Computer Physics Communications 297 (2024) 109065

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computer Programs in Physics

# Clinamen2: Functional-style evolutionary optimization in Python for atomistic structure searches ☆

Ralf Wanzenböck [a], Florian Buchner [a], Péter Kovács [a], Georg K.H. Madsen [a], Jesús Carrete [b,a,*]

[a] Institute of Materials Chemistry, TU Wien, A-1060 Vienna, Austria
[b] Instituto de Nanociencia y Materiales de Aragón (INMA), CSIC-Universidad de Zaragoza, 50009 Zaragoza, Spain

ARTICLE INFO

Dataset link: https://github.com/Madsen-s-research-group/clinamen2-public-releases

Dataset link: https://doi.org/10.5281/zenodo.10143313

*Keywords:*
CMA-ES
Optimization
Python
Atomistic calculations
Structure search

ABSTRACT

Clinamen2 is a versatile functional-style Python implementation of the covariance matrix adaptation evolution strategy (CMA-ES) utilizing Cholesky decomposition. On top of a problem-agnostic core algorithm, the software package offers a suite of utilities and library code enabling applications to important atomistic structure searches. Features include massively distributed computation and the BI-Population restart scheme. This article details the general code structure and introduces examples that illustrate some relevant applications for the materials science and chemistry worlds, including interfacing to density-functional-theory codes and machine-learned surrogate models. The functional design renders the code modular and adaptable, and makes the creation of interfaces to other atomistic software straightforward.

**Program summary**
*Program Title:* Clinamen2
*CPC Library link to program files:* https://doi.org/10.17632/x7syr2txsd.1
*Developer's repository link:* https://github.com/Madsen-s-research-group/clinamen2-public-releases
*Code Ocean capsule:* https://codeocean.com/capsule/4950229
*Licensing provisions:* Apache-2.0
*Programming language:* Python
*Supplementary material:*
*Nature of problem:* Find optimal atomistic structures retaining full flexibility in the choice of the optimization target, the methodological approach and its implementation (e.g. CPU- vs. GPU-heavy calculations). Enable interfacing with relevant software, including but not limited to density-functional-theory (DFT) codes and machine-learning (ML) solutions.
*Solution method:* The covariance matrix adaptation evolution strategy (CMA-ES) algorithm is implemented using Cholesky decompositions for efficiency. The core algorithm and application examples for specific problems are implemented in functional-style Python free of side effects, with data classes to keep track of the state of the evolution. Dask is used for job control to enable highly distributed workflows. Advanced strategies like BI-Population CMA-ES are easy to implement and illustrated in the examples.

## 1. Introduction

The covariance matrix adaptation evolution strategy (CMA-ES) [1–3] is a powerful and efficient tool for gradient-free optimization of high-dimensional problems, with a limited set of problem-specific hyperparameters. The main algorithm is based on drawing samples from a continuously updated multivariate normal distribution and has been extended and refined following different approaches, e.g., focusing on restarts with varying population and step sizes for optimizing multimodal functions [4–10]. Furthermore, much effort has gone into the advancement of the fundamental update scheme and the comparison of different implementations [6,8,11–14]. There are several CMA-ES codes available [15–22], often tailored to specific problems, and numerous applications to different subject matters [18,23–28].

☆ The review of this paper was arranged by Prof. Weigel Martin.

* Corresponding author at: Instituto de Nanociencia y Materiales de Aragón (INMA), CSIC-Universidad de Zaragoza, 50009 Zaragoza, Spain.
*E-mail address:* jcarrete@unizar.es (J. Carrete).

https://doi.org/10.1016/j.cpc.2023.109065
Received 7 June 2023; Received in revised form 16 November 2023; Accepted 13 December 2023
Available online 21 December 2023

The Python package Clinamen2 was developed to provide a versatile and extensible implementation of the CMA-ES. The wider aim was to offer the building blocks needed for a wide range of problems. The package follows a functional programming approach with building-block functions including the main algorithm, the encoding and decoding of structures as input vectors (degrees of freedom), interfaces to various codes for loss evaluation, as well as a number of convenience functions.

While aiming at a flexible code where any Python function with a compatible signature can be used for loss evaluation, we kept materials-science-related applications in focus. Finding minima of energy or free energy landscapes is a central problem when trying to describe the structure and dynamics of systems of atoms, whether periodic (derived from crystals), nonperiodic (molecules or clusters) or mixed (surfaces, both cleaved and with adsorbates, crystals with imperfections and so on). Given the extremely large number of minima involved in such searches, evolutionary algorithms are particularly appealing because of their ability to balance exploration and exploitation. Clinamen2 is the spiritual successor of Clinamen [18], an earlier Python code used for studying defect structures. It has been rewritten from scratch with an emphasis on adaptability. To illustrate its flexibility, example applications are included with the code and available on GitHub. Clinamen2 can interface to density-functional-theory (DFT) codes that are compatible with the atomic simulation environment (ASE) [29]. Interfaces to GPAW [30], NWChem [31] and VASP [32] are included with the software. Additionally, we provide a direct interface to a neural-network force field (NNFF) surrogate model, exemplified by a search of defect structures in silicon bulk, an evolution run of Lennard-Jones (LJ) clusters, and an evaluation of standard benchmark functions. In addition to the main CMA-ES algorithm, Clinamen2 implements several useful features such as covariance matrix decomposition [6,8] and the BI-Population restart scheme [5]. Clinamen2 utilizes the Dask Python library [33,34] for distributed computation on CPU and GPU clusters. We have included an example application that demonstrates the combination of Dask and the DFT codes NWChem and VASP for investigating atomistic structures of small silver clusters. Additionally, Clinamen2 provides convenience functions aimed at facilitating the implementation of applications and the quick evaluation of results, e.g., through the visualization of the loss trajectory.

## 2. Background

### 2.1. Covariance matrix adaptation evolution strategy

At its core, the original CMA-ES samples a population of $\lambda$ individuals $\boldsymbol{x}_k^g$, $k = 1, .., \lambda$ for every generation $g$ from the multivariate normal distribution

$$\boldsymbol{x}_k^{(g)} \sim \mathcal{N}\left(\boldsymbol{m}^{(g-1)}, \left[\sigma^{(g-1)}\right]^2 \boldsymbol{C}^{(g-1)}\right), \tag{1}$$

with distribution mean $\boldsymbol{m}$, step size $\sigma$ and covariance matrix $\boldsymbol{C}$. Typically, the user-defined input is limited to the initial mean (founder) $\boldsymbol{m}^{(0)}$, step size $\sigma^{(0)}$ and population size $\lambda$, and should be tuned for the specific problem [3]. Following Ref. [3], the default population size is set to $\lambda = 4 + \lfloor 3 \log d \rfloor$ based on the dimension $d$ of the problem. The advantage of increasing the population size for the investigation of multimodal functions has been demonstrated and different variations have been proposed [4,35–37].

The covariance matrix is updated by performing the so-called rank-one and rank-$\mu$ updates utilizing evolution paths [3]. As explained in Ref. [6], direct computation and storage of the covariance matrix $\mathbf{C}$ can be avoided by instead performing updates on its decomposition, e.g., on the triangular Cholesky factor $\boldsymbol{A}$ such that $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{A}^T$. In order to accomplish this, the rank-$\mu$ update needs to be decomposed into a series of rank-one updates. To that end, we follow Ref. [8] and implement the algorithms "Cholesky-CMA-ES" and "rankOneUpdate" defined therein.

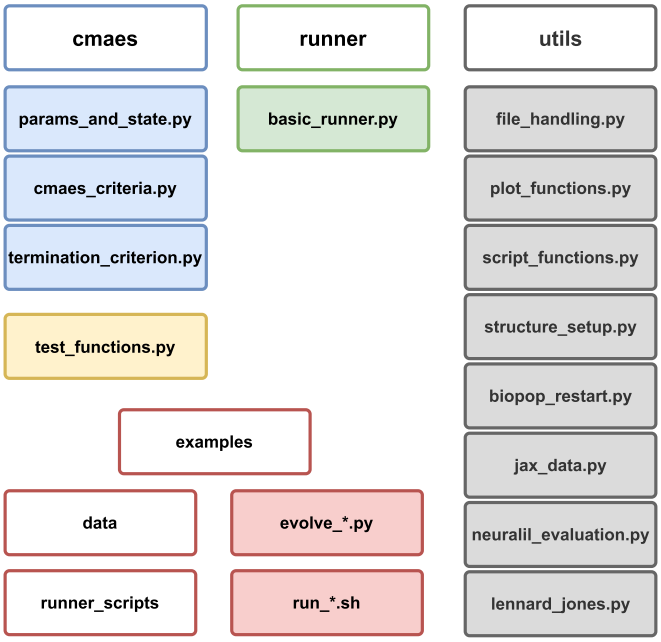


**Fig. 1.** The Clinamen2 package structure is divided into CMAES (core algorithm), RUNNER (functions for distributed computation), UTILS (utilities and tools for application implementation) with a set of TEST FUNCTIONS and EXAMPLES that are built utilizing the aforementioned code.

### 2.2. BI-population restart

In the context of the CMA-ES, a "restart" is a new evolution run, starting from either the same founder structure, i.e., the initial mean $\boldsymbol{m}^{(0)}$ in Eq. (1), as the original run or another randomly generated one with some parameters changed, e.g., the population size. Here, we utilize the BI-Population CMA-ES (BIPOP-CMA-ES) restart scheme [5,7]. This restart scheme includes two regimes that increase or decrease the population size with each restart, respectively, starting from the same founder structure. The first ("large") restart regime always doubles the population size to $\lambda_{\text{large}} = 2^{i_{\text{restart}}} \lambda_{\text{init}}$, where $\lambda_{\text{init}}$ is the initial population size chosen for the problem. The second ("small") restart scheme, in contrast, changes the population size to

$$\lambda_{\text{small}} = \left\lfloor \lambda_{\text{init}} \left( \frac{1}{2} \frac{\lambda_{\text{large}}}{\lambda_{\text{init}}} \right)^{U[0,1]^2} \right\rfloor, \tag{2}$$

and the step size to $\sigma_{\text{small}}^{(0)} = 10^{-2U[0,1]} \sigma_{\text{init}}^{(0)}$. Here, $U[0,1]$ is a random number uniformly drawn from $[0,1]$.

For every subsequent restart the scheme chooses the regime that has so far required the smaller number of total loss evaluations, with each restart running for a set number of generations or function evaluations or until one of the defined termination criteria is fulfilled. Usually, a maximum number of nine restarts in the "large" regime is performed [5].

## 3. General structure of Clinamen2

Fig. 1 illustrates the package structure of the Clinamen2 code, which is divided into four parts: CMA-ES core algorithm (CMAES), distributed computation (RUNNER), additional functionality (UTILS) and example applications. The implementation of the CMA-ES core algorithm is problem agnostic, i.e., an input vector and parameters are passed in and where these input values come from is of no relevance. Additional functions provide utilities and tools that can be used to apply the algorithm. A number of examples demonstrating the use of the algorithm

in conjunction with the additional functions is included on GitHub and described in section 4.

This implementation largely sticks to the principles of functional programming [38] re-popularized by modern ML frameworks. The two basic elements are frozen instances of *data classes* and free-floating *pure functions* operating on them. As the name suggests, data classes describe and hold data in a convenient, structured way. By freezing an instance of a data class, it becomes immutable. For brevity and in keeping with common Python usage, we refer to data classes whose instances are automatically frozen as "frozen data classes". Pure functions take those objects as strictly input parameters and, if necessary, construct and return new objects. The key feature of such functions is that they lack side effects. Therefore, state keeping and information flow are made explicit, which greatly simplifies optimization and parallelism. A third ubiquitous kind of entity, derived from those two, is a *closure*, i.e., the combination of a function with a frozen set of environment data controlling its behavior. Closures are typically returned by other functions.

### 3.1. CMA-ES algorithm

The frozen data classes AlgorithmParameters and AlgorithmState and related free-floating functions are at the core of the implementation of the main CMA-ES algorithm. While the AlgorithmParameters are fixed over the course of an evolution, a new instance of AlgorithmState, reflecting any changes, including the state of a random number generator, is returned by every function operating on it. For example,

```
def create_sample_from_state(
    parameters: AlgorithmParameters
) -> Callable:
```

returns a closure with access to all attributes of an AlgorithmParameters instance. The function created this way then returns a population of individuals from an AlgorithmState and the new state reflecting the sampling when called.

Since the main algorithm is problem agnostic, the means for loss evaluation have to be made available, e.g., constructed from the provided building blocks, when setting up an application. Clinamen2 includes a number of functions that create compatible closures to be used with the CMA-ES implementation. Each of these functions (create_sample_and_*) is tailored to a specific use case, with some of them utilized in the examples in section 4. For each new generation a population of individuals needs to be sampled from the AlgorithmState and evaluated. This might involve transformations of the input vector to make it compatible with the provided loss-evaluation function, which can be as simple as reshaping the input vector or arbitrarily complex. The simplest way to create sampling and evaluation closures is

```
def create_sample_and_sequential_evaluate(
    sample_individuals: Callable,
    evaluate_loss: Callable,
    input_pipeline: Callable = lambda x: x,
) -> Callable:
```

where in most applications sample_individuals is the standard function provided in Clinamen2, evaluate_loss can be any Python function with a compatible signature and input_pipeline transforms the input vector as needed. Convenient functions for updating the AlgorithmState and selected termination criteria are also available.

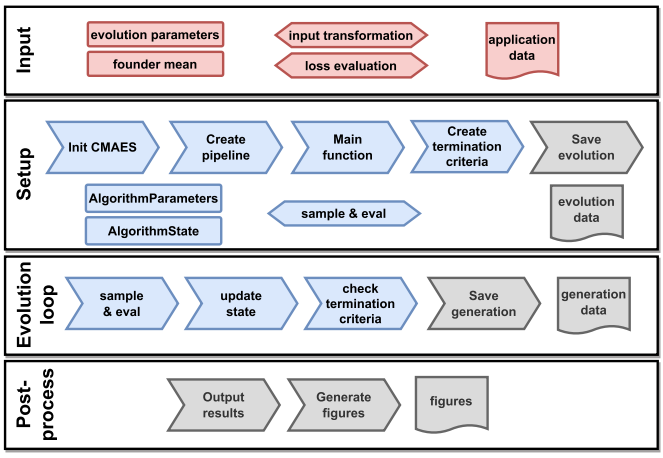


**Fig. 2.** The code structure of a basic Clinamen2 application. Colored rectangles represent objects or direct user input (arguments), hexagon-like elements are Python functions and document-style elements are files (either additional input or output). Red highlights are provided by the user, blue indicates main algorithm features, and utilities are shown in gray.

For an overview of the code structure of a basic Clinamen2 example application see Fig. 2. To start with, input has to be provided. This includes CMA-ES parameters, a founder structure, a function for input transformation, e.g., reshaping of the degrees of freedom into atom positions, and a function for loss evaluation. Depending on the use case the founder structure might be randomly generated at run-time or read from a file. Next, the building blocks of the core algorithm are used to set up all objects and closures needed to perform the evolution. With this, the evolution loop can be started and, again depending on the problem, the final result, every generation or every $n^{th}$ generation, is saved. Together with the evolution data saved during setup, any generation file can be used to continue the evolution from. During post-processing, the results of the evolution and any additional information can be printed to the shell or saved to files. It is often useful to generate simple trajectory figures to get an idea of how the evolution progressed. Relevant functions are provided in utils.plot_functions.

### 3.2. Distributed computation

For distributed computation, e.g., running CPU-heavy DFT calculations on a cluster, we rely on the Dask [33,34] Python package. In that framework, *clients* register with and submit jobs to a *scheduler*. The scheduler then distributes these jobs to suitable *workers*. Fig. 3 shows a schematic of this interplay between components. The calculation jobs themselves are implemented utilizing Jinja2 template scripts. Clinamen2 provides runner.basic_runner.FunctionRunner, which runs generic function calls, and runner.basic_runner.ScriptRunner, which is able to handle the execution of external programs.

The Dask scheduler is started from the command line, with output and errors written to LOG and LOGERR, respectively:

```
dask-scheduler
  -scheduler-file scheduler_nwchem.json
  -interface em2 1>LOG 2>LOGERR
```

Each worker is then usually submitted to a queuing system, e.g., Slurm, including the bash command:

```
dask-worker
  -nthreads 1 -nworkers 1
```

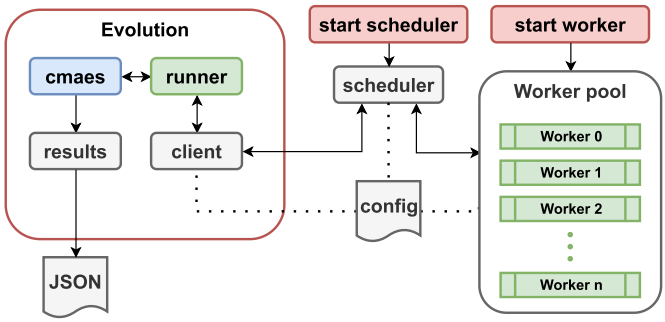


**Fig. 3.** A schematic of the interplay between components when performing distributed calculations with Dask. Those provided by the user are highlighted in red, and blue indicates main algorithm features, with the runner components in green and utilities shown in gray.

```
  -local-directory ${SOME_SCRATCH_SPACE}
  -scheduler-file scheduler_nwchem.json
```

The Dask client can be also instantiated from Python:

```
import dask.distributed

dask_client = dask.distributed.Client(
    scheduler_file="scheduler_nwchem.json",
)
```

For an end-to-end example of how Dask is used in Clinamen2, see section 4.2 and the associated online documentation and code example in the GitHub repository.

### 3.3. Utils

The Clinamen2 code features a suite of utilities (see gray rectangles in Fig. 1), some directed towards general convenience, while others are specific to problems or applications. UTILS.FILE_HANDLING provides the CMAFILEHANDLER with functions for the saving and loading of evolution and generation data. Utilizing these functions, UTILS.PLOT_FUNCTIONS offers an interface to load data pertaining to an evolution run and a selection of default plots, e.g., loss trajectories.

Additional UTILS packages include STRUCTURE_SETUP, BIPOP_RESTART and SCRIPT_FUNCTIONS containing, e.g., a default parser for command line arguments. For more details see the online documentation and the examples in section 4.

### 3.4. Checkpointing

In case an evolution was interrupted or additional generations are desired, a run can easily be resumed from the initial evolution data and any generation that has been saved to file. The checkpointing feature is used in the function-trial and LJ examples.

### 3.5. Interfaces and portability

The modular architecture of Clinamen2 greatly simplifies the use of individual components or subsets of those. Among other scenarios, this enables the easy development of new distributed backends, integration with workflow managers and use of arbitrary calculators. We briefly explore those possibilities in this subsection.

Although Dask can be deployed on a wide range of HPC and cloud infrastructure, it is conceivable that a different package such as Ray [39] or a lower-level queue manager is already available and optimized to run distributed jobs on a given target and that installing a second one is undesirable. Conveniently for this sort of situation, all the interaction with Dask is handled through the interface provided by the abstract RUNNER base class, so to create a different backend it is enough to subclass it. Moreover, the basic required interface consists in a single method to submit a new job and another one to fetch a result, meaning that the new implementation can be completed in a few hundred lines of code. Note, in particular, that Clinamen2 does not rely on the Dask features dealing with seamlessly distributed arrays or dataframes.

A more general use case is the inclusion of a Clinamen2 calculation as a step in a larger workflow, be it a high-throughput exploration of a library of materials or a single optimization effort. The functional, stateless design of the package facilitates the integration of individual components with object-oriented software keeping track of state. For instance, Clinamen2 can be used to add support for CMA-ES to the global optimization package AGOX [40] by creating a Clinamen2-backed SAMPLER class and letting either package handle the evaluation of configurations. Likewise, integration with workflow managers like AiiDA [41] or Fireworks [42] can be achieved at different levels: from using Clinamen2 as a standalone component in charge of the full optimization step to running individual CMA-ES through the corresponding component's interface and letting the workflow manager's queue system handle the distributed calculations.

Finally, although we have included very general RUNNER subclasses to deal with functions and scripts, and we provide an example of use of the popular ASE [29] CALCULATOR interface available for a plethora of atomistic programs, other communication channels are easily implemented. A new RUNNER subclass that completes its job through, e.g., interprocess communication mechanisms (sockets or pipes) or even an HTTP request to a REST interface can be quickly written and, as long as it complies with the interface of the basic class, will be seamlessly integrated in Clinamen2.

## 4. Example applications

In the following we exemplify the features of Clinamen2 using standard benchmark functions and the structures of small silver clusters, defects in silicon bulk and LJ clusters. The examples can be found on GitHub and the technical details on the implementation of the examples in the online documentation. The silicon bulk (section 4.3) and LJ-cluster (section 4.4) examples require Google JAX [43]. The DFT code utilized in the investigation of silver clusters (section 4.2) is NWChem [31], which needs to be installed separately (a variation on this example using VASP [32] instead is also provided with Clinamen2).

### 4.1. Benchmark functions

In order to test the validity of the presented CMA-ES implementation, optimization is performed on benchmark functions following the choices of Krause et al. in Ref. [8]. The function definitions are given in Table 1, and Fig. 4 shows the average number of generations required to arrive at $f(x) < 10^{-14}$ for 50 trial runs per function. Each function was evaluated for all input dimensions $d \in \{4, 8, 16, 32, 64, 128, 256\}$ with default population size and initial step size $\sigma^{(0)} = 1.0$. The initial mean $\mathbf{x}$ for each run was randomly drawn as $x_i \in \mathcal{N}(0, \mathbb{I})$ for the sphere function and $x_i \in [0, 1]$ for all others.

Furthermore, the Ackley function [44]

$$f(\mathbf{x}) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=0}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=0}^{d}\cos\left(cx_i\right)\right) + a + \exp(1), \tag{3}$$

**Table 1**
Benchmark functions reproduced from Ref. [8].

| Label | $f(x)$ |
|---|---|
| Cigar | $10^{-6}x_0^2 + \sum_{i=1}^{d} x_i^2$ |
| Different Powers | $\sum_{i=0}^{d} \lvert x_i \rvert^{\left(2+\frac{10i}{d-1}\right)}$ |
| Discus | $x_0^2 + \sum_{i=1}^{d} 10^{-6}x_i^2$ |
| Ellipsoid | $\sum_{i=0}^{d} 10^{\frac{-6i}{d-1}} x_i^2$ |
| Rosenbrock | $\sum_{i=0}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + \left(1 - x_i\right)\right]$ |
| Sphere | $\lvert\lvert\mathbf{x}\rvert\rvert^2$ |

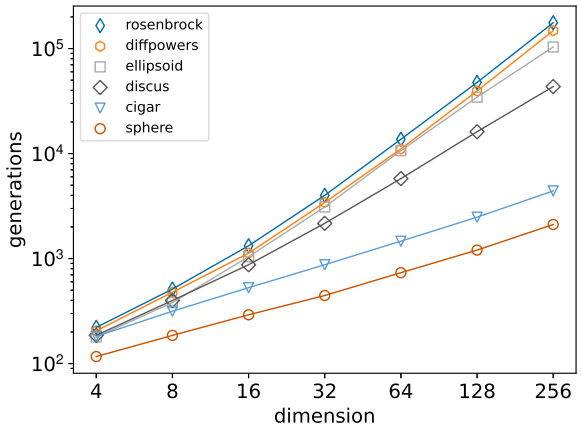


**Fig. 4.** The average number of generations required to reach a function value $f(x) < 10^{-14}$ over the dimension of the input vector. The benchmark functions are listed in Tbl. 1. 50 runs were performed for each combination.

with the common parametrization $a = 20$, $b = 0.2$ and $c = 2\pi$, was evaluated for $d \in \{128, 256, 512\}$ and the default population size [3], performing 50 trial runs for each combination. Twice the default population size was used when the success rate failed to reach 100 % for a problem dimension and, additionally, a third trial was performed for $d = 512$ and $\lambda_{512} = 3\lambda_{\text{default}}$.

Here, the founder configurations $\mathbf{x}$ were generated with the usual random drawing of $x_i \in [-32.768, 32.768]$ and an initial step size of $\sigma^{(0)} = 12.5$ was used, i.e. a fifth of the value range [5]. Fig. 5 shows the success rates and average number of function evaluations for each investigated combination of dimension and population size. Additional runs were performed for smaller problem dimensions, all arriving at 100 % success rate for default population size. While the CMA-ES performs very well for the small default population sizes, these results indicate the benefit of larger population sizes on the global optimization performance at least for multimodal functions [4]. For example, the default population size for dimension $d = 512$ is $\lambda_{512} = 22$, with the success rate of this combination being zero for the presented trial function. Consequently, increasing the population size parameter can be a powerful tool for exploring complex loss surfaces, as long as efficient evaluation is feasible, e.g., by incorporating machine-learned approaches. From the average number of function evaluations of successful runs it can be surmised that fewer generations are needed with a larger population size, 1485 generations for $\lambda_{512} = 66$ compared to 1900 for $\lambda_{512} = 44$. Therefore, even for the same success rate, the larger population size may be the more efficient choice, depending on the degree of parallelization of the function evaluations.

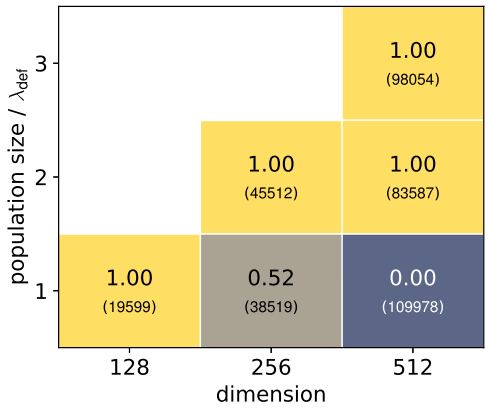


**Fig. 5.** The success rate for finding the global minimum of the Ackley function for given input dimensions and population sizes. Lighter colors indicate a higher success rate. Empty squares indicate that these combinations were not performed. The small numbers in parentheses show the average loss evaluations needed to arrive at a successful run. For success rate 0.00 the average over all runs is shown instead.

### 4.2. Ag cluster with DFT

To demonstrate how to interface to a DFT code and utilize the distributed computation capabilities of Dask [33,34], the structure of small silver clusters is investigated using NWChem [31] and VASP [32] for loss evaluation. This is not aimed at comparing the performance of the DFT codes, but at illustrating how different loss-calculation backends can be interchanged and used analogously.

We employ the workflow described in section 3.2, running as many worker processes as there are individuals in a generation, with each worker utilizing 16 CPU cores. This pool of workers registers with the scheduler which takes care of utilization and job distribution. In the example, which can be found on the GitHub repository, the evolution uses the worker pool exclusively and, while it is also possible to share the pool amongst evolution runs, this leads to a trade-off between reduced idle time and increased waiting periods.

Starting from configurations with atoms randomly placed evolution runs are performed for the $Ag_5$, $Ag_6$ and $Ag_7$ clusters. Running for 350 generations with the default population size, initial step size $\sigma^{(0)} = 1.0$ Å and no additional local optimization stable structures in good agreement with literature [45–47] are found. Fig. 6 shows a selection of results: $Ag_5$ and $Ag_7$ clusters evolved with NWChem starting from random positions in a sphere and two configurations of $Ag_6$ utilizing VASP, starting from random positions within a cube, uniformly distributed and utilizing PACKMOL, respectively.

### 4.3. Si bulk with neural-network force field

This example includes the use of a surrogate model for ab-initio calculations, in particular a neural-network force field (NNFF), and biasing of the initial covariance matrix. In Ref. [18] the authors perform a CMA-ES evolution on a supercell of pristine Si bulk utilizing DFT and observe different meta-stable defect structures along the evolution trajectory. They utilize unsupervised learning to cluster all sampled structures to then relax a representative selection with DFT. To reproduce these results, we train a NeuralIL committee [48,49] on a subset of this trajectory data and use it to drive an evolution for 1000 generations on the same Si supercell containing 64 atoms, with the degrees of freedom restricted to a sphere with a cutoff radius of 4.0 Å. The initial step size was $\sigma^{(0)} = 0.1$ Å. The initial covariance matrix $\boldsymbol{C}^{(0)}$ was biased according to Ref. [18], such that atoms closest to the center of this sphere, the focus $\boldsymbol{x}_f$, are the most volatile

$$\boldsymbol{C}^{(0)} = \mathbb{I} + \text{diag}\left(c_r^2\right), \tag{4}$$

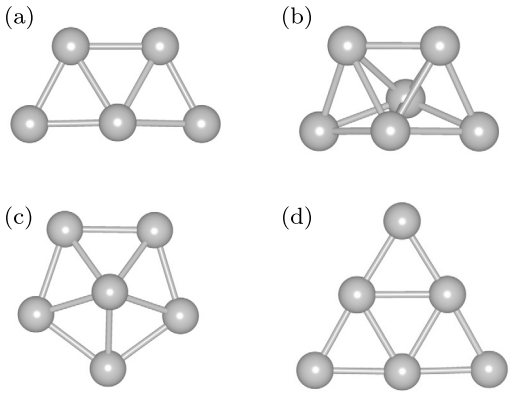


**Fig. 6.** Low-energy cluster configurations as identified via CMA-ES evolution without further local optimization. $Ag_5$ (a) and $Ag_7$ (b) starting from random positions within a sphere and evaluated with NWChem. $Ag_6$ in (c) and (d) evaluated with VASP and starting from PACKMOL-generated and uniformly-random sampled positions within a cube, respectively.

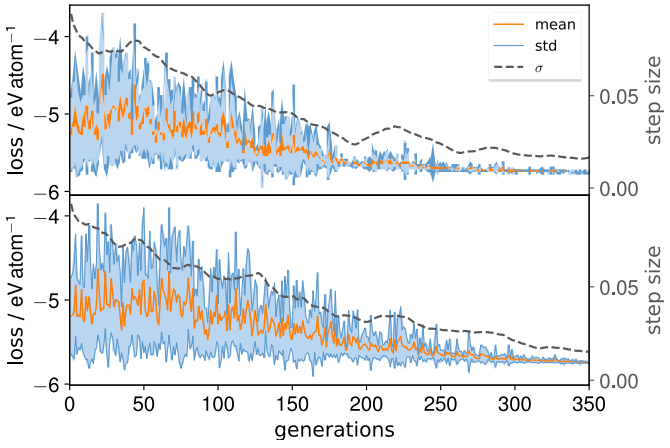


**Fig. 7.** Both figures show a trajectory (orange solid line) of the average loss, i.e. the energy per atom, of a population over 350 generations, for the Si bulk treated with a neural-network force field. The standard deviation within each generation is highlighted in blue and the CMA-ES step size $\sigma$ shown as a dark-gray dashed line. In the top panel the initial covariance matrix was biased by a factor $\propto r^{-2}$ and in the lower panel by a Gaussian.

$$c_r(i) = \frac{c_r}{\left[1 + ||\boldsymbol{x}_i^{(0)} - \boldsymbol{x}_f||_2 / (1\,\text{Å})\right]^2}. \tag{5}$$

We set the hyperparameter $c_r = 20.0$. With this, an evolution over 1000 generations completes in a matter of minutes utilizing a laptop and a single GPU. This statistic highlights the advantage of an accelerated machine-learning surrogate model with respect to direct DFT calculations for each configuration, a context in which energy evaluations overwhelmingly dominate the cost of the optimization process.

For comparison, with all other parameters unchanged, a second evolution run is performed with $\boldsymbol{C}^{(0)}$ biased by

$$c_r(i) = c_r \cdot \exp\left(-\frac{||\boldsymbol{x}_i^{(0)} - \boldsymbol{x}_f||_2^2}{2\sigma_{\text{bias}}^2}\right), \tag{6}$$

with $\sigma_{\text{bias}} = 1.25\,\text{Å}$ controlling the strength of the decay around focus $\boldsymbol{x}_f$. The trajectories of mean loss and step size $\sigma$ over the first 350 generations of both runs are shown in the top and bottom panels of Fig. 7, respectively. By running a fast local relaxation with the same trained NNFF and the FIRE [50] algorithm as implemented in the atomic simulation environment (ASE) [29], the extra step of clustering all sampled individuals can be forgone. Instead, the lowest-loss individual of each of the first 350 generations of the two runs is optimized. We see that both evolution runs contain structures that belong to loss basins of known defects, including the four-fold coordinated defect (FFCD) and Frenkel pair structures (see Fig. 8).

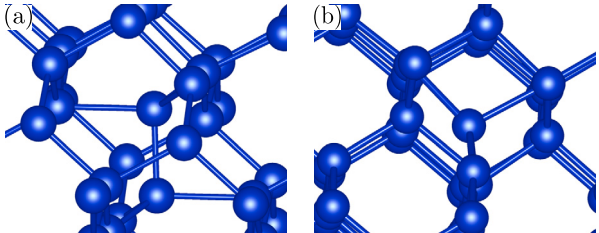


**Fig. 8.** Defects in stoichiometric Si supercells identified by local optimization of individuals along the evolution trajectory. (a) shows the FFCD configuration and (b) a Frenkel pair.

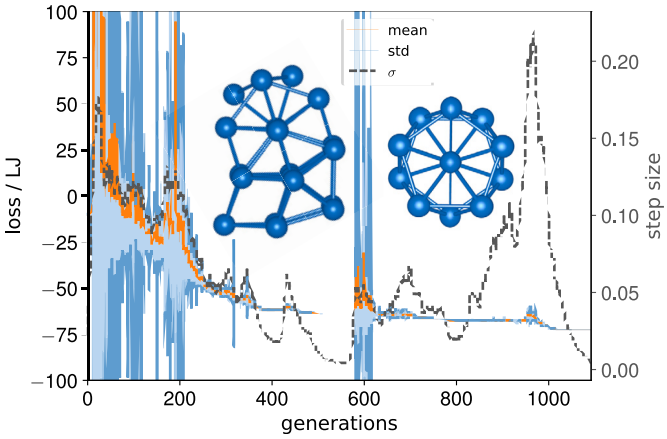


**Fig. 9.** Trajectory (orange solid line) of the average loss of a population over 1092 generations. The standard deviation within each generation is highlighted in blue and the CMA-ES step size $\sigma$ shown as a dark-gray dashed line. The y-axis focuses on a comparably narrow range to filter outliers and better illustrate the smaller oscillations in later generations. The structure on the left side is a local relaxation of generation 553 where the step size is at its lowest. The right-hand-side structure is the global optimum at generation 1092, without additional relaxation.

The two biasing functions BIAS_COVARIANCE_MATRIX_R and BIAS_COVARIANCE_MATRIX_GAUSS used in this example are included in UTILS.STRUCTURE_SETUP, along with additional functions utilizing ASE to calculate distances between atoms. The biased $\boldsymbol{C}^{(0)}$ is then transformed by calling SCIPY.LINALG.CHOLESKY [51].

### 4.4. Lennard-Jones cluster

LJ clusters [52] are some of the most studied model energy landscapes. They combine a formal simplicity and richness of features that make them ideal candidates for benchmarking optimization algorithms, including implementations of the CMA-ES [53,54]. In this work we use the 19-atom LJ cluster to illustrate the use of the implemented BIPOP restart [5,7] feature as described in section 2.2. For evaluating the LJ potential we repurpose code used in Ref. [55]. Reference values are taken from Ref. [52] and the implemented options for generating founder structures include PACKMOL [56].

We performed a number of evolution runs for different cluster sizes, varying algorithm settings, founder structures and restart parameters. The loss trajectory for one of these runs for an LJ cluster of 19 atoms is shown in Fig. 9. We chose this particular cluster because the number of degrees of freedom together with the rugged energy landscape pose a sufficiently difficult problem to necessitate a restart scheme. This spe-

cific evolution arrived at the global minimum [52] for population size $\lambda = 166$, step size $\sigma^{(0)} = 0.0201$ (in dimensionless LJ units) and random seed 45214. The widening of the standard deviation around generation 600 indicates that the algorithm managed to switch to a different loss basin. To further illustrate this, Fig. 9 shows two structures: the local relaxation of the best individual in generation 553 (left) and the global minimum in generation 1092 without further optimization (right). Upwards of generation 600 the best individuals predominately relax to the global minimum.

To get there, the BIPOP evolution performed ten restarts in total, with the highest population size for restart nine at $\lambda_{\text{large}} = 480$, i.e. the fourth of the large restarts. Overall, with the number of large restarts limited to five, one ninth (5/45) of the BIPOP evolution runs arrived at the global minimum. Restart strategies are especially useful when exploring multimodal loss landscapes as demonstrated here. The repeated searches start from different founder configurations and apply varied population- and step-size parameters, and greatly increase the diversity of visited structures by balancing exploration and exploitation [7].

## 5. Conclusions

We have presented Clinamen2, a versatile implementation of the Cholesky CMA-ES that provides convenient building block functions for applying the problem-agnostic core algorithm to various problems. Besides interfacing to, e.g., DFT codes or surrogate models like NNFFs, any Python function with a compatible signature may be used for loss evaluation and, therefore, to drive an evolution.

The GitHub repository contains example applications presented in this manuscript that - together with built-in utilities - enable the user to apply Clinamen2 to their specific use cases.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The latest version of Clinamen2, along with its documentation, is available at https://github.com/Madsen-s-research-group/clinamen2-public-releases. The version used for this manuscript is registered on Zenodo with https://doi.org/10.5281/zenodo.10143313. Both contain all the examples and the associated data files.

## Acknowledgements

This work was supported by the Austrian Science Fund (FWF) (SFB F81 TACO).

## References

[1] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 312–317.
[2] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195, https://doi.org/10.1162/106365601750190398.
[3] N. Hansen, The CMA evolution strategy: a tutorial, arXiv:1604.00772, Apr. 2016.
[4] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: 2005 IEEE Congress on Evolutionary Computation, Vol. 2, 2005, pp. 1769–1776, vol. 2, iSSN: 1941-0026.
[5] N. Hansen, Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed, in: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, GECCO '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 2397–2402.
[6] T. Suttorp, N. Hansen, C. Igel, Efficient covariance matrix update for variable metric evolution strategies, Mach. Learn. 75 (2) (2009) 167–197, https://doi.org/10.1007/s10994-009-5102-1.
[7] I. Loshchilov, CMA-ES with restarts for solving CEC 2013 benchmark problems, in: 2013 IEEE Congress on Evolutionary Computation, 2013, pp. 369–376, iSSN: 1941-0026.
[8] O. Krause, D.R. Arbonès, C. Igel, CMA-ES with optimal covariance update and storage complexity, in: Advances in Neural Information Processing Systems, Vol. 29, Curran Associates Inc., 2016.
[9] G. Arampatzis, D. Wälchli, P. Weber, H. Rästas, P. Koumoutsakos, $(\mu,\lambda)$-CCMA-ES for constrained optimization with an application in pharmacodynamics, in: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '19, Association for Computing Machinery, New York, NY, USA, 2019.
[10] Z. Li, Q. Zhang, Variable metric evolution strategies by mutation matrix adaptation, Inf. Sci. 541 (2020) 136–151, https://doi.org/10.1016/j.ins.2020.05.091.
[11] R. Ros, N. Hansen, A simple modification in CMA-ES achieving linear time and space complexity, in: Proceedings of the 10th International Conference on Parallel Problem Solving from Nature — PPSN X - Volume 5199, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 296–305.
[12] I. Loshchilov, A computationally efficient limited memory CMA-ES for large scale optimization, in: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 397–404.
[13] O. Krause, C. Igel, A more efficient rank-one covariance matrix update for evolution strategies, in: Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII, Association for Computing Machinery, 2015, pp. 129–136, Conference date: 17-01-2015 Through 20-01-2015.
[14] R. Biedrzycki, On equivalence of algorithm's implementations: the CMA-ES algorithm and its five implementations, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 247–248.
[15] N. Hansen, Y. Akimoto, P. Baudis, CMA-ES/pycma on Github, Zenodo, https://doi.org/10.5281/zenodo.2559634, Feb. 2019.
[16] N.E. Toklu, T. Atkinson, V. Micka, P. Liskowski, R.K. Srivastava, EvoTorch: scalable evolutionary computation in Python, arXiv preprint Https://arxiv.org/abs/2302.12600, 2023.
[17] R. Hamano, S. Saito, M. Nomura, S. Shirakawa, CMA-ES with margin: lower-bounding marginal probability for mixed-integer black-box optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 639–647.
[18] M. Arrigoni, G.K.H. Madsen, Evolutionary computing and machine learning for discovering of low-energy defect configurations, npj Comput. Mater. 7 (1) (2021) 1–13, https://doi.org/10.1038/s41524-021-00537-1.
[19] J. Blank, K. Deb, pymoo: multi-objective optimization in python, IEEE Access 8 (2020) 89497–89509.
[20] C. Igel, V. Heidrich-Meisner, T. Glasmachers Shark, J. Mach. Learn. Res. 9 (2008) 993–996.
[21] N. Khan, A parallel implementation of the covariance matrix adaptation evolution strategy, Tech. Rep. arXiv:1805.11201 [cs.math], May 2018, type: article, https://doi.org/10.48550/arXiv.1805.11201.
[22] M.K. Heris, CMA-ES in MATLAB, 2015.
[23] A.M. Vincent, P. Jidesh, An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms, Sci. Rep. 13 (1) (2023) 4737, https://doi.org/10.1038/s41598-023-32027-3.
[24] T. Sato, K. Watanabe, An evolutional topology optimization of electric machines for local shape modification and visualization of sensitivity distribution based on CMA-ES, IEEJ Trans. Electr. Electron. Eng. 18 (2) (2023) 286–293, https://doi.org/10.1002/tee.23721.
[25] E.R. Claussen, P.D. Renfrew, C.L. Müller, K. Drew, CMA-ES-Rosetta: blackbox optimization algorithm traverses rugged peptide docking energy landscapes, Tech. Rep., type: article, https://doi.org/10.1101/2022.12.19.521113, Dec. 2022.
[26] R. Wanzenböck, M. Arrigoni, S. Bichelmaier, F. Buchner, J. Carrete, G.K.H. Madsen, Neural-network-backed evolutionary search for srtio3(110) surface reconstructions, Digit. Discov. 1 (2022) 703–710, https://doi.org/10.1039/D2DD00072E.
[27] T.P. Baldão, M.R.O.A. Maximo, T. Yoneyama, Optimizing univector field navigation parameters using cma-es, in: 2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE), 2021, pp. 318–323.
[28] Y. Nagata, The lens design using the CMA-ES algorithm, in: K. Deb (Ed.), Genetic and Evolutionary Computation – GECCO 2004, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 1189–1200.
[29] 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, K.W. Jacobsen, The atomic simulation environment—a Python library for working with atoms, J. Phys. Condens. Matter 29 (27) (2017) 273002, https://doi.org/10.1088/1361-648X/aa680e.
[30] A.H. Larsen, M. Vanin, J.J. Mortensen, K.S. Thygesen, K.W. Jacobsen, Localized atomic basis set in the projector augmented wave method, Phys. Rev. B 80 (19) (2009) 195112, https://doi.org/10.1103/PhysRevB.80.195112.

[31] E. Aprà, E.J. Bylaska, W.A. de Jong, N. Govind, K. Kowalski, T.P. Straatsma, M. Valiev, H.J.J. van Dam, Y. Alexeev, J. Anchell, V. Anisimov, F.W. Aquino, R. Atta-Fynn, J. Autschbach, N.P. Bauman, J.C. Becca, D.E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G.N. Chuev, C.J. Cramer, J. Daily, M.J.O. Deegan, T.H. Dunning, M. Dupuis, K.G. Dyall, G.I. Fann, S.A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A.W. Götz, J. Hammond, V. Helms, E.D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B.G. Johnson, H. Jónsson, R.A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R.D. Lins, R.J. Littlefield, A.J. Logsdail, K. Lopata, W. Ma, A.V. Marenich, J. Martin del Campo, D. Mejia-Rodriguez, J.E. Moore, J.M. Mullin, T. Nakajima, D.R. Nascimento, J.A. Nichols, P.J. Nichols, J. Nieplocha, A. Otero-de-la Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R.M. Richard, P. Sadayappan, G.C. Schatz, W.A. Shelton, D.W. Silverstein, D.M.A. Smith, T.A. Soares, D. Song, M. Swart, H.L. Taylor, G.S. Thomas, V. Tipparaju, D.G. Truhlar, K. Tsemekhman, T. Van Voorhis, A. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K.D. Vogiatzis, D. Wang, J.H. Weare, M.J. Williamson, T.L. Windus, K. Woliński, A.T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, R.J. Harrison, Nwchem: past, present, and future, J. Chem. Phys. 152 (18) (2020) 184102, https://doi.org/10.1063/5.0004997.

[32] G. Kresse, J. Furthmüller, Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set, Phys. Rev. B 54 (16) (1996) 11169–11186, https://doi.org/10.1103/PhysRevB.54.11169.

[33] M. Rocklin, Dask: parallel computation with blocked algorithms and task scheduling, in: K. Huff, J. Bergstra (Eds.), Proceedings of the 14th Python in Science Conference, 2015, pp. 130–136.

[34] Dask Development Team, Dask: Library for dynamic task scheduling, 2016.

[35] G. Jastrebski, D. Arnold, Improving evolution strategies through active covariance matrix adaptation, in: 2006 IEEE International Conference on Evolutionary Computation, 2006, pp. 2814–2821, iSSN: 1941-0026.

[36] M. Preuss, Niching the CMA-ES via nearest-better clustering, in: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 1711–1718.

[37] I. Loshchilov, M. Schoenauer, M. Sebag, Alternative restart strategies for CMA-ES, in: C.A.C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, M. Pavone (Eds.), Parallel Problem Solving from Nature - PPSN XII, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 296–305.

[38] D. Mertz, Functional Programming in Python, O'Reilly Media, Inc., Sebastopol, 2015.

[39] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, W. Paul, M.I. Jordan, I. Stoica, Ray: a distributed framework for emerging AI applications, arXiv preprint Https://arxiv.org/abs/1712.05889, 2017.

[40] M.-P.V. Christiansen, N. Rønne, B. Hammer, Atomistic global optimization X: a Python package for optimization of atomistic structures, J. Chem. Phys. 157 (2022) 054701, https://doi.org/10.1063/5.0094165.

[41] M. Uhrin, S.P. Huber, J. Yu, N. Marzari, G. Pizzi, Workflows in AiiDA: engineering a high-throughput, event-based engine for robust and modular computational workflows, Comput. Mater. Sci. 187 (2021) 110086, https://doi.org/10.1016/j.commatsci.2020.110086.

[42] A. Jain, S.P. Ong, W. Chen, B. Medasani, X. Qu, M. Kocher, M. Brafman, G. Petretto, G.-M. Rignanese, G. Hautier, D. Gunter, K.A. Persson, FireWorks: a dynamic workflow system designed for high-throughput applications, Concurr. Comput., Pract. Exp. 27 (2015) 5037–5059, https://doi.org/10.1002/cpe.3505.

[43] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs, 2018.

[44] D.H. Ackley, The model, in: A Connectionist Machine for Genetic Hillclimbing, Springer, 1987, pp. 29–70.

[45] R. Fournier, Theoretical study of the structure of silver clusters, J. Chem. Phys. 115 (5) (2001) 2165–2177, https://doi.org/10.1063/1.1383288.

[46] S. Garg, N. Kaur, N. Goel, M. Molayem, V.G. Grigoryan, M. Springborg, Properties of naked silver clusters with up to 100 atoms as found with embedded-atom and density-functional calculations, Molecules 28 (7) (2023), https://doi.org/10.3390/molecules28073266.

[47] S. Manna, Y. Wang, A. Hernandez, P. Lile, S. Liu, T. Mueller, A database of low-energy atomically precise nanoclusters, Sci. Data 10 (1) (2023) 308, https://doi.org/10.1038/s41597-023-02200-4, Publisher: Nature Publishing Group.

[48] H. Montes-Campos, J. Carrete, S. Bichelmaier, L.M. Varela, G.K.H. Madsen, A differentiable neural-network force field for ionic liquids, J. Chem. Inf. Model. 62 (1) (2022) 88–101, https://doi.org/10.1021/acs.jcim.1c01380.

[49] J. Carrete, H. Montes-Campos, R. Wanzenböck, E. Heid, G.K.H. Madsen, Deep ensembles vs committees for uncertainty estimation in neural-network force fields: comparison and application to active learning, J. Chem. Phys. 158 (2023) 204801, https://doi.org/10.1063/5.0146905.

[50] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, P. Gumbsch, Structural relaxation made simple, Phys. Rev. Lett. 97 (17) (2006) 170201, https://doi.org/10.1103/PhysRevLett.97.170201.

[51] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 contributors, SciPy 1.0: fundamental algorithms for scientific computing in Python, Nat. Methods 17 (2020) 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[52] D.J. Wales, J.P.K. Doye, Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms, J. Phys. Chem. A 101 (28) (1997) 5111–5116, https://doi.org/10.1021/jp970984n.

[53] T. Schaul, T. Glasmachers, J. Schmidhuber, High dimensions and heavy tails for natural evolution strategies, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 845–852.

[54] C.L. Müller, I.F. Sbalzarini, Energy landscapes of atomic clusters as black box optimization benchmarks, Evol. Comput. 20 (4) (2012) 543–573, https://doi.org/10.1162/EVCO_a_00086.

[55] S. Bichelmaier, J. Carrete, G.K.H. Madsen, Evaluating the efficiency of power-series expansions as model potentials for finite-temperature atomistic calculations, Int. J. Quant. Chem. 123 (11) (2023) e27095, https://doi.org/10.1002/qua.27095.

[56] L. Martínez, R. Andrade, E.G. Birgin, J.M. Martínez, PACKMOL: a package for building initial configurations for molecular dynamics simulations, J. Comput. Chem. 30 (13) (2009) 2157–2164, https://doi.org/10.1002/jcc.21224.