The Journal of Chemical Physics

The OpenKIM processing pipeline: A cloud-based automatic material property computation engine

Cite as: J. Chem. Phys. **153**, 064104 (2020); https://doi.org/10.1063/5.0014267
Submitted: 18 May 2020 . Accepted: 20 July 2020 . Published Online: 10 August 2020

D. S. Karls, M. Bierbaum, A. A. Alemi, R. S. Elliott, J. P. Sethna, and E. B. Tadmor

AIP Publishing

The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

The OpenKIM processing pipeline: A cloud-based automatic material property computation engine

Cite as: J. Chem. Phys. 153, 064104 (2020); doi: 10.1063/5.0014267
Submitted: 18 May 2020 • Accepted: 20 July 2020 •
Published Online: 10 August 2020

D. S. Karls,[1] M. Bierbaum,[2] A. A. Alemi,[3] R. S. Elliott,[1] J. P. Sethna,[4] and E. B. Tadmor[1,a)]

AFFILIATIONS

[1]Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, Minnesota 55455, USA
[2]Department of Information Science, Cornell University, Ithaca, New York 14850, USA
[3]Google Research, Mountain View, California 94043, USA
[4]Department of Physics, Cornell University, Ithaca, New York 14850, USA

Note: This paper is part of the JCP Special Topic on Classical Molecular Dynamics (MD) Simulations: Codes, Algorithms, Force Fields, and Applications.
[a)]**Author to whom correspondence should be addressed:** tadmor@umn.edu

ABSTRACT

The Open Knowledgebase of Interatomic Models (OpenKIM) is a framework intended to facilitate access to standardized implementations of interatomic models for molecular simulations along with computational protocols to evaluate them. These protocols include tests to compute material properties predicted by models and verification checks to assess their coding integrity. While housing this content in a unified, publicly available environment constitutes a major step forward for the molecular modeling community, it further presents the opportunity to understand the range of validity of interatomic models and their suitability for specific target applications. To this end, OpenKIM includes a computational pipeline that runs tests and verification checks using all available interatomic models contained within the OpenKIM Repository at https://openkim.org. The OpenKIM Processing Pipeline is built on a set of Docker images hosted on distributed, heterogeneous hardware and utilizes open-source software to automatically run test–model and verification check–model pairs and resolve dependencies between them. The design philosophy and implementation choices made in the development of the pipeline are discussed as well as an example of its application to interatomic model selection.

I. INTRODUCTION

As computational resources become more powerful, cheaper, and more prevalent, the use of molecular simulations is becoming increasingly prominent in the understanding and prediction of material properties. The most accurate methods used in this domain are first principles approaches based on a fully quantum mechanical model of the potential energy surface, but these remain prohibitively expensive for many problems of interest. Often, in order to reduce computational complexity, approximate interatomic models (referred to as *interatomic potentials* or *force fields*) are developed that eschew the electronic degrees of freedom in favor of a purely classical coarse-grained description of atomic interactions. The predictive power of these simulations hinges delicately on a number of factors including the form of the model and its parameters, the physical properties under scrutiny, and the simulation method.

The development of new interatomic models is a daunting task requiring a great deal of expertise and time. It is therefore common for researchers to adopt models for their simulations developed by other groups and published in the literature. This can be difficult in practice, since in many cases the computer code used to generate the published results is not available with the article and may not even be archived by the authors themselves. Implementations of the model may exist in simulation packages, but these are often unverified with unreliable provenance and so may not be consistent with

the published work. This leaves other researchers to independently implement and test interatomic models based on the description found in the literature, adding greatly to the barrier to adoption.

The Open Knowledgebase of Interatomic Models (OpenKIM, KIM)[1,2] established in 2009 and funded through the U.S. National Science Foundation aims to solve the scientific and practical issues of material simulations that use interatomic models through a comprehensive cyber infrastructure. OpenKIM is hosted at https://openkim.org and includes a repository for storing computer implementations of interatomic models, computational protocols to evaluate them including tests to compute materials property predictions and verification checks to assess their coding integrity, and first-principles and experimental results that serve as reference data for comparison. The computational protocols can be standalone but are typically applied through an existing molecular simulation platform ("simulator"). The process necessary for these computations to run with an interatomic model is managed through a lightweight middleware library known as the KIM Application Programming Interface (API).[3] The KIM API formally defines an abstract representation of the data and processing directives necessary to perform a molecular simulation, and provides a programmatic cross-language implementation capable of efficiently communicating them between models and simulators. Any interatomic model code and simulator that conform to the KIM API standard are thus capable of functioning together seamlessly; currently supported simulators include ASAP,[4] ASE,[5,6] DL_POLY,[7] GULP,[8] Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS),[9] libatoms/QUIP,[10] MDStressLab++,[11,12] potfit,[13–15] pyiron,[16] and quasicontinuum (QC).[17,18]

The importance of archiving interatomic models has been recognized by others who have, in turn, established similar projects including the NIST Interatomic Potentials Repository (IPR)[19,20] and Jarvis-FF.[21,22] However, there are two significant differences between these projects and OpenKIM. First, as alluded to above, an interatomic model archived in OpenKIM is a software package that includes all the code necessary to evaluate the model to obtain the energy, forces, stresses and related values for a given atomic configuration. This should be contrasted with repositories that only archive model parameter files to be used with implementations in specific molecular simulation codes. Archiving the model code is important, not only because it allows the model to function as a self-contained library that can be used in a portable fashion with many simulators, but also because the implementation of a model is typically complex, making it susceptible to programming errors and often requiring optimization. This complexity gives rise to subtle effects in some cases, e.g., the specifics of the splines comprising the functional forms in a tabulated interatomic model have been shown to affect its predictions for some properties.[23] Maintaining this code (and its history) is paramount in avoiding duplicated development effort. A second major distinction, and the focal point of this work, is that all of the models and computational protocols in OpenKIM are paired with one another and executed in a completely automated manner via a distributed, cloud-based platform known as the *OpenKIM Processing Pipeline* (hereafter, "the pipeline"). Material property predictions computed in this fashion are inserted into a publicly accessible database alongside corresponding first-principles and experimental data, and aid in the analysis of individual models as well as the comparison of different models. These results are available through a publicly accessible mongo database hosted at https://query.openkim.org and a simplified query API through the KIM-query python package[24] and integrated within some simulators such as LAMMPS.[9]

II. OVERVIEW

A. Content in KIM

Before turning attention to the pipeline itself, it is first necessary to survey the various types of content in OpenKIM that pass through it. (Note that below, the standard KIM terminology is indicated using a san-serif font, e.g., Model refers to an interatomic model in the OpenKIM system.) The following are items of the OpenKIM content addressed by the pipeline:

- Model

  An algorithm representing a specific interaction between atoms, e.g., an interatomic potential or force field. There are two primary types of Models: *portable models*, which can be used with any KIM API-compliant simulation code, and *simulator models*, which only work with a specific simulation code. Portable models can either be *standalone* or *parameterized*. Standalone models consist of both a parameter file and the corresponding source code that implements the functional form of an interatomic model. Because the same source code is often reused across multiple parameter sets, KIM also allows it to be encapsulated in a Model Driver, and parameterized models thus contain only parameter files and a reference to their driver. Simulator models also contain only a parameter file but instead of referencing a Model Driver, they include a set of commands that invoke the implementation of a model found in a particular simulator, e.g., LAMMPS.

- Test

  A computer program that when coupled with a suitable Model, possibly including additional input, calculates a specific prediction (material property) for a particular configuration. Similar to the case of models, the code that performs the computation can typically be reused with different parameter sets, e.g., a code that calculates the lattice constant of face-centered cubic (fcc) Al could, with minor alterations, do the same for fcc Ni. Accordingly, a Test can either be standalone in nature or consist of a parameter file specifying the calculation that is read in by a Test Driver. Each material property computed by a KIM Test conforms to a *property definition*[25] schema defined by the Test developer for that property and archived in OpenKIM. This makes it possible to automatically compare property predictions across different Models and with first-principles or experimental reference data and enables dependencies between Tests (see Sec. V).

- Verification Check

  A computer program that when coupled with a Model examines a particular aspect of its coding correctness. This includes checks for programming errors, failures to satisfy required behaviors such as invariance principles, and determination of general characteristics of the Model's functional form such as smoothness. For example, a Verification Check might

check whether the forces reported by a Model are consistent with the energy it reports, i.e., whether the forces are the negative derivatives of the energy.

All of the above items (including Model Drivers and Test Drivers) are assigned a unique identifier (or "KIM ID") in the OpenKIM repository that includes a three-digit version extension to record their evolution over time. Furthermore, each version is assigned its own digital object identifier (DOI) for persistent accessibility.

The objective of the pipeline is to automatically pair Tests and Verification Checks with compatible Models and execute them. A Test and Model are compatible and can be executed if (1) they are written for compatible versions of the KIM API, and (2) if the atomic species involved in the calculation of the Test are all supported by the Model. Verification Checks are designed to work with any atomic species supported by the Model, and so their compatibility is determined only based on criterion (1). The material property instances generated by executing a specific Test–Model pair are collectively referred to as a Test Result, while the result generated by a Verification Check–Model pair is termed a Verification Result. In either case, if a pair fails to successfully generate a result, it produces an Error. The execution time required to produce each Test Result, Verification Result, or Error is collected and normalized with respect to a whetstone benchmark[26] so as to give a hardware-independent estimation of the computing resources that were consumed.

B. Pipeline architecture

All Models, Tests, and Verification Checks are submitted to the OpenKIM repository through a web application ("Web App") that serves the openkim.org domain and interfaces with the pipeline. Once a submitted item has completed an editorial review process and been approved, a page is created for it that contains metadata associated with the item and links to its source code. The Web App proceeds to notify a separate *Gateway* machine of the new item, which then retrieves the item and inserts it into a publicly accessible database. Next, the Gateway sends a request to a third machine termed the *Director*, whose purpose is to determine the set of all current compatible items that it can be run with. For each compatible match that it finds, the Director creates a *job* (a message corresponding to a Test–Model or Verification Check–Model pair that is to be run) that it communicates back to the Gateway. Each job is claimed by one member of a fleet of *Worker* machines that fetches the corresponding items from the Gateway and executes it; once a given job is completed, its results are synchronized back to the Gateway. After inserting the results into its database, the Gateway returns them to the Web App. A schematic of these machines, the roles they play, and their connectivity is shown in Fig. 1.

To make this concrete, consider a new Model for aluminum (Al) [e.g., an embedded-atom method (EAM) potential[27]] is added to the OpenKIM system. There are many Tests in the system designed to work with Al models. One example is a test that computes the cohesive energy (energy per atom) of Al in the face-centered cubic (fcc) structure in its equilibrium configuration. The Director will create a job coupling the Al fcc cohesive energy test with the new EAM Al potential that will be queued by the Gateway. A worker will pick up this job and perform the computation. The result will be the prediction of the new EAM potential for the cohesive energy of fcc Al. This piece of information (encapsulated in a standard format explained below) will be returned to the Gateway and from there passed on to the Web App for display on openkim.org. Similar calculations will be performed for all Tests that compute Al properties. In addition, the new potential will be subjected to all Verification Checks. The specifics of how such calculations are orchestrated in practice are described in Sec. IV.

Drawing on best practices in API design,[28] the guiding principle of the pipeline architecture is encapsulation: the Web App, Director, and Worker all have specific tasks to carry out on Models, Tests, and Verification Checks, while the primary focus of the Gateway is to keep each of these elements isolated from one another. This division of the pipeline into modular components based on a clear separation of responsibilities is advantageous for two reasons. First, it reaps all of the usual benefits that accompany encapsulation. Only simple public interfaces are exposed by each component, while private data and functions internal to them remain protected from mutation or misuse. This enables changes of arbitrary complexity to the private data structures and functions of

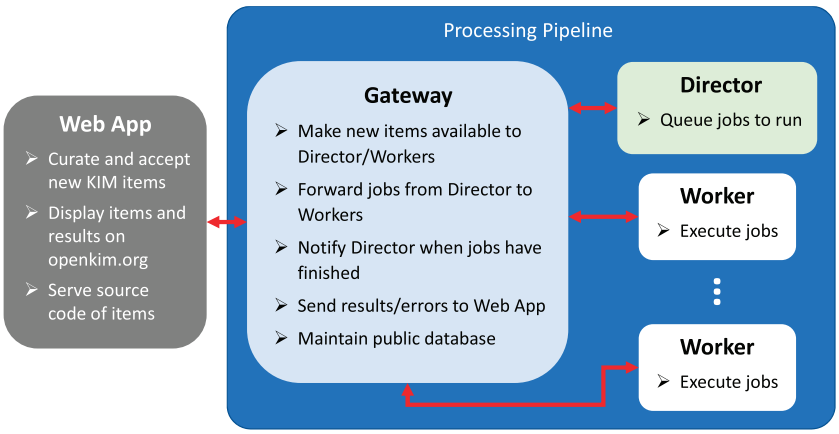

FIG. 1. Abstract architecture of the pipeline and the responsibilities of each component. Arrows indicate connectivity.

a component, which may be necessary for bug fixes or to accommodate changes in software dependencies, without affecting the interaction with neighboring components. The result is comprehensible, maintainable code that is practical to adapt in response to changing design requirements. A secondary advantage of encapsulation is that it naturally facilitates scalability. For example, horizontal scaling of Workers or addition of Directors to accommodate increasing computational demands is straightforward and can be done in a dynamic fashion. High-Performance Computing (HPC) can be accommodated by Workers geared to submission and retrieval of tasks from HPC resources.

III. IMPLEMENTATION

The implementation of the conceptual architecture described in Sec. II B is motivated by three main design objectives:

- **Provenance**—ability to track the origin of and recreate every Test Result, Verification Result, and Error.
- **Flexibility**—ability to run on a wide range of hardware in different physical locations and scale with computational demand.
- **Ease of development**—minimization of initial and ongoing development and maintenance costs.

The first two of these objectives are satisfied with the aid of virtualization. While this could be accomplished using full-fledged virtual machines, the pipeline is instead built upon a basis of Docker images,[29] which have several practical advantages in the pipeline setting. Each individual component is provisioned and stored as a version-controlled Docker image based on Linux from which a container process is spawned that runs the component. The stack-like structure of Docker images is designed to maximize reuse of files, minimizing the amount of data that must be sent over the network when deploying new images to the components. More importantly, because the specific Docker image used to create a component contains a complete specification of its environment, each component and any task it performs is reproducible. In particular, the outcome of any job (Test Result or Verification Result) can be reproduced based on the version of the Docker image used to create the Worker container that ran it. Containerizing the pipeline components using Docker also provides fluid portability because containers can be run on nearly any modern hardware.[30] In the event that the components are run on shared hardware, the process isolation of containers minimizes the risk of interference.

The third objective in the pipeline implementation is ease of development. Because there are various operations specific to the OpenKIM framework and its contents that must be carried out, it is necessary to develop and maintain custom software for the pipeline. The Gateway, Director, and Workers are all based on a single object-oriented codebase written in Python that features classes for the different types of KIM items (Models, Tests, etc.) as well as the Gateway, Director, and Workers themselves, that allow them to perform the tasks shown in Fig. 1. However, aside from this custom software, widely-used packages and protocols are used to the maximum extent possible in order to lower the burden of development and maintenance. The rsync[31] utility is used to transfer the Models, Model Drivers, Tests, Test Drivers, and Verification Checks between the local repositories of KIM items on the Gateway, Director, and Workers. Tornado[32] is used to provide an authenticated web-based HTTPS control API at pipeline.openkim.org accessible to the Web App for submitting new items, as well as a web interface to the public database run by the Gateway, which is implemented using MongoDB,[33] at query.openkim.org. The Director uses SQLite[34] to maintain an internal database for keeping track of jobs and dependencies between them (to be discussed in Sec. V). Finally, Workers include copies of the KIM API-compliant molecular simulation codes mentioned in Sec. I.

The most critical external software packages leveraged in the pipeline are those that connect all of its components: Celery[35] and RabbitMQ.[36] Celery is an open-source distributed task queuing framework written in Python. In this context, a *task* can be thought of as an arbitrary function to be executed on some arguments. In the case of the pipeline, the classes that define the Gateway, Director, and Workers each have a number of member functions that perform some manner of processing on KIM items. Those member functions that must be invoked by other components of the pipeline are thus registered as Celery tasks. Celery prompts the actual execution of its registered tasks by way of message passing. On each component, a Celery daemon is run that waits to receive a message requesting that it execute a specific task with some arguments. For example, a Celery daemon runs on each Worker that waits for a message asking it to execute a specific job. Such a message, which is created by the Director, contains as its arguments the names of the specific Test or Verification Check and Model that are to be run together.

Message passing in Celery is orchestrated by a *message broker*. Although multiple message brokers are available to be used with Celery, RabbitMQ was chosen for the pipeline because of its robustness and extensive feature set. Written in Erlang, RabbitMQ implements message passing using what is known as the advanced message queuing protocol (AMQP).[37] AMQP is a protocol that adheres to the *publisher–subscriber* messaging pattern. Rather than sending messages directly from one component to another, they are placed in extensible buffers called *queues* that are polled by *subscribers* that acquire and process them. In fact, messages are not even sent directly to queues, but rather to *exchanges* that can implement different logic for routing messages to the queues that are bound to it. In the pipeline, however, there is only a single exchange with three queues bound to it: one to which the Gateway subscribes, one to which the Director subscribes, and one to which all of the Workers subscribe. The Gateway publishes messages to the Director queue when it requests that it create jobs for a newly approved KIM item or when a job has finished running, the Director publishes the jobs it creates as messages in the Worker queue, and the Workers publish messages to the Gateway queue as they finish executing jobs. All flow of control in the pipeline is conducted by RabbitMQ, while all execution is handled by Celery.

IV. COMPUTATIONAL WORKFLOW

In order to gain a better understanding of the components of the pipeline and their internals, consider the sequence of operations that occur when a new Test is uploaded to OpenKIM and approved. For the purposes of this example, suppose the newly approved Test,

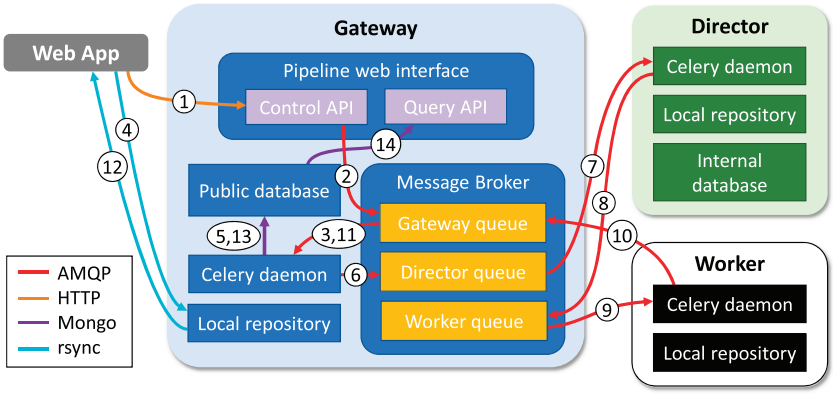

FIG. 2. Internals of the pipeline components and the communication between them when a new item is submitted. See Sec. IV for details. Note that when multiple Workers are running, they all read to and write from the same queues, and the broker ensures that each job is only acquired by a single Worker.

T, computes the lattice constant of fcc Al and there is only a single Model, *M*, for Al that exists in the OpenKIM repository. As pictured in Fig. 2, the Web App begins the submission of *T* to the pipeline by ① notifying its Control API by sending an HTTP request to pipeline.openkim.org.[38] The pipeline control API responds by ② placing a message on the Gateway queue indicating a new item has been submitted. The Celery daemon running on the Gateway polls this queue and ③ acquires the message, causing it to ④ rsync the item from the official OpenKIM repository on the Web App to its own local repository. After ⑤ inserting the item into the public database, the Gateway Celery daemon ⑥ places a message on the Director queue to inform it of the new item. The Director Celery daemon, polling the Director queue, ⑦ acquires this message and rsyncs the item from the local repository of the Gateway to its own local repository. Since the newly received item was a Test, the Director proceeds to loop over all Models that might be compatible with *T*. Finding *M* is compatible with *T*, the Director daemon creates a job message for the pair *T*–*M* and ⑧ places it on the Worker queue. The Worker daemon ⑨ acquires this message from the Worker queue and subsequently executes the job. Once the job has finished running, the Worker announces so by ⑩ placing a corresponding message on the Gateway queue. The Gateway daemon ⑪ acknowledges this message and rsyncs the directory containing the output of the job, which could be either a Test Result or Error, from the local repository of the Worker to its own local repository. The Gateway daemon then ⑫ rsyncs the job output directory from its local repository to the Web App to be placed in the OpenKIM repository and displayed on openkim.org. Finally, the Gateway daemon ⑬ inserts the Test Result or Error into the public-facing database where ⑭ it can be accessed by the Query API hosted at query.openkim.org. A similar process takes place when a new Model or Verification Check is uploaded.

V. DEPENDENCIES

One subtlety not illustrated in the preceding example is that Tests in OpenKIM are allowed to make use of Test Results computed by other Tests. Indeed, this is encouraged whenever possible because creating Tests is typically complicated and they can be expensive to run against even simple Models. Such dependencies between Tests are made possible by the fact that all Test Results (and Verification Results) contain, at a minimum, a file that includes one or more *property instances*,[39] numerical realizations of *property definitions*.[25] Property definitions are intended to embody all physical information necessary to define a material property while ignoring any algorithmic or implementational details related to how they are computed. Each contains a set of keys that represent physical quantities that have a well-defined data type and unit specification, and are either required to be reported in each corresponding property instance or may optionally be supplied. For example, the cohesive energy of a cubic crystal is defined by four required keys: the lattice constant of the conventional unit cell, basis atom coordinates, basis atom species, and the cohesive energy itself. Optional keys include a human-readable name for the crystal type and keys for a precise Wyckoff representation of the crystal structure. By storing Test Results in an explicit, machine-readable format in the public database of the pipeline, other Tests can use them for their own purposes with appropriately crafted queries. These queries can be done in several ways, including simulator-native commands or the KIM-query python package.[24]

The existence of dependencies between Tests places restrictions on the order in which jobs can be scheduled in the pipeline. To manage this, each Test is required to provide a file that lists which other Tests it depends on results from, which we refer to as its *upstream dependencies*.[40] Conversely, the set of Tests that rely on the results of a given Test are termed its *downstream dependents*. Altogether, this means that the collection of all Tests in OpenKIM can be thought of as a directed acyclic graph. There are two mechanisms employed by the pipeline to traverse this structure as it executes jobs, both of which are carried out by the Director: *upstream resolution* and *downstream resolution*. Upstream resolution occurs when a compatible Test–Model pair is first found. Before creating a job for the pair, the Director inspects the dependency file of the Test. If there are Test Results for each pairing of the Tests listed with the Model in question, the job is placed on the Worker queue. However, if any are missing, the Director performs upstream resolution for those pairs. This continues recursively to identify the set of all unique Test–Model pairs that are indirect upstream dependencies of the original Test–Model pair and whose own upstream dependencies are all satisfied. Finally, jobs are created for each pair in this list and placed on the Worker queue. Once the Gateway notifies

the Director of a newly generated Test Result, downstream resolution is carried out. The Director first reads the Test and Model used to generate the Test Result from the message placed on its queue by the Gateway. It then searches its internal database for any Tests that are downstream dependents of the Test indicated in the Test Result message. Any downstream dependents that have any of the others as an upstream dependency are discarded before proceeding.[41] Each remaining downstream dependent is coupled with the Model and upstream resolution is performed on each pair in order to arrive at a unique list of Test–Model pairs to run. Once all of the downstream dependents have been iterated over, jobs are queued for all pairs in the list.

An explicit example is shown in Fig. 3. Suppose there exist several Tests that calculate properties of fcc Al at zero temperature: one that computes the lattice constant (T_{LC}), one that computes the elastic constants (T_{EC}), and one that computes the stress field surrounding a monovacancy using a linear elastic model (T_{V}). The elastic constants Test has the lattice constant Test as its upstream dependency, whereas the vacancy Test has both the elastic constants and lattice constants Tests as its upstream dependencies. Next, assume that a new Model M for Al has been uploaded to the OpenKIM Repository. When the Director is notified of the new model, it begins looping over all current Tests to determine which of them are compatible with the model. For the purposes of this example, assume that the first Test the Director visits is T_{V}. The first phase of dependency resolution is shown in Fig. 3(a) (the circled numbers below refer to dependency resolution steps in the figure). After determining it is a compatible match with M, the Director begins iterating over its upstream dependencies to see if they are satisfied. In the case of a Test with multiple dependencies, the order in which it lists them in its dependency file is arbitrary. Supposing that T_{EC} is listed first, the Director attempts to match it with M and performs upstream resolution on this pair ①. Although it is found to be compatible, ② the Director finds that the upstream dependency of T_{EC}, T_{LC}, has not yet been run against M. Recursing once more, the Director matches T_{LC} with the M and performs upstream resolution on the pair. This time, since T_{LC} has no upstream dependencies, it is determined that the pair is ready to run and it is passed back down to the original upstream resolution that was started at T_{V} to be added to the run list. Having looped over T_{EC} during the original upstream resolution, ③ the Director attempts upstream resolution on T_{V}'s other dependency, T_{LC}. Although it finds that T_{LC} is ready to run against M, the pair is already found in the run list, and so it is ignored. Having completed the upstream resolution from T_{V}, ④ a job is created for the pair T_{LC}–M and pushed to the Worker queue. The next phase of dependency resolution is shown in Fig. 3(b). Assuming the job produces a Test Result (rather than an Error), ⑤ the Director is notified and begins downstream resolution for T_{LC}. Observing that T_{EC} is an upstream dependency of T_{V}, the latter is discarded from consideration, leaving only downstream resolution to T_{EC}. ⑥ Upstream resolution on the pair T_{EC}–M confirms that T_{LC} has been run and that there are no other upstream dependencies, and so ⑦ a job for the pair is created and queued. The final phase of dependency resolution is shown in Fig. 3(c). Once the Test Result corresponding to T_{EC}–M is returned to the Director, ⑧ downstream resolution leads the Director to T_{EC}'s one downstream dependent T_{V}. Now, ⑨ and ⑩ upstream resolution of T_{V}–M indicates that all of its upstream dependencies are met and ⑪ it is run.

VI. APPLICATION TO MODEL SELECTION

A practical application of the data produced by the OpenKIM pipeline is the selection of an interatomic model for a specific target application. To aid in this process, the "KIM Compare" tool[42] aggregates Test Results for a set of properties of interest for a range of Models and displays them to the user in the form of dynamic tables and graphs. The first step is to identify a set of N_{props} properties deemed important for a model to reproduce accurately for the fidelity of the target application, and for which first principles or experimental reference data are available. The absolute relative error between the model prediction and the reference data for each property is defined as

$$e_p^M := \left| \frac{V_p^M - R_p}{R_p} \right|, \tag{1}$$

where V_p^M is the prediction of model M for property p and R_p is a reference value. In order to compare between models, a cost function is defined as a weighted sum (with weights $w_p > 0$) over the relative errors, so that for model M the error cost is

$$\zeta^M := \frac{\sum_{p=1}^{N_{\mathrm{props}}} w_p e_p^M}{\sum_{p=1}^{N_{\mathrm{props}}} w_p}. \tag{2}$$

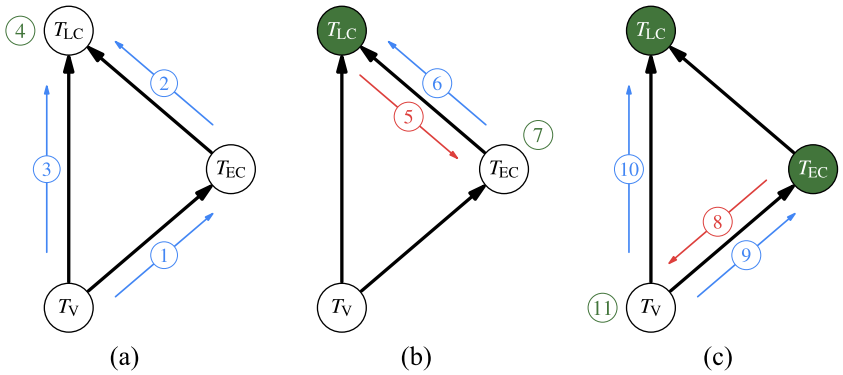

FIG. 3. Example of dependency resolution when a new Model is uploaded to the processing pipeline. Black arrows indicate upstream dependencies while blue and red arrows represent upstream and downstream resolution, respectively. (a) Upstream resolution begins from T_{V}–M and leads to T_{LC}–M being run. (b) Downstream resolution begins from T_{LC}–M and leads to T_{EC}–M being run. (c) Downstream resolution begins from T_{EC}–M and leads to T_{V}–M being run.

The lower the cost ζ^M the more accurate the model is overall. The weights in Eq. (2) are selected based on domain expertise and intuition regarding the relative importance of the properties for the target application. An area of active research in OpenKIM is to develop more rigorous methods for identifying properties of importance and associated weights for an arbitrary target application.[43]

In addition to accuracy, computational cost is also an important consideration when selecting a model. As a measure of the speed of a model, its average execution time over all N_{props} properties is computed. For model M, this is

$$\bar{t}^M := \frac{1}{N_{\text{props}}} \sum_{p=1}^{N_{\text{props}}} t_p^M, \tag{3}$$

where t_p^M is the execution time for computing property p with model M normalized by the whetstone benchmark (see Sec. II). By using normalized time, computations performed on Workers running on different architectures are considered on equal footing.

A model can be selected from a pool of available candidates by examining the results from Eqs. (2) and (3) on a cost vs time plot generated by the KIM Compare tool. A recent real-world example of usage of this tool was the selection of a copper (Cu) model for a large-scale molecular dynamics (MD) simulation of crystal plasticity at Lawrence Livermore National Laboratory (LLNL).[44–46] The objective was to find a model that was as inexpensive as possible in order to maximize the size of the simulation while still being sufficiently accurate for the material properties being studied. Crystal plasticity in fcc crystals is governed by dislocation nucleation and interaction. Key properties for obtaining correct behavior include the elastic constants that govern the long–range interaction between dislocations, the intrinsic stacking fault energy that governs the splitting distance in dissociated dislocation cores, and basic crystal properties including the equilibrium lattice constant and cohesive energy. In addition, it is important that the likelihood of dislocation nucleation relative to competing mechanisms such as deformation twinning or brittle fracture is captured. This is governed by the unstable stacking energy,[47] unstable twinning energy,[48] and surface energies of potential cleavage planes.

The cost vs computation time for 30 EAM and Finnis–Sinclair (FS) potentials archived in OpenKIM for many of the aforementioned properties is shown in Fig. 4 (see the supplementary material). These properties were calculated at zero temperature, although a better estimate of a given model's accuracy could be gained by examining the values of these properties at a temperature closer to that of the target application. Only EAM and FS potentials were considered since they are known to provide acceptable accuracy for fcc metals and are significantly less expensive than more accurate options. The differences in computation time between the models is related to details such as the employed cutoff radius, functional forms, and in the case of tabulated functions, the number of data points. Based on these results, model "P" by Mishin *et al.*[49–51] was selected by the LLNL researchers because it provided a good compromise in terms of relative speed and accuracy.

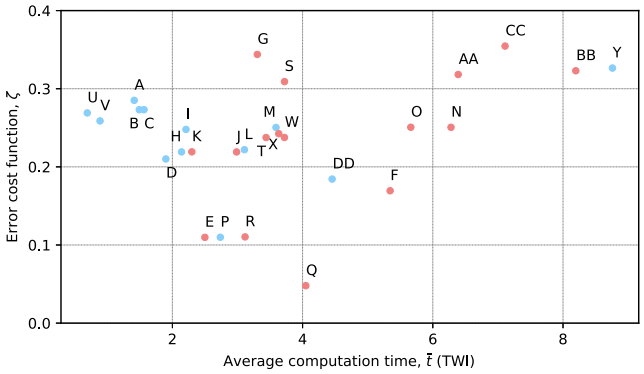

FIG. 4. Cost function ζ^M defined in Eq. (2) vs the average computation time in Eq. (3), given in units of Tera-Whetstone Instructions (TWI), for 30 EAM and FS potentials for Cu archived in OpenKIM and 13 material properties of fcc Cu. Monospecies models (blue) display similar accuracy to multispecies models (red), but are typically computationally less expensive. See the supplementary material for a definition of the model labels.

VII. CONCLUSIONS AND FUTURE WORK

The OpenKIM Pipeline is a distributed infrastructure to orchestrate the computation of compatible Models, Tests, and Verification Checks in the OpenKIM repository. This infrastructure is divided into different encapsulated components based on a clear separation of responsibilities. Each component is implemented as a Docker container, providing reproducibility of their environment and the tasks they perform, as well as portability across heterogeneous hardware. Moreover, common software packages and protocols are leveraged not only in the majority of the individual components but also in the networking that allows them to communicate with one another. Altogether, the design choices support the project-wide goals of provenance, flexibility, and ease of development. The results from the calculations performed by the pipeline are archived at openkim.org and are used by the KIM Compare tool to help users select models for applications of interest.

Further work is needed to implement a more sophisticated algorithm for job scheduling that excludes the possibility of jobs being rerun unnecessarily in the case of pathological dependency structures. Support must also be added for jobs that require HPC resources, including those external to the pipeline itself. This may entail a revision of the containerization approach so that a Docker image is created for each individual job.[52] It also brings forward the need for a job prioritization system, which might take into account profiling information for jobs previously run for each Test in order to predict the computational demand of future jobs. Vertical scaling of the individual components of the pipeline is becoming increasingly important to accommodate increased community uptake. In addition, growth in the size of the OpenKIM repository highlights the need for automated horizontal scaling based on work load. Finally, the development of intelligent tools for model comparison and selection that can assist users in this process remains a challenging and important area for continuing work.

SUPPLEMENTARY MATERIAL

A spreadsheet containing the full listing of the material properties, models, and reference data used to construct Fig. 4 is provided as the supplementary material. The weights used in computing the cost function ζ in Eq. (2) can be manipulated in the spreadsheet to see how this affects model selection.

ACKNOWLEDGMENTS

This research was partly supported by the National Science Foundation (NSF) under Grant Nos. DMR-1834251 and DMR-1834332. The authors acknowledge the Minnesota Supercomputing Institute (MSI) at the University of Minnesota for providing resources that contributed to the results reported in this paper. The authors thank Ronald Miller, Noam Bernstein, Mingjian Wen, and Yaser Afshar for helpful discussions and for contributing to this effort.

DATA AVAILABILITY

The data that support the findings of this study are available within the article and its supplementary material.

REFERENCES

[1]E. B. Tadmor, R. S. Elliott, J. P. Sethna, R. E. Miller, and C. A. Becker, JOM **63**, 17 (2011).
[2]E. B. Tadmor, R. S. Elliott, S. R. Phillpot, and S. B. Sinnott, COSSMS **17**, 298 (2013).
[3]R. S. Elliott and E. B. Tadmor, "Knowledgebase of interatomic models (KIM) application programming interface (API)" (OpenKIM, 2011) https://doi.org/10.25950/ff8f563a.
[4]See https://wiki.fysik.dtu.dk/asap for Asap—As Soon As Possible.
[5]A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, J. Phys.: Condens. Matter **29**, 273002 (2017).
[6]S. R. Bahn and K. W. Jacobsen, Comput. Sci. Eng. **4**, 56 (2002).
[7]I. T. Todorov, W. Smith, K. Trachenko, and M. T. Dove, J. Mater. Chem. **16**, 1911 (2006).
[8]J. D. Gale, J. Chem. Soc., Faraday Trans. **93**, 629 (1997).
[9]S. Plimpton, J. Comput. Phys. **117**, 1 (1995).
[10]A. P. Bartók *et al.*, "LibAtoms+QUIP: A software library for carrying out molecular dynamics simulations," http://www.libatoms.org/.
[11]N. C. Admal and E. B. Tadmor, J. Elasticity **100**, 63 (2010).
[12]N. C. Admal and E. B. Tadmor, J. Chem. Phys. **134**, 184106 (2011).
[13]P. Brommer and F. Gähler, Philos. Mag. **86**, 753 (2006).
[14]P. Brommer and F. Gähler, Modell. Simul. Mater. Sci. Eng. **15**, 295 (2007).
[15]P. Brommer, A. Kiselev, D. Schopf, P. Beck, J. Roth, and H.-R. Trebin, Modell. Simul. Mater. Sci. Eng. **23**, 074002 (2015).
[16]J. Janssen, S. Surendralal, Y. Lysogorskiy, M. Todorova, T. Hickel, R. Drautz, and J. Neugebauer, Comput. Mater. Sci. **163**, 24 (2019).
[17]E. B. Tadmor, M. Ortiz, and R. Phillips, Philos. Mag. A **73**, 1529 (1996).
[18]E. B. Tadmor, F. Legoll, W. K. Kim, L. M. Dupuy, and R. E. Miller, Appl. Mech. Rev. **65**, 010803 (2013).
[19]C. A. Becker, F. Tavazza, Z. T. Trautt, and R. A. Buarque de Macedo, Curr. Opin. Solid State Mater. Sci. **17**, 277 (2013), frontiers in Methods for Materials Simulations.
[20]L. M. Hale, Z. T. Trautt, and C. A. Becker, Modell. Simul. Mater. Sci. Eng. **26**, 055003 (2018).
[21]K. Choudhary, F. Y. P. Congo, T. Liang, C. Becker, R. G. Hennig, and F. Tavazza, Sci. Data **4**, 160125 (2017).
[22]K. Choudhary, A. J. Biacchi, S. Ghosh, L. Hale, A. R. H. Walker, and F. Tavazza, J. Phys.: Condens. Matter **30**, 395901 (2018).
[23]M. Wen, S. M. Whalen, R. S. Elliott, and E. B. Tadmor, Modell. Simul. Mater. Sci. Eng. **23**, 074008 (2015).
[24]D. S. Karls, https://github.com/openkim/kim-query.
[25]E. B. Tadmor, R. S. Elliott, and D. S. Karls, https://openkim/properties.
[26]H. J. Curnow and B. A. Wichmann, Comput. J. **19**, 43 (1976), https://academic.oup.com/comjnl/article-pdf/19/1/43/1057793/190043.pdf.
[27]M. S. Daw, S. M. Foiles, and M. I. Baskes, Mater. Sci. Rep. **9**, 251 (1993).
[28]M. Reddy, *API Design for C++*, 1st ed. (Morgan Kaufmann, Burlington, MA, 2011).
[29]D. Merkel, Linux J. **2014**, see https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.
[30]For HPC environments, Singularity images can be constructed from Docker images.
[31]See https://rsync.samba.org for more information on the "rsync" file synchronization utility.
[32]See https://www.tornadoweb.org for more information on the Tornado web server.
[33]See https://www.mongodb.com for more information on the MongoDB database application.
[34]See https://www.sqlite.org for more information on the SQLite database application.
[35]A. Solem *et al.*, "Celery distributed task queue," www.celeryproject.org.
[36]See https://www.rabbitmq.com for more information on the RabbitMQ message broker.
[37]Currently, RabbitMQ features native support only for AMQP version 0.9.1, employed here.
[38]There are only two parts of the process shown in Fig. 2 that the Web App is aware of: (1) that a new item has been submitted, at which point it notifies the Gateway's control API in step 1; (2) it periodically checks to see if new results or errors have been uploaded by the Gateway by scanning the contents of some of its directories. This loose coupling obviates the need to deal with synchronization between the Web App and the Gateway that would otherwise be necessary.
[39]Note that the KIM-property python package (https://github.com/openkim/kim-property) can be used to create and write property instances. A native implementation in LAMMPS is also available.
[40]Strictly speaking, what is listed are *lineages* of `Tests`, which encompass all versions of that `Test`. The dependency is always taken to correspond to the latest existing version in that lineage.
[41]This is applicable in the event where a new version of an existing `Test` is uploaded, which forces its downstream dependents to be rerun. The reason is that jobs associated with the downstream dependents being removed from the list could otherwise eventually be run twice when downstream resolution is performed on the `Test Results` of jobs associated with the others. However, this mechanism can fail if more complicated structures exist in the dependency graph. A point of future work is to address this shortcoming with a global graph traversal method, e.g. a topological sorting algorithm, while taking care not to needlessly sequentialize jobs in independent branches.
[42]See https://openkim.org/compare for to use the model comparison tool.
[43]D. S. Karls, "Transferability of empirical potentials and the knowledgebase of interatomic models," Ph.D. thesis, University of Minnesota, Minneapolis, MN, USA, 2016.
[44]V. V. Bulatov, private communication (2020).
[45]L. A. Zepeda-Ruiz, A. Stukowski, T. Oppelstrup, N. Bertin, N. R. Barton, R. Freitas, and V. V. Bulatov, "Metal hardening in atomistic detail," arXiv:1909.02030 [cond-mat.mtrl-sci] (2019).

[46]L. A. Zepeda-Ruiz, A. Stukowski, T. Oppelstrup, and V. V. Bulatov, Nature **550**, 492 (2017).
[47]J. R. Rice, G. E. Beltz, and Y. Sun, J. Mech. Phys. Solids **40**, 239 (1992).
[48]E. B. Tadmor and S. Hai, J. Mech. Phys. Solids **51**, 765 (2003).
[49]Y. Mishin, "EAM potential (LAMMPS cubic hermite tabulation) for Cu developed by Mishin, Mehl, and Papaconstantopoulos (2001) v005," OpenKIM, https://doi.org/10.25950/bbcadadf (2018).
[50]R. S. Elliott, "EAM model driver for tabulated potentials with cubic Hermite spline interpolation as used in LAMMPS v005," OpenKIM, https://doi.org/10.25950/bbcadadf (2018).
[51]Y. Mishin, M. J. Mehl, D. A. Papaconstantopoulos, A. F. Voter, and J. D. Kress, Phys. Rev. B **63**, 224106 (2001).
[52]K. M. D. Sweeney and D. Thain, in *Proceedings of the 9th Workshop on Scientific Cloud Computing, ScienceCloud'18* (Association for Computing Machinery, New York, NY, USA, 2018).