**Response to the Reviewers**

We are grateful to the Reviewers for their valuable suggestions and comments. We have carefully considered all comments and suggestions and have revised the manuscript accordingly. The following details the changes made and brief responses to the questions raised. All the significant changes are marked yellow in the revised manuscript.

**Reviewer(s)' Comments to Author:**

**Reviewer: 1**  
Comments to the Author

The authors present an implementation of Behler and Parinello type symmetry function descriptors, referred to as TensorAlloy, which has been implemented in the TensorFlow machine learning framework. The authors claim to provide techniques which allow for the efficient calculation of forces and virials within ML frameworks. They provide validation of the code on two publicly available benchmarks, QM7 and the Ni-Mo alloy data set.  
  
**Minor points:**

1. Please heavily edit the article for better grammar and sentence structure before publication. There are mistakes in places that can affect the understanding of the science in presented.

***Authors’ response:*** *Thanks very much for your comments. The core sections (Method, Discussion) are almost re-written from scratch. We believe readers can have a better understanding of our method.*

1. In the abstract and conclusions it is stated that: "TensorAlloy can reach state-of-the-art performance within just one GPU hour." This statement is not very useful, and perhaps a bit misleading. For example, if this code package were tested on harder benchmarks, such as the TensorMol ChemSpider benchmark (2M+ structures) or the ANI-1 dataset (20M+ structures), both of which are publicly available, then it is likely that 1 GPU hour would not be sufficient for training a model with TensorAlloy. TensorMol (also programmed in TensorFlow) took weeks time on GPUs train to their 2M molecule ChemSpider benchmark. This was including force training as is included in this work. Statements such as that given above should only be done so within the context of a given benchmark, and it should be made clear that those times are only valid on that given benchmark.

***Authors’ response:*** *Thanks very much for pointing out this. Due to some restrictions we cannot make our program open-source. Thus, emphasizing training time or comparing with other programs may not be that convincing in this paper. However, speed is a very important factor in machine learning. So, we create a demo (https://github.com/Bismarrck/vap)to show the efficiency of our program. A benchmark function is also provided. The prediction phase benchmark results are also discussed in the revised manuscript. All of our demo need is a Python-3.7 environment with TensorFlow 1.13, Numpy, Scikit-learn and ASE. The energy & force & stress calculations of the (MoNi)64000 system cost approximately 80 seconds on the MacBook Pro 2018 (6 Cores, 2.6 GHz): 75 seconds for building GSL arrays, 5 seconds for executing TF graph. In the training phase, GSL arrays can be pre-built and loaded from cache (like “tfrecords”) directly. The benchmark results may (indirectly) explain why the experiments in this paper only costs 1-2 GPU hours.*

*Note: a natively compiled TensorFlow is strongly recommended.*

1. When training to QM7, do you use the loss function as provided in Equation 22. If so, what QM stress are you training to?

***Authors’ response:*** *Thanks very much for your comments. We added a statement to make this clear: “The force and stress losses of Equation 27 are disabled for this experiment.”*

**Major points:**

1. The use of python based machine learning frameworks for atomistic potentials is not new. Even symmetry function based models have been implemented in such frameworks. Please see the open source packages: Tensormol and TorchANI. Also see SchNet pack, which is not symmetry function based but has similar claims. Tensormol and SchNet pack have publications associated with them, which the authors fail to mention in the context of having used ML frameworks to build a computational graph which allows for easy access to quantities such as forces and virials. I would like to see an explanation of how the authors technique, i.e. the "virtual atom" approach, differs from existing open source packages.

***Authors’ response:*** *Thanks very much for your comments. To answer this question, we have to introduce the design principles of TensorAlloy:*

*Our program is designed to have two phases: 1) the training phase and 2) the prediction phase. Both phases have the same fundamental requirement: the ability of building direct computation graph from atomic positions to total energy. However, each phase has their own concerns:*

1. *The training phase:*
2. *Stoichiometry-free: the training dataset should not have any stoichiometry restriction. Any type of solid or molecule is acceptable. A universal approach of expressing different structures in a single reference system may be necessary.*
3. *Mini-batch training: mini-batch based stochastic training is currently the most efficient way to train neural networks on large datasets. However, batch training requires vectorized and aligned expressions. Here aligned means feature arrays of structures of different stoichiometries share the same shape.*
4. *Cache: some intermediate arrays may be pre-computed and stored in cache files. During training, these values can be loaded from cache directly, thus saving significantly amount of resources.*
5. *The prediction phase:*
6. *Stoichiometry: a trained model ABC can make predictions on arbitrary AxByCz where 0 <= x, y, z.*
7. *Standalone: the prediction phase can be implemented in a standalone program. Only the exported trained model is required. Collaborators or others can focus on using the model.*

*The virtual-atom approach plays a central role in handling these technical requirements. It provides a way to describe structures of various sizes and stoichiometries in unified and vectorized forms. In our TensorAlloy program, a structure must be mapped to the unified vectorized inputs and then the symmetry function descriptors are calculated based on these unified inputs. In the training phase, this may waste some computation resources because the size of the final unified expression depends on the largest structure. But the implementations within TensorFlow become much easier. In the revised manuscript, the unified vectorized expressions are labeled with “GSL” which is short for “global symbol list”.*

*To make the approach clear, we provide a Python implementation in the appendix and a full demo on* [*GitHub*](https://github.com/Bismarrck/vap) *(https://github.com/Bismarrck/vap).*

**

*TorchANI, the successor of TensorMol, is really a well-designed machine learning package based on PyTorch. PyTorch is based on the so-called “dynamic graph” while TensorFlow uses “static graph”. Thus, the fundamental design principles of TorchANI differs with our TensorAlloy significantly. Here are some differences: TorchANI do not use global reference system (GSL) or similar unified expression approach to align training batches. Virial stress is not yet implemented in TorchANI. We should also notice that TorchANI mainly focuses on molecules. All their uploaded examples are molecules.*

1. The authors should make it clear that what is being done in this work to allow for the efficient computation of forces and stress is simply providing a forward pass through the Behler and Parinello type descriptors. All actual force and stress calculations are obtained nearly for free through autograd which is built into TensorFlow. The proposed abilities to compute/train to forces and stress are therefore not new capabilities, especially considered they have been exploited elsewhere before.

***Authors’ response:*** *Thanks very much for your comments. We have mentioned this in the revised manuscript.*

1. This article claims to be producing a "potential" but then employs the leaky ReLU activation function, which cannot conserve energy since the first derivative of this active function is discontinuous. In my own personal experience, I have seen that LReLU provides better fits to atomistic data sets, yet for mathematically obvious reasons fails to conserve energy. For this reason, I have to question the validity of the results presented. An appropriate "smooth" activation with continuous first derivatives (there are many options in published literature) should be selected and the benchmarks should be recomputed to provide realistic results. Otherwise, users of your package may decide to pick up your code and use LReLU without realizing this lack of conservation could affect their results.

***Authors’ response:*** *Thanks very much for this valuable suggestion. We did some activation function tests (tanh, LReLU, softplus). Our results suggest “softplus” (a smooth and differentiable ReLU) is a very good choice (Section 3.G in the revised manuscript).*

1. It is claimed that this algorithm is efficient and a lot of figures show the timing of the code. However, without putting this in context of existing codes that do similar things I cannot gauge the relative efficiency of the presented "virtual atom" approach. To me, it is not even clear how this approach is even different than any other publicly available code base.

***Authors’ response:*** *Thanks very much for your comments. We fully understand this concern. So we did some major changes in the revised manuscript:*

1. *The entire “Method” section of the revised manuscript is re-written. We believe readers can have better understandings of our approach.*
2. *We provide a virtual-atom approach based Python implementation of the symmetry function descriptors on* [*GitHub*](https://github.com/Bismarrck/vap) *(*[*https://github.com/bismarrck/vap*](https://github.com/bismarrck/vap)*). Test codes are also given so that readers and users can use our codes to compute symmetry function descriptors of arbitrary structure. Feel free to use our codes in your own research.*
3. *We provide a benchmark function of the prediction phase. Figure 8 of the revised manuscript is obtained with this function. You can just run this benchmark on your own machine.*

**Reviewer: 2**  
Comments to the Author

However, there seem to be some misconceptions already in the abstract! For example, the authors may not be aware that the virial for ML potentials has already been worked out here:

Thompson et al, "General formulation of pressure and stress tensor for arbitrary many-body interaction potentials under periodic boundary conditions", JCP (2009)

https://doi.org/10.1063/1.3245303

This approach works very well in a variety of "descriptor" based approaches, e.g. the SNAP method of the same authors. In a very general sense, it is possible to "backpropagate" the derivatives through any choice of descriptors.

So at the very least, the abstract should be substantially revised to clarify the actual contribution of this new work.

***Authors’ response:*** *Thanks very much for your comments. In fact, our virial equation:*

*A close up of a logo

Description automatically generated*

*is directly derived from the equation in this JCP paper (see the appendix of this manuscript):*

*A picture containing object, clock

Description automatically generated*

*Our equation is just an alternative expression of the JCP equation. We have modified our statements in the revised abstract & manuscript to make clear of this.*

*However, the original JCP form is not that suitable for our program due to technical reason: computing dE/dh is much easier because of the AutoGrad feature of TensorFlow.*

*As introduced in this paper, with the help of the virtual-atom approach, we can build a direct computation graph from atomic positions & cell to total energy. Then, the derivative, dE/dh, can be obtained from TensorFlow with just one line:*

*dEdh = tf.gradients(energy, cell)*

*Here ‘tf.gradients’ is a function provided by TensorFlow. Thus, we can have a simplified and concise implementation:*

*A screenshot of a cell phone

Description automatically generated*

*For more about our algorithm, you can look into the newly uploaded demo on GitHub: https://github.com/Bismarrck/vap*