AFLOW-ML: A RESTful API for machine-learning predictions of materials properties

Eric Gossett,[1, 2] Cormac Toher,[1, 2] Corey Oses,[1, 2] Olexandr Isayev,[3] Fleur Legrain,[4, 5] Frisco Rose,[1, 2] Eva Zurek,[6] Jesús Carrete,[7] Natalio Mingo,[4] Alexander Tropsha,[3] and Stefano Curtarolo[1, 2, 8, *]

[1] *Department of Mechanical Engineering and Materials Science, Duke University, Durham, North Carolina 27708, USA*
[2] *Center for Materials Genomics, Duke University, Durham, North Carolina 27708, USA*
[3] *Laboratory for Molecular Modeling, Division of Chemical Biology and Medicinal Chemistry, UNC Eshelman School of Pharmacy, University of North Carolina, Chapel Hill, North Carolina 27599, USA*
[4] *LITEN, CEA-Grenoble, 38054 Grenoble, France*
[5] *Universiteé Grenoble Alpes, 38000 Grenoble, France*
[6] *Department of Chemistry, State University of New York at Buffalo, Buffalo, New York 14260, USA*
[7] *Institute of Materials Chemistry, TU Wien, A-1060 Vienna, Austria*
[8] *Fritz-Haber-Institut der Max-Planck-Gesellschaft, 14195 Berlin-Dahlem, Germany*

(Dated: June 21, 2018)

Machine learning approaches, enabled by the emergence of comprehensive databases of materials properties, are becoming a fruitful direction for materials analysis. As a result, a plethora of models have been constructed and trained on existing data to predict properties of new systems. These powerful methods allow researchers to target studies only at interesting materials — neglecting the non-synthesizable systems and those without the desired properties — thus reducing the amount of resources spent on expensive computations and/or time-consuming experimental synthesis. However, using these predictive models is not always straightforward. Often, they require a panoply of technical expertise, creating barriers for general users. AFLOW-ML (AFLOW Machine Learning) overcomes the problem by streamlining the use of the machine learning methods developed within the AFLOW consortium. The framework provides an open RESTful API to directly access the continuously updated algorithms, which can be transparently integrated into any workflow to retrieve predictions of electronic, thermal and mechanical properties. These types of interconnected cloud-based applications are envisioned to be capable of further accelerating the adoption of machine learning methods into materials development.

I. INTRODUCTION

Since their inception, high throughput materials science frameworks such as AFLOW [1–8] have been amassing large databases of materials properties. For instance, the AFLOW database [9–12] alone contains over 1.7 million material compounds with over 170 million calculated properties, generated from the Inorganic Crystal Structure Database (ICSD) [13–15], as well as by decorating crystal structure prototypes [16]. Combined with other online databases and high-throughput frameworks such as the Materials Project [17–20], NoMaD [21], OQMD [22, 23], the Computational Materials Repository [24, 25], and AiiDA [26, 27], materials data is abundant and available. As a result, machine learning (ML) methods [28–30] have emerged as the ideal tool for data analysis [31–38], by identifying the key features in a data set [39] to construct a model for predicting the properties of a material. Recently, several models were developed to predict the properties of various material classes such as perovskites [40, 41], oxides [42], elpasolites [43, 44], thermoelectrics [45–47], and metallic glasses [48]. Additionally, generalized approaches have been devised for inorganic materials [49–60] and for systematically identifying efficient physical models of materials properties [61].

While predictions are powerful tools for rational materials design, the discipline is still reasonably new within the realm of materials science. As a result, a working understanding of machine learning principles, along with a high level of technical expertise, is required for using code bases effectively. This inhibits accessibility, where an average end user aims to utilize the codes to retrieve predictions with as little complication as possible. With the ever increasing number of predictive models, a unique challenge emerges: how does one create an accessible means to integrate machine learning frameworks into a materials discovery workflow?

AFLOW-ML alleviates this issue by providing a simplified abstraction on top of sophisticated predictive models. Predictions are exposed through a web accessible Application Programming Interface (API) where functionality is distilled down to its essence: from the user input, return a prediction. AFLOW-ML can be added into any code base through use of an HTTP request library, native to most languages. Alternatively, AFLOW-ML can be utilized through use of the included Python client and command line interface. Through such abstractions, AFLOW-ML is accessible to a wide audience: it unburdens the user from having to understand the intricacies of machine learning and eliminates the technical expertise required to set up such codes.

II. REST API

The AFLOW-ML API is structured around a REpresentational State Transfer (REST) architecture, which allows resources to be accessed using HTTP request methods. Each resource is located at an endpoint, which is identified by a URL comprised of descriptive nouns.

* stefano@duke.edu

Action	Endpoint	Object
POST	`/plmf/prediction`	task
POST	`/mfd/prediction`	task
GET	`/prediction/result/{id}`	status or prediction

TABLE I. A list of all endpoints in the API. Actions specify the supported HTTP request method, endpoints define the URL and objects are the returned resource. An endpoint with {...} denotes a path parameter.

A URL may also include special variables in the form of path parameters and a query string. A path parameter is a segment of the URL that specifies a particular resource and is denoted by a noun within braces ({...}) inside the endpoint. The query string is a list of key-value pairs placed at the end of a URL, which controls the representation (e.g. format) of the resource. Typically, it is used to apply filters or define the structure of the returned representation.

Resources within the API are represented in JavaScript Object Notation (JSON), and are referred to as objects. Once at an endpoint, the user must specify how to interact with the object. This is referred to as an action, and is an HTTP request method. The API supports the two most common HTTP request methods, `GET` and `POST`, where `GET` fetches an object and `POST` sends user defined data to the server. Therefore, users interact with the API by performing actions (`GET`, `POST`) on endpoints (URLs) to retrieve objects (resources).

III. API STRUCTURE

The overall structure of the API can be seen in Table I. All endpoints are located at the base URL **aflow.org/API/aflow-ml/v1.0/** and are organized by the model and the returned object.

AFLOW-ML currently supports two models: molar fraction descriptor (`mfd`) [50] and property-labeled materials fragments (`plmf`) [49]. It is designed to be extensible, and additional models will be added in the future as they are developed.

The `mfd` model [50] predicts the material properties based on the chemical formula only: the vector of descriptors has 87 components, each component b_i being the mole fraction of element Z_i in the compound (Z_1 is H, Z_2 is He, etc.). The model is built with nonlinear support vector machines [29] and a radial basis function kernel. The model is trained using a data set of 292 randomly selected compounds of the ICSD for which the vibrational properties are computed with DFT calculations. Pearson and Spearman correlations for k-fold cross-validation ($k = 5 \rightarrow 14$) are in excess of 0.9 for all predicted properties, while the mean average errors are 13.2 meV/atom (vibrational free energy) and 0.037 meV/(atom · K) (vibrational entropy) and the root mean square errors are 18.8 meV/atom (vibrational free energy) and 0.052 eV/(atom · K) (vibrational entropy). Further details on the model training and validation can be found in Ref. 50.

The `plmf` model [49] represents each crystal structure as a colored graph, where the atomic vertices are decorated by the reference properties of the corresponding elemental species. Topological neighbors are determined using a Voronoi tessellation, and these nodes are connected to form the graph. The final feature vector for the ML model is obtained by partitioning the full graph into smaller subgraphs, termed fragments in analogy with the fragment-based descriptors used in cheminformatics [62]. All `plmf` models are built with the Gradient Boosting Decision Tree (GBDT) method [30]. Models for electronic and thermomechanical properties were trained on 26,674 and 2,829 materials entries from the AFLOW repository, respectively. All models are validated through Y-randomization (label scrambling) and five-fold cross validation, with coefficient of determination (r^2) values in excess of 0.9 for most quantities, while the mean average errors are 0.035 eV (electronic band gap), 8.68 GPa (bulk modulus), 10.6 GPa (shear modulus), 35.9 K (Debye temperature), 0.05 k_B/atom (heat capacity at constant pressure) and 0.04 k_B/atom (heat capacity at constant volume), and the root mean square errors are 0.51 eV (electronic band gap), 14.3 GPa (bulk modulus), 18.4 GPa (shear modulus), 57.0 K (Debye temperature), 0.09 k_B/atom (heat capacity at constant pressure) and 0.07 k_B/atom (heat capacity at constant volume). Further details on the model training and validation can be found in Ref. 49.

In general, API usage involves uploading a material structure to a `POST` endpoint, `<model>/prediction`, and retrieving a prediction object from a `GET` endpoint, `/prediction/result/{id}`, as shown in the flowchart in Figure 1. `POST` endpoints are responsible for the submission of a material structure for a prediction. In their request body, the `file` keyword is required. It must contain a string representation of the material's crystal structure, in POSCAR 5 format (the lattice geometry and atomic position input file for version 5 of the VASP DFT package [63, 64]). Upon receiving a request, the response body returns a task object containing information about the submitted structure, which has the following format:

```
{
  "id": String,
  "model": String,
  "results_endpoint": String
}
```

When a material is posted to the API, a prediction task is created and added to a queue. Each task is assigned an identifier, the `id` keyword, used to fetch the the prediction object at the endpoint referenced in the `results_endpoint` keyword. This endpoint, `/prediction/result/{id}`, supports the `GET` method and requires the id as a path parameter. Depending on

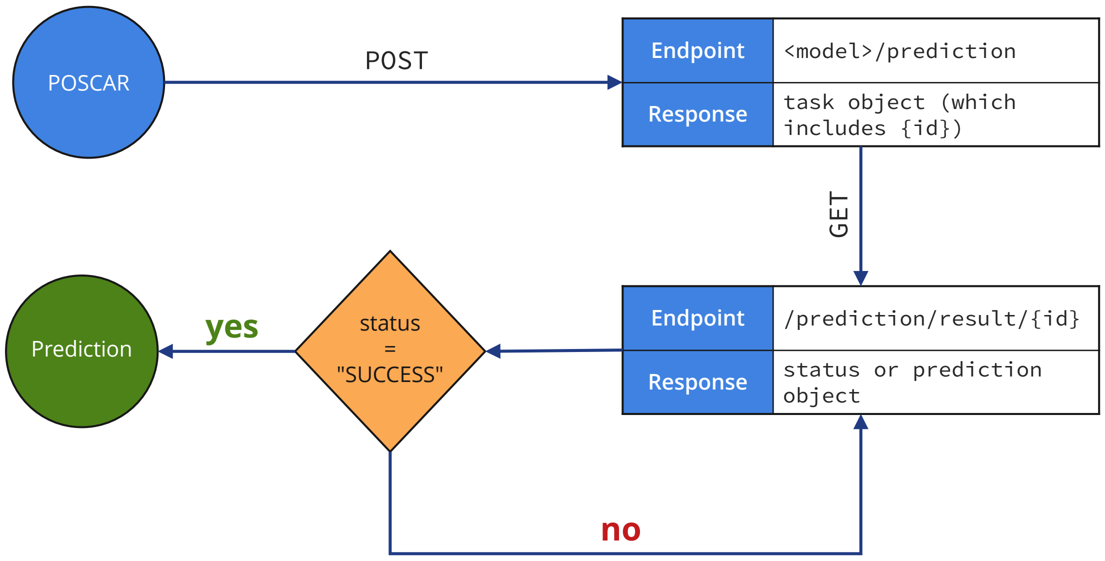


FIG. 1. A flowchart illustrating the typical use-case of the API. First a POSCAR file is posted to the `<model>/prediction` endpoint to retrieve a task object. The `id` field of the task object is used to poll the `<model>/prediction/{id}` for the status of the prediction. If the status field of the response equals `SUCCESS` then the prediction object is returned.

the status of the prediction task, the response body returns a status object or prediction object. When the task is still in the queue, the status object is returned:

```
{
  "status": String,
  "description": String
}
```

The status object details the state of the prediction task. The state is identified by the keyword `status` which takes one of the following values: `STARTED`, `PENDING`, `SUCCESS` and `FAILURE`, while a description of each status type is given by the `description` keyword. When attempting to retrieve the prediction object, it is best to poll the endpoint periodically to check the status. Tasks that are still within the queue are given the `STARTED` or `PENDING` status, while a completed task status reads `SUCCESS`. In instances where the uploaded file is incorrectly formatted, a failed task occurs, status: `FAILURE`. When the task is completed, the response contains the prediction object. The prediction object is an extension of the status object and contains different keywords depending on the model used. For `plmf`, the prediction object, known as a `plmf` prediction, takes the following form:

```
{
  "status": String,
  "description": String,
  "model": String,
  "citation": String,
  "ml_egap_type": String,
  "ml_egap": Number,
  "ml_energy_per_atom": Number,
  "ml_ael_bulk_modulus_vrh": Number,
  "ml_ael_shear_modulus_vrh": Number,
  "ml_agl_debye": Number,
  "ml_agl_heat_capacity_Cp_300K": Number,
  "ml_agl_heat_capacity_Cv_300K": Number,
  "ml_agl_heat_capacity_Cp_300K_per_atom":
      Number,
  "ml_agl_heat_capacity_Cv_300K_per_atom":
      Number,
  "ml_agl_thermal_conductivity_300K": Number,
  "ml_agl_thermal_expansion_300K": Number
}
```

The `mfd` model returns an `mfd` prediction object:

```
{
  "status": String,
  "description": String,
  "model": String,
  "citation": String,
  "ml_Cv": Number,
  "ml_Fvib": Number,
  "ml_Svib": Number
}
```

The details of each object and their keywords are found in the List of API Endpoints and Objects section.

IV. USING THE API

The process to retrieve a prediction is as follows: First, the contents of a POSCAR 5 file, titled test.poscar, are uploaded to the submission endpoint. This can be achieved by using an HTTP library such as Requests (Python) [65], URLSession (iOS SDK) [66], HttpURLConnection (Java) [67], Fetch (JavaScript) [68] or using a command line tool such as Wget or cURL. For this example, cURL is used. The contents of the POSCAR are posted to the submission endpoint as follows:

```
curl  http://aflow.org/API/aflow-ml/v1.0/plmf/prediction --data-urlencode file@test.poscar
```

where the `--data-urlencode` flag handles encoding the contents of the POSCAR, located in the current directory, and associating it to the `file` keyword. Note that as mentioned previously, the `POST` returns a JSON response including the task id. The task id is then used to poll the results endpoint:

```
curl  http://aflow.org/API/aflow-ml/v1.0/prediction/result/{id}
```

The status keyword is used as an indicator to determine if any additional polling is required. Depending on the status of the job, the endpoint returns either a task status object or a prediction object. If the status keyword's value is `SUCCESS` then no additional polling is required, since the endpoint returns a prediction object, which is an extension of the task status object.

V. LIST OF API ENDPOINTS AND OBJECTS

This section includes the details of each endpoint and object accessible in the API. Endpoint information contains the associated HTTP method, request parameters, request body data and response object for each endpoint. Object properties are listed along with their type and description.

A. Endpoints

- `POST  plmf/prediction`
  - *Description.* Uploads the contents of a POSCAR 5 to retrieve a prediction using the `plmf` model.
  - *Query parameters.*
    - `file` (required) - The contents of the POSCAR 5.
  - *Response format.* On success, the response header contains the HTTP status code 200 OK and the response body contains a task object, in JSON format.
  - *Example.*

    ```
    curl http://aflow.org/API/aflow-ml/
    v1.0/plmf/prediction
    --data-urlencode file@test.poscar
    ```

- `POST  mfd/prediction`
  - *Description.* Uploads the contents of a POSCAR 5 to retrieve a prediction using the `mfd` model.
  - *Query parameters.*
    - `file` (required) - The contents of the POSCAR 5.
  - *Response format.* On success, the response header contains the HTTP status code 200 OK and the response body contains a task object, in JSON format.
  - *Example.*

    ```
    curl http://aflow.org/API/aflow-ml/
    v1.0/mfd/prediction
    --data-urlencode file@test.poscar
    ```

- `GET  /prediction/result/{id}`
  - *Description.* Fetches the status object or returns the prediction object when the task is completed.
  - *Path parameters.*
    - `id` (required) - The unique identifier retrieved from the task object on submission.
  - *Query string arguments.*
    - `fields` - A comma separated list of the fields to include in the JSON response object. Note that specified fields only affects the prediction object.
  - *Response format.* On success, the response header contains the HTTP status code 200 OK. If the task is still pending, the response body contains a task status object in JSON format. Upon completion the response body contains a prediction object in JSON format.

- *Example.*

```
curl http://aflow.org/API/aflow-ml/
v1.0/prediction/result
/59ea0f78-4868-4a1e-9a20-e7343f00907d
```

B. Objects

- Task
  - *Description.* Describes the task for a submitted prediction. Includes the unique identifier for the prediction and results endpoint.
  - *Keys.*
    - ∗ `id`
      - · *Description.* The unique identifier of the prediction task. Used at the fetch prediction endpoint to retrieve the status of a prediction and return the results on completion.
      - · *Type.* String.
    - ∗ `results_endpoint`
      - · *Description.* The path of the endpoint where the prediction task status and results are retrieved.
      - · *Type.* String.
    - ∗ `model`
      - · *Description.* The name of the machine learning model used to generate the prediction.
      - · *Type.* String.
  - *Example.*

```
{
    "id":      "d29af704-06bf
               -4dc8-8928
               -cd2c41aea454",
    "model":    "plmf",
    "results_endpoint":
               "/prediction/result/
               d29af704-06bf
               -4dc8-8928-
               cd2c41aea454"
}
```

- Status
  - *Description.* Provides the status of a (prediction) task.
  - *Keys.*
    - ∗ `status`
      - · *Description.* The status of the task. Takes the following values: `STARTED`, `PENDING`, `SUCCESS` and `FAILURE`. When a task is added to the queue its status reads `PENDING`. Once it reaches the top of the queue the status reads `STARTED` and the prediction will run. If the prediction is successful the status reads `SUCCESS`.
      - · *Type.* String.
    - ∗ `Description`
      - · *Description.* Describes the status of the task.
      - · *Type.* String.
  - *Example.*

```
{
    "status":        "PENDING"
    "description":   "The calculation
                      is running"
}
```

- `plmf` prediction
  - *Description.* The results of the prediction using the `plmf` model. This is an extension of the task status object.
  - *Keys.*
    - ∗ `status`
      - · *Description.* The status of the task. Takes the following values: `STARTED`, `PENDING`, `SUCCESS` and `FAILURE`. When a task is added to the queue its status reads `PENDING`. Once it reaches the top of the queue the status reads `STARTED` and the prediction will run. If the prediction is successful the status reads `SUCCESS`.
      - · *Type.* String.
    - ∗ `description`
      - · *Description.* Describes the status of the task.
      - · *Type.* String.
    - ∗ `model`
      - · *Description.* The model used in the prediction.
      - · *Type.* String.
    - ∗ `citation`
      - · *Description.* The DOI for the model's publication.
      - · *Type.* String.
    - ∗ `ml_egap_type`
      - · *Description.* Specifies if the material is a metal or an insulator. Takes the following values: `Metal`, `Insulator`.

- *Type.* String.

* `ml_egap`
  - *Description.* The electronic band gap.
  - *Units.* eV.
  - *Type.* Number.
* `ml_energy_per_atom`
  - *Description.* The energy per atom.
  - *Units.* eV/atom.
  - *Type.* Number.
* `ml_ael_bulk_modulus_vrh`
  - *Description.* The bulk modulus, trained on data calculated with the Automatic Elasticity Library (AEL). [69]
  - *Units.* GPa.
  - *Type.* Number.
* `ml_ael_shear_modulus_vrh`
  - *Description.* The shear modulus, trained on data calculated with AEL.
  - *Units.* GPa.
  - *Type.* Number.
* `ml_agl_debye`
  - *Description.* The Debye temperature, trained on data calculated with the Automatic GIBBS Library (AGL) [70].
  - *Units.* K.
  - *Type.* Number.
* `ml_agl_heat_capacity_Cp_300K`
  - *Description.* The heat capacity at 300K and constant pressure, trained on data calculated with AGL.
  - *Units.* k_B/cell.
  - *Type.* Number.
* `ml_agl_heat_capacity_Cp_300K_per_atom`
  - *Description.* The heat capacity per atom at 300K and constant pressure, trained on data calculated with AGL.
  - *Units.* k_B/atom.
  - *Type.* Number.
* `ml_agl_heat_capacity_Cv_300K`
  - *Description.* The heat capacity at 300K and constant volume, trained on data calculated with AGL.
  - *Units.* k_B/cell.
  - *Type.* Number.
* `ml_agl_heat_capacity_Cv_300K_per_atom`
  - *Description.* The heat capacity per atom at 300K and constant volume, trained on data calculated with AGL.
  - *Units.* k_B/atom.
  - *Type.* Number.
* `ml_agl_thermal_conductivity_300K`
  - *Description.* The lattice thermal conductivity at 300K, trained on data calculated with AGL.
  - *Units.* W/(m K).
  - *Type.* Number.
* `ml_agl_thermal_expansion_300K`
  - *Description.* The thermal expansion coefficient at 300K, trained on data calculated with AGL.
  - *Units.* K^{-1}.
  - *Type.* Number.

– *Example.*

```
{
    "status":
                "SUCCESS"
    "description":
                "The job has
                 completed.",
    "model": "plmf",
    "citation": "10.1038/ncomms15679",
    "ml_egap_type":
                "Insulator",
    "ml_egap":
                0.923,
    "ml_energy_per_atom":
                -5.760,
    "ml_ael_bulk_modulus_vrh":
                178.538,
    "ml_ael_shear_modulus_vrh":
                140.121,
    "ml_agl_debye":
                713.892,
    "ml_agl_heat_capacity_Cp_300K":
                23.362,
    "ml_agl_heat_capacity_Cp_300K
    _per_atom":
                2.333,
    "ml_agl_heat_capacity_Cv_300K":
                22.625,
    "ml_agl_heat_capacity_Cv_300K
    _per_atom":
                2.311,
    "ml_agl_thermal_conductivity
    _300K":
                2.792,
    "ml_agl_thermal_expansion_300K":
                6.093e-05
}
```

- `mfd` prediction
  - *Description.* The results of a prediction using the `mfd` model. This is an extension of the task status object.
  - *Keys.*
    * `status`

    · *Description.* The status of the task. Takes the following values: `STARTED`, `PENDING`, `SUCCESS` and `FAILURE`. When a task is added to the queue its status reads `PENDING`. Once it reaches the top of the queue the status reads `STARTED` and the prediction will run. If the prediction is successful the status reads `SUCCESS`.
    · *Type.* String.
  * `description`
    · *Description.* Describes the status of the task.
    · *Type.* String.
  * `model`
    · *Description.* The model used in the prediction.
    · *Type.* String.
  * `citation`
    · *Description.* The DOI for the model's publication.
    · *Type.* String.
  * `ml_Cv`
    · *Description.* The heat capacity at constant volume.
    · *Units.* meV/(atom · K).
    · *Type.* Number.
  * `ml_Fvib`
    · *Description.* The vibrational free energy per atom.
    · *Units.* meV/atom.
    · *Type.* Number.
  * `ml_Svib`
    · *Description.* The vibrational entropy per atom.
    · *Units.* meV/(atom · K).
    · *Type.* Number.

– *Example.*

```
{
    "description":
        "The job has completed.",
    "model": "mfd",
    "citation":
         "10.1021/acs.chemmater.
         7b00789",
    "status": "SUCCESS",
    "ml_Cv": 0.221,
    "ml_Fvib": 21.188,
    "ml_Svib": 0.211
}
```

VI. PYTHON CLIENT

A Python client is available for the AFLOW-ML REST API that provides researchers and developers a means to integrate AFLOW-ML into their applications or workflows, such as AFLOWπ [8]. The client includes the `AFLOWmlAPI` class which provides all the functionality needed to interface with the AFLOW-ML API, and can be downloaded at `aflow.org/src/aflow-ml`. From the client, a prediction is retrieved by passing the contents of a POSCAR to the `get_prediction` method. The `AFLOWmlAPI` class can be incorporated into a Python framework using code similar to the example illustrated below.

```
from aflowml.client import AFLOWmlAPI

with open('test.poscar', 'r') as input_file:
  aflowML = AFLOWmlAPI()
  data = aflowML.get_prediction(
    input_file.read(),
    'plmf'
  )
```

This method takes two arguments: `poscar` and `model`, where `poscar` is the content from the file `test.poscar` and `model` is a string specifying the model to use (`plmf` or `mfd`). This method returns a Python dictionary, in which the keys and respective predicted values are model dependent. For a list of each prediction object's key and value pair, please refer to the previous section.

The client's `AFLOWmlAPI` class includes two additional methods, `submit_job` and `poll_job`, that provide more control when submitting a prediction, and which can be used in place of the `get_prediction` method in the previous example. The `submit_job` method targets the `<model>/prediction` endpoint and returns the jobs task id. From the id, the job can be polled using the `poll_job` method which returns a prediction object upon completion. These methods are ideal for cases where the user would prefer to postpone polling to a later time.

VII. COMMAND LINE INTERFACE

Upon installation, the Python AFLOW-ML client provides a command line interface (CLI) titled `aflow-ml`. The CLI exposes all the functionality of the Python client and is targeted at users who are not familiar with Python or using REST APIs. To receive a prediction the path of the POSCAR 5 file is passed to the CLI as a positional argument. Additionally, the model type is specified via the `--model` flag:

```
aflow-ml  test.poscar --model=plmf
```

By default, the CLI outputs the results to the terminal. The default functionality is modified by the use of additional flags. For instance, results can be saved to an out file by use of the `-s` flag:

```
aflow-ml  test.poscar --model=plmf -s
```

where the predicted results are saved to a file titled `prediction.txt`. Additional flags exist which provide various levels of customization such as specifying the predicted values to return or the format of the output. A list of each flag is found below. This list is also viewable from the CLI using the `-h` or `--help` flags.

A. CLI flags

- Model
  - *Flag.* `-m` or `--model`
  - *Description.* (Required) Specifies the model to use in the prediction.
  - *Example.*
    `aflow-ml test.poscar --model=plmf`
- Save
  - *Flag.* `-s` or `--save`
  - *Description.* Saves the prediction to a file. If the out file is not specified contents are saved to a file named `prediction.txt`.
  - *Example.*
    `aflow-ml test.poscar -m plmf -s`
- Outfile
  - *Flag.* `--outfile`
  - *Description.* Specifies the path and name of the out file.
  - *Example.*
    `aflow-ml test.poscar -m plmf -s --outfile=prediction.txt`
- Format
  - *Flag.* `--format`
  - *Description.* Specifies the format of the out file. Currently, text and JSON are supported.
  - *Example.*
    `aflow-ml test.poscar -m plmf -s --format=json`
- Fields
  - *Flag.* `--fields`
  - *Description.* State the predicted fields to show in the output. Expects fields as a comma separated list. If the flag is not present, all fields are shown.
  - *Example.*
    `aflow-ml test.poscar -m plmf -s --fields=ml_egap,ml_egap_type`
- Verbose
  - *Flag.* `-v` or `--verbose`
  - *Description.* Toggle verbose mode. When enabled the CLI logs the progress of the prediction.
  - *Example.*
    `aflow-ml test.poscar -m plmf -v`

VIII. EXAMPLE APPLICATIONS

In order to demonstrate the utility of the AFLOW-ML API, two examples are provided, one each for the `plmf` and `mfd` models. These examples demonstrate how the API can be used to access machine-learning predictions, and to rapidly identify property trends for different classes of materials.

A. Predicting shear and bulk moduli for MoTi

For the first example, the AFLOW-ML API and the AFLOW data API are used to retrieve AFLOW-ML predictions for the shear and bulk moduli for the structures in the MoTi alloy system, which is one of the most populated binary alloy systems in the AFLOW database and is expected to form several ordered compounds [71]. Specifically, the shear and bulk moduli for the compositions that comprise the convex hull are investigated. First, the convex hull for MoTi is calculated using AFLOW via the following command:

```
aflow --chull --alloy=MoTi --output=json
```

The command outputs convex hull information in JSON format to the file `aflow_MoTi_hull.json`, containing an array of objects detailing information on each point in the convex hull as depicted in Figure 2(a). Here, the points on the hull are extracted by filtering the array for entries in which the key `ground_state` is true.

Next, the AFLOW data API is used to fetch the AFLOW Unique Identifier (AUID) [11], species, stoichiometry and POSCAR for every MoTi entry in the AFLOW database.

```
import urllib2
import json

BASE_URL = 'http://aflowlib.duke.edu/
    ↪ AFLOWDATA/LIB2_RAW/Mo_pvTi_sv'

# Fetch a list of all Mo Ti aflowlib entries
    ↪  by prototype
req = urllib2.Request(BASE_URL + '?
    ↪ aflowlib_entries')
res = urllib2.urlopen(req).read()
```

```
aflowlib_entries = res.replace('\n', '').
    ↪ split(',')
print len(aflowlib_entries), " MoTi entries
    ↪ found"
print 'Fetching data:'

# For each entry get its json
entries_data = []
count = 1
for entry in aflowlib_entries:
    obj = {}
    obj['prototype'] = entry

    # auid
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?auid')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['auid'] = res

    # get species
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?species')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['species'] = res.split(',')

    # get composition
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?composition')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['composition'] = [int(d) for d in
        ↪ res.split(',')]

    # get stoichiometry
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?stoichiometry')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['stoichiometry'] = [float(d) for d
        ↪ in res.split(',')]


    # formation entropy per atom
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?enthalpy_formation_atom
        ↪ ')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['enthalpy_formation_atom'] = float(
        ↪ res)

    # sg relax
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '?spacegroup_relax')
    res = urllib2.urlopen(req).read().
        ↪ replace('\n', '')
    obj['spacegroup_relax'] = int(res)

    # POSCAR
    req = urllib2.Request(BASE_URL + '/' +
        ↪ entry + '/CONTCAR.relax')
    res = urllib2.urlopen(req).read()
    poscar_lines = res.split('\n')
    poscar_lines.insert(5, ' '.join(obj['
        ↪ species']))
    poscar = '\n'.join(poscar_lines)
    obj['poscar'] = poscar

    print '%s of %s              %s' % (
        count,
        len(aflowlib_entries),
        BASE_URL + '/' + entry
    )
    count += 1
    entries_data.append(obj)


print 'Writing to json'
with open('entries_data.json', 'w') as
    ↪ outfile:
    json.dump(
        entries_data,
        outfile,
        sort_keys=True,
        indent=2,
        ensure_ascii=False
    )
```

The resulting JSON file, `entries_data.json`, contains an array of objects holding the desired set of information for each MoTi entry. Finally, each POSCAR is passed to the AFLOW-ML API client to retrieve the predictions.

```
from AFLOWml.client import AFLOWmlAPI
import json

with open('entries_data.json', 'r') as
    ↪ json_file:
    entries_data = json.load(json_file)
    predictions = []
    ml = AFLOWmlAPI()
    count = 1
    print "Fetching predictions:"
    for entry in entries_data:
        print '%s of %s              %s' %
            ↪ (
            count,
            len(entries_data),
            entry['prototype']
        )
        count += 1
        prediction = ml.get_prediction(entry
            ↪ ['poscar'], 'plmf')
        # merge entry and prediction dict
        merged = prediction.copy()
        merged.update(entry)
        predictions.append(merged)


print 'Writing predictions.json'
with open('predictions.json', 'w') as
    ↪ outfile:
    json.dump(
        predictions,
        outfile,
        sort_keys=True,
        indent=2,
        ensure_ascii=False
    )
```

Predictions are obtained from the `plmf` model and saved to the JSON file, `predictions.json`. This file also holds an array of objects, where each object contains the prediction results merged with the entry information found in `entries_data.json`. At this point, the AUIDs for each hull point, retrieved by filtering `aflow_MoTi_hull.json`, are used to identify their predictions found within `predictions.json`.

The shear and bulk moduli vs. composition are plotted for each composition on the convex hull in Figure 2. The top two panels (b,c) show the predicted moduli for each convex hull point (orange points) along with the range of predicted values for each entry that shares the same composition, while the bottom panels (d,e) display the distribution for equi-composition.

The plots in Figure 2(b,c) indicate that the elastic moduli generally decrease with reducing Mo content. Furthermore, Figure 2(d,e) demonstrate that while the range of predicted moduli for hull member compositions is large, the majority of predicted values cluster around the values for the entries on the hull. This suggests that the mechanical properties of the hull entries are generally representative of all entries sharing the same composition, and may also be partly due to the relaxation of many initial structures to the lowest energy one. Additionally, it is evident that the hull point bulk modulus values are always near the maximum predicted value, which is expected since it is known that this property correlates strongly with cohesive energy [53], of which the formation enthalpy is a component. This is supported by the strong dependence of the bulk modulus on the formation enthalpy for a given stoichiometry, as indicated by Figure 2(f).

B. Vibrational free energy and entropy of perovskite oxides (ABO_3)

The second example leverages the `mfd` ML model to investigate the vibrational free energy and entropy for perovskite oxides (ABO_3). POSCAR files are generated using the `aflow --proto` command [16]:

```
aflow --proto=AB3C_cP5_221_a_c_b:A:O:B
--params=3.795 > AOB.poscar
```

where A and B are set for each combination of the following elements: Al, Si, S, P, Pd, Se, Ni, N, B, C, H, Ga, Pt, Sn, In, Ge, Er, Rh, Li, Zn, Te, K, Cu, Au, Ir, Ca, Ho, Br, Y, As, Pb, La, Ti, Sb, Bi, Cs, Ba, Tl, Cl, Co, Ce, Sr, Na, Rb, V, Cd, Ta and Sc. This results in 1,126 POSCAR files, stored in a directory named `POSCARS`. Note that the generated POSCAR files are not in POSCAR 5 format and require conversion. Next, the AFLOW-ML API client is used to retrieve predictions for the vibrational free energy and entropy using the `mfd` model. As in the previous example, output is saved in JSON format to a file titled `predictions.json`.

```
import json
import os
from AFLOWml.client import AFLOWmlAPI

predictions = []
ml = AFLOWmlAPI()
for root, dirs, files in os.walk('POSCARS'):
    count = 1
    for f in files:
        if f.endswith('.poscar'):
            print '%s of %s        %s' % (
                count,
                len(files),
                f
            )
            count += 1
            with open(root + '/' + f, 'r')
               ↪ as poscar:
                prediction = ml.
                    ↪ get_prediction(poscar
                    ↪ .read(), 'mfd')
                prediction['name'] = f.
                    ↪ replace('.poscar', ''
                    ↪ ) + 'O'
                predictions.append(
                    ↪ prediction)


print 'Writing predictions.json'
with open('predictions.json', 'w') as
    ↪ outfile:
    json.dump(
        predictions,
        outfile,
        sort_keys=True,
        indent=2,
        ensure_ascii=False
    )
```

Heat maps for each AB combination are plotted in Figure 3 for the predicted vibrational free energy, F_{vib}, and entropy, S_{vib}. Two tendencies are visible: F_{vib} decreases and S_{vib} increases with atomic size; and local minima in F_{vib} and maxima in S_{vib} occur at compositions containing alkali metals. In general, heavy atoms with large radii result in lower vibrational frequencies, resulting in lower values of internal vibrational energy $U_{\text{vib}}(\sim \hbar\omega)$, and higher values of S_{vib}. Since $F_{\text{vib}} = U_{\text{vib}} - TS_{\text{vib}}$, this results in F_{vib} having lower values for materials containing heavier atoms (towards the bottom right of the plot), as well as for compositions containing alkali metals that have large radii resulting in weak interatomic bonds and thus low vibrational frequencies. The inverse of this relationship is seen in panel (b), with maxima in S_{vib} occurring for compositions containing alkali metals or very heavy elements such as Tl and Pb.

Rapidly generated plots such as these facilitate the discovery of design rules, enabling the identification of suitable compositions for specific applications. It also provides insight into the criteria used by the machine-learning algorithms to predict materials properties.

IX. CONCLUSION

AFLOW-ML enhances materials discovery by providing streamlined open access to predictive models. The REST API promotes resource sharing, where any application, workflow

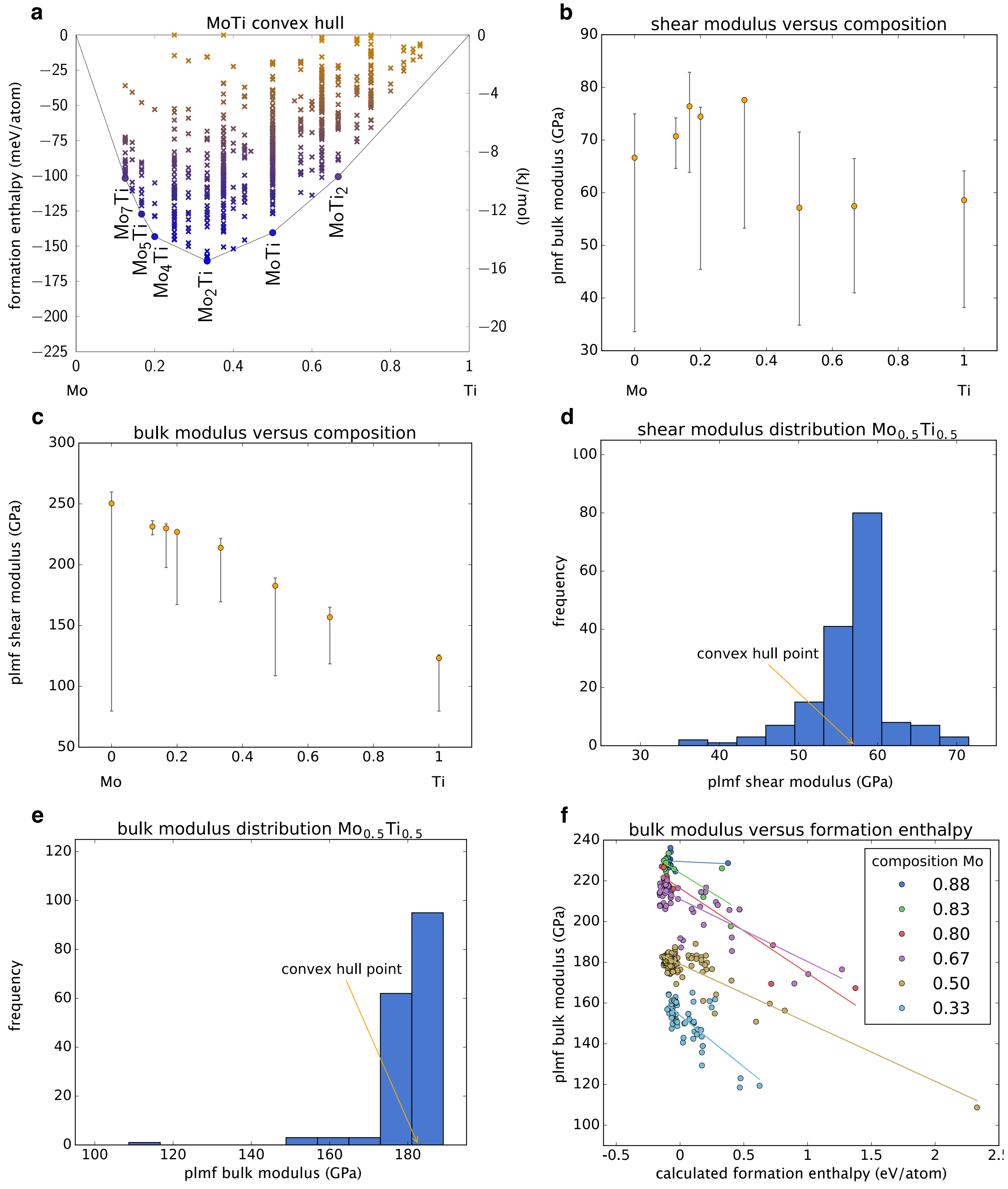
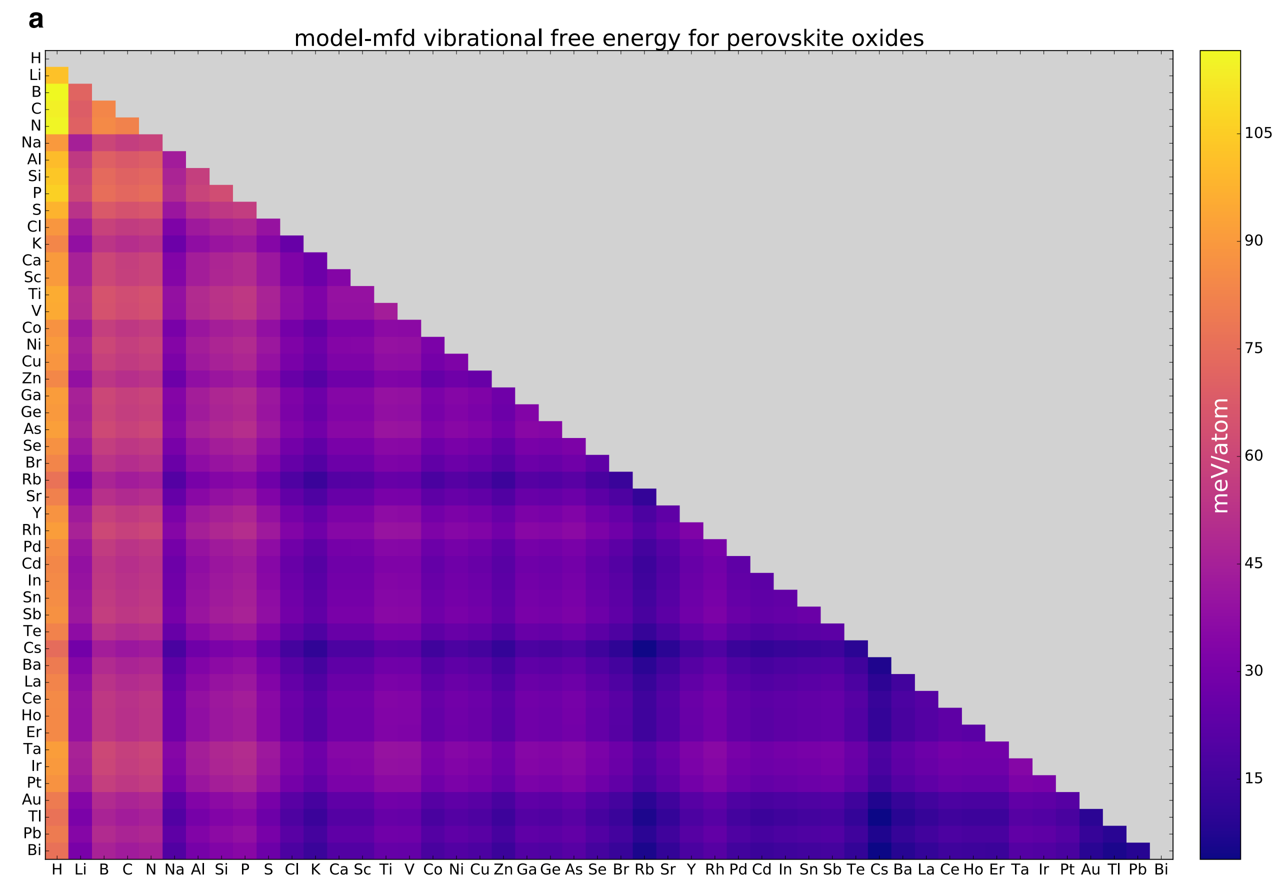


FIG. 2. The predicted shear and bulk modulus for MoTi. (**a**) The convex hull for MoTi. (**b**, **c**) The predicted moduli vs. composition for all entries sharing a hull member composition, where the vertical bars indicate the range of values for all of the structures in AFLOW database for a given stoichiometry. (**d**, **e**) The distribution for equi-composition. (**f**) The predicted bulk modulus vs. the calculated formation enthalpy for each hull point composition.



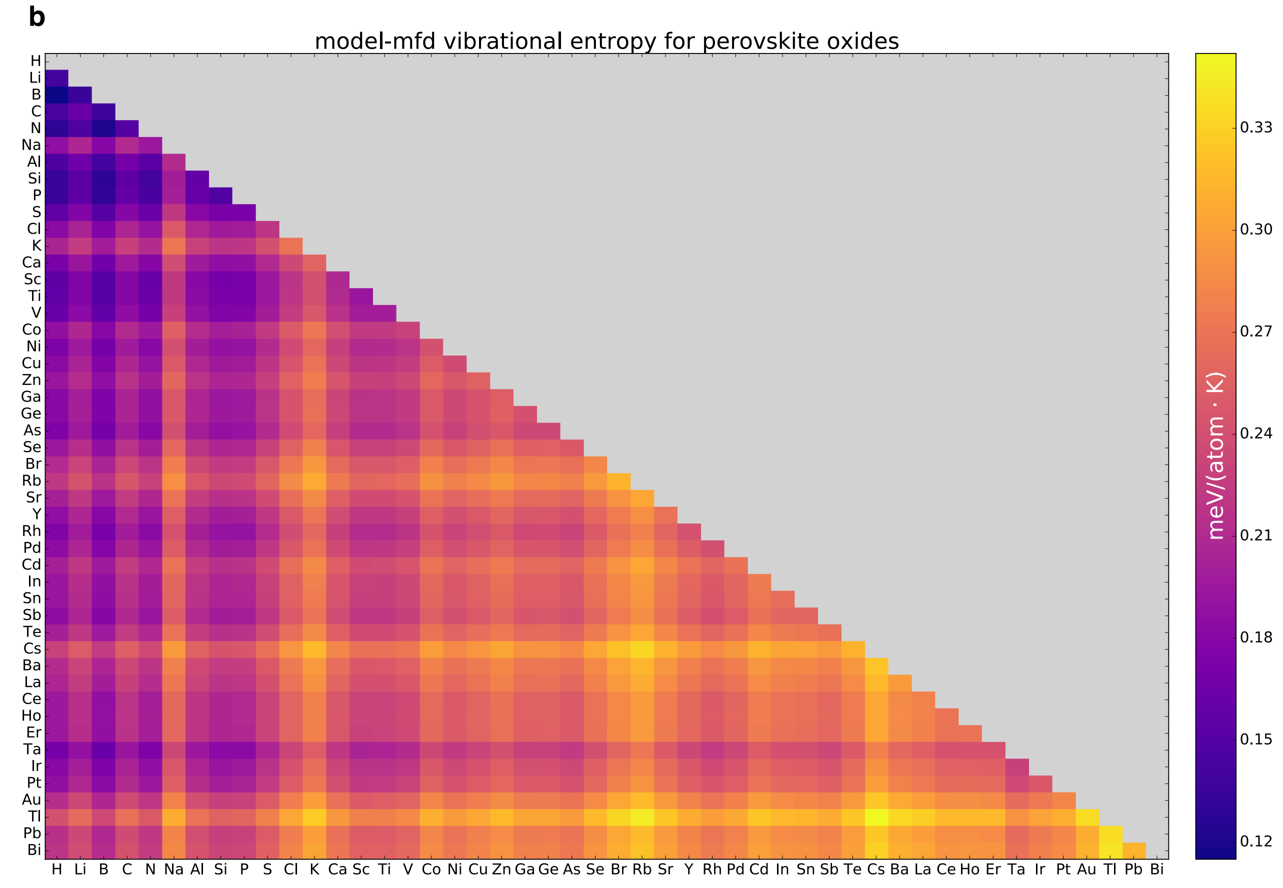


FIG. 3. The predicted vibrational free energy (**a**) and entropy (**b**) for perovskite oxides (ABO_3).

or website may leverage our models. Additionally, the Python client provides a closed solution, which requires little programming knowledge to get started. With this flexibility, AFLOW-ML presents the accessible option for machine learning in the materials design community.

ACKNOWLEDGMENTS

This work is partially supported by DOD-ONR (N00014-14-1-0526, N00014-16-1-2326, N00014-16-1-2583, N00014-17-1-2090), by NSF (DMR-1436151), and by the Duke University—Center for Materials Genomics. C.O. acknowledges support from the National Science Foundation Graduate Research Fellowship under Grant No. DGF-1106401. A.T. and O.I. acknowledge support from DOD-ONR N00014-16-1-2311 and Eshelman Institute for Innovation Award. O.I. acknowledges Extreme Science and Engineering Discovery Environment (XSEDE) award DMR-110088, which is supported by National Science Foundation grant number ACI-1053575. F.L. and N.M. acknowledge support from m-era.net project "ICETS" (ANR-14-MERA-0003-03) and ANR Carnot "MAPPE". S.C. acknowledges the Alexander von Humboldt Foundation and the Max Planck Society for financial support.

DATA AVAILABILITY

The calculated data required to reproduce the convex hull in Figure 2(a) and the structural input data for the AFLOW-ML API are available to download from `http://www.aflow.org/` using the AFLOW data API [11], with detailed instructions provided in Section VIII. The AFLOW-ML predicted data required to reproduce these findings can be generated and retrieved using the AFLOW-ML API from `http://www.aflow.org/aflow-ml/`, following the instructions provided in Section VIII.

[1] S. Curtarolo, W. Setyawan, G. L. W. Hart, M. Jahnátek, R. V. Chepulskii, R. H. Taylor, S. Wang, J. Xue, K. Yang, O. Levy, M. J. Mehl, H. T. Stokes, D. O. Demchenko, and D. Morgan, *AFLOW: An automatic framework for high-throughput materials discovery*, Comput. Mater. Sci. **58**, 218–226 (2012).

[2] K. Yang, C. Oses, and S. Curtarolo, *Modeling Off-Stoichiometry Materials with a High-Throughput Ab-Initio Approach*, Chem. Mater. **28**, 6484–6492 (2016).

[3] C. E. Calderon, J. J. Plata, C. Toher, C. Oses, O. Levy, M. Fornari, A. Naturen, M. J. Mehl, G. L. W. Hart, M. Buongiorno Nardelli, and S. Curtarolo, *The AFLOW standard for high-throughput materials science calculations*, Comput. Mater. Sci. **108 Part A**, 233–238 (2015).

[4] O. Levy, M. Jahnátek, R. V. Chepulskii, G. L. W. Hart, and S. Curtarolo, *Ordered Structures in Rhenium Binary Alloys from First-Principles Calculations*, J. Am. Chem. Soc. **133**, 158–163 (2011).

[5] O. Levy, G. L. W. Hart, and S. Curtarolo, *Structure maps for hcp metals from first-principles calculations*, Phys. Rev. B **81**, 174106 (2010).

[6] O. Levy, G. L. W. Hart, and S. Curtarolo, *Uncovering Compounds by Synergy of Cluster Expansion and High-Throughput Methods*, J. Am. Chem. Soc. **132**, 4830–4833 (2010).

[7] G. L. W. Hart, S. Curtarolo, T. B. Massalski, and O. Levy, *Comprehensive Search for New Phases and Compounds in Binary Alloy Systems Based on Platinum-Group Metals, Using a Computational First-Principles Approach*, Phys. Rev. X **3**, 041035 (2013).

[8] A. R. Supka, T. E. Lyons, L. S. I. Liyanage, P. D'Amico, R. Al Rahal Al Orabi, S. Mahatara, P. Gopal, C. Toher, D. Ceresoli, A. Calzolari, S. Curtarolo, M. Buongiorno Nardelli, and M. Fornari, *AFLOWπ: A minimalist approach to high-throughput ab initio calculations including the generation of tight-binding hamiltonians*, Comput. Mater. Sci. **136**, 76–84 (2017).

[9] S. Curtarolo, W. Setyawan, S. Wang, J. Xue, K. Yang, R. H. Taylor, L. J. Nelson, G. L. W. Hart, S. Sanvito, M. Buongiorno Nardelli, N. Mingo, and O. Levy, *AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations*, Comput. Mater. Sci. **58**, 227–235 (2012).

[10] W. Setyawan and S. Curtarolo, *High-throughput electronic band structure calculations: Challenges and tools*, Comput. Mater. Sci. **49**, 299–312 (2010).

[11] R. H. Taylor, F. Rose, C. Toher, O. Levy, K. Yang, M. Buongiorno Nardelli, and S. Curtarolo, *A RESTful API for exchanging materials data in the AFLOWLIB.org consortium*, Comput. Mater. Sci. **93**, 178–192 (2014).

[12] F. Rose, C. Toher, E. Gossett, C. Oses, M. Buongiorno Nardelli, M. Fornari, and S. Curtarolo, *AFLUX: The LUX materials search API for the AFLOW data repositories*, Comput. Mater. Sci. **137**, 362–370 (2017).

[13] G. Bergerhoff, R. Hundt, R. Sievers, and I. D. Brown, *The inorganic crystal structure data base*, J. Chem. Inf. Comput. Sci. **23**, 66–69 (1983).

[14] A. D. Mighell and V. L. Karen, *NIST Materials Science Databases*, Acta Crystallogr. Sect. A **49**, c409 (1993).

[15] A. Belsky, M. Hellenbrandt, V. L. Karen, and P. Luksch, *New developments in the Inorganic Crystal Structure Database (ICSD): accessibility in support of materials research and design*, Acta Crystallogr. Sect. B **58**, 364–369 (2002).

[16] M. J. Mehl, D. Hicks, C. Toher, O. Levy, R. M. Hanson, G. L. W. Hart, and S. Curtarolo, *The AFLOW Library of Crystallographic Prototypes: Part 1*, Comput. Mater. Sci. **136**, S1–S828 (2017).

[17] A. Jain, G. Hautier, C. J. Moore, S. P. Ong, C. C. Fischer, T. Mueller, K. A. Persson, and G. Ceder, *A high-throughput infrastructure for density functional theory calculations*, Comput. Mater. Sci. **50**, 2295–2310 (2011).

[18] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. A. Persson, *Commentary: The Materials Project: A materials genome approach to accelerating materials innovation*, APL Mater. **1**, 011002 (2013).

[19] S. P. Ong, W. D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V. L. Chevrier, K. A. Persson, and G. Ceder, *Python Materials Genomics (py-*

matgen): A robust, open-source python library for materials analysis, Comput. Mater. Sci. **68**, 314–319 (2013).

[20] *The Materials Project*, www.materialsproject.org.

[21] M. Scheffler, C. Draxl, and Computer Center of the Max-Planck Society, Garching, *The NoMaD Repository*, http://nomad-repository.eu (2014).

[22] J. E. Saal, S. Kirklin, M. Aykol, B. Meredig, and C. Wolverton, *Materials Design and Discovery with High-Throughput Density Functional Theory: The Open Quantum Materials Database (OQMD)*, JOM **65**, 1501–1509 (2013).

[23] *The Open Quantum Materials Database*, wwww.oqmd.org.

[24] S. R. Bahn and K. W. Jacobsen, *An object-oriented scripting interface to a legacy electronic structure code*, Comput. Sci. Eng. **4**, 56–66 (2002).

[25] D. D. Landis, J. Hummelshøj, S. Nestorov, J. Greeley, M. Dułak, T. Bligaard, J. K. Nørskov, and K. W. Jacobsen, *The Computational Materials Repository*, Comput. Sci. Eng. **14**, 51–57 (2012).

[26] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, *AiiDA: automated interactive infrastructure and database for computational science*, Comput. Mater. Sci. **111**, 218–230 (2016).

[27] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, *AiiDA*, http://www.aiida.net (2016).

[28] L. Breiman, *Random Forests*, Mach. Learn. **45**, 5–32 (2001).

[29] C. Cortes and V. Vapnik, *Support-vector networks*, Mach. Learn. **20**, 273–297 (1995).

[30] J. H. Friedman, *Greedy Function Approximation: A Gradient Boosting Machine*, Ann. Stat. **29**, 1189–1232 (2001).

[31] H. K. D. H. Bhadeshia, *Neural Networks in Materials Science*, ISIJ Int. **39**, 966–979 (1999).

[32] H. K. D. H. Bhadeshia, R. C. Dimitriu, S. Forsik, J. H. Pak, and J. H. Ryu, *Performance of neural networks in materials science*, Mater. Sci. Technol. **25**, 504–510 (2009).

[33] Y. Zeng, S. J. Chua, and P. Wu, *On the Prediction of Ternary Semiconductor Properties by Artificial Intelligence Methods*, Chem. Mater. **14**, 2989–2998 (2002).

[34] T. Gu, W. Lu, X. Bao, and N. Chen, *Using support vector regression for the prediction of the band gap and melting point of binary and ternary compound semiconductors*, Solid State Sci. **8**, 129–136 (2006).

[35] L. M. Ghiringhelli, J. Vybiral, S. V. Levchenko, C. Draxl, and M. Scheffler, *Big Data of Materials Science: Critical Role of the Descriptor*, Phys. Rev. Lett. **114**, 105503 (2015).

[36] E. O. Pyzer-Knapp, K. Li, and A. Aspuru-Guzik, *Learning from the Harvard Clean Energy Project: The Use of Neural Networks to Accelerate Materials Discovery*, Adv. Func. Mater. **25**, 6495–6502 (2015).

[37] R. Gómez-Bombarelli, J. Aguilera-Iparraguirre, T. D. Hirzel, D. Duvenaud, D. Maclaurin, M. A. Blood-Forsythe, H. S. Chae, M. Einzinger, D.-G. Ha, T. Wu, G. Markopoulos, S. Jeon, H. Kang, H. Miyazaki, M. Numata, S. Kim, W. Huang, S. I. Hong, M. Baldo, R. P. Adams, and A. Aspuru-Guzik, *Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach*, Nature Mater. **15**, 1120–1127 (2016).

[38] M. Ziatdinov, A. Maksov, and S. V. Kalinin, *Learning surface molecular structures via machine vision*, NPJ Comput. Mater. **3**, 31 (2017).

[39] S. R. Michalski, G. J. Carbonell, and M. T. Mitchell, eds., *Machine Learning an Artificial Intelligence Approach Volume II* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1986).

[40] A. van Roekeghem, J. Carrete, C. Oses, S. Curtarolo, and N. Mingo, *High-Throughput Computation of Thermal Conductivity of High-Temperature Solid Phases: The Case of Oxide and Fluoride Perovskites*, Phys. Rev. X **6**, 041061 (2016).

[41] G. Pilania, A. Mannodi-Kanakkithodi, B. P. Uberuaga, R. Ramprasad, J. E. Gubernatis, and T. Lookman, *Machine learning bandgaps of double perovskites*, Sci. Rep. **6**, 19375 (2016).

[42] G. Hautier, C. C. Fischer, A. Jain, T. Mueller, and G. Ceder, *Finding Naturere's Missing Ternary Oxide Compounds using Machine Learning and Density Functional Theory*, Chem. Mater. **22**, 3762–3767 (2010).

[43] F. A. Faber, A. Lindmaa, O. A. von Lilienfeld, and R. Armiento, *Machine Learning Energies of 2 Million Elpasolite (ABC_2D_6) Crystals*, Phys. Rev. Lett. **117**, 135502 (2016).

[44] G. Pilania, J. E. Gubernatis, and T. Lookman, *Multi-fidelity machine learning models for accurate bandgap predictions of solids*, Comput. Mater. Sci. **129**, 156–163 (2017).

[45] J. Carrete, W. Li, N. Mingo, S. Wang, and S. Curtarolo, *Finding Unprecedentedly Low-Thermal-Conductivity Half-Heusler Semiconductors via High-Throughput Materials Modeling*, Phys. Rev. X **4**, 011019 (2014).

[46] J. Carrete, N. Mingo, S. Wang, and S. Curtarolo, *Nanograined Half-Heusler Semiconductors as Advanced Thermoelectrics: An Ab Initio High-Throughput Statistical Study*, Adv. Func. Mater. **24**, 7427–7432 (2014).

[47] A. Furmanchuk, J. E. Saal, J. W. Doak, G. B. Olson, A. Choudhary, and A. Agrawal, *Prediction of seebeck coefficient for compounds without restriction to fixed stoichiometry: A machine learning approach*, J. Comput. Chem. **39**, 191–202 (2018).

[48] Y. T. Sun, H. Y. Bai, M. Z. Li, and W. H. Wang, *Machine Learning Approach for Prediction and Understanding of Glass-Forming Ability*, J. Phys. Chem. Lett. **8**, 3434–3439 (2017).

[49] O. Isayev, C. Oses, C. Toher, E. Gossett, S. Curtarolo, and A. Tropsha, *Universal fragment descriptors for predicting electronic properties of inorganic crystals*, Nature Commun. **8**, 15679 (2017).

[50] F. Legrain, J. Carrete, A. van Roekeghem, S. Curtarolo, and N. Mingo, *How Chemical Composition Alone Can Predict Vibrational Free Energies and Entropies of Solids*, Chem. Mater. **29**, 6220–6227 (2017).

[51] A. Furmanchuk, A. Agrawal, and A. Choudhary, *Predictive analytics for crystalline materials: bulk modulus*, RSC Adv. **6**, 95246–95251 (2016).

[52] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, *A general-purpose machine learning framework for predicting properties of inorganic materials*, NPJ Comput. Mater. **2**, 16028 (2016).

[53] M. de Jong, W. Chen, R. Notestine, K. A. Persson, G. Ceder, A. Jain, M. D. Asta, and A. Gamst, *A Statistical Learning Framework for Materials Science: Application to Elastic Moduli of k-nary Inorganic Polycrystalline Compounds*, Sci. Rep. **6**, 34256 (2016).

[54] D. W. Davies, K. T. Butler, A. J. Jackson, A. Morris, J. M. Frost, J. M. Skelton, and A. Walsh, *Computational Screening of All Stoichiometric Inorganic Materials*, Chem **1**, 617–627 (2016).

[55] W. F. Reinhart, A. W. Long, M. P. Howard, A. L. Ferguson, and A. Z. Panagiotopoulos, *Machine learning for autonomous crystal structure identification*, Soft Matter **13**, 4733–4745 (2017).

[56] O. Isayev, D. Fourches, E. N. Muratov, C. Oses, K. Rasch, A. Tropsha, and S. Curtarolo, *Materials Cartography: Representing and Mining Materials Space Using Structural and Electronic Fingerprints*, Chem. Mater. **27**, 735–743 (2015).

[57] L. Ward, R. Liu, A. Krishna, V. I. Hegde, A. Agrawal, A. Choudhary, and C. Wolverton, *Including crystal structure attributes in machine learning models of formation energies via Voronoi tessellations*, Phys. Rev. B **96**, 024104 (2017).

[58] T. Xie and J. C. Grossman, *Crystal Graph Convolutional Neural Networks for Accurate and Interpretable Prediction of Material Properties*, submitted arxiv.org/abs/1710.10324 (2017).

[59] G. Pilania, C. Wang, X. Jiang, S. Rajasekaran, and R. Ramprasad, *Accelerating materials property predictions using machine learning*, Sci. Rep. **3**, 2810 (2013).

[60] B. Meredig, A. Agrawal, S. Kirklin, J. E. Saal, J. W. Doak, A. Thompson, K. Zhang, A. Choudhary, and C. Wolverton, *Combinatorial screening for new materials in unconstrained composition space with machine learning*, Phys. Rev. B **89**, 094104 (2014).

[61] R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L. M. Ghiringhelli, *SISSO: a compressed-sensing method for systematically identifying efficient physical models of materials properties*, submitted arxiv.org/1710.03319 (2017).

[62] F. Ruggiu, G. Marcou, A. Varnek, and D. Horvath, *ISIDA Property-Labelled Fragment Descriptors*, Mol. Informatics **29**, 855–868 (2010).

[63] G. Kresse and J. Hafner, *Ab initio molecular dynamics for liquid metals*, Phys. Rev. B **47**, 558–561 (1993).

[64] G. Kresse and J. Furthmüller, *Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set*, Phys. Rev. B **54**, 11169–11186 (1996).

[65] K. Reitz, *Python Requests Library*, `http://docs.python-requests.org/en/master/`.

[66] Apple, *URLSession*, `https://developer.apple.com/documentation/foundation/urlsession/`.

[67] Oracle, *HttpURLConnection*, `https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html`.

[68] The Web Hypertext Application Technology Working Group. (WHATWG), *Fetch*, `https://fetch.spec.whatwg.org/`.

[69] C. Toher, C. Oses, J. J. Plata, D. Hicks, F. Rose, O. Levy, M. de Jong, M. D. Asta, M. Fornari, M. Buongiorno Nardelli, and S. Curtarolo, *Combining the AFLOW GIBBS and Elastic Libraries to efficiently and robustly screen thermomechanical properties of solids*, Phys. Rev. Mater. **1**, 015401 (2017).

[70] C. Toher, J. J. Plata, O. Levy, M. de Jong, M. D. Asta, M. Buongiorno Nardelli, and S. Curtarolo, *High-throughput computational screening of thermal conductivity, Debye temperature, and Grüneisen parameter using a quasiharmonic Debye model*, Phys. Rev. B **90**, 174107 (2014).

[71] S. Barzilai, C. Toher, S. Curtarolo, and O. Levy, *Molybdenum-titanium phase diagram evaluated from ab initio calculations*, Phys. Rev. Mater. **1**, 023604 (2017).