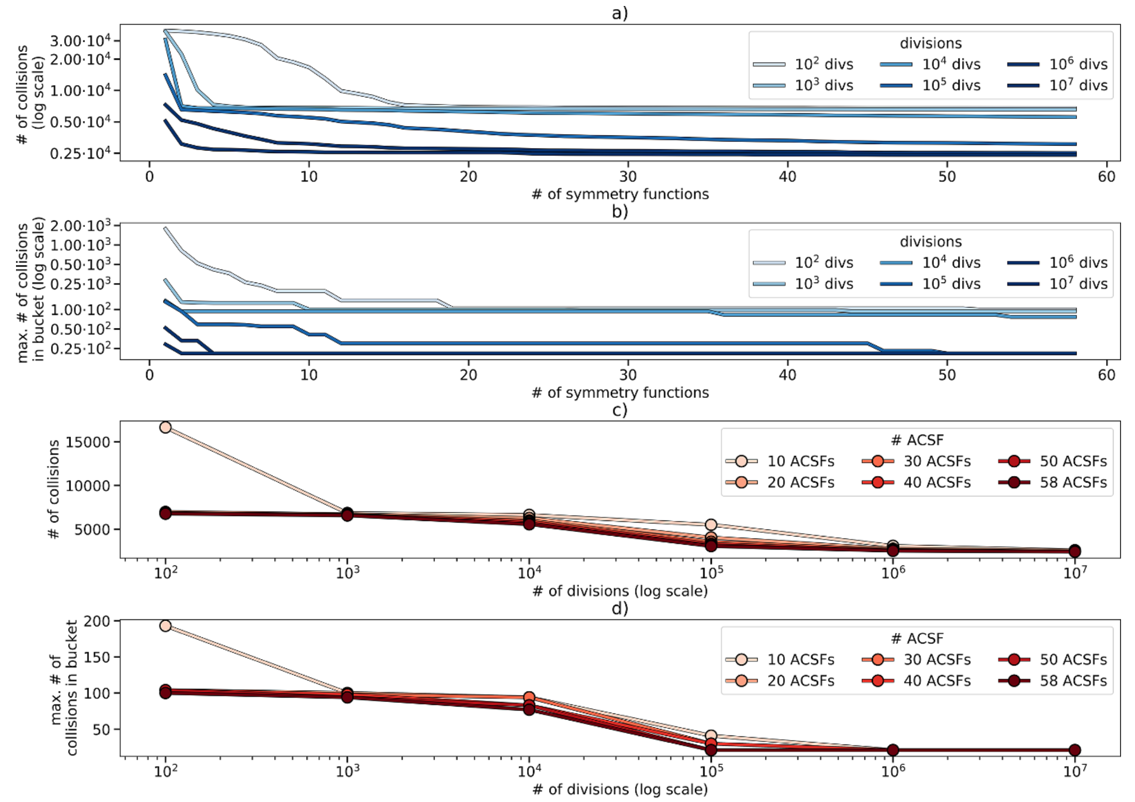


**Figure 7.** Panels (a) and (b) show the total and maximum number of hash table collisions, i.e. configurations that hash into the same bucket due to similarity of their ACSF vectors, vs. number of ACSFs, for different binning divisions. Panels (c) and (d) show the same properties as a function of the number of binning divisions, for different numbers of ACSFs.

consideration. Being able to differentiate chemical environments is a necessary (but not sufficient) condition for a good HDNNP fit, in which case the BAH algorithm could be utilized to identify a minimum floor to the size of the ACSF set.

At this point, the question arises of which subdivision range is ‘best’ to describe a given data set, and whether this is actually dependent on the specific data set. As can be seen from figure 5, the number of subdivisions roughly corresponds to the symmetry function space distance between the collided atomic environments. As such the ‘right’ subdivision range depends on whether we want to detect environments that are only roughly similar or exactly the same, and there is not a single ideal value. For the type of analysis presented in figure 7, a lower number of subdivisions (in the range of $10^2$–$10^4$) provides a more granular behavior in the number of collisions vs. symmetry functions utilized, which results in an easier to analyze trend. For detecting contradictions (see section 3.4 we require environments that are either extremely similar or exactly the same, in which case the upper range of subdivisions ($10^6$–$10^7$) is better suited.

Whether the number of subdivisions required depends on the specific data set is harder to evaluate. Since our data sets are derived from physically ‘reasonable’ configurations corresponding to chemical systems, they share roughly the same properties, with some differences depending on the involved elements, states of matter present, energy ranges covered, etc. The parameters of the trendlines in figure 5 might depend on the specific composition of the data in the data set, but as long as the relationship with ACSF space distance remains, the specific parameters are not crucial.

In the end no specific number of subdivisions is ideal for every situation, and this has to be tested with each data set and adapted to each desired analysis, but the BAH process is so fast that binning a data set multiple times is not a problem. Our recommendation is to test three widely separated orders of magnitude of subdivisions ($10^3$–$10^5$–$10^7$), and refine according to the results.

### 3.4. Comparison of atomic environments and conflicting information

The result of running the BAH algorithm is a list of environments that fall into the same bucket. That is, we obtain a list of collisions representing structurally similar atomic environments as defined above. This is valuable information and can be used to predict if a new configuration obtained from a simulation employing the HDNNP is sufficiently different from the available data to justify an inclusion in the reference data set to refine the potential. All the atomic environments in a large number of structures structure

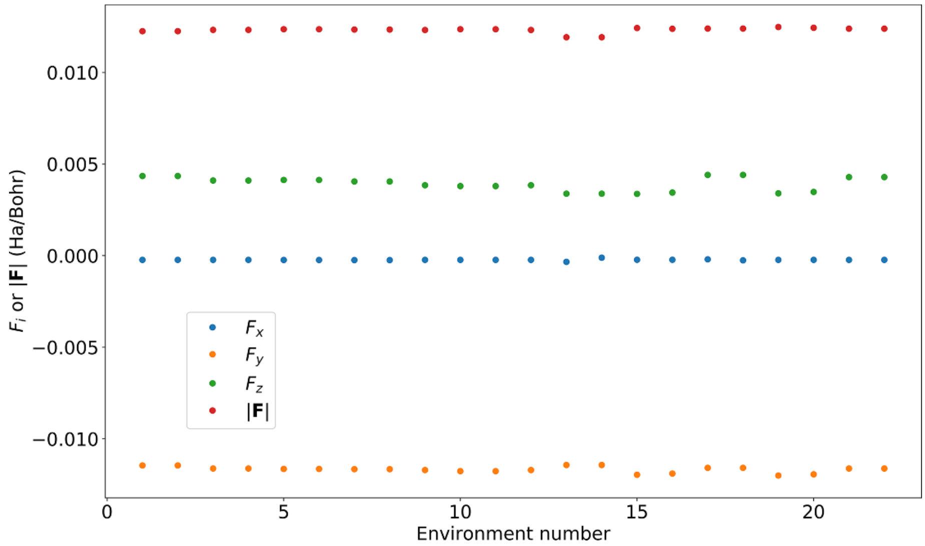


**Figure 8.** Force components and force vector magnitude for 22 environments found in a collision bucket. Note that although the ACSF vector for all environments is identical, there are slight differences in force values arising from numerical noise in the DFT calculations.

obtained in long validation simulations can be screened in this way, and for a most efficient use of subsequent electronic structure calculations it is possible to identify those structures from this pool, in which the highest fraction of environments is sufficiently different for the existing reference data.

Another possibility is the search for contradictions in the data set. Contradictions in this case means atoms whose ACSF sets are similar, but their derived properties (any per atom predicted property, such as force, spin, charge, etc) differ by more than an acceptable threshold. This could be due to a too small ACSF set or cutoff radius of the ACSFs, which does not allow one to correctly distinguish chemically different atomic environments, due to the neglect of long-range interactions beyond the cutoff radius, or due to incorrect electronic structure data resulting, e.g. from a poor convergence level. Contradictions are detrimental to the fitting process, since in case of conflicting data the HDNNP cannot reach a high fitting accuracy [54].

If we apply this analysis to our data set, with $10^5$ binning divisions we find that the bucket with most collisions contains 22 environments. The ACSF vector of these configurations is identical, but plotting their DFT force components [70] and magnitude results in figure 8. We can see that the forces are not exactly identical, but they are within the expected error margin for the HDNNP [71], i.e. below about 100 meV/Bohr. In this case, no contradiction is detected, but in other situations we found structures that have not properly been converged for various reasons. Identifying and eliminating these data substantially improved the HDNNPs in this case. For larger data sets, the points within buckets could be automatically analyzed, and a contradiction warning raised if the force difference is above a given threshold.

### 3.5. The BAH algorithm and quality of NNP fit

The question now arises, whether the BAH algorithm can be actually used to systematically improve the training procedure of a NNP or the constitution of a dataset. To prove that this is possible, we have first continued the line of thought in figure 7. This figure shows that there is, as could be expected, a relation between the number of similar environments detected and the number of ACSFs utilized to describe a collection of atomic environments. Now, figure 9 shows the interrelation between number of descriptors, detected collisions, and accuracy of a fit. Subfigure (a) shows that as the number of utilized ACSF increases, the root mean squared error (RMSE) of a NNP fit for the ZnO dataset decreases, both for predicted energies and forces, plateauing at around 30 ACSFs. Since the number of detected similar environments is also related to the number of utilized symmetry functions, we can find a correlation between RMSE and detected collisions, as plotted in subfigure (b) for different number of divisions in the BAH algorithm. The curves for $10^5$ and $10^4$ divisions show the highest ‘sensitivity’ here, covering a larger range in the $x$-axis. Subfigure (c) plots the curve for $10^5$ divisions on its own, and adds a third axis with the corresponding number of ACSF to each collisions data point, highlighting the three-way relationship between the involved quantities.

One needs to be careful with the analysis here, since correlation does not necessarily imply causation. But a logical connection between the three quantities can be easily found: when not enough descriptors are utilized for a given dataset, the NNP cannot correctly differentiate between similar atomic environments, and thus has problems correctly assigning and predicting energies and forces. This also happens to the BAH

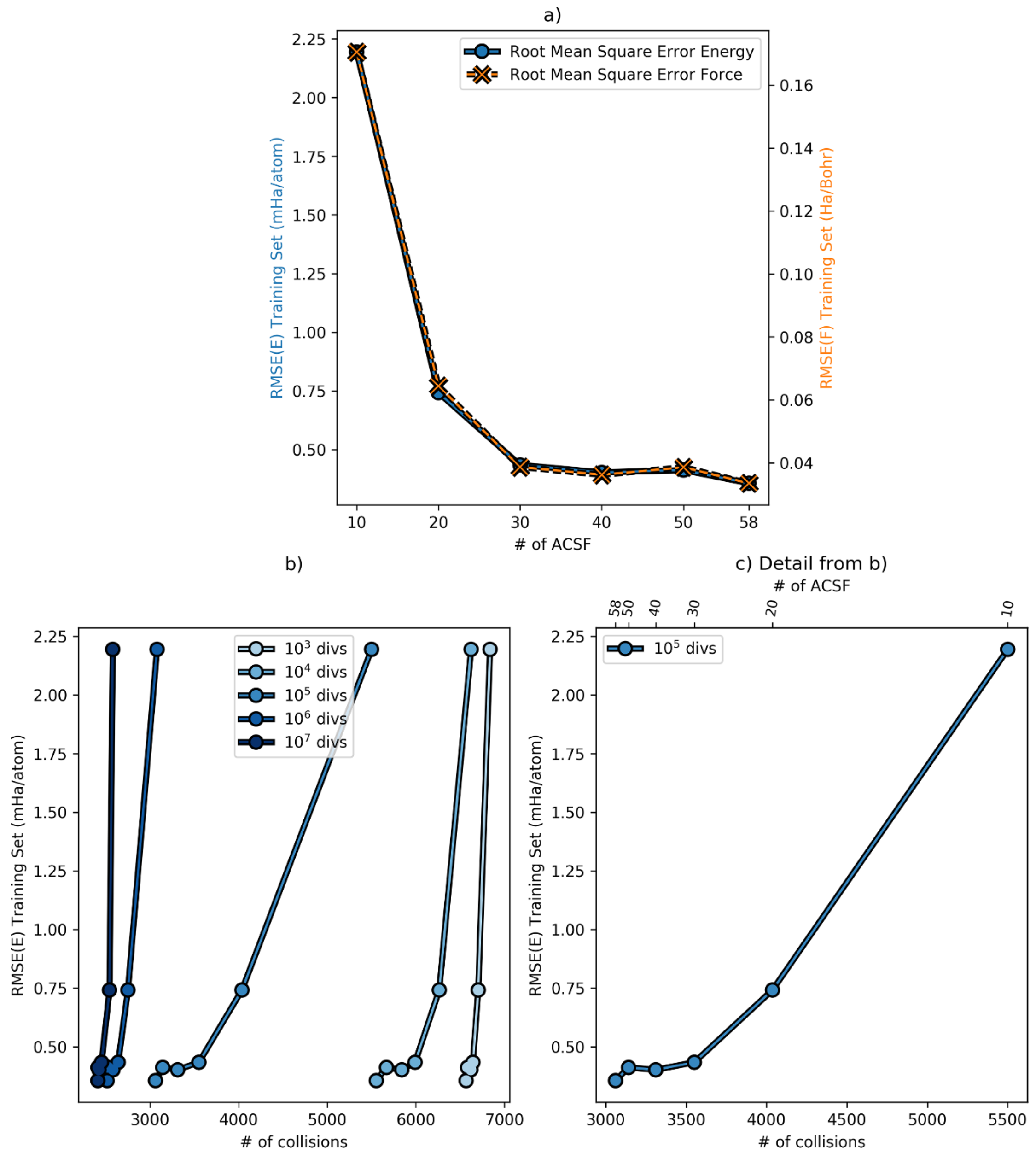


**Figure 9.** (a) Root mean squared error (RMSE) for energy and forces for the ZnO dataset with different number of ACSFs as descriptors. (b), (c) Energy RMSE at different ACSF numbers vs. number of collisions at the same ACSF number detected by the BAH algorithm with different numbers of divisions. Plots of force RMSE have a similar shape. The dependence of the number of collisions and RMSE on the number of ACSF descriptors is made more explicit in (c) for the $10^5$ divisions curve.

algorithm, which detects more collisions when atomic environments cannot be fully resolved. Thus the BAH method can be used as a rough estimate of the quality of a given descriptor set, without the need to perform costly fits. This is important since each extra descriptor in a NNP adds computational cost, not only during the fitting procedure, but also when utilizing the NNP to conduct simulations. There is thus a competition between higher fit accuracy (which always has diminishing returns past a certain number of ACSFs), and computational efficiency.

As a final test of the utility of the BAH algorithm, we can attempt to use it to ‘curate’ a dataset. In normal operation, this would be done by generating new structures during a simulation, applying the BAH algorithm to detect those with novel atomic environments, processing those with the electronic structure method of choice, and adding them to the dataset to repeat the procedure.

To emulate this procedure, structures have been removed from the current dataset, the remaining dataset used to fit a NNP, and the predictive power of this NNP on the removed structures evaluated once again through the RMSE. If the removed structures contain unique environments that are not present in the remaining dataset, the NNP will have trouble predicting energies and forces of the removed structures. On the other hand, if they contain repeated environments, the NNP is able to learn those environments from the

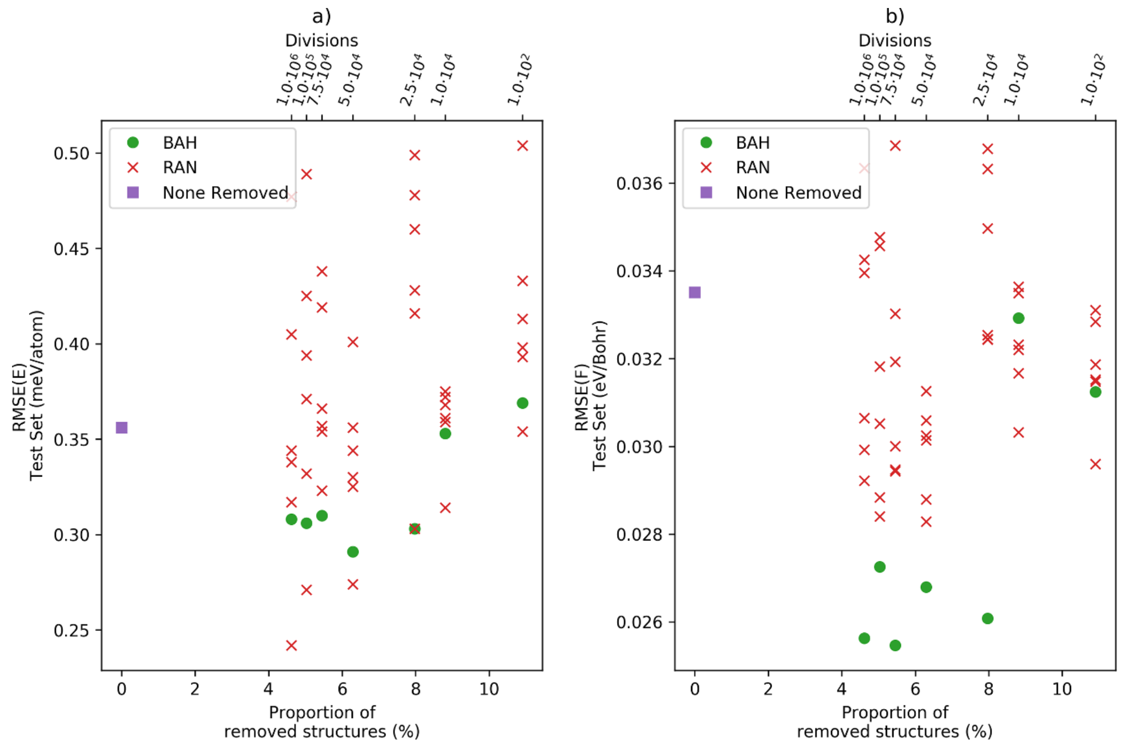


**Figure 10.** RMSEs for (a) energies and (b) forces, for structures removed from the ZnO dataset utilizing the BAH algorithm at different numbers of divisions, or a random selection approach. A reference point for the accuracy of the NNP on the whole dataset is also presented. BAH performs similar or better to random sampling for energies, and better than random until 8% of removed structures for forces.

remaining dataset and accurately predict properties for the removed structures. Due to how the dataset is set up, unfortunately entire structures need to be removed instead of single atomic environments, but for other systems it might be possible to instead construct a library of isolated atomic environments and thus operate at this lower level.

Two structure removal methods are compared. In the first method, the BAH algorithm is applied to the dataset with different numbers of divisions, structures are ordered by the number of atomic environments shared with other structures, and then structures are removed from the top of this list. Every time a structure is removed, the number of matches of other structures is also reduced, so the ordering of this list needs to be updated each time. Deletion proceeds until every structure has a given maximum number of matches. In the second method, as a control, structures are simply removed at random, until the same number of structures as with the BAH method have been removed. Since the random method can result in varying qualities of fits, and the training procedure in itself has some non-deterministic elements, it is repeated six times for better sampling.

Figures 10 and 11 show the results of this comparison. As can be seen in figure 10, the dataset curated with BAH can predict energies in the removed (test) dataset with either similar or better accuracy than the random curation, in the range of 4%–12% removed structures. For the forces, which are harder to predict, the BAH curation overperforms random sampling up to 8% of removed structures. The goal of this figure is not to show that BAH selection improves the predictive power of the NNP, but that similarity detection and removal with BAH does not results in a *worse* dataset, particularly when compared to a random selection approach. Figure 11 shows a similar outlook, but now comparing the energy and force RMSE differences (both absolute and relative) across 20 fitting epochs. The NNP fits with similar accuracy on the remaining training set for both methods, but the error on the test removed structures is up to 20% worse for the random case.

It is thus possible to assert that the BAH algorithm has managed to choose those structures that can be removed from the dataset without affecting predictive quality of the NNP, that is, it has correctly detected those environment that are repeated and thus superfluous in the dataset. This is true up to a certain number of removed structures: this number depends on the divisions utilized when applying the BAH algorithm, and as divisions become larger more matches are found and more structures removed. Presumably, past a certain point the resolution of the algorithm becomes too rough and it starts removing environments that are not similar enough (read as, unique), and thus predictive power becomes comparable to a random selection.

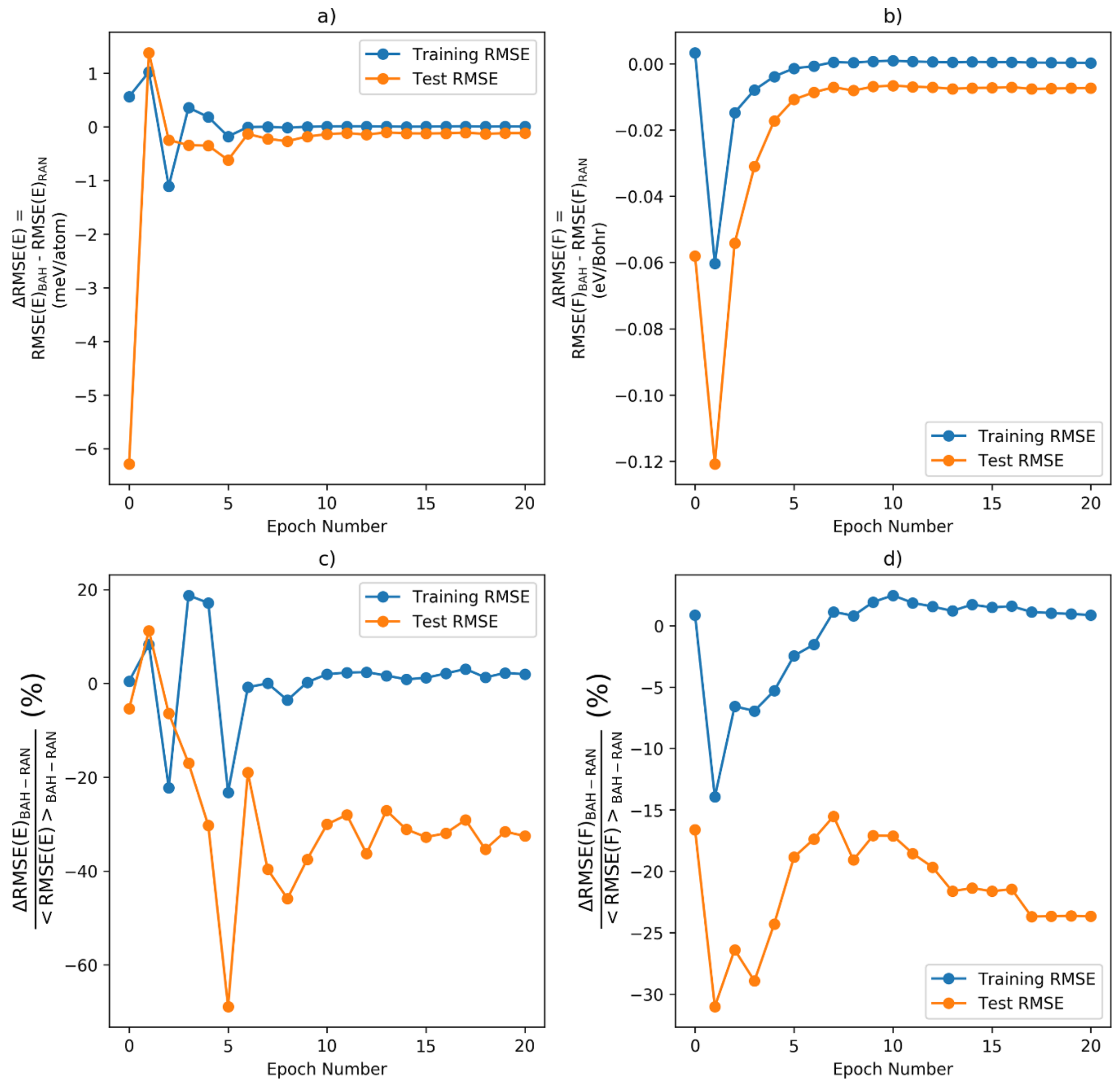


**Figure 11.** Comparison of the accuracy of a NNP trained on a modified dataset by removing 5% of the structures with a BAH (with $10^5$ divisions) or a random approach. The removed structures are put into a test dataset. (a), (b) Absolute difference in energy and force RMSE vs. training epoch ($\Delta\mathrm{RMSE}_{\mathrm{BAH-RAN}} = \mathrm{RMSE}_{\mathrm{BAH}} - \mathrm{RMSE}_{\mathrm{RAN}}$, negative values mean that BAH is more accurate). Both approaches result in a similar accuracy for the training set, but BAH is favored in the removed/test set. (c), (d) As in the previous subfigures, but a percentage difference (with $\langle \mathrm{RMSE}_{\mathrm{BAH-RAN}} \rangle = 0.5 \cdot (\mathrm{RMSE}_{\mathrm{BAH}} + \mathrm{RMSE}_{\mathrm{RAN}})$, once again negative values mean BAH is more accurate). Once again both methods perform similarly for the train data, but BAH performs up to 20% better on the test set.

This is still a good result: removing 5%–8% of a dataset without apparent loss of quality is a marked achievement, and proves how the algorithm could be used in a practical situation to filter new structures before performing costly fits or even costlier reference calculations.

## 4. Supporting information

In the supporting information we present:

- A list of ACSF parameters for the studied ZnO slab data set.
- The code utilized to perform the scaling tests in section 3.1.

## 5. Conclusions

In this work we have presented a BAH method, which allows a computationally very efficient comparison of a large number of geometric atomic environments, which are used in the construction of modern MLPs. In case of HDNNPs, which we use as a typical example here, these environments are usually described by vectors of ACSFs. We show that the ability of the method to identify similar atomic environments can be systematically controlled by the number of subdivisions used in the binning process of the ACSF vectors, but also a large number of alternative descriptors proposed in the literature is equally applicable.

The method is fast, simple and robust with many applications in the construction of MLPs. One example is the identification of redundant atomic environments in the reference data sets used for the construction of the potential as a basis for the decision which structures should be included in the training set. This is an essential step, as a systematic coverage of the configuration space is very important for obtaining reliable potentials, which an excessive amount of data would turn the construction and use of the potentials unfeasible. Due to the use of hash functions and tables, the method can process millions of candidate atomic environments in a number of minutes, being much faster than a naive direct comparison approach. The obtained information can be stored in data libraries that can be efficiently searched at a later stage if needed. We note that in this context the BAH algorithm is complementary to the use of active learning, as the BAH algorithm is based on the geometric structure and its description, while it does not require the availability of trained MLPs as no property evaluations are needed. Active learning on the other hand is based on the comparison of predicted properties, which allows one to focus on the reliability of the target property, while it depends on the availability of preliminary models and their evaluation.

Another application is the validation of the structural resolution capabilities of the descriptors used for the discrimination of different atomic environments. Poor descriptor sets result in a large number of environments appearing erroneously to be structurally similar although local physical properties like forces substantially differ. Finally, the method can be used to identify conflicting data in the training set, which might result from an insufficient convergence level of the reference electronic structure calculations and other types of errors resulting in inconsistent information. Consequently, the BAH method has been found to be a useful tool for solving a variety of challenges emerging in the construction of MLPs, with many additional potential applications in other fields requiring the efficient comparison of structural features, such as genetic algorithms [72], minima hopping [73], and kinetic Monte Carlo [74] simulations.

## Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

## Acknowledgments

We thank the Deutsche Forschungsgemeinschaft (DFG) for financial support (Be3264/10-1, Project Number 289217282 and INST186/1294-1 FUGG, Project Number 405832858). J B gratefully acknowledges a DFG Heisenberg professorship (Be3264/11-2, Project Number 329898176). We would also like to thank the North-German Supercomputing Alliance (HLRN) under project number NIC00046 for computing time.

## ORCID iDs

Martín Leandro Paleico https://orcid.org/0000-0002-8427-0221
Jörg Behler https://orcid.org/0000-0002-1220-1542

## References

[1] Behler J 2016 Perspective: machine learning potentials for atomistic simulations *J. Chem. Phys.* **145** 170901
[2] Botu V, Batra R, Chapman J and Ramprasad R 2017 Machine learning force fields: construction, validation and outlook *J. Phys. Chem.* C **121** 511–22
[3] Deringer V L, Caro M A and Csányi G 2019 Machine learning interatomic potentials as emerging tools for materials science *Adv. Mater.* **31** 1902765
[4] Hohenberg P and Kohn W 1964 Inhomogeneous electron gas *Phys. Rev.* **136** B864–71
[5] Kohn W and Sham L J 1965 Self-consistent equations including exchange and correlation effects *Phys. Rev.* **140** A1133–8
[6] Blank T B, Brown S D, Calhoun A W and Doren D J 1995 Neural network models of potential energy surfaces *J. Chem. Phys.* **103** 4129–37
[7] Behler J and Parrinello M 2007 Generalized neural-network representation of high-dimensional potential-energy surfaces *Phys. Rev. Lett.* **98** 146401
[8] Jiang B and Guo H 2013 Permutation invariant polynomial neural network approach to fitting potential energy surfaces *J. Chem. Phys.* **139** 054112
[9] Lorenz S, Groß A and Scheffler M 2004 Representing high-dimensional potential-energy surfaces for reactions at surfaces by neural networks *Chem. Phys. Lett.* **395** 210–15
[10] Manzhos S and Carrington Jr T 2008 Using neural networks, optimized coordinates and high-dimensional model representations to obtain a vinyl bromide potential surface *J. Chem. Phys.* **129** 224104
[11] Unke O T and Meuwly M 2019 Physnet: a neural network for predicting energies, forces, dipole moments and partial charges *J. Chem. Theory Comput.* **15** 3678–93

[12] Schütt K T, Sauceda H E, Kindermans P-J, Tkatchenko A and Müller K-R 2018 Schnet—a deep learning architecture for molecules and materials *J. Chem. Phys.* **148** 241722
[13] Zhang L, Han J, Wang H, Car R and E W 2018 Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics *Phys. Rev. Lett.* **120** 143001
[14] Smith J S, Isayev O and Roitberg A E 2017 ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost *Chem. Sci.* **8** 3192–203
[15] Bartók A P, Payne M C, Kondor R and Csányi G 2010 Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons *Phys. Rev. Lett.* **104** 136403
[16] Bartók A P and Csányi G 2015 Gaussian approximation potentials: a brief tutorial introduction *Int. J. Quant. Chem.* **115** 1051–7
[17] Shapeev A V 2016 Moment tensor potentials: a class of systematically improvable interatomic potentials *Multiscale Model. Simul.* **14** 1153–73
[18] Thompson A P, Swiler L P, Trott C R, Foiles S M and Tucker G J 2015 Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials *J. Comput. Phys.* **285** 316–30
[19] Jenke J, Subramanyam A P A, Densow M, Hammerschmidt T, Pettifor D G and Drautz R 2018 Electronic structure based descriptor for characterizing local atomic environments *Phys. Rev.* B **98** 144102
[20] Balabin R M and Lomakina E I 2011 Support vector machine regression (LS-SVM)-an alternative to artificial neural networks (ANNS) for the analysis of quantum chemistry data? *Phys. Chem. Chem. Phys.* **13** 11710
[21] Gastegger M, Behler J and Marquetand P 2017 Machine learning molecular dynamics for the simulation of infrared spectra *Chem. Sci.* **8** 6924
[22] Darley M G, Handley C M and Popelier P L A 2008 Beyond point charges: dynamic polarization from neural net predicted multipole moments *J. Chem. Theor. Comput.* **4** 1435–48
[23] Pereira F and Aires-de Sousa J 2018 Machine learning for the prediction of molecular dipole moments obtained by density functional theory *J. Cheminformatics* **10** 43
[24] Artrith N, Morawietz T and Behler J 2011 High-dimensional neural-network potentials for multicomponent systems: applications to zinc oxide *Phys. Rev.* B **83** 153101
[25] Morawietz T, Sharma V and Behler J 2012 A neural network potential-energy surface for the water dimer based on environment-dependent atomic energies and charges *J. Chem. Phys.* **136** 064103
[26] Yao K, Herr J E, Toth D W, Mckintyre R and Parkhill J 2018 The tensor Mol-0.1 model chemistry: a neural network augmented with long-range physics *Chem. Sci* **9** 2261–9
[27] Bereau T, Andrienko D and von Lilienfeld O A 2015 Transferable atomic multipole machine learning models for small organic molecules *J. Chem. Theory Comput.* **11** 3225–33
[28] Faraji S, Ghasemi S A, Rostami S, Rasoulkhani R, Schaefer B, Goedecker S and Amsler M 2017 High accuracy and transferability of a neural network potential through charge equilibration for calcium fluoride *Phys. Rev.* B **95** 104105
[29] Lee J, Seko A, Shitara K, Nakayama K and Tanaka I 2016 Prediction model of band gap for inorganic compounds by combination of density functional theory calculations and machine learning techniques *Phys. Rev.* B **93** 115104
[30] Pilania G, Gubernatis J E and Lookman T 2017 Multi-fidelity machine learning models for accurate bandgap predictions of solids *Comput. Mater. Sci.* **129** 156–63
[31] Eckhoff M, Lausch K N, Blöchl P E and Behler J 2020 Predicting oxidation and spin states by high-dimensional neural networks: applications to lithium manganese oxide spinels (arXiv:2007.00335)
[32] Rupp M, Tkatchenko A, Müller K-R and von Lilienfeld O A 2012 Fast and accurate modeling of molecular atomization energies with machine learning *Phys. Rev. Lett.* **108** 058301
[33] Pártay L B, Bartók A P and Csányi G 2010 Efficient sampling of atomic configurational spaces *J. Phys. Chem.* B **114** 10502–12
[34] Kolsbjerg E L, Peterson A A and Hammer B 2018 Neural-network-enhanced evolutionary algorithm applied to supported metal nanoparticles *Phys. Rev.* B **97** 195424
[35] Jennings P C, Lysgaard S, Hummelshøj J S, Vegge T and Bligaard T 2019 Genetic algorithms for computational materials discovery accelerated by machine learning *npj Comput. Mater.* **5** 1–6
[36] Ceriotti M, Tribello G A and Parrinello M 2011 Simplifying the representation of complex free-energy landscapes using sketch-map *Proc. Natl Acad. Sci. USA* **108** 13023–8
[37] De S, Musil F, Ingram T, Baldauf C and Ceriotti M 2017 Mapping and classifying molecules from a high-throughput structural database *J. Cheminformatics* **9** 6
[38] Sadeghi A, Ghasemi S A, Schaefer B, Mohr S, Lill M A and Goedecker S 2013 Metrics for measuring distances in configuration spaces *J. Chem. Phys.* **139** 184118
[39] Zhu L *et al* 2016 A fingerprint based metric for measuring similarities of crystalline structures *J. Chem. Phys.* **144** 034203
[40] De S, Bartók A P, Csányi G and Ceriotti M 2016 Comparing molecules and solids across structural and alchemical space *Phys. Chem. Chem. Phys.* **18** 13754–69
[41] Parsaeifard B, De D S, Christensen A S, Faber F A, Kocer E, De S, Behler J, von Lilienfeld A, and Goedecker S 2020 An assessment of the structural resolution of various fingerprints commonly used in machine learning (arXiv:2008.03189 [cond-mat, physics: physics])
[42] Hutter F, Lücke J and Schmidt-Thieme L 2015 Beyond manual tuning of hyperparameters *Künstl Intell.* **29** 329–37
[43] Luo G 2016 A review of automatic selection methods for machine learning algorithms and hyper-parameter values *Netw. Model. Anal. Health Inform. Bioinform.* **5** 18
[44] Klein A, Falkner S, Bartels S, Hennig P and Hutter F 2017 Fast Bayesian optimization of machine learning hyperparameters on large datasets *Artificial Intelligence and Statistics* pp 528–36
[45] Gastegger M, Schwiedrzik L, Bittermann M, Berzsenyi F and Marquetand P 2018 wACSF—weighted atom-centered symmetry functions as descriptors in machine learning potentials *J. Chem. Phys.* **148** 241709
[46] Browning N J, Ramakrishnan R, von Lilienfeld O A and Roethlisberger U 2017 Genetic optimization of training sets for improved machine learning models of molecular properties *J. Phys. Chem. Lett.* **8** 1351–9
[47] Imbalzano G, Anelli A, Giofré D, Klees S, Behler J and Ceriotti M 2018 Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials *J. Chem. Phys.* **148** 241730
[48] Behler J 2017 First principles neural network potentials for reactive simulations of large molecular and condensed systems *Angew. Chem., Int. Ed.* **56** 12828–40
[49] Bartók A P, Kondor R and Csányi G 2013 On representing chemical environments *Phys. Rev.* B **87** 184115

[50] Pronobis W, Tkatchenko A and Mueller K-R 2018 Many-body descriptors for predicting molecular properties with machine learning: analysis of pairwise and three-body interactions in molecules *J. Chem. Theory Comput.* **14** 2991–3003

[51] Jindal S, Chiriki S and Bulusu S S 2017 Spherical harmonics based descriptor for neural network potentials: structure and dynamics of $Au_{147}$ nanocluster *J. Chem. Phys.* **146** 204301

[52] Kocer E, Mason, J K and Erturk H 2019 A novel approach to describe chemical environments in high-dimensional neural network potentials *J. Chem. Phys.* **150** 154102

[53] Faber F A, Christensen A S, Huang B and von Lilienfeld O A 2018 Alchemical and structural distribution based representation for universal quantum machine learning *J. Chem. Phys.* **148** 241717

[54] Behler J 2011 Atom-centered symmetry functions for constructing high-dimensional neural network potentials *J. Chem. Phys.* **134** 074106

[55] Behler J 2014 Representing potential energy surfaces by high-dimensional neural network potentials *J. Phys.: Condens. Matter* **26** 183001

[56] Behler J 2015 Constructing high-dimensional neural network potentials: a tutorial review *Int. J. Quantum Chem.* **115** 1032–50

[57] Seung H S, Opper M and Sompolinsky H 1992 Query by committee *Proc. 5th Annual Workshop on Computational Learning Theory* pp 287–94

[58] Artrith N and Behler J 2012 High-dimensional neural network potentials for metal surfaces: aprototype study for copper *Phys. Rev.* B **85** 045439

[59] Podryabinkin E V and Shapeev A V 2017 Active learning of linearly parametrized interatomic potentials *Comp. Mater. Sci.* **140** 171–80

[60] Zhang L, Lin D-Y, Wang H, Car R and E W 2019 Active learning of uniformly accurate interatomic potentials for materials simulation *Phys. Rev. Mater.* **3** 023804

[61] Schran C, Behler J and Marx D 2020 Automated fitting of neural network potentials at coupled cluster accuracy: protonated water clusters as testing ground *J. Chem. Theory Comput.* **16** 88–99

[62] Cormen T H, Leiserson C E, Rivest R L and Stein C 2009 *Introduction to Algorithms* (Cambridge, MA: MIT Press)

[63] Bentley J L 1975 Multidimensional binary search trees used for associative searching *Commun. ACM* **18** 509–17

[64] Pearson K 1901 On lines and planes of closest fit to systems of points in space *London, Edinburgh Dublin Phil. Mag. J. Sci.* **2** 559–72

[65] Hotelling H 1933 Analysis of a complex of statistical variables into principal components *J. Educ. Psychol.* **24** 417–41

[66] Frenkel D and Smit B 2002 *Understanding Molecular Simulations* (New York: Academic)

[67] Aggarwal C C, Hinneburg A and Keim D A 2001 On the surprising behavior of distance metrics in high dimensional space *Database Theory—ICDT 2001* (*Lecture Notes in Computer Science*) ed J Van den Bussche and V Vianu (Berlin: Springer) pp 420–34

[68] In this text we use the term collision for two different but similar phenomena. Hash collisions occur when two objects that are different result in the same hash value, which is possible due to the space reduction that happens in hash functions. This behavior is a property of the hash function itself, it is in principle unavoidable and undesirable but can be worked around. In the text we mostly care about collisions that happen when two SF vectors that are close in SF space are binned into the same integer vector, and thus will end up with the same has values. This is desired behavior, and in fact the core of the BAH algorithm. The difference between the two concepts is *where* the 'collision' is happening: for hash collisions, it happens directly from the object itself; for SF collisions, it happens after the vectors have been pre-processed by binning.

[69] Python 3.8.5 documentation 5. Data structures (available at: docs.python.org/3/tutorial/datastructures.html)

[70] Note: When comparing force components directly, care should be taken. ACSF vectors are invariant with respect to rotations and translations in coordinate space, but forces are *not*. This is due to the derivatives involved in going from energy to forces, which add a direction component. The result is that with the same ACSF vector, one can have different force vector orientations, that is, the components of the force vector might not match. The predicted magnitude of the force vector should on the other hand remain consistent since it is directionless. A trivial example of this is an unrelaxed unmodified slab with two interfaces: atoms in the top and bottom surfaces will have identical environments as described by their ACSFs, but the $Z$-component of their force vectors will necessarily, due to symmetry, be opposite. This becomes more complicated for more homogeneous systems such as liquids and amorphous solids, where the same atomic environment might be found in a variety of orientations. Thus only force vector magnitudes should be compared, or a consistent orientation of the environments should be achieved in some way.

[71] Weinreich J, Römer A, Paleico M L and Behler J 2020 Properties of $\alpha$-Brass nanoparticles. 1. Neural network potential energy surface *J. Phys. Chem.* C **124** 12682–95

[72] Deaven D M and Ho K M 1995 Molecular geometry optimization with a genetic algorithm *Phys. Rev. Lett.* **75** 288–91

[73] Goedecker S 2004 Minima hopping: an efficient search method for the global minimum of the potential energy surface of complex molecular systems *J. Chem. Phys.* **120** 9911–17

[74] Voter A F 2007 Introduction to the kinetic Monte Carlo method *Radiation Effects in Solids* (*NATO Science Series*) ed K E Sickafus, E A Kotomin and B P Uberuaga (Dordrecht: Springer) pp 1–23