Rocketsled: a software library for optimizing high-throughput computational searches

JPhys Materials

OPEN ACCESS

RECEIVED
16 January 2019

REVISED
28 February 2019

ACCEPTED FOR PUBLICATION
4 March 2019

PUBLISHED
12 April 2019

PAPER

Rocketsled: a software library for optimizing high-throughput computational searches

Alexander Dunn[1,2,3], Julien Brenneck[2] and Anubhav Jain[2]

[1] Department of Materials Science and Engineering, University of California, Berkeley CA 94720, United States of America
[2] Lawrence Berkeley National Laboratory, Energy Technologies Area, 1 Cyclotron Road, Berkeley, CA, 94720, United States of America
[3] Author to whom any correspondence should be addressed.

E-mail: ardunn@lbl.gov

Keywords: machine learning, optimization, adaptive design, Bayesian optimization, superhard, high-throughput, active learning

Supplementary material for this article is available online

Abstract
A major goal of computation is to optimize an objective for which a forward calculation is possible, but no inverse solution exists. Examples include tuning parameters in a nuclear reactor design, optimizing structures in protein folding, or predicting an optimal materials composition for a functional application. In such instances, directing calculations in an optimal manner is important to obtaining the best possible solution within a fixed computational budget. Here, we introduce Rocketsled, an open-source Python-based software framework to help users optimize arbitrary objective functions. Rocketsled is built upon the existing FireWorks workflow software, which allows its computations to scale to supercomputing centers and for its objective functions to be complex, long-running, and error-prone workflows. Other unique features of Rocketsled include its ability to easily swap out the underlying optimizer, the ability to handle multiple competing objectives, the possibility to inject domain knowledge into the optimizer through feature engineering, incorporation of uncertainty estimates, and its parallelization scheme for running in high-throughput at massive scale. We demonstrate the generality of Rocketsled by applying it to optimize several common test functions (Branin-Hoo, Rosenbrock 2D, and Hartmann 6D). We highlight its potential impact through two example use cases for computational materials science. In a search for photocatalysts for hydrogen production among 18 928 perovskites previously calculated with density functional theory, the untuned Rocketsled Random Forest optimizer explores the search space with approximately 6–28 times fewer calculations than random search. In a search among 7394 materials for superhard candidates, Rocketsled requires approximately 61 times fewer calculations than random search to discover interesting candidates. Thus, Rocketsled provides a practical framework for establishing complex optimization schemes with minimal code infrastructure and enables the efficient exploration of otherwise prohibitively large search spaces.

1. Introduction

High throughput computing (HTC) has emerged as an indispensable framework for computational scientific inquiry, finding broad use from genotype–phenotype mapping in phenomics [1] to screening-based materials discovery [2–4]. HTC is defined by its ability to robustly run collections of independent computational tasks over distributed resources; modern highly-parallel computing resources make this a natural abstraction for many computational tasks. With cumulative run times of potentially millions of CPU hours, an instinctive question is to ask how HTC experiments can be optimized to reduce total computational time (or equivalently, how to maximize the utility of a given computational budget). Furthermore, even when one plans to comprehensively compute an entire search space, the use of optimization can help find a desired solution earlier in the process. This can be particularly useful if a comprehensive search potentially involves weeks or months of calculation. A large class of problems in HTC can be viewed fundamentally as optimization problems over a

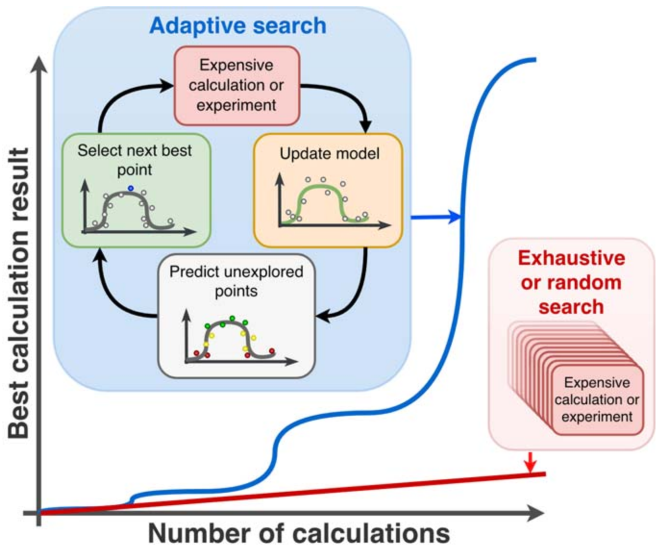

Adaptive search
Expensive calculation or experiment
Select next best point
Update model
Predict unexplored points
Exhaustive or random search
Expensive calculation or experiment
Best calculation result
Number of calculations

Figure 1. Motivation behind adaptive design. The adaptive strategy, also known as an optimization loop, uses a surrogate model (e.g. Bayesian or machine learning) trained on previous information to select the next calculation or experiment to run. This process is repeated in an iterative fashion by updating the model with new information. Adaptive search strategies can produce better results than other search strategies (e.g. random) with a given number of calculations, which makes them attractive for problems where exhaustive searches are impractical.

search space. Formally, this is solving

$$\underset{x \in D}{\operatorname{argmin}} f(x),$$

where f is the objective or loss function to be optimized (e.g. a simulation), and D is the domain of $\boldsymbol{x}$ to search. This is considered a global optimization problem, a type of black-box optimization that makes minimal assumptions about the search space or 'black-box' objective function, and is concerned only with global optima, as opposed to local optima. The goal of global optimization is finding global optima using as few function evaluations as possible. When considered sequentially, this type of problem is amenable to adaptive design, a search strategy that updates a predictive model as more information becomes available. Figure 1 illustrates this process: in adaptive design, each successive calculation is used to build a model that guides the next calculation (or set of calculations).

One of the most popular methods for global optimization is Bayesian optimization (BO) [5]. Some of the major advantages of BO are that it does not require derivatives or gradients, supports both discontinuous and discrete dimensions in parameter space, and allows for a potentially stochastic objective function. The most common form of BO uses a Gaussian process (GP) [6] model; alternate models include Random Forests (RF) [7] and Parzen Trees [5]. Many popular Bayesian optimization software implementations exist, including Bayesian optimization [8], MOE [9], Spearmint [10], SigOpt, Google Vizier [11], pySOT [12], Bayesim [13], and skopt [14]. There also exist software for global optimization that are targeted to a particular domain, such as the COMBO [15] and FUELS [16] (Citrination) software for materials science. Despite this growing trend, there exists to our knowledge no optimization framework that adequately integrates with an established workflow software. Such integration would make it possible to optimize objective functions that involve evaluating complex, error-prone, and computationally expensive workflows (requiring perhaps multiple long-running jobs with complex data handling for a single function evaluation) on large supercomputing centers and distributed clusters.

Here, we introduce Rocketsled v2019.2.27, an optimization framework designed to automatically deploy adaptive design workflows in high-throughput. The major goal of Rocketsled is to provide an extensible framework for using optimization in an HTC environment. Figure 2 illustrates the process by which new inputs are selected, distributed over nodes, and fed into an optimization engine for the selection of subsequent inputs. This cycle is repeated in a continuous feedback loop. Rocketsled is integrated with the computational workflow tool FireWorks [17], which makes it capable of handling millions of error-prone jobs on distributed computing resources with high parallelism. In particular, FireWorks has previously been used to run hundreds of millions of CPU-hours of calculations on a variety of international supercomputing centers [18–23] and represents an established, mature workflow platform geared to HTC workloads. This is a distinct advantage over existing software packages, where execution is intended to be local, not distributed. Rocketsled is robust to many kinds of optimization problems, including problems with multiple disparate objectives, discontinuous domains, and

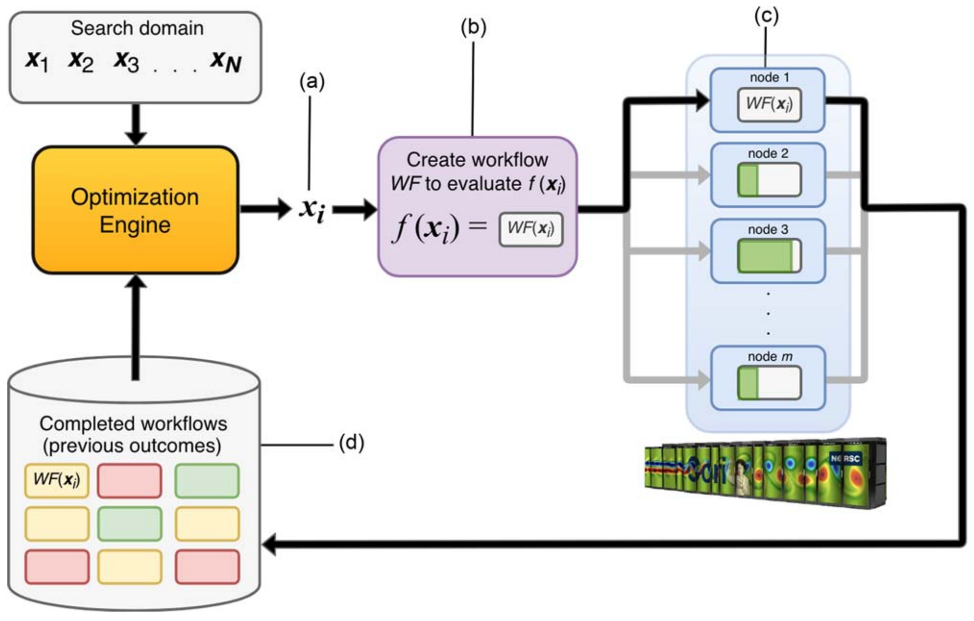


Figure 2. An optimization loop for high-throughput computational problems. (a) The optimization engine uses the results of previous outcomes to select a set of input parameters $\boldsymbol{x_i}$ from the domain of possible inputs. (b) The set of input parameters is converted into a corresponding workflow representing the function evaluation, $f(\boldsymbol{x_i})$. (c) The workflow is submitted to a computing center, where expensive workflows can finish asynchronously across nodes. (d) The results of the function evaluation are stored persistently to be used in the optimization engine's next prediction.

high-dimensional search spaces. Rocketsled also provides a method of including domain knowledge to improve optimization performance through a feature we call '$\boldsymbol{z}$-descriptors'. We note that the goal of Rocketsled is not to implement a new algorithm for optimization; instead, Rocketsled comes bundled with several built-in BO engines and is capable of using external optimization engines such as skopt, MOE, SigOpt, or a user-defined algorithm. Rocketsled is thus able to support many kinds of scientific workflows, hardware configurations, optimization engines, and distributed execution modes that are not possible with existing software.

In this paper we provide an overview of the design principles behind Rocketsled, highlighting software design aspects of the HTC framework, and brief mathematical details of the optimization. We then demonstrate performance of Rocketsled in optimizing generic mathematical test functions as well as example use cases in two high-throughput computing applications in inorganic materials discovery. Finally, we discuss the distinctions and limitations of Rocketsled. Rocketsled is released open-source under a BSD-style license and can be obtained at www.github.com/hackingmaterials/rocketsled; the current version at the time of this writing is v2019.2.27.

2. Methods

The goal of Rocketsled is to reduce the time scientists spend designing, writing, and debugging code for optimization loops and ultimately create a high-level and automatic optimization engine for HTC problems. In this section, we summarize the main methods and software Rocketsled uses, and we provide more comprehensive details in the supplementary information (available online at stacks.iop.org/JPMATER/2/034002/mmedia).

2.1. Mechanics of rocketsled execution: high-throughput computing with fireworks

Rocketsled is integrated with the open-source scientific workflow code FireWorks. This allows Rocketsled to be hardware-agnostic and workflow-agnostic and to separate the adaptive design component (i.e. building of surrogate model and selection of next calculation to run) from the management and execution of the workflows on various computing platforms. For example, through FireWorks, expensive parts of the workflows, such as physics simulations, can be executed on clusters or supercomputers, whereas less expensive parts, such as optimizations, can be executed locally or on clusters, with all workflows managed through a central database server (MongoDB [24]). Because the design of Rocketsled integrates closely with that of FireWorks, we briefly describe FireWorks' core design below. An in-depth description of FireWorks can be found in the original publication [17] and online documentation (https://materialsproject.github.io/fireworks/).

A FireWorks workflow is composed of one or more jobs (called 'Fireworks') that may have a complex set of dependencies and data passing between them (figure 3(a)). Each Firework is run on a single computing platform,

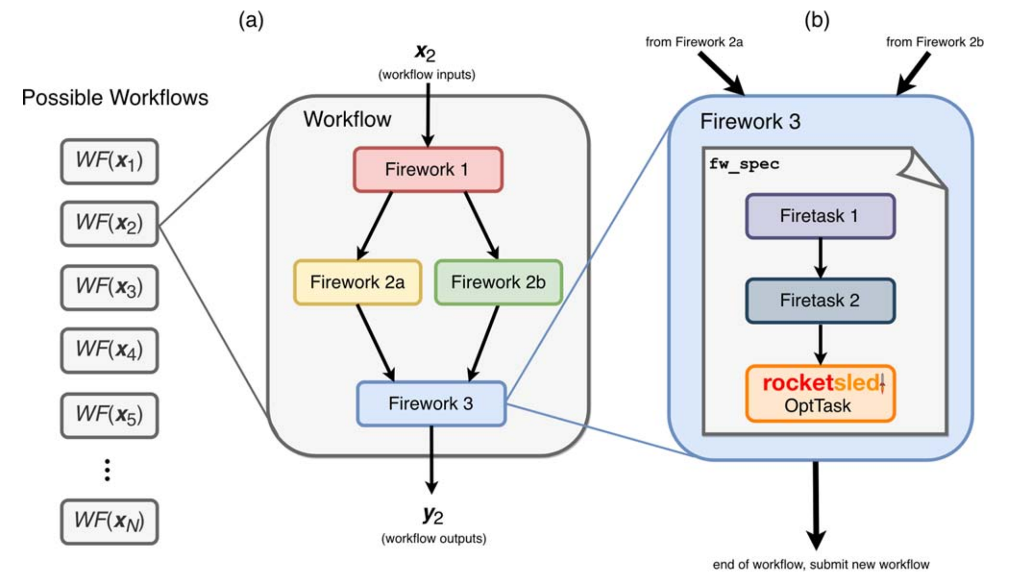


Figure 3. A schematic of FireWorks' primary objects (Workflows, Fireworks/jobs, and Firetasks) and Rocketsled's optimization Firetask (OptTask). (a) A search space of many workflows, where each workflow is determined by its input parameters, and an individual workflow in directed acyclic graph (DAG) form. Workflows can have many constituent jobs (Fireworks), such as a physics simulation, and each job can be executed conditionally or on different computing resources. Running a workflow is analogous to evaluating an objective function. This particular workflow takes in an input vector $\boldsymbol{x}_2$, runs four Fireworks, and outputs an objective function response $\boldsymbol{y}_2$. (b) The final Firework job in the workflow from (a), containing OptTask, the Rocketsled optimization object, as the final Firetask. OptTask reads $\boldsymbol{x}_2$ (the input parameters in this workflow) and $\boldsymbol{y}_2$ (and output of the function evaluation) from the workflow using the Firework's 'spec' (a set of shared variables common to all the tasks in a Firework job) and submits the optimal next workflow (e.g. $\boldsymbol{x}_5$) for execution.

but different Fireworks within the same workflow can run in parallel on different platforms if the workflow graph and the user specification allow. Each Firework is in turn composed of one or multiple Firetasks, which are Python functions or runnable scripts that are executed sequentially (figure 3(b)). A workflow can thus be as simple as a single Firetask within a single Firework (e.g. a single script to run), or as complex as hundreds or thousands of individual operations that pass data between them. The FireWorks software allows users to define a set of workflows, store them in a central database (LaunchPad), and distribute execution over multiple computing centers. The database manages the state of all the workflows and allows users to inspect progress, re-run failed jobs, and perform other tasks.

Rocketsled is implemented as a Firetask called *OptTask*; when designing a workflow, the user will typically place the OptTask at the very end of the workflow representing the function evaluation (figure 3(b)). The operation of OptTask consists of 3 stages (figure 4). In the first stage (figure 4(a)), the OptTask will read in the data generated by the current workflow—namely, the set of input parameters that were chosen for this evaluation (the vector $\boldsymbol{x}$) and the value of output response (the vector $\boldsymbol{y}$, with a single value for conventional optimization and multiple values for multi-objective optimization) for this particular input choice. It will also generate any domain-specific descriptors describing the input space that can be used to help the surrogate model (the vector $\boldsymbol{z}$), as will be discussed later. The OptTask next stores this information in a MongoDB database and retrieves all previously generated inputs, outputs, and descriptors (the matrices $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, respectively) to assemble a full record of information known thus far. In the second stage (figure 4(b)), the OptTask will build a surrogate model that can be used to predict the values and uncertainties of all remaining inputs in the domain. Rocketsled provides many different algorithms for generating this surrogate model. In the third and final stage (figure 4(c)), OptTask will evaluate the remaining points in the space (accounting for uncertainties) to determine the best of the remaining inputs to run via one of several acquisition functions (described later), and automatically submit a new workflow for this best input. We emphasize that, once set up, no user intervention is needed to maintain the system. Rocketsled will continue submitting calculations automatically, improving the accuracy of the surrogate model with every new calculation, until directed to stop (e.g. through stopping the FireWorks 'rocket launches') or until the search space has been exhausted.

2.2. Optimization backend

Rocketsled operates in a manner that is agnostic to the choice of underlying optimization engine. The requirements for an optimizer are that, given the dimensions of the search space and a set of previously evaluated inputs $\boldsymbol{X}$ with the corresponding $f(\boldsymbol{x})$ responses $\boldsymbol{Y}$, it returns at least one $\boldsymbol{x}$ in the unexplored space with which

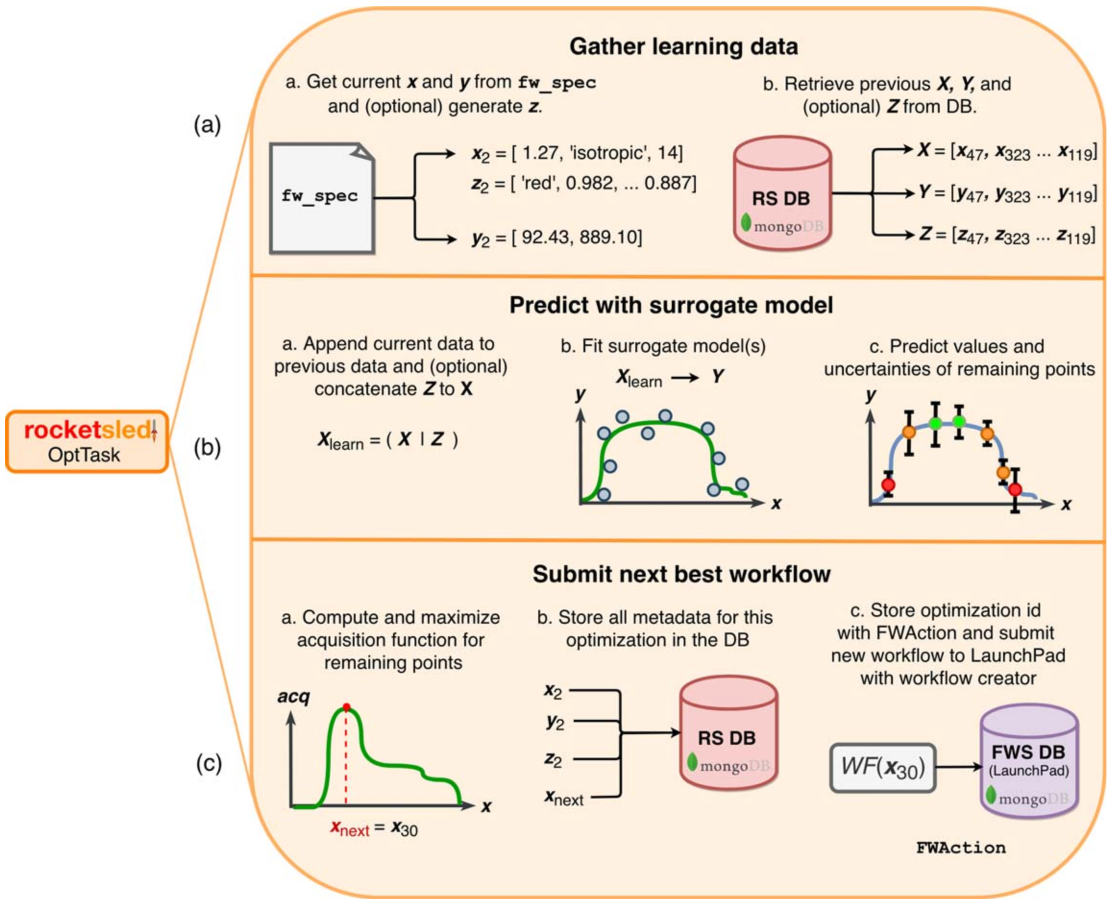

Gather learning data
a. Get current x and y from fw_spec and (optional) generate z.
b. Retrieve previous X, Y, and (optional) Z from DB.
(a)
fw_spec
x2 = [1.27, 'isotropic', 14]
z2 = ['red', 0.982, ... 0.887]
y2 = [92.43, 889.10]
RS DB
mongoDB
X = [x47, x323 ... x119]
Y = [y47, y323 ... y119]
Z = [z47, z323 ... z119]
Predict with surrogate model
a. Append current data to previous data and (optional) concatenate Z to X
Xlearn = (X | Z)
b. Fit surrogate model(s)
Xlearn → Y
c. Predict values and uncertainties of remaining points
rocketsled OptTask
(b)
Submit next best workflow
a. Compute and maximize acquisition function for remaining points
acq
xnext = x30
b. Store all metadata for this optimization in the DB
x2
y2
z2
xnext
RS DB
c. Store optimization id with FWAction and submit new workflow to LaunchPad with workflow creator
WF(x30)
FWS DB (LaunchPad)
FWAction
(c)

Figure 4. A summary of the primary responsibilities of OptTask, Rocketsled's optimization task. (a) OptTask gathers all previous input and output data from the central database (i.e. from the execution of other workflows) and extracts the current input and objective response from the workflow. (b) OptTask uses surrogate models trained on explored (evaluated) points to predict the values and uncertainties of unexplored points. (c) OptTask determines the next best workflow to run by first computing the $\boldsymbol{x}$ vector most likely to have a favorable (according to a chosen acquisition function) result, and then submitting the corresponding workflow to the central workflow database, the FireWorks LaunchPad.

to start the next workflow. Rocketsled supports the ability to use one of several built-in Bayesian optimizers or to integrate external custom optimizers written as Python functions. The built-in optimizers are selected with arguments to OptTask and are based on scikit-learn [25] regressors, a full list of which can be found in the supplement. These optimizers possess the full scope of Rocketsled's abilities including tolerance-based duplicate prevention and selection of acquisition function. Custom optimizers retain the core Rocketsled features, including FireWorks execution, separate optimization management, use of surrogate functions, and multi-objective mode.

The built-in Rocketsled optimizers operate by fitting regression models to the available data, predicting $f(\boldsymbol{x})$ for points in the unexplored space, generating uncertainty estimates for each point, and selecting the next best $\boldsymbol{x}$ to run. The uncertainty estimates are either taken directly from the model (if the model itself provides such estimates) or bootstrapped [26] if no uncertainty is directly provided by the model. Once uncertainty estimates have been generated, Rocketsled selects the next best $\boldsymbol{x}$ by maximizing the chosen acquisition function. Acquisition functions are statistical methods for selecting the next point to run given limited knowledge about the objective function. Rocketsled comes with five acquisition functions, the most well-known being Expected Improvement (EI) [27, 28]. EI evaluates $f(\boldsymbol{x})$ at points with both a high probability of improving and large margin of improvement, and is mathematically defined as [29]:

$$\mathrm{EI}(x) = \begin{cases} (\mu(x) - f(x_{\min}) - \chi)\cdot\Phi(Z) + \sigma(x)\phi(Z), & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases},$$

$$Z = \begin{cases} \dfrac{\mu(x) - f(x_{\min}) - \chi}{\sigma(x)}, & \sigma(x) > 0 \\ 0, & \sigma(x) = 0 \end{cases},$$

where $\mu(x)$ and $\sigma(x)$ are the mean and variance of the $f(x)$ prediction, Φ and ϕ are the cumulative distribution function and probability distribution function respectively, $\boldsymbol{x}_{\mathbf{min}}$ is the vector for which $f(\boldsymbol{x})$ is minimized, and χ is a user-tunable parameter controlling exploration versus exploitation. High values of χ correspond to highly exploratory strategies; it has been shown that $\chi = 0.01$ works well for a diverse range of problem sets [28]. The other acquisition functions available for single objectives are Probability of Improvement [30], Lower Confidence Bound [31], and greedy selection (a strategy in which the top prediction is always picked and uncertainty is not taken into account).

Rocketsled's optimization backend also supports multi-objective optimization. In multi-objective optimization, we seek the largest set of Pareto-optimal solutions, particularly those solutions with the smallest hypervolume (i.e. minimization in each objective). A Pareto-optimal solution is a point which is not dominated in all objectives by any other point; the Pareto frontier is the set of all Pareto-optimal solutions, a succinct formal definition of which is presented in Cui *et al* [32]. At the time of this writing, Rocketsled has two acquisition functions for selecting the next best guess when there are multiple competing objectives. The first is a greedy strategy that selects the guess with the highest probability of being Pareto-optimal. The second strategy is the Expected Maximin Improvement Scheme [33, 34] which seeks to maximize the *minimum* EI gain among objectives. In order to generate predictions and uncertainties in a similar manner to single-objective mode, Rocketsled repeats the single-objective procedure across all objectives, and thus for prediction purposes, considers objectives independent of one another.

2.3. Incorporating domain knowledge with *z*-descriptors

Optimization problems may benefit from the injection of domain knowledge. One way to represent domain knowledge is to allow the optimizer to incorporate features or descriptors, i.e. computationally inexpensive functions incorporating domain knowledge which is not directly found in the $\boldsymbol{x}$ vector. To state it another way, rather than fitting a surrogate model directly as a function of the $\boldsymbol{x}$ vector, one can introduce additional attributes to simplify the model. These functions are of the form $\boldsymbol{z} = g(\boldsymbol{x})$, where $\boldsymbol{z}$ is a vector representing the information encoded by $\boldsymbol{x}$ and g is computationally cheap to evaluate. While the $\boldsymbol{x}$ vector must be unique, $\boldsymbol{z}$ has no such requirement. For example, if one is optimizing molecules for stability, $\boldsymbol{x}$ may define the molecule uniquely by its atomic positions or SMILES [35] string, but $\boldsymbol{z}$ provides chemically-relevant data based on $\boldsymbol{x}$ such as bond lengths, angles, and functional group orientations which allow the optimizer to better predict the molecule's stability. Such use of feature extraction techniques is a very common practice for improving conventional machine learning results but is typically not supported by optimization packages, which operate directly on the $\boldsymbol{x}$ vector alone. We present specific examples making use of this technique in a later section and demonstrate that the use of domain-specific $\boldsymbol{z}$ features can have a large impact on overall results.

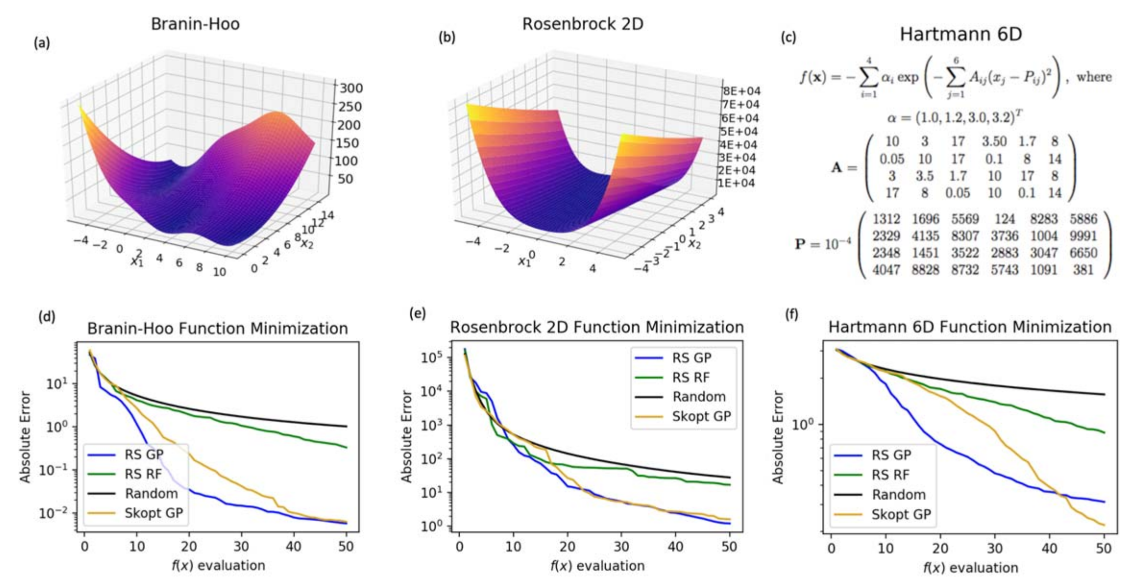
3. Results

We showcase the use of Rocketsled in two major areas. First, we benchmark Rocketsled on a set of well-known test functions. Second, we demonstrate how Rocketsled can lead to more efficient searches for functional materials for photocatalytic water splitting and for superhard materials. The objective of these case studies is to demonstrate how the built-in optimizers and capabilities for managing high-throughput workflows with Rocketsled may be useful. We also provide results regarding the computational expense of the optimizer itself.

3.1. Optimization of common test functions

The primary concern of a black-box optimization algorithm is its ability to accurately forecast objective function values with limited information. To estimate its usefulness in real-world optimization problems, Rocketsled's optimization engine was evaluated on three common single-objective test functions [36–38] with known global minima. Two optimizers based on machine learning models built into Rocketsled's framework, GP and RF, were independently tested and compared to a similar, popular open-source software, Scikit-Optimize [14] (skopt) under identical constraints. For each benchmark function, the algorithm was allowed fifty function evaluations to predict the function minimum. The absolute error in the optimizer's predictions are plotted in figure 5.

The objective of these tests was to affirm suitable performance of Rocketsled's built-in optimization engine using default (untuned) hyperparameters across several objective functions. We emphasize that Rocketsled supports any custom optimizer, and its main purpose is not to achieve the best optimization performance on test functions but rather to better support adaptive design in HTC workloads. By the 50th function evaluation, both Rocketsled algorithms and the Skopt GP perform better than random search on average on the three benchmark functions. The GP-based optimization methods typically perform near an order of magnitude better than the RF optimization on these tests.



$$f(\mathbf{x}) = -\sum_{i=1}^{4}\alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right), \text{ where}$$

$$\alpha = (1.0, 1.2, 3.0, 3.2)^T$$

$$\mathbf{A} = \begin{pmatrix} 10 & 3 & 17 & 3.50 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

$$\mathbf{P} = 10^{-4}\begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$$

Figure 5. (a)–(b) Plots of the Branin-Hoo and Rosenbrock 2D test functions. (c) Closed form of the Hartmann 6D function. (d)–(f) Absolute error of the Rocketsled Random Forest (RS RF), Rocketsled Gaussian Process (RS GP), and Skopt Gaussian Process (Skopt GP) optimization algorithms on each function compared with random guess and averaged over 100 independent runs (100 000 runs for random guess). Closed forms and optima of all benchmarking functions are provided in the supplement.

3.2. Application to the materials science domain: photocatalysis and superhard materials

Although the Rocketsled framework is completely agnostic of application, we believe a major use case will be in the field of materials science (in-line with the user base of the underlying FireWorks software). The growing toolkit for running high-throughput calculations [39–42], the potentially immense search space (e.g. often more than 10^{10} viable compounds [43]), and the growing number of studies using adaptive design in the materials domain to guide both simulations [34, 44–59] as well as experiments [44, 60–63] makes computational materials design an exciting field to explore with Rocketsled. In the two following case studies, we demonstrate the applicability of Rocketsled to adaptive design for materials discovery.

We first examine the application of Rocketsled to the discovery of one-photon water-splitting perovskites, a photocatalytic technology useful for hydrogen production. A set of 18 928 perovskite compositions were previously calculated by Castelli *et al* [64] in a comprehensive materials screening procedure, where the chemical search space was exhaustively explored with density functional theory (DFT) calculations. Among the candidates, only 20 were identified as candidates for further study, 13 of which were previously unknown by the research community. In the following use case, we compare the performance of the RF optimizer in finding all 20 photocatalytic candidates to random search, a set of chemical rules, and a tuned genetic algorithm (GA) previously reported by Jain *et al* [65] that used *a priori* knowledge of results to determine an upper bound on GA performance. This GA's hyperparameters such as elitism, selection function, population size, and crossover and fitness functions were tuned by searching 2952 parameter combinations; the 'best GA' was optimized via *a posteriori* analysis (such as one- and two-parameter ANOVA), where the hyperparameters were changed based on the results of previous GA runs. Such hyperparameter tuning is not optimal for adaptive design, since optimal hyperparameters cannot be determined without prior knowledge of the objective function response.

Every compound in the search space ($N = 18\,928$) is encoded by its chemical formula *ABX*, where *A* and *B* are elements and *X* is a ternary anion. The 3-tuple [*A B X*] is the $\boldsymbol{x}$ vector, with the Mendeleev number of the element for *A* and *B* and the average Mendeleev number for *X*. The fitness function (taken from Jain *et al* [65]) response is the objective function, ranging from 0 (worst possible performance) to 30 (one of the 20 candidates) and assessing band gap, stability, and band edge position. The results are precomputed, so to evaluate the objective function we merely need to look up an entry's data. In a live optimization run, evaluating the objective function would consist of running a DFT workflow. Calculation and fitness function details can be found in the supplementary information.

We test seven optimization strategies to search the perovskite space. The first strategy is random search. The second strategy is to use a set of 'chemical rules' regarding charge balance, even–odd valence rule, and use of Goldschmidt tolerance factor [66] in ranking candidates. These are designed as a crude way to mimic selections from a human expert in the field [65]. The third strategy is a RF-based BO with Rocketsled, using only the 3-tuple $\boldsymbol{x}$ vector for learning and prediction. In a fourth strategy, we again use Rocketsled but add to each $\boldsymbol{x}$ vector a $\boldsymbol{z}$ vector; these are an additional 132 features that represent composition-based chemical information such as statistics on electronegativity, oxidation state, and common ionic radii [67, 68]. The set of features used for $\boldsymbol{z}$ were previously published as the MagPie descriptor set [68] independent of this problem, therefore we can consider them both generally representative of the chemical information of each composition yet 'blind' to the specifics of the problem. The fifth strategy is the same as the fourth (Rocketsled Random Forest + $\boldsymbol{z}$), and we further exclude the 11 587 compounds failing two simple chemical stability rules from the search. The sixth and seventh strategies are the GA previously reported by Jain *et al* [65], with either no chemical rules applied (strategy 6) or using chemical rules to exclude the 11 587 compounds failing stability rules (strategy 7). We note that both the GA results from the prior study optimized the hyperparameters of the GA using the problem itself, thus these should be considered as upper bounds on performance as those hyperparameters would likely not have been known in advance. In summary, the seven strategies used are (1) random search, (2) chemical rules, (3) Rocketsled RF with $\boldsymbol{x}$ only (RS RF), (4) RS RF + $\boldsymbol{z}$ features, (5) RS RF + $\boldsymbol{z}$ features + chemical rules, (6) GA (upper bound), and (7) genetic algorithm + chemical rules (upper bound).

We evaluate the performance of an optimization strategy by its 'speedup':

$$S_n = \frac{N_{ran}^n}{N_{opt}^n},$$

where S_n is the speedup in finding n solutions, N_{opt}^n is the number of objective function evaluations (DFT calculations) necessary for the optimizer to find n solutions, and N_{ran}^n is the number of evaluations necessary for a random search to find n solutions. A random search has a speedup of 1 by definition while an optimization strategy finding candidates in fewer expensive calculations than random search has speedup larger than 1. For example, a search strategy finding 10 solutions in half the calculations of random search has $S_{10} = 2.0$. N_{ran}^n is defined as

$$N_{ran}^n = \frac{n(x + 1)}{s + 1},$$

where x is the size of the search space (18 928), and s is the total number of solutions (20) [65]. For repeated, non-deterministic runs, the mean and variance of speedup can be calculated by first calculating the speedup of each individual run, and then computing statistics from this set.

The mean speedups (from 20 independent runs) of the four RF experiments to 10 and 20 solutions are depicted in figure 6 along with results from chemical-rules-only. For all experiments, the S_{20} (speedup of finding all 20 solutions in the search space) of all Rocketsled-based searches exceeded that of random search by a factor of 5.84 and chemical-rules-only by a factor of 1.30. The RS default RF has performance similar to the best GA reported previously across 10- and 20-solution speedup, even without any tuning to the problem.

For all Rocketsled-guided search strategies, adding additional information via the cheap surrogate $\boldsymbol{z}$ model increased optimization performance. Adding $\boldsymbol{z}$ features alone gave a 76.4% increase in S_{20} and a 162.0% increase in S_{10}. When chemical rules were further applied to restrict the search space, the resulting models were by far the top-performing models of those tested, with S_{20} reaching 15.99 and S_{10} reaching 28.26. A more detailed treatment of $\boldsymbol{z}$ features, and a separate study with a manually selected set of statistics for the $\boldsymbol{z}$ features, is presented in the supplementary information.

Figure 6 also plots the values previously reported by Jain *et al* for GAs [65] on the same problem. We note that the GA value reported is the best among more than 2500 hyperparameter combinations tested. This hyperparameter search, which separately studied various crossover rates, mutation rates, and population sizes on GA performance, required prior knowledge of all results so that the best combination could be selected. When these parameters were not tuned using knowledge of the results, the reported speedups in the prior study were typically about half that of the best GA [65], making them significantly lower than that of untuned Rocketsled results. Nevertheless, the Rocketsled results compare favorably to or exceed even the fully tuned GA results even without any prior knowledge of the problem, indicating the robustness of the approach.

In figure 7, we plot the performance of the RF optimizers in terms of candidates found as a function of function evaluations. Steeper lines indicate higher speedup. The RF optimizer, when provided neither $\boldsymbol{z}$ nor chemical heuristics, is generally poorer than the chemical-rules-only search, but has a standard deviation overlapping the S_{20} of chemical-rules-only. The localized drop in performance around 11 candidates can be explained by the clustered distribution of candidates in Mendeleev space (see supplementary information), given that the only information this strategy has is the 3-vector of Mendeleev numbers representing A, B, and X. All chemical knowledge-restricted or $\boldsymbol{z}$-directed optimizations with Rocketsled exceed the performance of the chemical heuristics alone in S_{20}. For these strategies, the variance observed in S_{20} over the 20 independent runs is sufficiently small such that the mean is not within one standard deviation of the chemical-rules only strategy, indicating significant advantage.

All strategies vastly outperform random search. For example, with only the encoded Mendeleev numbers for perovskite representation, Rocketsled optimization finds all candidates with less than one-fifth of the DFT

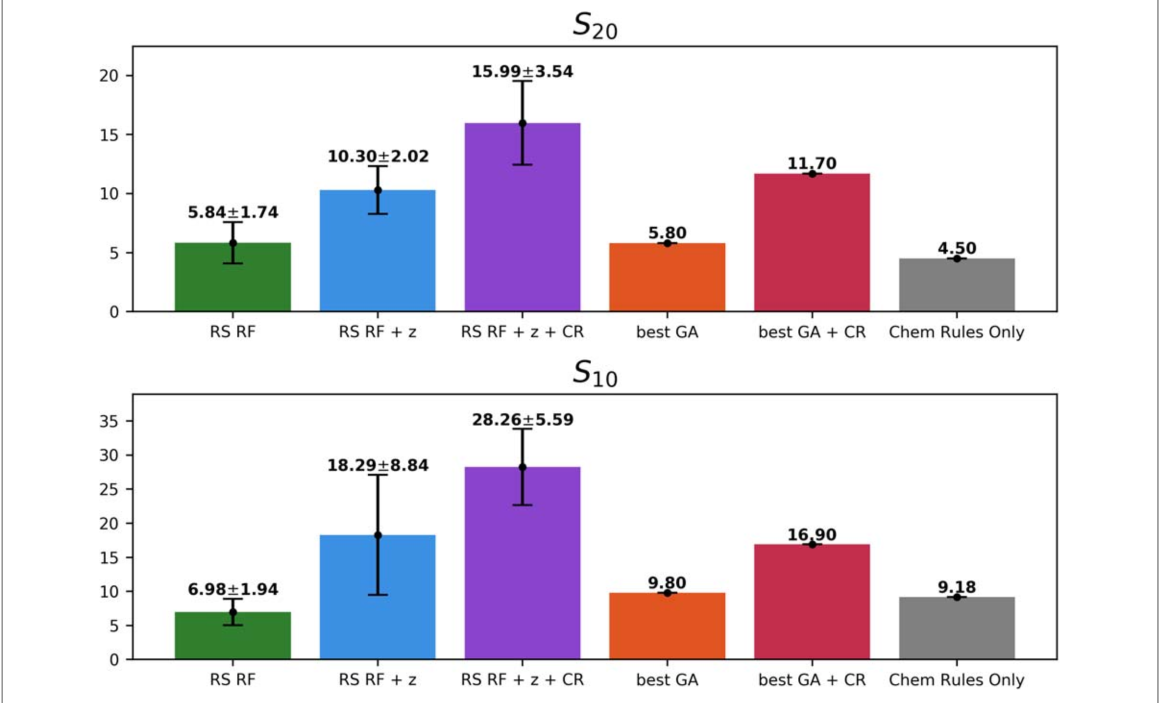


Figure 6. Comparison of different optimization algorithms for 10-solution (S_{10}) and 20-solution (S_{20}) speedups. Rocketsled Random Forest (RS RF) is shown both without the additional z-descriptors and with chemical rules (CR); all Rocketsled results are without any specific tuning to the problem. Results from a previous study [65] reporting the best genetic algorithm (GA) results after *a posteriori* tuning across over 2500 hyperparameter combinations are also shown with and without chemical rules. Note that random search has a speedup of 1.0 by definition. The best strategies involve use of Rocketsled with one or more forms of domain knowledge applied.

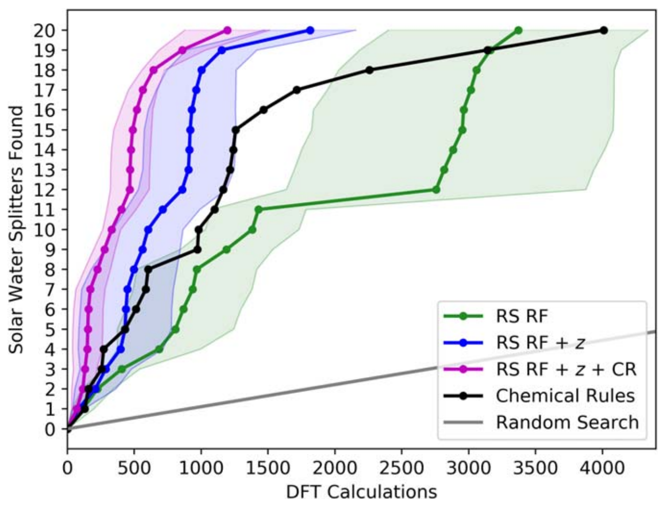


Figure 7. Calculations needed to produce solutions to the solar water splitting problem for four different Rocketsled Random Forest strategies over 20 independent runs (for details of each strategy, see the main text). The shaded areas are error bars representing one standard deviation from the mean performance. The RF incorporating expert knowledge (purple) from chemical rules (CR) outperforms those which do not incorporate expert knowledge (blue, green). The RFs incorporating extra z features (purple, blue) outperform equivalent RFs without z features (green).

calculations random searching requires. Using z information or chemical information independently with the optimization, we can reduce this number to less than 10%. If the optimizer can use both z information and chemical heuristics, we use only 6.25% of the calculations, and save 16 901 DFT calculations on average. If we are interested in simply finding half the candidates while incorporating chemical knowledge and z information, we require only 3.4% of the calculations random search requires. Thus, very significant reductions in the computational cost needed to perform materials searches are possible using the Rocketsled framework.

We next demonstrate the examine the application of Rocketsled to a multi-objective problem: searching for eight known superhard materials among a set of 7394 heterogeneous, k-nary structures from the Materials

Table 1. List of superhard candidate materials as eight solutions to a multi-objective optimization problem.

MP id string	Formula	Space group	K (GPa)	G (GPa)	Common name (if available) and prior studies of superhardness
mp-47	C	194	435.661	522.922	Londsdaleite [comp 72]
mp-66	C	227	435.686	520.267	Diamond [exp 73]
mp-1985	C_3N_4	176	408.925	312.428	ß-C_3N_4 [comp 74]
mp-1019055	ReN_2	127	379.804	253.458	Rhenium Nitride[a]
mp-1894	WC	187	385.194	278.960	Tungsten carbide [exp 73]
mp-49	Os	194	401.328	258.697	Osmium [exp 75]
mp-2653	BN	186	373.241	383.285	*w*-BN [comp 72, exp 76]
mp-1018649	BC_5	156	378.000	347.000	Diamondlike-Boron Carbide [exp 77]

Exp.—Experimentally observed as superhard by hardness testing.
Comp.—Theorized as superhard by computational studies.
[a] A computational study [78] has proposed similar Re_xN_y stoichiometries as superhard materials.

Project. Discovering new superhard materials is imperative to reducing the cost of industrial cutting and polishing tools. The hardness indicator of a polycrystalline inorganic solid is typically estimated computationally by first determining its full elastic tensor with DFT, a relatively expensive *ab initio* procedure, and then calculating the Voight–Reuss–Hill [69] average of the bulk modulus (K) and shear modulus (G). Materials are canonically classified as superhard if their Vicker's indentation hardness, a metric related to having both high K and high G, exceeds 40 GPa [70]. In the same manner as an investigation by Tehrani *et al* [71], we will consider a material 'superhard' if it is both incompressible (high K) and resistant to shear (high G).

Our dataset, which includes data from a previous high-throughput elastic tensor study [20], was downloaded from the Materials Project. Structures having negative K or G and compositions containing noble gases were removed as were 2D materials. The resulting 7394 unique structures are composed of 56% ternary, 39% binary, 2.5% unary, 2.1% quaternary, 0.2% quinary, and ~0.02% senary stoichiometries. Among these structures, 167 spacegroups and 6238 unique compositions are represented. Spacegroup 225 (Fm-3m) is the mode spacegroup of the set and composes 15% of all the structures; 22 of the spacegroups appear in the dataset with a frequency higher than 1%.

The objective is to identify the crystal structures with both high K and high G without the optimization algorithm having problem-specific or *a priori* knowledge, and using as few simulated elastic computations as possible. The search can be considered a multi-objective optimization: for a structure to be considered a candidate, it must have both $K > 350$ GPa and $G > 250$ GPa. To reduce the number of polymorphs in the final set of superhard candidates, we will only consider the top performing (Pareto-optimal with respect to K and G) structures for each composition as solutions to the optimization. For example, while there are seven carbon polymorphs matching the K and G criteria, there are only two which are Pareto-optimal. The final list of solutions, shown in table 1, is composed of eight superhard candidates:

Previous studies by de Jong *et al* [79] and Tehrani *et al* [71] applied statistical learning techniques to smaller —yet similarly diverse—elastic datasets, using sets of compositional and structural descriptors. In similar fashion, we generate basic statistics from the composition and structure of each material which we use as the optimization algorithm's $\boldsymbol{z}$ vector. The 132 composition features were generated using the matminer implementation[67] of Ward *et al*'s MagPie descriptor set [68], which includes means, average deviations, modes, and ranges of elemental properties such as melting temperature, atomic weight, and covalent radii. The remaining 52 structural descriptors were the space group number, the structure's centrosymmetry, and statistics (taken over sites) based on local order parameters [80] implemented in matminer [67]. Rocketsled's multi-objective mode was used with the RF optimizer and the Maximin EI acquisition function. No encoding of the materials was used besides the $\boldsymbol{z}$ vector.

In figure 8 we plot the progress of the Rocketsled optimizer in finding all candidates in fewer computations than random search for twenty independent runs. On average, we can find all the materials in table 1 using 261 elastic calculations instead of 6573 calculations with random search. We find the peak speedup of 61.1 ± 33.1 for 5 candidates, and the lowest speedup of 28.3 ± 9.2 for all 8 candidates. The higher speedups with $n < 6$ may be explained by the presence of carbon in five of the eight materials; the search algorithm can preferably look for carbon-containing compounds once it has identified them. More dissimilar materials, such as ReN_2, are found with lower speedup if not predicted as high-scoring candidates early in the optimization run. Figure 9 provides visual snapshot of which candidates in the search space are likely to be explored with both random and Rocketsled strategies at a state of progress of only 200 calculations. This plot demonstrates that Rocketsled is exploring the known 8 solutions with very high probability even at this early stage.

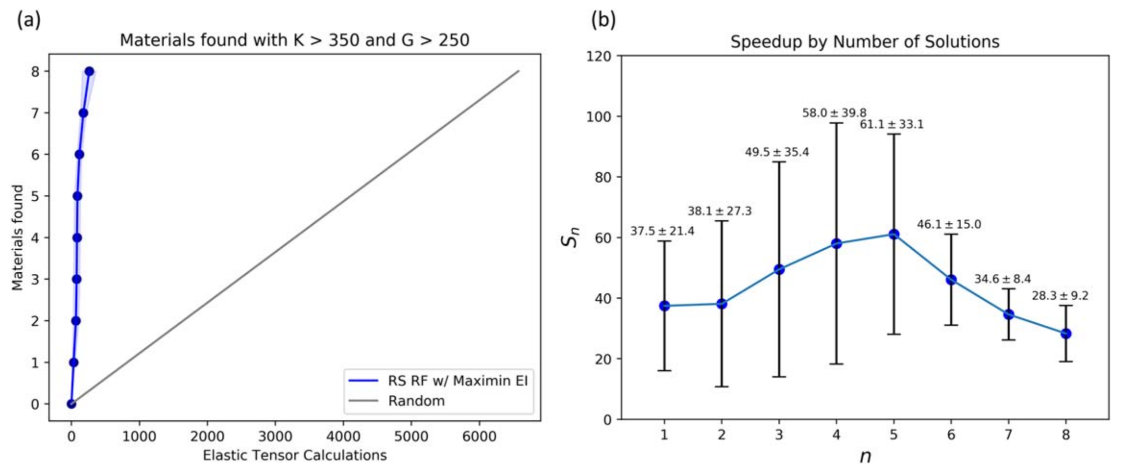


Figure 8. (a) Rocketsled optimization strategy applied to finding the eight high *K* high and *G* materials (potentially superhard) among 7394 possible structures. The number of candidates found (averaged over 20 independent optimizations) is plotted as a function of number of function evaluations. The thin shaded area represents one standard deviation from the mean. (b) The speedup for each number of candidates, 1–8. Error bars represent one standard deviation.

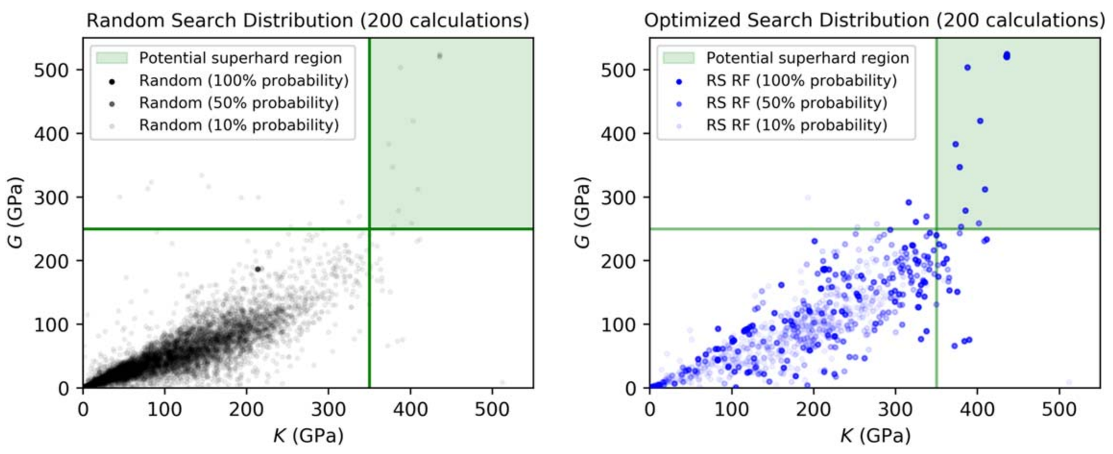


Figure 9. Distributions of search strategies for 200 allowed function evaluations (DFT calculations) searching for superhard (high *K*, high *G*) materials. The optimized (untuned Rocketsled Random Forest) searches are shown as a distribution over 20 independent runs; more opaque shades of blue indicate a higher probability of being evaluated. Random search (black) has the same probability of being evaluated for each material; note that on average, it takes random search $N_{ran}^{1} = 821$ calculations to find one superhard candidate, whereas the Rocketsled RF search will find on average all 8 candidates within 261 calculations.

3.3. Performance and computational overhead

A major consideration when incorporating a black-box optimization engine is the computational overhead required for training and updating the surrogate model. Here, we show the overhead of the Rocketsled and FireWorks framework as a whole for local execution on a 2017 Macbook Pro with 3.5 GHz Intel i7 processor. We use the EI acquisition function that requires uncertainty estimation. The Rocketsled GP, RF, and random predictors overhead times are shown independently. Due to the small, constant time required for selecting a random guess, random predictor times essentially represent the non-machine learning portions of the optimization procedure including communication with the database, manipulation of the training and prediction matrices in memory, and FireWorks operations. At each function evaluation, the models are retrained using all previous calculations available for training and 1,000 points in the search space are predicted. For the purpose of this overhead benchmark, a mock objective function was used which accepts any vector and returns a random scalar between 0 and 1.

As plotted in figure 10(a), the RF model has a much higher training time for small numbers of function evaluations. This is because the acquisition functions such as EI require estimates of prediction uncertainty as input. With a Gaussian Process model, uncertainty estimates can be directly taken from the trained model; however, with a RF model, bootstrapped training and prediction is used to estimate the uncertainty of predictions made on unexplored points. Bootstrapping the training/prediction cycle in this manner introduces

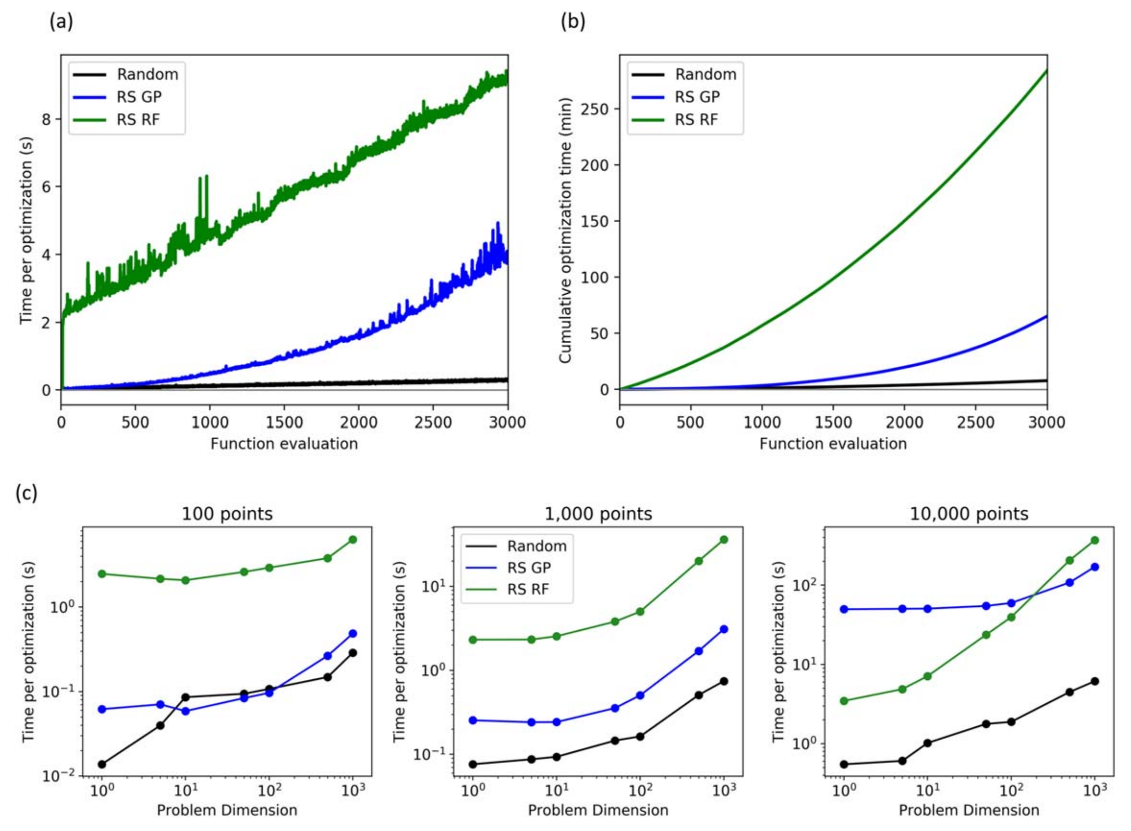


Figure 10. (a) Rocketsled overhead time per optimization, evaluated on a mock 1D objective function. (b) Cumulative time taken during optimization on a 1D objective function. (c) Time required per optimization for each optimizer for various combinations of problem dimension (D) and number of training points (N). All benchmarks were performed on a 2017 Macbook Pro with 3.5 GHz Intel i7 processor.

additional computational requirements during prediction. Overall, the overhead of the ML-enabled optimizations is dominated by the computational complexities of the underlying models, which are $O(n^2 \log n)$ (RF) [81] and $O(n^3)$ (GP) [82] when the problem dimension is held constant.

Black-box optimization problems may be high-dimensional, so in figure 10(c) we illustrate the computational overhead of each predictive algorithm as a function of problem dimension, up to 1000 dimensions, for various set numbers of training points (up to 10 000). The random model (representing non-ML Rocketsled operations) is typically near or less than 1 s per optimization unless $N > 10\ 000$ and the model is more than 100-dimensional, but the optimizers implementing machine learning methods are less robust to increasing problem dimension. For the bootstrapped RF, prediction tends to become computationally intensive ($>$10 s/optimization) for $N > 1000$ with $D > 50$. The GP optimizer is more robust to increases in problem dimension, but is still computationally intensive for large N and $D > 100$. Again, the overhead is dominated by the complexity of the underlying ML models. If the evaluation cost of the objective function is still significantly greater than these per-optimization times, the optimizers could be reasonably used with larger numbers of training points and more search dimensions. The worst cases generally represent 10–100 s of overhead per calculation, which is comparable to modern packages such as COMBO [15] and skopt [14] under the same conditions. Thus, the more expensive it is to perform a full function evaluation, the larger of a problem size that can be efficiently addressed with the Rocketsled framework. We note that the built-in optimizers' hyperparameters (e.g. number of training points, number of testing points, number of bootstraps) can be changed to increase the computational efficiency of the optimization if required.

4. Discussion, practical considerations, and limitations

While there are specialized black-box optimization packages already available, to our knowledge none adequately handle optimization for HTC. Most existing software is intended to execute the black-box function on the user's local computer, which limits their capability to manage complex production workflows on high performance systems. In these packages, the execution of the workflows is strictly sequential, meaning that the expensive black-box function and optimization are executed one after another, limiting parallelism.

Additionally, existing packages are often not robust to heterogeneous, discontinuous, or non-continuous search domains. Furthermore, few existing packages contain multi-objective capabilities or the ability to include domain knowledge through the addition of features or descriptors.

In contrast, Rocketsled's built-in optimizers are designed to balance performance and ease-of-use across a wide range of optimization problems. Integration with the FireWorks workflow software gives the user maximum control over the execution of the expensive black-box function and the optimization. Rocketsled is not limited to sequential function evaluation and optimization; it allows for black-box function evaluations in parallel across arbitrary computing resources as well as optimization in parallel, if desired (a detailed discussion of parallelism features is presented in the supplementary information). Built-in optimizers are robust to many kinds of search spaces, including discrete, combinatorically-generated search domains (sets of allowed values in each dimension) and discrete, non-combinatorically-generated search domains (a set of allowed points). Built-in optimizers also work on spaces defined with a combination of bounded and discrete dimensions (including categorical variables) and all optimizers can function in single or multi-objective mode. To our knowledge, there are no existing software packages robust to the variety of optimization requirements that Rocketsled supports.

Applying adaptive design problems to high-throughput computing problems is growing in popularity, particularly for exploration of novel inorganic materials. A previous study identified stable structures for stage-I and stage-II lithium-graphite intercalation compounds using 4%–6% of the calculations required to explore the entire search space, a combinatorial problem containing more than 16.7 million total possibilities [51]. Others have found adaptive design can accelerate searches—particularly those where exhaustive computation would be problematic—for a variety of applications including novel crystalline interfaces [56], elastic properties [52], high-pressure Mg-silicate phases [47], ultra-low thermal conductivity structures [53], inorganic/organic molecular interfaces [54], layered materials [50], stable carbide/nitrides [57], piezoelectrics [34], Poisson–Schrödinger simulations of LEDs [48], and stable crystal structures [55] for Y_2Co_{17}. In this study, we demonstrated two additional proofs-of-concept searching for superhard materials and photocatalysts. These studies strongly suggest that future use of the Rocketsled framework can lead to more efficient and ultimately more successful searches of materials space.

Rocketsled is the most useful when the following criteria are met: (i) full function evaluations are relatively expensive (e.g. $>$100 s per evaluation), (ii) the number of workflows to be run is on the scale of 100–1 000 000, (iii) workflows and their results are determined by their input parameters (non-stochastic), and (iv) one wants to take advantage of the workflow features built into the FireWorks platform such as distributed execution and comprehensive job management. Further details on limitations and usage of Rocketsled are presented in the supplementary information.

5. Conclusion

Rocketsled represents a robust, open-source framework capable of running a diverse range of black-box optimization problems on HTC resources. It contains a combination of features not found in other similar frameworks, notably: (i) distributed parallel execution on supercomputing platforms, (ii) support for multiple optimizers and acquisition functions, (iii) ability to handle heterogeneous search spaces, (iv) the ability to incorporate domain knowledge through descriptors, and (v) multi-objective optimization routines. We presented two applications in computational materials science demonstrating automatic optimization capabilities, demonstrating very high potential speedups compared to random searches, chemical-intuition based searches, or other methods such as GAs. We also have provided benchmarks regarding the computational cost of optimization. We believe the growth of data-driven scientific computing will enable many future use cases for Rocketsled, particularly in materials science, where the underlying FireWorks library is well-established.

Acknowledgments

This work was primarily funded and intellectually led by the United States Department of Energy, Office of Basic Energy Sciences, Early Career Research Program, which provided funding for AJ, AD, and JB. JB was supported in part by the US Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship (SULI) program. Lawrence Berkeley National Laboratory is funded by the DOE under award DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a US Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. We would also like to thank Juliane Mueller for her continued expert guidance and Saraubh Bajaj for code commits and review during early stages of development.

ORCID iDs

Alexander Dunn https://orcid.org/0000-0002-8567-1879

References

[1] Lussier Y A and Liu Y 2007 Computational approaches to phenotyping: high-throughput phenomics *Proc. Am. Thorac. Soc.* **4** 18–25
[2] Jain A *et al* 2011 A high-throughput infrastructure for density functional theory calculations *Comput. Mater. Sci.* **50** 2295–310
[3] Jain A, Shin Y and Persson K A 2016 Computational predictions of energy materials using density functional theory *Nat. Rev. Mater* **1** 15004
[4] Curtarolo S *et al* 2013 The high-throughput highway to computational materials design *Nat. Mater.* **12** 191
[5] Shahriari B, Swersky K, Wang Z, Adams R P and Freitas N. de 2016 Taking the human out of the loop: a review of bayesian optimization *Proc. IEEE* **104** 148–75
[6] Rasmussen C E and Williams C K I 2005 *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)* (Cambridge, MA: MIT Press)
[7] Breiman L 2001 Random forests *Mach. Learn.* **45** 5–32
[8] Github BayesianOptimization https://github.com/fmfn/BayesianOptimization (Accessed: 2 October 2018)
[9] GitHub Yelp MOE https://github.com/Yelp/MOE (Accessed: 10 February 2018)
[10] Snoek J, Larochelle H and Adams R P 2012 Practical bayesian optimization of machine learning algorithms *Proc. 25th Int. Conf. on Neural Information Processing Systems* vol 2 (Curran Associates Inc) pp 2951–9
[11] Golovin D *et al* 2017 Google Vizier : a service for black-box optimization *KDD '17: Proc. of the 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (Halifax, NS, August 13–17 2017)* (New York: ACM) pp 1487–95
[12] Eriksson D, Bindel D and Shoemaker C GitHub - Surrogate Optimization Toolbox (pySOT) https://github.com/dme65/pySOT (Accessed: 15 January 2019)
[13] Kurchin R, Romano G and Buonassisi T 2018 Bayesim: a tool for adaptive grid model fitting with Bayesian inference 1–8
[14] GitHub Scikit-Optimize https://github.com/scikit-optimize/scikit-optimize (Accessed: 12 December 2018)
[15] Ueno T, Rhone T D, Hou Z, Mizoguchi T and Tsuda K 2016 COMBO: an efficient Bayesian optimization library for materials science *Mater. Discov.* **4** 18–21
[16] Ling J, Hutchinson M, Antono E, Paradiso S and Meredig B 2017 High-dimensional materials and process optimization using data-driven experimental design with well-calibrated uncertainty estimates *Integr. Mater. Manuf. Innov.* **6** 207–217
[17] Jain A *et al* 2010 FireWorks: a dynamic workflow system designed for high-throughput applications *Concurr. Comput. Pract. Exp.* **27** 5037–59
[18] Mathew K *et al* 2016 MPInterfaces: a materials project based python tool for high-throughput computational screening of interfacial systems *Comput. Mater. Sci.* **122** 183–90
[19] Petretto G *et al* 2018 High-throughput density-functional perturbation theory phonons for inorganic materials *Sci. Data* **5** 180065
[20] De Jong M *et al* 2015 Charting the complete elastic properties of inorganic crystalline compounds *Sci. Data* **2** 1–13
[21] Chen W *et al* 2016 Understanding thermoelectric properties from high-throughput calculations: trends, insights, and comparisons with experiment *J. Mater. Chem.* C **4** 4414–26
[22] Tran R *et al* 2016 Surface energies of elemental crystals *Sci. Data* **3** 160080
[23] Qu X *et al* 2015 The electrolyte genome project: a big data approach in battery materials discovery *Comput. Mater. Sci.* **103** 56–67
[24] MongoDB www.mongodb.com (Accessed: 15 January 2019)
[25] Pedregosa F *et al* 2012 Scikit-learn: machine learning in python *J. Mach. Learn. Res.* **12** 2825–30
[26] Ramprasad R, Batra R, Pilania G, Mannodi-Kanakkithodi A and Kim C 2017 Machine learning and materials informatics: recent applications and prospects *npj Comput. Mater.* **3** 54
[27] Jones D R, Schonlau M and Welch W J 1998 Efficient global optimization of expensive black-box functions *J. Glob. Optim.* **13** 455–92
[28] Lizotte D J 2008 Practical Bayesian optimization *PhD Thesis* University of Alberta
[29] Brochu E, Cora V M and de Freitas N 2010 A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning (arXiv:1012.2599)
[30] Kushner H J 1964 A new method of locating the maximal point of an arbitrary multipeak curve in the presence of noise *J. Basic Eng.* **86** 97–106
[31] Cox D D and John S 2002 A statistical method for global optimization *IEEE Int. Conf. on Systems, Man, and Cybernetics (Chicago, IL, 18–21 October 1992)* pp 1241–6
[32] Cui Y, Geng Z, Zhu Q and Han Y 2017 Review: multi-objective optimization methods and application in energy saving *Energy* **125** 681–704
[33] Balling R 2003 The maximin fitness function; multi-objective city and regional planning. *Evolutionary Multi-Criterion Optimization* ed C M Fonseca *et al* (Berlin: Springer) pp 1–15
[34] Gopakumar A M, Balachandran P V, Xue D and Gubernatis J E 2018 Multi-objective optimization for materials discovery via adaptive design *Sci. Rep.* **8** 3738
[35] Weininger D 1988 SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules *J. Chem. Inf. Comput. Sci.* **28** 31–6
[36] Rosenbrock H H 1960 An automatic method for finding the greatest or least value of a function *Comput. J.* **3** 175–84
[37] Dixon L C W and Szegö G P 1978 *Towards Global Optimisation 2* (Amsterdam: North-Holland) pp 1–15
[38] Surjanovic S, Bingham D and Simon F U Virtual Library of Simulation Experiments: Test Functions and Datasets https://sfu.ca/~ssurjano/optimization.html (Accessed: 10 April 2018)
[39] Mathew K *et al* 2017 Atomate: a high-level interface to generate, execute, and analyze computational materials science workflows *Comput. Mater. Sci.* **139** 140–52
[40] Pizzi G, Cepellotti A, Sabatini R, Marzari N and Kozinsky B 2016 AiiDA : automated interactive infrastructure and database for computational science *Comput. Mater. Sci.* **111** 218–30
[41] Mayeshiba T *et al* 2017 The materials simulation toolkit (MAST) for atomistic modeling of defects and diffusion *Comput. Mater. Sci.* **126** 90–102
[42] Curtarolo S *et al* 2012 AFLOW: An automatic framework for high-throughput materials discovery *Comput. Mater. Sci.* **58** 218–26
[43] Davies D W *et al* 2016 Computational screening of all stoichiometric inorganic materials *Chem* **1** 617–27
[44] Xue D *et al* 2016 Accelerated search for materials with targeted properties by adaptive design *Nat. Commun.* **7** 1–9
[45] Oda H, Kiyohara S, Tsuda K and Mizoguchi T 2017 Transfer learning to accelerate interface structure searches *J. Phys. Soc. Japan* **86** 1–4

[46] Solomou A *et al* 2018 Multi-objective Bayesian materials discovery: application on the discovery of precipitation strengthened NiTi shape memory alloys through micromechanical modeling arXiv:1807.06868

[47] Wu S Q *et al* 2014 An adaptive genetic algorithm for crystal structure prediction *J. Phys.: Condens. Matter* **26**

[48] Rouet-Leduc B, Hulbert C, Barros K, Lookman T and Humphreys C J 2017 Automatized convergence of optoelectronic simulations using active machine learning *Appl. Phys. Lett.* **111** 043506

[49] Lookman T, Balachandran P V and Xue D 2019 Active learning in materials science with emphasis on adaptive sampling using uncertainties for targeted design *npj Comput. Mater* **5** 21

[50] Bassman L *et al* 2018 Active learning for accelerated design of layered materials *npj Comput. Mater.* **4** 74

[51] Okamoto Y 2017 Applying bayesian approach to combinatorial problem in chemistry *J. Phys. Chem.* A **121** 3299–304

[52] Balachandran P V, Xue D, Theiler J, Hogden J and Lookman T 2016 Adaptive strategies for materials design using uncertainties *Sci. Rep.* **6** 1–9

[53] Seko A *et al* 2015 Prediction of low-thermal-conductivity compounds with first-principles anharmonic lattice-dynamics calculations and bayesian optimization *Phys. Rev. Lett.* **115** 1–5

[54] Todorović M, Gutmann M U, Corander J and Rinke P 2017 Bayesian inference of atomistic structure in functional materials *npj Comput. Mater.* **5** 35

[55] Yamashita T *et al* 2018 Crystal structure prediction accelerated by Bayesian optimization *Phys. Rev. Mater.* **2** 013803

[56] Kiyohara S, Oda H, Tsuda K and Mizoguchi T 2016 Acceleration of stable interface structure searching using a kriging approach *Japan. J. Appl. Phys.* **55** 3–6

[57] Talapatra A *et al* 2018 Towards an autonomous efficient materials discovery framework: an example of optimal experiment design under model uncertainty *Phys. Rev. Mater.* **2** 113803

[58] Cerqueira T F T *et al* 2015 Materials design on-the-fly *J. Chem. Theory Comput.* **11** 3955–60

[59] Lookman T, Balachandran P V, Xue D, Hogden J and Theiler J 2017 Statistical inference and adaptive design for materials discovery *Curr. Opin. Solid State Mater. Sci.* **21** 121–8

[60] Balachandran P V, Kowalski B, Sehirlioglu A and Lookman T 2018 Experimental search for high-temperature ferroelectric perovskites guided by two-step machine learning *Nat. Commun.* **9**

[61] Raccuglia P *et al* 2016 Machine-learning-assisted materials discovery using failed experiments *Nature* **533** 73–6

[62] Dieb T M and Tsuda K 2018 *Machine Learning-Based Experimental Design in Materials Science BT - Nanoinformatics* ed I Tanaka (Singapore: Springer) pp 65–74

[63] Frazier P I and Wang J 2016 *Bayesian Optimization for Materials Design* (*Information Science for Materials Discovery and Design*) ed T Lookman, F J Alexander and K Rajan (Berlin: Springer) pp 45–75

[64] Castelli I E *et al* 2012 New cubic perovskites for one- and two-photon water splitting using the computational materials repository *Energy Environ. Sci.* **5** 9034–43

[65] Jain A, Castelli I E, Hautier G, Bailey D H and Jacobsen K W 2013 Performance of genetic algorithms in search for water splitting perovskites *J. Mater. Sci.* **48** 6519–34

[66] Goldschmidt V M 1926 Die gesetze der krystallochemie *Naturwissenschaften* **14** 477–85

[67] Ward L *et al* 2018 Matminer: an open source toolkit for materials data mining *Comput. Mater. Sci.* **152** 60–9

[68] Ward L, Agrawal A, Choudhary A and Wolverton C 2016 A general-purpose machine learning framework for predicting properties of inorganic materials *npj Comput. Mater* **2** 1–7

[69] Hill R 1952 The elastic behaviour of a crystalline aggregate *Proc. Phys. Soc. Sect.* A **65** 349

[70] Vepřek S 1999 The search for novel, superhard materials *J. Vac. Sci. Technol.* A **17** 2401–20

[71] Mansouri Tehrani A *et al* 2018 Machine learning directed search for ultraincompressible, superhard materials *J. Am. Chem. Soc.* **140** 9844–53

[72] Pan Z, Sun H, Zhang Y and Chen C 2009 Harder than diamond: superior indentation strength of wurtzite BN and lonsdaleite *Phys. Rev. Lett.* **102** 1–4

[73] Riedel R 2000 *Handbook of Ceramic Hard Materials* (New York: Wiley-VCH)

[74] Liu A Y and Cohen M L 1989 Prediction of new low compressibility solids *Science* **245** 841–2

[75] Tabor D 2000 *The Hardness of Metals* (Oxford: Clarendon Press, Oxford University Press)

[76] Liu G *et al* 2015 Submicron cubic boron nitride as hard as diamond *Appl. Phys. Lett.* **106** 121901

[77] Solozhenko V L, Kurakevych O O, Andrault D, Le Godec Y and Mezouar M 2009 Ultimate metastable solubility of boron in diamond: Synthesis of superhard diamondlike BC5 *Phys. Rev. Lett.* **102** 1–4

[78] Zhao Z *et al* 2014 Nitrogen concentration driving the hardness of rhenium nitrides *Sci. Rep.* **4** 4797

[79] De Jong M *et al* 2016 A statistical learning framework for materials science: application to elastic moduli of *k*-nary inorganic polycrystalline compounds *Sci. Rep.* **6** 34256

[80] Zimmermann N E R, Horton M K, Jain A and Haranczyk M 2017 Assessing local structure motifs using order parameters for motif recognition, interstitial identification, and diffusion path characterization *Frontiers Mater.* **4** 34

[81] Louppe G 2014 Understanding random forests: from theory to practice (https://doi.org/10.13140/2.1.1570.5928)

[82] Hensman J, Fusi N and Lawrence N D 2013 Gaussian processes for big data *Proc. Twenty-Ninth Conf. on Uncertainty in Artificial Intelligence* pp 282–90