Benchmarking Inverse Optimization Algorithms for Materials Design

Hanfeng Zhai[*†1], **Hongxia Hao**[‡2], **and Jingjie Yeo**[§1]

[1]Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, New York 14850, USA
[2]Microsoft Research AI4Science, Beijing, China

ABSTRACT

Machine learning-based inverse materials discovery has attracted enormous attention recently due to its flexibility in dealing with black box models. Yet, many metaheuristic algorithms are not as widely applied to materials discovery applications as machine learning methods. There are ongoing challenges in applying different optimization algorithms to discover crystals with single- or multi-elemental compositions and how these algorithms differ in mining the ideal materials. We comprehensively compare 11 different optimization algorithms for the design of single- and multi-elemental crystals with targeted properties. By maximizing the bulk modulus and minimizing the Fermi energy through perturbing the parameterized elemental composition representations, we estimated the unique counts of elemental compositions, mean density scan of the objectives space, mean objectives and frequency distributed over the materials' representations and objectives. We found that nature-inspired algorithms contain more uncertainties in the defined elemental composition design tasks, which correspond to their dependency on multiple hyperparameters. Runge Kutta optimization (RUN) exhibits higher mean objectives whereas Bayesian optimization (BO) displayed low mean objectives compared with other methods. Combined with materials count and density scan, we propose that BO strives to approximate a more accurate surrogate of the design space by sampling more elemental compositions and hence have lower mean objectives, yet RUN will repeatedly sample the targeted elemental compositions with higher objective values. Our work shed light on the automated digital design of crystals with single- and multi-elemental compositions and is expected to elicit future studies on materials optimization such as composite and alloy design based on specific desired properties.

Highlights

- Present a testbed that can implement many state-of-the-art optimization algorithms, and benchmarked 11 different optimizations spanning from Bayesian optimization to genetic algorithms, particle swarm optimization, etc. by evaluating multiple proposed metrics.

- Hybrid grey wolf optimization, hybrid improved blue whale optimization, and genetic algorithms generally display higher variances from repeated experiments for the unique elemental count, objective space density scan, and mean objective values. Bayesian optimization and Runge Kutta optimization generally show lower variances.

- Runge Kutta optimization exhibits higher mean objective values and Bayesian optimization shows lower mean objectives, compared with other optimizations benchmarked.

- By combining estimations from materials count and normalized frequency, one deduces Bayesian optimization aims to approximate a more accurate surrogate of the design space and hence sample more elemental compositions, whereas Runge Kutta optimization tends to repeatedly sample the discovered optimal materials multiple times.

1 Introduction

Designing materials with desired tailored properties has always been a key goal in human society, from ancient metallurgy to modern nanotechnology. Traditionally, most research conducted in the field of designing crystals with multi-elemental compositions relies on costly experimental synthesis and time-consuming trial-and-error processes. Advances in computational engineering can greatly accelerate materials design efficiency. To achieve the design goal, many optimization methods were developed for the design process. These methods can be categorized as first and zeroth orders, where first-order optimization

---

[*]Email: hz253@cornell.edu
[†]Present Address: Department of Mechanical Engineering, Stanford University
[‡]Email: hongxiahao@microsoft.com
[§]Email: jingjieyeo@cornell.edu

methods are gradient-based methods, such as gradient descent and Newton-Raphson, that rely on the problem being first-order differentiable. First-order methods are widely applied to topology optimization coupled with finite element methods for mechanical design, including classical problems like the design of beams[1] and trusses[2]. However, the primary drawback of first-order optimization is that the differential form of the problems may not exist, or be easily obtained, in many (or most) real-world application scenarios. In many cases, the governing equations for the problem do not exist as differentiable forms. Moreover, first-order optimization tends to get "trapped" in local optima close to the initial design which heavily relies on prior physical understanding of the problem to ameliorate. Hence, metaheuristic optimization methods, such as genetic algorithms and particle swarm optimization, are more flexible and adaptive to complex problem domains. These characteristics enable metaheuristics to be applicable to multiscale materials design problems, such as molecular simulations, machine learning surrogates, and density functional theory calculations. For example, molecular dynamics (MD) simulations model the interactions of particles based on interatomic potentials[3,4], DFT simulations calculate the energy based on pseudo-potentials for the Hamiltonian[5], and ML surrogates represent the material's structure-property map with a neural network[6]. In these situations, obtaining first-order derivatives is either an extremely taxing task or may even be intractable. To overcome these challenges, recent literature highlights the utility of metaheuristics for elemental composition design, in contrast to gradient-based optimizations. For example, Ramprasad and coworkers developed genetic algorithm methods coupled with deep learning surrogate models for designing polymer dielectrics[7] and used similar strategies to design polymers with high glass transition temperatures and bandgaps[8]. Winter et al. combined machine learning surrogate with particle swarm optimization (PSO) to design drugs[9]. Weiel et al. used PSO to select parameters in molecular dynamics simulations[10].

Many numerical methods exist for design optimization and two of the most widely applied methods are Bayesian optimization (BO) and deep reinforcement learning (DRL). BO leverages Gaussian process regression to fit a surrogate model of the design space and leverages Bayesian statistics to iteratively improve the accuracy of such a surrogate by sampling more points via an acquisition function. BO is frequently applied in materials design. For example, Shin et al. combined micromechanical experiments and BO to design spiderweb nanomechanical resonators[11]. Chen and coworkers developed a series of BO frameworks for mechanical design and elemental composition design using finite element simulations[12], molecular simulations[13,14], and machine learning surrogate models[15,16]. Zhai and Yeo combined BO and individual-based modeling of bacteria to design antimicrobial nanosurfaces[17]. In another popular method, DRL utilizes a deep neural network (DNN) as an agent to interact with the environment to receive rewards from different output properties and learn the best strategy to achieve the design goal. When applying DRL for materials design, the evaluations of the material properties are treated as the environment for the agent to learn the correct policy to achieve the design goal. For instance, Gu and coworkers developed a DRL framework for designing tough graphene oxide[18] and continuum composite materials[19]. Farimani and coworkers developed DRL frameworks for porous graphene for water desalination[20].

Can these optimization methods be uniformly applied to the same problem and how do they differ? To answer these questions, benchmark studies are urgently needed. One particular reason for the urgency is efficiency. Optimization methods are intended to improve the efficiency of the design problems. However, when companies or research groups want to employ design optimization for their problems of interest, it is of utmost importance to select optimization methods that are best suited to their specific problems. Selecting ideal optimization methods from the massive amounts of existing literature is non-trivial: one may fall into the "trial-and-error" trap and the efficiency of solving the problem is significantly reduced. Moreover, ML-based studies for materials design are often focused on either large-scale molecular structures (i.e., polymers) or continuum structures. This is in part because it is easier to extract structure-property relationships for larger structural bodies in the context of continuum modeling, where first-order derivatives are easy to obtain. Specifically for materials composed of multiple elements, many studies leverage molecular graphs or use natural language processing to find this structure-property map. In contrast, while there are some studies on the design of small molecules using language representation with generative models[21], a huge gap exists in applying general optimization methods for designing such multi-elemental compositions. Therefore, following our previous discussion, the specific problem to be posed herein is how different optimization methods differ in the context of design optimization.

To address these challenges, we propose and conduct a benchmark study by designing a framework to implement most state-of-the-art optimization methods. We benchmark many widely employed metaheuristic optimizations, including Bayesian optimization, genetic algorithms, particle swarm optimization, and simulated annealing. We further compared novel optimization methods, such as the arithmetic optimization algorithm and Runge Kutta optimization, for a total of 11 different optimization algorithms. We took advantage of the large numbers of open-source software for computational elemental composition design, including pymatgen[22], Materials Project[23], matminer[24], and mealpy[25]. The material evaluations are enabled by surrogate models: pretrained graph neural network models to predict the elemental compositions' properties based on input graph structure. The general schematic of our study is shown in Figure 1. We begin with inspiration from the general premise of industrial or fundamental research problems: one needs to design elemental compositions based on specific applications. Given this application, the targeted physical properties must be identified based on the design requirements, such as high
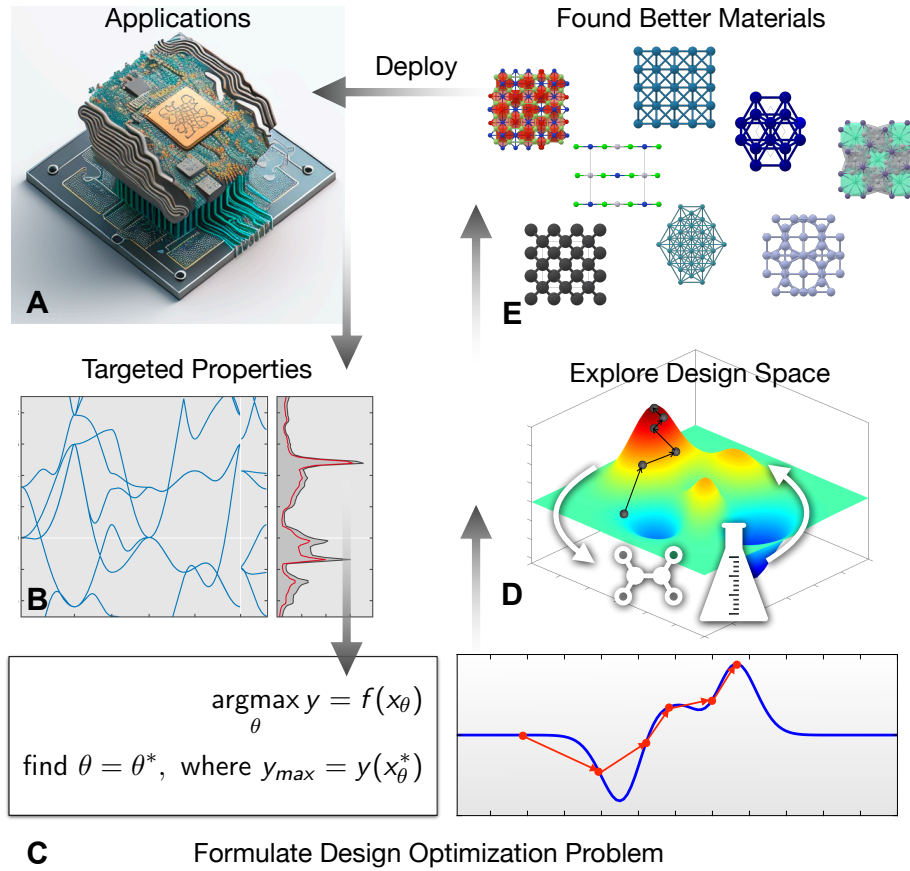
Figure 1. The general overview of the materials design process. **(A)** Schematic for defining the intended industrial application. **(B)** The process of identifying the desired material properties such as high bulk modulus and low Fermi energy. **(C)** The schematic for formulating a digital design optimization problem using numerical models to discover or design elemental composition with the desired properties. **(D)** Different optimization methods use exploration/exploitation strategies to generate a design space for exploring the best materials. **(E)** The optimal elemental compositions are extracted from the design space exploration and are ready for real-world applications. In this study, we focus on formulating an optimization framework that enables the application of the most current popular optimization methods to search for desired elemental compositions and compare different optimizations based on specific metrics. Finally, the "mined" elemental compositions are analyzed.

toughness or low thermal conductivity. Based on these targeted applications and properties, the design problem can then be formulated mathematically together with the corresponding optimization algorithms. The design space can be explored to extract the optimal elemental compositions from the materials database(s). Here, we will mainly focus on the optimization process (Figure 1 **C**, **D**, & **E**) to propose simplified target material properties and focus on the algorithmic implementations and benchmarking. The significance of this work lies in that there is a lack of studies that evaluate the performance of different optimization methods, in contrast with the abundance of published literature that applies specific optimization methods for tailored materials design tasks. Our study may benefit future research in ways such as: (1) Researchers can select different algorithms that correspond well with their potential applications based on our evaluation results; (2) Researchers can use or refer to our framework and benchmark other optimization algorithms of their interests. Our major contributions are to (1) propose a framework that can implement different algorithms, which generalizes the materials' inverse design tasks, (2) evaluate different optimization methods based on defined metrics, and (3) highlight ideal algorithms for different potential applications based on our evaluations. Some results were unexpected: the commonly employed BO did not stand out in terms of the mean objective evaluation, while surprisingly RUN outperformed on both the stability and objective evaluations.

This paper is articulated as follows. We present an overview of the underlying methodology, describe the workflow of connecting machine-learned models of materials properties with different optimization methods, and benchmark the results obtained from 11 different optimization algorithms in Section 3. We then discuss the algorithms that are developed to perform the automated optimizations in Section 4. A detailed explanation of the methodologies employed is presented in Section 2 and further information is contained in the ESI.
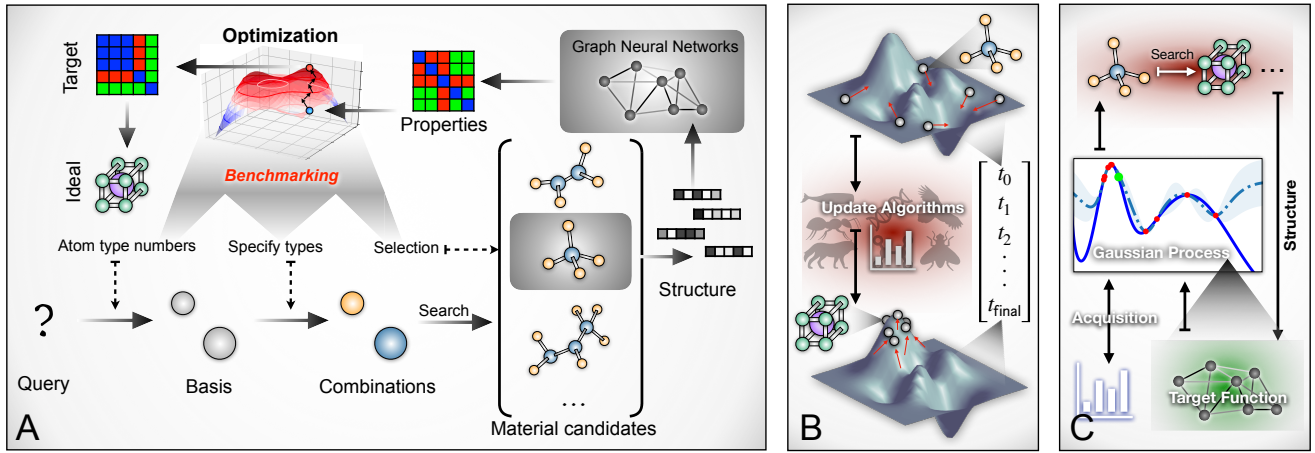

Figure 2. Schematic for optimization benchmarking and different optimization algorithms employed in the study. **(A)** Workflow for discovering elemental compositions by combining graph neural network (GNN) surrogate models with defined design variable randomization to select optimal elemental compositions with targeted material properties. See Section 2 & ESI for details. **(B)** Schematic for optimization through metaheuristic algorithms, starting with a set of random particles (or population) sampled in the design space, to iteratively update and eventually cluster around globally optimal values, indicating the search for elemental compositions with desired properties. **(C)** Bayesian optimization methods for elemental composition discovery, seeking to build a surrogate model of the design space by employing Gaussian process regression and searching for the next material evaluations based on an acquisition function. Optimal elemental compositions can be extracted from the design space surrogate.

2 Methods

2.1 Inverse Optimization Framework

2.1.1 Problem Formulation

Our goal is to benchmark design optimization algorithms for inverse elemental composition design, in which we formulate the problem as maximizing an objective function corresponding to materials properties. We benchmark using a model problem of maximizing the bulk modulus while simultaneously minimizing the Fermi energy. While not specific to any applications, our prototype model is useful for benchmarking and unveiling characteristics of different optimization algorithms for elemental composition design and discovery. To maximize an objective function, $\mathscr{J}$, as a single-objective optimization problem which is

defined as the difference between the bulk modulus and the Fermi energy, $\mathcal{J} = K - E_{Fermi}$:

$$
\begin{aligned}
\underset{n_{atom},\,\xi_n,\,\eta}{\arg\max} \quad & \mathcal{J} = K - E_{Fermi}, \\
\text{where} \quad & K, E_{Fermi} = MEGNet(\mathcal{G}_{\Theta}), \\
\rightarrow & \mathcal{G} = \Omega(n_{atom},\,\xi_n,\,\eta);\ \Theta = [n_{atom}, \xi_n, \eta] \\
\text{subject to} \quad & n_{atom} \in [1,4] \text{ or } \equiv 1,\ \xi_n \in [0,100],\ \eta \in [0,100]
\end{aligned}
\tag{1}
$$

Here, the problem is defined as a discrete constrained optimization problem. The input graph structure $\mathcal{G}$ is related to the properties K and E_{Fermi}, which can be considered as a forward mapping. The forward mapping of fast properties predictions is enabled based on the graph neural network structure MatErials Graph Network (MEGNet)[26]. Further details are provided in Section 2 & ESI.

The optimization problem can be solved using various optimization algorithms. Here, we explored 11 different state-of-the-art optimizations to conduct benchmarking for materials discovery. The methods include Bayesian Optimization[27], Deep Reinforcement Learning[28], Genetic Algorithm[29], Particle Swarm Optimization[30], Hybrid Grey Wolf Optimization[31], (Hybrid-Improved) Whale Optimization Algorithm[32], Ant Colony Optimization[33], Differential Evolution[34], Simulated Annealing[35], Bees Optimization[36], Arithmetic Optimization[37], and Runge Kutta (RUN) Optimization[38]. These algorithms fall into two main categories: machine learning-based optimizations and metaheuristic optimizations (Figure 2 **B** & **C**). We use different strategies to perturb Θ to maximize $\mathcal{J}$, which will be elaborated in Section 2 & ESI. Here, we consider single and multi-element composition optimization scenarios. For single-element composition optimization, the design variable Θ is restricted to 1, and there are only two design variables ξ_n and η. For multi-element composition optimization, the screened elemental compositions contain $n_{atom} \in [1,4]$. The materials libraries as indicated in Figure 2 consist of a series of MP IDs and the randomly generated η is used to select the corresponding MP ID.

Following the discussion from Equation (1), the optimization is benchmarked and connected through the material evaluations. We utilize MEGNet for materials properties evaluation mainly for two considerations: (1) Fast inference. By using the pretrained MEGNet model (mp-2019.4.1), the properties can be obtained "on the fly" given the input graph structure, getting rid of the lengthy computational process via traditional numerical methods like molecular dynamics (MD) or density functional theory (DFT). (2) Better screen-ability of the design space. Given the pretrained MEGNet model, one can infer the corresponding properties given generated input graph $\mathcal{G}$ and hence allowing the evaluation of a large amount of "untouched" materials. On the other hand, due to the lack of existing data for the properties of many potential materials, one may not directly obtain the properties given the input elements just from a query. Also, due to the lack of pseudo- and/or interatomic potentials, MD or DFT calculations may not be conducted. Here, we employ MEGNet due to its reported accuracy and stability. As reported by Chen et al.[26], the MAE for the prediction for the Fermi energy and bulk modulus are around 0.028 and 0.05, respectively. MEGNet is not the only option for materials properties inference, other existing frameworks with competitive performance include CGCNN[39], MODNet[39], H-CLMP(T)[40], and Crystal graph attention networks[41]. Since our focus is on the optimization methods, we do not emphasize the surrogate model evaluations. Notably, many recent works focus on materials property prediction benchmarks[42, 43].

For this problem, the inverse optimization problem is to directly perturb the graph structure such that the materials' properties can be tailored. However, it is difficult to represent different materials' graph structures containing different elemental numbers in the same dimensions for optimization. Hence, we parameterize the graph structure $\mathcal{G} = \mathcal{G}_{\Theta}$ so that the algorithms tune the design variables $\Theta = [n_{atom}, \xi_n, \eta]$ to change the elemental composition $\mathcal{G}_{\Theta}$. Here, as visualized in Figure 2 **A**, the optimization begins with variable n_{atom}, as the number representing different types of elements in the elemental compositions. We may denote the different atoms in the elemental compositions as the *atomic dimensions*. Follows is a random number distributed in $[0,100]$ as ξ_n to assign atomic types to the atomic dimensions, which we may define as the *material basis*. For example, if the atomic dimension $n_{atom} = 3$ we may obtain an element basis from ξ_n like {C-H-O}. Given this material basis, one can generate a list of potential elemental compositions from the *Materials Project*, for which we denote as the *materials library*. One can then use η to select the candidate elemental compositions from the materials library. If following our previous example, the materials library may look like {CH_3O, C_2H_6O, C_3H_6O, ...} and our selected material may be C_2H_6O. For how ξ_n and η determine the atomic graph structure $\mathcal{G}$ in the algorithm, please refer to *Extension on Problem Formulation* in ESI.

2.1.2 Workflow Automation

One of the main contributions of our work is the development of an optimization framework that is flexible for many implementations of state-of-the-art optimization methods. The key point in such a framework is the automation of the optimization, where the update of the next set and the current set of the materials evaluation are connected. The automation is enabled through the graph neural network surrogate model as formulated in Equation (1), the MEGNet. The whole workflow
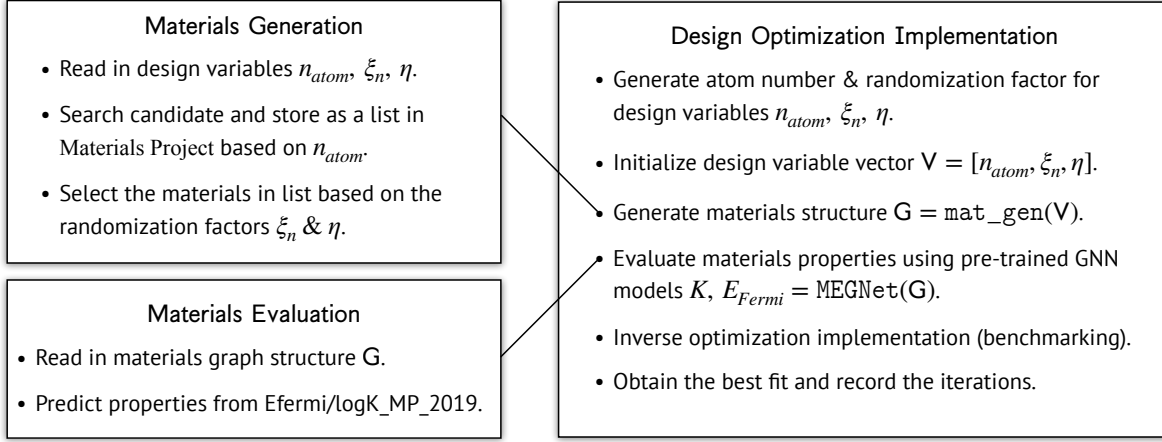
Figure 3. Algorithmic flow chart for the optimization framework, corresponding to Equations (1) & (2) and Figure 2.

can be simplified to the expression:

$$\overbrace{\underbrace{(MEGNet(\Theta_{n+1})}^{\text{evaluation}} \leftarrow \Theta_{n+1})}_{\text{exploration}} \underset{\text{feedback}}{\overset{\text{search}}{\rightleftharpoons}} \underbrace{IOA(\Theta_n, \overbrace{MEGNet(\Theta_n)}^{\text{evaluation}}; \mathbf{P}_{IOA})}_{\text{exploitation}} \tag{2}$$

where *IOA* stands for "inverse optimization algorithms". Assumes the algorithm evaluates the nth set of elemental compositions with the design variable Θ_n, obtaining the corresponding properties $MEGNet(\Theta_n)$, by employing the hyperparameters $\mathbf{P}_{IOA}$ specified by from different optimization methods, one then obtain the design variables for the next set of evaluation Θ_{n+1}, for which the evaluated properties $MEGNet(\Theta_{n+1})$ then feeds back to the IOA and iteratively search for the next set. This loop allows one to get rid of the traditional "trial-and-error" approach and only need to set the hyperparameters to obtain the optimal elemental compositions given the targeted properties.

The algorithmic flowchart is visualized in Figure 3 (Equations (1) & (2)). The formulated basic optimization method is shown in the right block, while the materials generation and evaluation processes are laid out in the left blocks. In our approach, the algorithm is initialized and repeated based on different random seeds for the five attempts in Figure 5. The hyperparameters for each algorithm are provided in the ESI and were selected from commonly used default values for each algorithm.

2.2 Optimization Algorithms

2.2.1 Bayesian Optimization

The core of machine learning (ML)–based optimization algorithms is to employ the explored materials data to fit a surrogate assisting the optimization process. Here, there are two main ways of adopting ML methods for design optimization: (1) building up a surrogate model of the design space for final exploration, namely the Bayesian optimization method; (2) utilizing a deep neural network (DNN) to learn the interactions of input elemental compositions and their corresponding material properties, which can be considered as the interaction of actions and their environment in which this DNN is considered as an "agent". This method is known as deep reinforcement learning. The detailed mathematical formulation and derivations are shown in the *Machine Learning Based Algorithms & Metaheuristic Optimization* in ESI.

For approach (1), following the formulated problem in Equation (1), we now assume the optimization algorithm is initiated by a set of m materials evaluations from MEGNet: $K_1, K_2, K_3, ..., K_m, (E_{\text{Fermi}})_1, (E_{\text{Fermi}})_2, (E_{\text{Fermi}})_3, ..., (E_{\text{Fermi}})_m = MEGNet(\Omega(\Theta_1), \Omega(\Theta_2), ..., \Omega(\Theta_m))$; one then obtains data lies in the mapping of $M : \Theta \rightarrow \mathscr{J}$, in which we assume M exist and continuous. Here, this M represents the design space, i.e., the projection from the input design variables to the objectives. M can be approximated using Gaussian process regression (GPR) from the generated data in the materials evaluation, which can be simplified to the form:

$$(\mathscr{J}_1, \mathscr{J}_2, \mathscr{J}_3, ..., \mathscr{J}_m) = GP(\Theta_1, \Theta_2, ..., \Theta_m; \mathbf{p}_{GP}) \tag{3}$$

where $\mathbf{p}_{GP}$ are all the hyperparameters involved in setting up the regression model, elaborated in detail in the ESI. Given a surrogate model of the design space built from GPR, one can improve the accuracy of such a surrogate by iterative sampling more points. The core of BO here would be the strategies to sample the next set of material evaluations from Θ.

The next set of design variables Θ_{next} is selected via maximizing the acquisition function $\mathscr{A}$, writes:

$$\Theta_{next} = \arg\max \mathscr{A}(\Theta_1, ..., \Theta_m, \mathscr{J}_1, ..., \mathscr{J}_m; \mathbf{p}_{Aq}) \tag{4}$$

where we use the upper confidence bound as the acquisition function (see *Bayesian Optimization* in ESI). BO utilizes GPR and acquisition function to efficiently sample the design space for a more accurate approximation towards the formulated optimization goal (max $\mathscr{J}$).

2.2.2 Metaheuristic Optimizations

Metaheuristic optimization is a family of optimization techniques that are used to find the optimal solution by exploring a large search space. They are often inspired by natural processes or social behavior, such as animal behavior (e.g., ant, whale, wolf), physical process (i.e., annealing of metals), biological processes (i.e., gene mutation), or fundamental mathematics.

Metaheuristic algorithms are typically applied to complex optimization problems where traditional methods are either not feasible or inefficient. They are highly flexible and can be adapted to a wide range of problem domains, in which elemental composition design is our target scenario. Metaheuristics are often considered "black box" optimization methods, as they do not require knowledge of the problem structure. More specifically, they do not require the existence of the first-order differentiation form of the governing equations of the problem.

If we assume the simplest form, one can simplify and interpret the metaheuristic optimization process by sampling a set of random particles in the design space and the optimization process refers to the update of these sampled particles (For the generalization of this form see *Methodology Generalization* in ESI). The update rules may be inspired by different natural processes, corresponding to the optimization methods employed. If one assumes there are a total of d dimension of the problem domain, and denotes the rule as $\mathscr{R}$, the update rules for the ith particle write:

$$X_i^d \leftarrow X_i^d + \mathscr{R}(X_i^d, X_j^d, \mathbf{p}_{MH}) \tag{5}$$

where X is the location of the particle, and the ith particle's location may be related to it adjacent particle(s) j. $\mathbf{p}_{MH}$ denotes the hyperparameters employed in the metaheuristic optimizations. We here only present the generalized form to identify the key essence of metaheuristics (details of each algorithm see *Metaheuristic Optimization* in ESI).

3 Results

3.1 Elemental Composition Design Benchmarks

Figure 4 shows the benchmarks of the 11 different inverse optimization algorithms (IOAs) and their objective properties explorations (Column **A**), the distribution of the material evaluation numbers with the corresponding mean values (Column **B**), and the distributions of scanned density with overall mean values (Column **C**), for both the single (Row **1**) and multi-element (Row **2**) elemental composition design. In the objective distributions, different IOAs are marked in different colors, and the corresponding optimization methods are abbreviated. Here, the materials count represents the total number of unique elemental compositions scanned by the different IOAs. The mean densities of objective spaces are computed to estimate the repetitive evaluations of the same elemental compositions by computing the mean value of the density scan space (see Section 2 & ESI). Both metrics help us understand the "strategy" of different algorithms when they scan the chemical design space to search for the ideal elemental compositions.

There are five major findings from Figure 4. (1) Comparing **A1** and **A2**, the scanned objective spaces are smaller for GWO compared with other optimization methods for both the single and multi-element composition design cases. For the single-element composition design case: (2) the evaluated elemental composition numbers are similar for most methods, whereas SA and AOA are observably smaller. Both GWO and SA display relatively high variances compared with other methods. (3) The mean scanned densities are relatively higher for SA and AOA methods, with both displaying higher variances. Both SA and AOA possess higher variances. For the multi-element composition design cases: (4) RUN has the lowest evaluated elemental composition numbers. GWO and BWO have the highest variances. (5) For the mean scanned densities for multi-element compositions, GWO and RUN have the highest evaluated elemental composition numbers and variances.

From Figure 5, we can draw five key observations. (1) When comparing **A1** and **A2**, the mean objective distributions differ for the single-element and multi-element composition design cases. (2) For single-element design, GA, SA, and RUN display high mean objective values. SA and RUN also exhibit relatively low variances. GWO shows relatively high mean objectives but with higher variances, while BO demonstrates a stable performance with a low mean objective value and low variance. (3) DE and AOA exhibit repetitive scans in the same composition region, indicating a tendency to get "trapped" in a local space. This aligns with their relatively lower mean objectives in **A1**. Moving on to multi-element design, RUN achieves the highest mean objective values with low variances. GA, GWO, and BWO show higher variances. (4) AOA and SA again exhibit repeated scanning of the same elemental compositions, suggesting a tendency for optimization methods to get trapped in local

Optimizations	MC (Single-element)	MDS (Single-element)	MC (Multi-element)	MDS (Multi-element)
BO	0.2	6.5×10^{-2}	104.3	10.1×10^{-2}
GA	0.3	42.9×10^{-2}	179704.5	101.7×10^{-2}
PSO	1.7	28.9×10^{-2}	12575.7	7.1×10^{-2}
GWO	14.3	13.3×10^{-2}	771728.3	1019.3×10^{-2}
BWO	0.7	26.0×10^{-2}	402679.3	131.2×10^{-2}
ACO	0	9.7×10^{-2}	427.3	15.8×10^{-2}
DE	0	79.6×10^{-2}	73394.7	56.8×10^{-2}
SA	45.2	3576.7×10^{-2}	19843.8	12.5×10^{-2}
BOA	0.2	8.4×10^{-2}	29907.2	5.2×10^{-2}
AOA	5.8	1156.6×10^{-2}	150198.5	44.1×10^{-2}
RUN	0	1098.7×10^{-2}	27895.5	218.3×10^{-2}

Table 1. The variances for the five repeated experiments of different optimization algorithms for single- and multi-element composition design, expressed with single-digit precision. Note that MC denotes "Materials Count", and MDS stands for "Mean Density Scan". The four columns correspond to Figure 4 **B** & **C**. For better presentation, we rescale the variances of MDS 100 times larger (indicated in the form $\times 10^{-2}$).

Optimizations	MO (Single-element)	Vr (Single-element)	MO (Multi-element)	Vr (Multi-element)
BO	91.9	58.7×10^{-2}	84.5	57.5×10^{-2}
GA	97.2	1097.1×10^{-2}	91.0	3726.5×10^{-2}
PSO	93.0	90.7×10^{-2}	84.0	8.8×10^{-2}
GWO	96.8	580.6×10^{-2}	89.8	3816.3×10^{-2}
BWO	94.9	838.0×10^{-2}	91.0	3719.7×10^{-2}
ACO	92.5	148.3×10^{-2}	84.0	307.5×10^{-2}
DE	94.2	557.5×10^{-2}	89.2	593.5×10^{-2}
SA	97.6	81.5×10^{-2}	82.6	145.8×10^{-2}
BOA	93.9	404.5×10^{-2}	87.3	314.1×10^{-2}
AOA	94.7	840.2×10^{-2}	83.6	523.2×10^{-2}
RUN	96.7	429.3×10^{-2}	92.8	166.6×10^{-2}

Table 2. The mean objective values and variances for the five repeated experiments of different optimization algorithms for single- and multi-element composition design, expressed with single-digit precision. Note that MO denotes "Mean Objective", and Vr stands for "Variances". The four columns correspond to Figure 5 **A**. For better presentation, we rescale the variances of MDS 100 times larger (indicated in the form $\times 10^{-2}$).
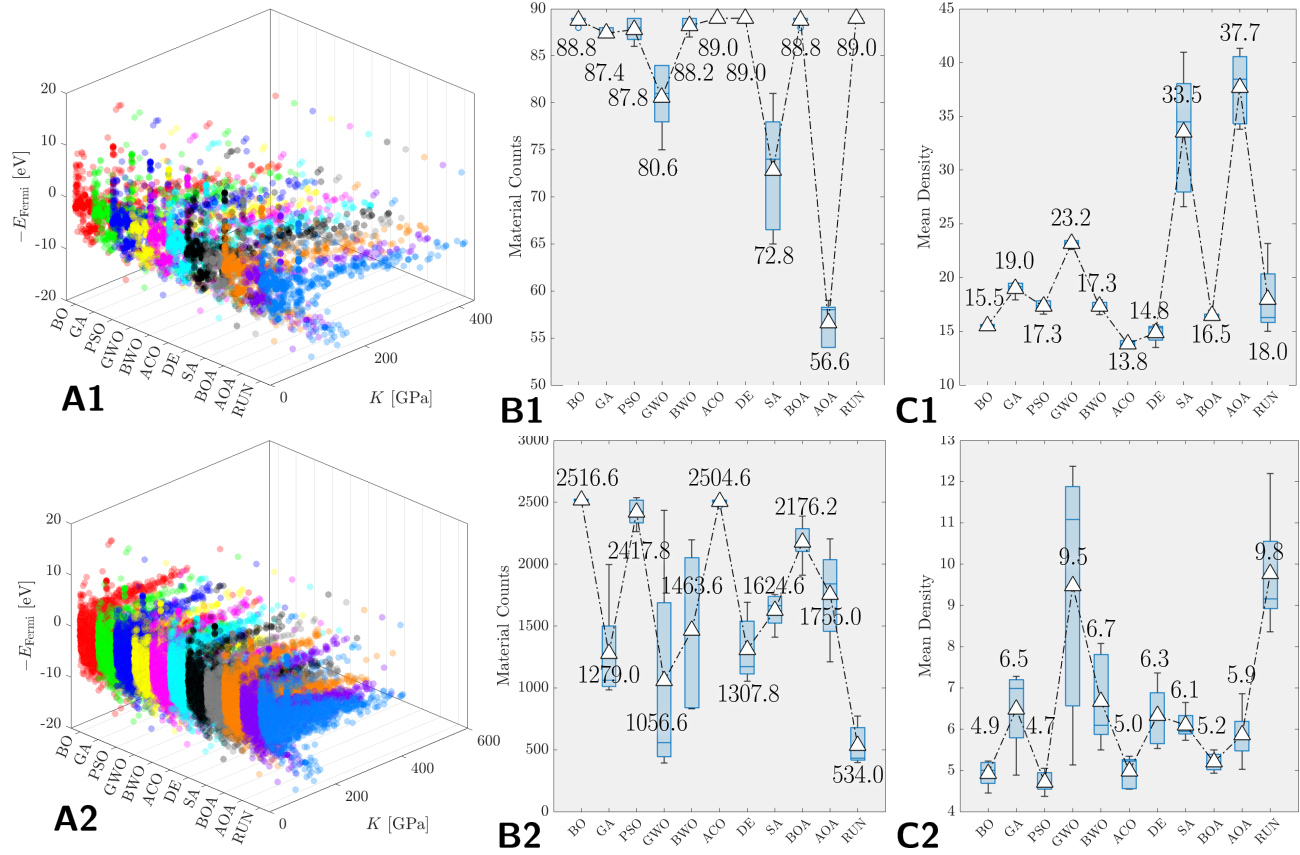
Figure 4. Objective properties exploration, objective value evaluation, and targeted design space count for benchmarking different optimization methods for single and multi-element composition design. **(A1)** Overall design space benchmarking for the 11 different optimization schemes used, marked in different colors for the single-element composition design (For the projections in 2D, please refer to Figure S6). **(B1)** The distribution (for the 5 repeated experiments) and mean values of the overall material counts during the evaluation processes. **(C1)** The distribution (for the 5 repeated experiments) and mean values of the scanned material evaluation densities (see Section 2 & *Density Scan* in ESI for the calculation of "Mean Density") during the evaluation processes. (Related details can be found in the analysis of Figure S7). **(A2)** Overall design space benchmarking for the multi-element composition design (refer to Figure S6 for details). **(B2)** The distribution and mean values of the overall material counts. **(C2)** The distribution and mean values of the scanned material evaluation densities. The white triangular dots are the mean values for the 11 evaluations in both **(B)** & **(C)**. The corresponding numbers are marked on top of the bar plot for both **(B)** & **(C)**. BO, GA, PSO, GWO, HIWOA, ACO, DE, SA, BOA, AOA, & RUN denote Bayesian Optimization, Genetic Algorithm, Particle Swarm Optimization, Hybrid Grey Wolf Optimization, Hybrid-Improved Whale Optimization Algorithm, Ant Colony Optimization, Differential Evolution, Simulated Annealing, Bees Optimization, Arithmetic Optimization, and Runge Kutta Optimization, respectively.
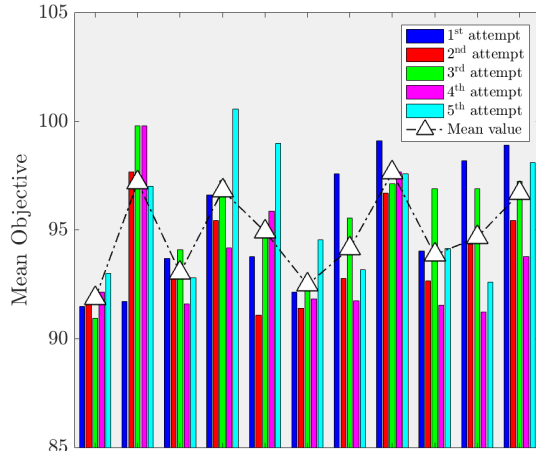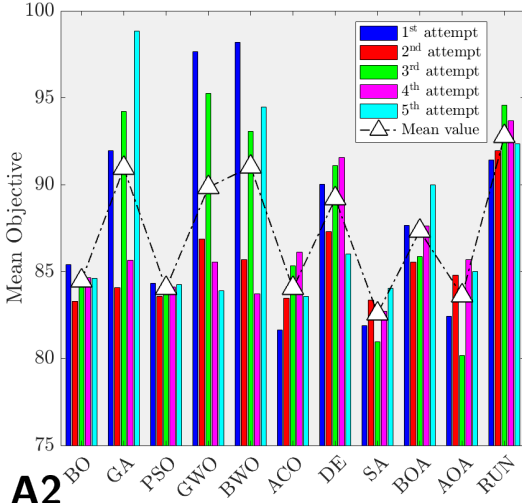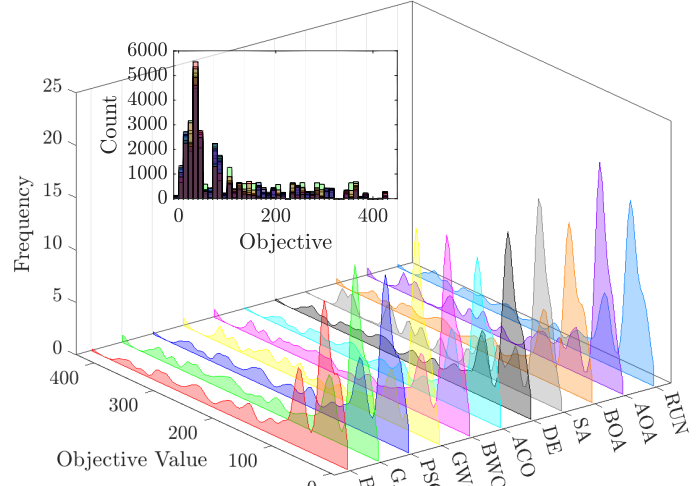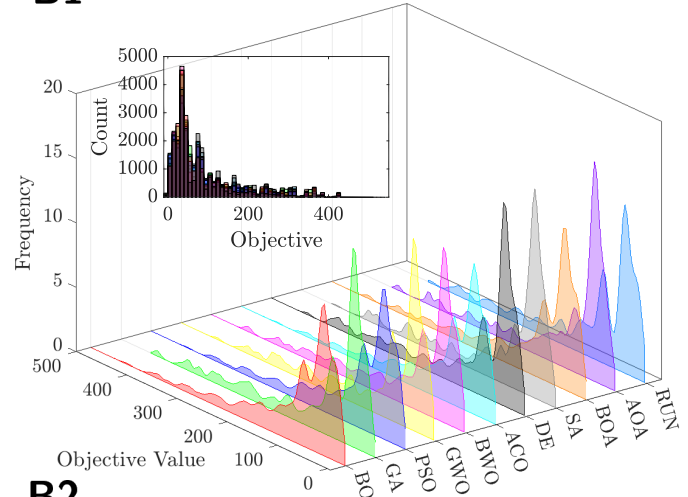
Figure 5. The statistical evaluations of the optimization processes. **(A1)** The distribution of the mean objective values (from 5 attempts) for the 11 evaluated optimization algorithms for single-element composition cases. Different colors mark different attempts, as indicated in the legend in the top-right corner of the figure. The white triangular dots stand for the mean values of the five attempts. The "Mean Objectives" are calculated as $\frac{\sum(\text{objectives})}{\text{attempts}}$ per optimization attempt (under the same random seed). **(B1)** The frequency distributions (i.e. repeated evaluations) using kernel fits of different objective values for the 11 different optimization algorithms (Details see Section 2 & *Additional Results* in ESI). Note that the original data distribution is shown in the top left corner of the subfigure. **(A2)** The distribution of the mean objective values for the 11 evaluated optimization algorithms for multi-element composition cases, where the white triangular dots are the mean values. **(B2)** The frequency distributions of different objective values for the 11 optimization algorithms.

composition spaces. (5) When considering both single and multi-element cases, different optimization algorithms demonstrate similar elemental composition search strategies, as evidenced by similar frequency distributions.

To further quantitatively support our observations on the uncertainties of different optimizations for repeated elemental composition design experiments in Figures 4 & 5, Tables 1 & 2 show the variances among different experiments per different optimizations. Both Table 1 & Figure 4 show that the materials count variances are very different for single- and multi-element compositions, due to the very limited single-element compositions existing in the chemistry table. For multi-element composition design, BO has the lowest variance and GWO has the highest variance of materials count, with GWO's variance approximately 7398 times higher than BO. Other than that, GWO's variance is approximately 27 times higher than that of RUN of materials count. GA and AOA have approximately the same level of uncertainty for materials count. The relative magnitude of the variances of the materials count and the mean density scan does not follow the exact same trend. Yet for multi-element composition design, GWO still exhibits the highest variance, with 366.93% higher than that of BO and 9992.08% higher than that of BO. Generally, it can still be deduced from Table 1 that, for both single- and multi-element composition design, nature-inspired algorithms (i.e., GA, GWO, BWO) display relatively higher variances, indicating higher uncertainties, which will be further discussed in Section 4. Table 2 further confirms the observation that nature-inspired algorithms present higher uncertainties. For both single- and multi-element composition designs, GA, GWO, & BWO show evidently higher variances of the mean objectives than that of other optimizations. For single-element composition design, GA presents 155.56% and 1093.8% relatively higher variances, GWO presents 35.24% and 531.77% relatively higher variances, and BWO presents 95.20% and 811.86% relatively higher variances than that of RUN and BO, respectively. For multi-element composition design, GA presents 2136.79% and 6380.87% relatively higher variances, GWO presents 2190.70% and 6537.04% relatively higher variances, and BWO presents 2132.71% and 6369.04% relatively higher variances than that of RUN and BO, respectively.

Figure 6 visualizes the material distribution generated by all the optimization algorithms, the changes in objective values during material evaluations benchmarked against silicon, and the reconstruction of the design space averaged across the 11 different optimizations. In Figure 6 **A**, the crystal phases of Xe and Kr are repeatedly scanned by multiple algorithms, despite their instability under room temperature. This raises an intriguing question for computational optimization: How can complex factors such as cost and industrial applications be incorporated into the optimization framework to enhance its realism for real-world deployment? This may be left for future explorations. AOA and GWO show an increasing trend in the number of evaluated elemental compositions in the target space for bulk modulus, while BO, GA, and GWO exhibit the same trend for Fermi energy. Figure 6 **C** visually depicts the design space, conveying a clear message that it is highly nonconvex and presents significant challenges in searching for the optimal point. Moreover, the non-continuous contours observed in the averaged design spaces based on evaluated elemental compositions further complicate the search for the global optimum. For a detailed discussion on the design spaces of all 11 optimization algorithms (and DRL), please refer to the ESI.

From Figure 6, we can derive five main observations. (1) Comparing **A** and **B**, we observe that the top evaluated elemental compositions are similar. (2) The majority of the top chemical compounds are binary alloys, consisting of elements present in the top single-element compositions such as Cs, Ce, Ba, and others. (3) RUN demonstrates effective scanning strategies and consistently converges towards local optimal elemental compositions. (4) SA demonstrates similar search strategies as RUN, evidenced by the high normalized frequency along the materials list. (5) BO appears to struggle in identifying the "ideal" elemental compositions and tends to scan the same region repeatedly, indicating a potential trap in the local optimal area. Based on Figures 4-6, we have also added qualitative comparison and suggested potential applications in Table 3. Since we hope to design materials digitally with low uncertainty and targeted desired properties quantified through objectives, low variance, and high mean objectives can be viewed as "advantages" of the algorithm. "Targeted materials selection" is recommended due to repeated materials sampling and "Large-scale exploration" is recommended based on more materials candidates sampled with decent objective values.

Among all the metrics we evaluated, it can be deduced that RUN achieved superior optimization results compared with the others (Figures 4 & 5, Tables 1 & 2). This out-performance may be due to several reasons. (1) RUN is free of hyperparameters. The update process of RUN does not depend on fine-tuned hyperparameters, making it more robust especially for highly nondeterministic situations like elemental composition design. (2) The updating strategies of RUN are very different than that of other methods. The RUN optimization searches for the next set of evaluations by mimicking the update strategies of Runge Kutta's discretization of differential equations (see ESI), which are fundamentally different from other metaheuristic methods. (3) From (2), the enhanced solution quality (ESQ) mechanism is well suited for our formulated elemental composition discovery cases. In the authors' original proposal[38], the ESQ mechanism is employed to avoid the local optimal solutions and increase convergence speed. In our case, as the material evaluation numbers are fixed at 5000, the faster convergence induced by ESQ enables the robust performance of the RUN algorithm in our benchmark cases. Further comprehensive characteristics are discussed in Section 4.
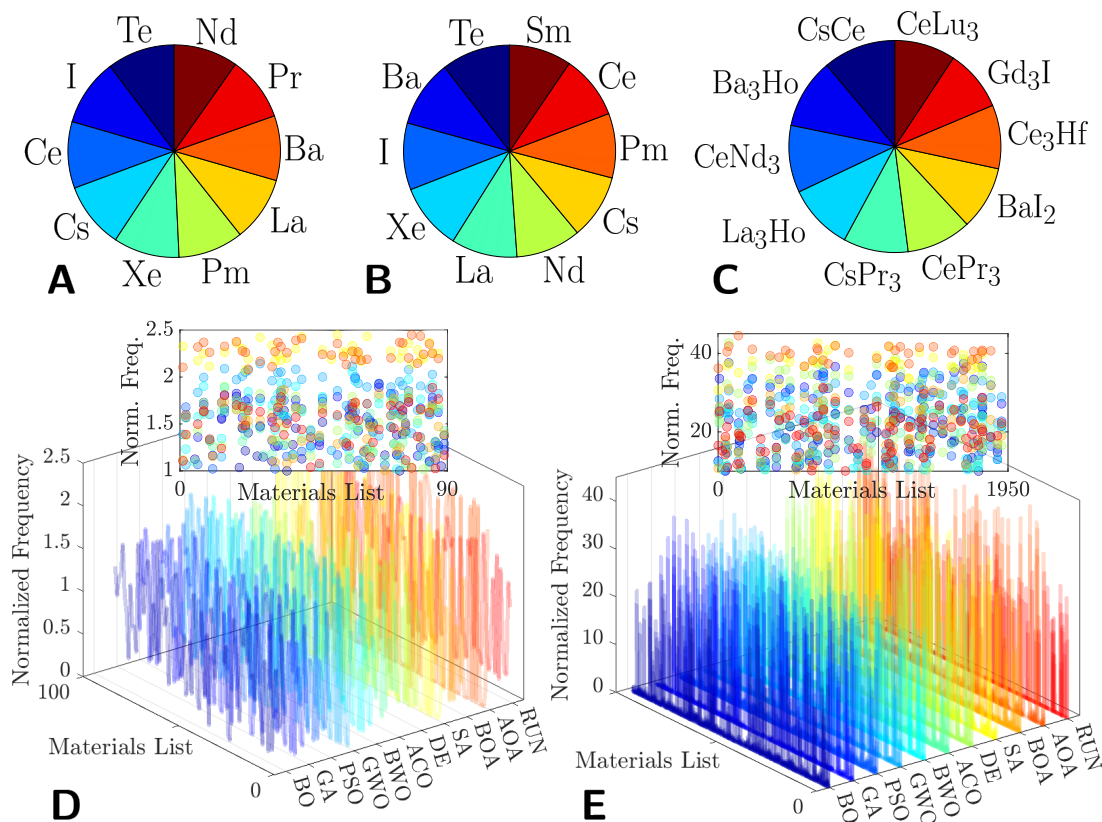
Figure 6. Extracted elemental composition evaluations for the optimization benchmarking. **(A)** The top ten elemental compositions extracted from the single-element design case. **(B)** The top ten elemental compositions extracted from the multi-element composition design cases. **(C)** The top ten chemical compounds extracted from the multi-element composition design case. **(D)** The normalized frequency (see *Additional Results* in ESI for extended analysis for frequency evaluations of optimizations) for different extracted elemental compositions under the single-element composition design cases. The top subfigure dots stand for the projection of the normalized frequency in the range of $[1, 2.5]$. **(E)** The normalized frequency under the multi-element composition design cases. The top subfigure dots stand for the projection of the normalized frequency in the range of $[10, 45]$.

Optimizations	Advantages	Disadvantages	Potential Use
BO	Low variances	Low objective	Large-scale materials exploration
GA	High objectives	High variances	Targeted materials selection
PSO	Low variances	Low objective	Large-scale materials exploration
GWO	High objective	High variances	Not recommended
BWO	High objective	High variances	Not recommended
ACO	Low variances	Low objective	Large-scale materials exploration
DE	Decent objective	Relatively high variances	Not recommended
SA	High selectivity	Problem-specific	Not recommended
BOA	Decent objective	Low selectivity	Not recommended
AOA	High selectivity	High variance	Targeted materials selection
RUN	High objective & stable	Low selectivity	Materials exploration & design

Table 3. Assessment and conclusion from the comparison results for optimization methodologies benchmarked. Note that the (dis)advantages concluded from the previous results may be subjective. Note that the "Low selectivity" for RUN indicates the variance is relatively low compared with DE and AOA.

4 Discussion

Based on the results presented earlier, our findings raise several intriguing questions, for which we proffer the following discussions. Firstly, we observe variations in the performance of different optimization algorithms across the two elemental composition design scenarios, such as the contrasting results of material counts and mean density of RUN and AOA (Figure 4 **B** & **C**). One explanation is that algorithm effectiveness is problem-specific. Moreover, the optimization methods may also be sensitive to the dimensionality of the problem, such that changing from single-element to multi-element composition design induces the change of metrics evaluation shown in Figures 4 & 5. Additionally, in the case of single-element composition design, the limited number of available elemental compositions can impact the performance of certain algorithms, potentially influencing their search capabilities. An interesting observation pertains to the consistent success of the RUN algorithm in achieving high mean objective values in both problem scenarios, especially in the second case (Figure 5 **A**). This achievement can be attributed to the fact that the sampling strategy employed by RUN is hyperparameter-free and adept at storing information, which proves advantageous in handling high-dimensional spaces.

Another intriguing finding is that BO exhibits lower mean objectives accompanied by low variances in both cases. This trend becomes more apparent when considering Figures 5 **B** & 6 **E**. A plausible explanation would be BO is sampling a wider range of diverse elemental compositions during the optimization process. This behavior can be attributed to the uncertainty-based sampling strategy employed by BO, which facilitates the exploration of a diverse design space and aids in constructing a more accurate surrogate model. AOA tends to repeatedly sample similar elemental compositions over an extensive number of evaluations (Figure 5 **B**). This behavior suggests that AOA may become trapped within a specific region of the design space, necessitating further investigation to understand the underlying causes and identify potential remedies. Finally, we explore the major disparity in SA's performance between the two elemental composition discovery cases. A possible explanation for this discrepancy is that SA is sensitive to the dimension of the optimization problem, leading to its different performances tested from the designed metrics in Figures 4 & 5.

Each optimization algorithm demonstrates unique characteristics and strategies that contribute to its specific outcomes. The low variances of the objectives in Figure 5 could be explained by BO's low dependence on hyperparameters. BO aims to approximate an accurate surrogate model to optimize the objective, resulting in small mean objective values. This strategy utilizes uncertainty-based sampling, which decreases the overall objective value. The low uncertainty and variance in the final results are attributed to the small number of hyperparameters and the large sampling base. Additionally, BO exhibits high material counts, as observed in Figure 4 **B**. GA heavily relies on hyperparameters such as mutation, offspring generation, and randomization (details see ESI). Strong dependence on hyperparameters leads to high uncertainty in the mean objective values for both cases (Figure 5 **A1** & **A2**). Moreover, GA repeated samples of the same elemental compositions once ideal properties are detected, resulting in high sampling density compared with BO, PSO, and other related methods (Figure 5 **B1** & **B2**). PSO, on the other hand, exhibits lower uncertainty and variance as it has fewer hyperparameters, contributing to its more stable performance, as depicted in Figure 5 **A1** & **A2**.

GWO is similar to GA in terms of hyperparameter dependency, resulting in high uncertainty and sampling density focused on specific objective regions (Figure 5). Notably, GWO displays higher uncertainty even in material counts and mean density, which differentiates it from GA (Figure 4). Similarly, BWO exhibits high uncertainty in mean objective values (Figure 5 **A**), which may also be attributed to its dependence on hyperparameters and very similar searching strategies of mimicking animal dynamical behavior. However, BWO displays more material counts and less sampling density scan (Figure 4 **B** & **C**), potentially attributed to the setting of the encircling prey parameter, which enhances sampling stability. ACO employs continuous probability-based sampling, resulting in relatively less uncertainty in the mean objective values (Figure 5 **A**). ACO has similar mean objective values as BO and PSO while exhibiting lower sampling density scans (Figure 4 **C1**), indicating that ACO avoids oversampling the same regime for the single-element composition design case. DE algorithmically operates as a variant of GA, yielding comparable mean objective results (Figure 5 **A**). With fewer hyperparameters compared to GA, DE exhibits lower uncertainty (Figure 5 **A**). The evolutionary strategies of DE demonstrate similar material counts to GA, as seen in Figure 4, and repeated sampling for favorable materials, as depicted in Figure 6. SA's performance is observed to be more problem-specific, as both the material counts and density scan vary per different design cases (Figure 4 **B** & **C** and Figure 5 **A**). SA exhibits low uncertainty for the mean objective values (Figure 5 **A**) due to fewer hyperparameters and the implementation of a Markov chain for the acceptance-rejection sampling criteria (details see ESI). BOA shares similarities with GWO and BWO in terms of high uncertainty in the mean objectives (Figure 5 **A**) and low material counts for multi-element composition design (Figure 4 **B2**), as they use particle-based, animal behavior-inspired search criteria (details see ESI). However, BOA exhibits relatively lower variance (Figure 5 **A**), indicating lower dependence on, and having fewer, hyperparameters.

AOA uses a distinct searching strategy based on mathematical symbolized operations. AOA exhibits high uncertainty with relatively low mean objectives (Figure 5 **A**). AOA tends to oversample the local regime, leading to low material counts (Figure 5 **B**). This behavior can be attributed to a large number of hyperparameters and the overdependence on certain factors such as `MOA` and `MOP` (details see *Arithmetic Optimization Algorithm* in ESI), resulting in a scaling factor that underestimates

previous locations. Finally, RUN achieves high objectives with low uncertainty and avoids oversampling in specific objective regions (Figure 5). The material count is sensitive to dimensionality, yet all dimensions yield successful search results, as judged by the high overall mean objectives (Figure 5 **A**). RUN is characterized by being hyperparameter-free, incorporating numerous intermediate hyperparameters to increase the dimensionality of the iteration scheme, and relying on previous solutions for updates based on search history. Following our previous discussion on RUN's ESQ search mechanism and being hyperparameter-free, we would like to examine why RUN had outperformed other algorithms by having high objective values and low uncertainties. RUN employs an update scheme that has four update parameters[1], each containing randomization coefficients[2] and are connected. This randomization and "parameter-connection" could help explain the low uncertainty observed in our numerical experiments. For the high objective, RUN's ability to avoid local optima as reported by Ahmadianfar et al.[38] could be a possible explanation, in which they contend that the adaptive mechanism employed to update the parameter and the ESQ are the main mechanisms that assure a good transition from exploration to exploitation.

5 Conclusion

In this paper, we propose a framework that can incorporate most state-of-the-art optimization algorithms to perform multi-elemental composition discovery and design for crystals. Our framework parameterizes the input elemental graph structure to tailor the corresponding bulk modulus K and Fermi energy E_{Fermi}. The key goal is to conduct a comprehensive benchmark for many of the state-of-the-art optimization methods and provide further insights for future implementation of these algorithms for elemental composition design. We optimized both single and multi-element composition discovery for maximized K and minimized E_{Fermi} by formulating a single objective optimization. We benchmarked 11 different inverse optimization algorithms by fixing the material evaluations. We found that GA, GWO, and BWO exhibit higher variances, and BO and RUN display generally lower variances for materials count, density scan (Table 1), and mean objectives (Table 2). We further conclude that nature-inspired algorithms contain more uncertainties in elemental design cases, which can be attributed to the dependency on hyperparameters. RUN exhibits higher mean objectives whereas BO displayed low mean objectives compared with other optimization methods. Combined with materials count and density scan, it is proposed that BO samples more elemental compositions aiming to approximate a more accurate surrogate of the design space, and hence have lower mean objectives, yet RUN will repeatedly sample the discovered optimal elemental compositions for the successful search strategy (Figures 5 **B** and 6 **D** & **E**). We also proffer detailed discussions on the exhibited results that correspond to each optimization's characteristics.

6 Acknowledgements

7 Author contributions statement

H.Z. and H.H. conceived the research. H.Z. designed and conducted the experiment(s). H.Z., H.H., and J.Y. analyzed the results. J.Y. acquired the funding and resources. All authors reviewed the manuscript.

8 Additional information

Accession codes: The codes for the optimization benchmarking are available at https://github.com/hanfengzhai/materials-discovery-benchmark/tree/master/inverse_opt.
Competing interests: The authors declared no competing interests.

References

1. Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. S. & Sigmund, O. Efficient topology optimization in MATLAB using 88 lines of code. *Struct. Multidiscip. Optim.* **43**, 1–16, DOI: 10.1007/s00158-010-0594-7 (2010).

2. Bendsøe, M. P. & Kikuchi, N. Generating optimal topologies in structural design using a homogenization method. *Comput. Methods Appl. Mech. Eng.* **71**, 197–224, DOI: 10.1016/0045-7825(88)90086-2 (1988).

---

[1] k_1, k_2, k_3, k_4 in Ref.[38]
[2] $rand_1$ & $rand_2$ in Ref.[38]

3. Zhai, H. & Yeo, J. Multiscale mechanics of thermal gradient coupled graphene fracture: A molecular dynamics study. *Int. J. Appl. Mech.* DOI: 10.1142/s1758825123500448 (2023).

4. Ng, T., Yeo, J. & Liu, Z. A molecular dynamics study of the thermal conductivity of graphene nanoribbons containing dispersed stone–thrower–wales defects. *Carbon* **50**, 4887–4893, DOI: 10.1016/j.carbon.2012.06.017 (2012).

5. Ritz, E. T. & Benedek, N. A. Interplay between phonons and anisotropic elasticity drives negative thermal expansion in pbtio$_3$. *Phys. Rev. Lett.* **121**, DOI: 10.1103/physrevlett.121.255901 (2018).

6. Buehler, M. J. Modeling atomistic dynamic fracture mechanisms using a progressive transformer diffusion model. *J. Appl. Mech.* **89**, DOI: 10.1115/1.4055730 (2022).

7. Mannodi-Kanakkithodi, A., Pilania, G., Huan, T. D., Lookman, T. & Ramprasad, R. Machine learning strategy for accelerated design of polymer dielectrics. *Sci. Reports* **6**, DOI: 10.1038/srep20952 (2016).

8. Kim, C., Batra, R., Chen, L., Tran, H. & Ramprasad, R. Polymer design using genetic algorithm and machine learning. *Comput. Mater. Sci.* **186**, 110067, DOI: 10.1016/j.commatsci.2020.110067 (2021).

9. Winter, R. *et al.* Efficient multi-objective molecular optimization in a continuous latent space. *Chem. Sci.* **10**, 8016–8024, DOI: 10.1039/c9sc01928f (2019).

10. Weiel, M. *et al.* Dynamic particle swarm optimization of biomolecular simulation parameters with flexible objective functions. *Nat. Mach. Intell.* **3**, 727–734, DOI: 10.1038/s42256-021-00366-3 (2021).

11. Shin, D. *et al.* Spiderweb nanomechanical resonators via bayesian optimization: Inspired by nature and guided by machine learning. *Adv. Mater.* **34**, 2106248, DOI: 10.1002/adma.202106248 (2021).

12. Zhang, Y., Tao, S., Chen, W. & Apley, D. W. A latent variable approach to gaussian process modeling with qualitative and quantitative factors. *Technometrics* **62**, 291–302, DOI: 10.1080/00401706.2019.1638834 (2019).

13. Ghumman, U. F. *et al.* Designing active layer of organic solar cells using multi-fidelity molecular simulations and spectral density function. *Comput. Mater. Sci.* **211**, 111491, DOI: 10.1016/j.commatsci.2022.111491 (2022).

14. Comlek, Y., Pham, T. D., Snurr, R. & Chen, W. Rapid design of top-performing metal-organic frameworks with qualitative representations of building blocks, DOI: 10.48550/ARXIV.2302.09184 (2023).

15. Zhang, H., Chen, W., Iyer, A., Apley, D. W. & Chen, W. Uncertainty-aware mixed-variable machine learning for materials design. *Sci. Reports* **12**, DOI: 10.1038/s41598-022-23431-2 (2022).

16. Iyer, A. *et al.* Data centric nanocomposites design via mixed-variable bayesian optimization. *Mol. Syst. Des. & Eng.* **5**, 1376–1390, DOI: 10.1039/d0me00079e (2020).

17. Zhai, H. & Yeo, J. Computational design of antimicrobial active surfaces via automated bayesian optimization. *ACS Biomater. Sci. & Eng.* **9**, 269–279, DOI: 10.1021/acsbiomaterials.2c01079 (2022).

18. Zheng, B., Zheng, Z. & Gu, G. X. Designing mechanically tough graphene oxide materials using deep reinforcement learning. *npj Comput. Mater.* **8**, DOI: 10.1038/s41524-022-00919-z (2022).

19. Sui, F., Guo, R., Zhang, Z., Gu, G. X. & Lin, L. Deep reinforcement learning for digital materials design. *ACS Mater. Lett.* **3**, 1433–1439, DOI: 10.1021/acsmaterialslett.1c00390 (2021).

20. Wang, Y., Cao, Z. & Farimani, A. B. Efficient water desalination with graphene nanopores obtained using artificial intelligence. *npj 2D Mater. Appl.* **5**, DOI: 10.1038/s41699-021-00246-9 (2021).

21. De Cao, N. & Kipf, T. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973* DOI: 10.48550/ARXIV.1805.11973 (2018).

22. Ong, S. P. *et al.* Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Comput. Mater. Sci.* **68**, 314–319, DOI: 10.1016/j.commatsci.2012.10.028 (2013).

23. Jain, A. *et al.* Commentary: The materials project: A materials genome approach to accelerating materials innovation. *APL Mater.* **1**, 011002, DOI: 10.1063/1.4812323 (2013).

24. Ward, L. *et al.* Matminer: An open source toolkit for materials data mining. *Comput. Mater. Sci.* **152**, 60–69, DOI: 10.1016/j.commatsci.2018.05.018 (2018).

25. Thieu, N. V. & Mirjalili, S. MEALPY: a Framework of The State-of-The-Art Meta-Heuristic Algorithms in Python, DOI: 10.5281/zenodo.6684223 (2022).

26. Chen, C., Ye, W., Zuo, Y., Zheng, C. & Ong, S. P. Graph networks as a universal machine learning framework for molecules and crystals. *Chem. Mater.* **31**, 3564–3572, DOI: 10.1021/acs.chemmater.9b01294 (2019).

27. Frazier, P. I. A tutorial on bayesian optimization, DOI: 10.48550/ARXIV.1807.02811 (2018).

28. Sutton, R. S. & Barto, A. G. *Reinforcement Learning: An Introduction* (MIT press, 2018).

29. Carr, J. An introduction to genetic algorithms. Whitman College Department of Mathematics (2014).

30. Liang, J., Qin, A., Suganthan, P. & Baskar, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evol. Comput.* **10**, 281–295, DOI: 10.1109/tevc.2005.857610 (2006).

31. Obadina, O. O., Thaha, M. A., Althoefer, K. & Shaheed, M. H. Dynamic characterization of a master–slave robotic manipulator using a hybrid grey wolf–whale optimization algorithm. *J. Vib. Control.* **28**, 1992–2003, DOI: 10.1177/10775463211003402 (2021).

32. Tang, C., Sun, W., Wu, W. & Xue, M. A hybrid improved whale optimization algorithm. In *2019 IEEE 15th International Conference on Control and Automation (ICCA)*, DOI: 10.1109/icca.2019.8900003 (IEEE, 2019).

33. Socha, K. & Dorigo, M. Ant colony optimization for continuous domains. *Eur. J. Oper. Res.* **185**, 1155–1173, DOI: 10.1016/j.ejor.2006.06.046 (2008).

34. Mohamed, A. W., Hadi, A. A. & Jambi, K. M. Novel mutation strategy for enhancing SHADE and LSHADE algorithms for global numerical optimization. *Swarm Evol. Comput.* **50**, 100455, DOI: 10.1016/j.swevo.2018.10.006 (2019).

35. Bertsimas, D. & Tsitsiklis, J. N. Simulated annealing. *Stat. science* **8**, 10–15 (1993).

36. Karaboga, D. & Basturk, B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **39**, 459–471, DOI: 10.1007/s10898-007-9149-x (2007).

37. Abualigah, L., Diabat, A., Mirjalili, S., Elaziz, M. A. & Gandomi, A. H. The arithmetic optimization algorithm. *Comput. Methods Appl. Mech. Eng.* **376**, 113609, DOI: 10.1016/j.cma.2020.113609 (2021).

38. Ahmadianfar, I., Heidari, A. A., Gandomi, A. H., Chu, X. & Chen, H. RUN beyond the metaphor: An efficient optimization algorithm based on runge kutta method. *Expert. Syst. with Appl.* **181**, 115079, DOI: 10.1016/j.eswa.2021.115079 (2021).

39. De Breuck, P.-P., Hautier, G. & Rignanese, G.-M. Materials property prediction for limited datasets enabled by feature selection and joint learning with modnet. *npj Comput. Mater.* **7**, DOI: 10.1038/s41524-021-00552-2 (2021).

40. Kong, S., Guevarra, D., Gomes, C. P. & Gregoire, J. M. Materials representation and transfer learning for multi-property prediction. *Appl. Phys. Rev.* **8**, DOI: 10.1063/5.0047066 (2021).

41. Schmidt, J., Pettersson, L., Verdozzi, C., Botti, S. & Marques, M. A. L. Crystal graph attention networks for the prediction of stable materials. *Sci. Adv.* **7**, DOI: 10.1126/sciadv.abi7948 (2021).

42. Fung, V., Zhang, J., Juarez, E. & Sumpter, B. G. Benchmarking graph neural networks for materials chemistry. *npj Comput. Mater.* **7**, DOI: 10.1038/s41524-021-00554-0 (2021).

43. Dunn, A., Wang, Q., Ganose, A., Dopp, D. & Jain, A. Benchmarking materials property prediction methods: the matbench test set and automatminer reference algorithm. *npj Comput. Mater.* **6**, DOI: 10.1038/s41524-020-00406-3 (2020).

44. Lillicrap, T. P. *et al.* Continuous control with deep reinforcement learning, DOI: 10.48550/ARXIV.1509.02971 (2015).

45. Mirjalili, S., Mirjalili, S. M. & Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **69**, 46–61, DOI: 10.1016/j.advengsoft.2013.12.007 (2014).

SUPPLEMENTARY INFORMATION: Benchmarking Inverse Optimization Algorithms for Materials Design

Hanfeng Zhai⋆[3], Hongxia Hao†[4], Jingjie Yeo⋆[5]

⋆Cornell University, Ithaca, NY, USA
†Microsoft Research AI4Science, Beijing, China

Summary

This supplementary information contains the extension for our problem formulation for detailed explanations. We then provide further details on hyperparameters and basic derivation for the 12 optimization algorithms employed (the 11 employed optimizations for repeated experiments with deep reinforcement learning). We then proffer some additional results in supplementary to our proposition in the main article.

Additional Results

Figure S1 shows the statistical distribution of the objective values over the five experiments with different random seeds. Figure S2 shows the statistical distribution of the materials count w.r.t. the objective values over the five experiments with different random seeds. It can be observed from both figures that the five repeated experiments generally match pretty well, indicating the robustness of the proposed framework in benchmarking optimization methods.
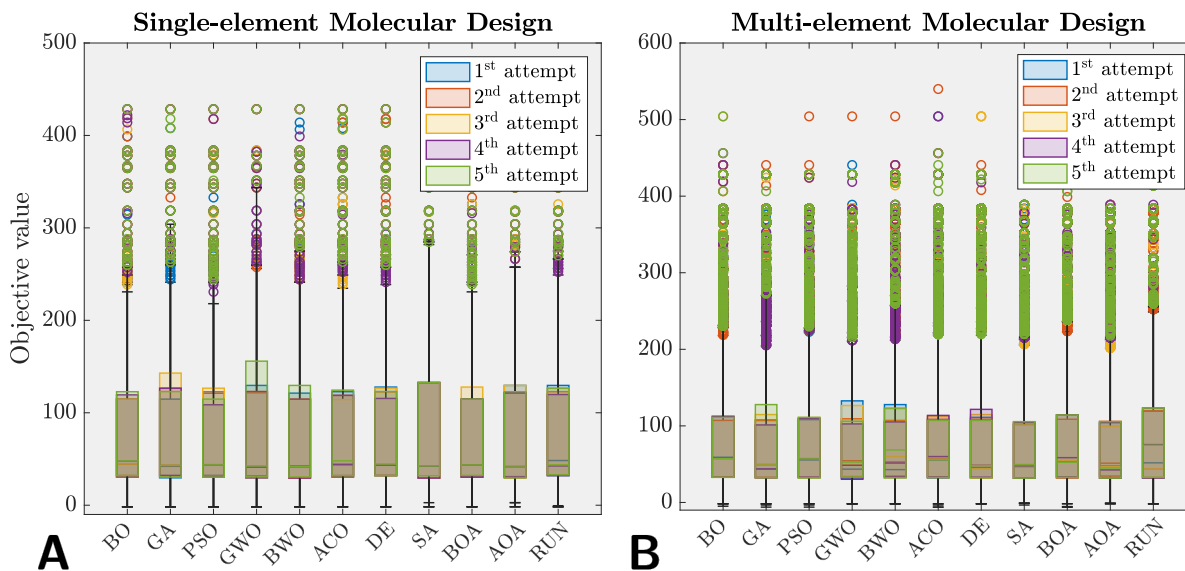

FIGURE S1. The objective value distribution over the five different experiments with different random seeds, in which the single-element composition design experiments are shown in **A** and multi-element case in **B**.

Figure S3 showcases the evaluation processes and the extracted element composition from both BO and RUN. The molecular evaluations in the objective spaces are marked with color gradients indicating the process, marked in the color bar on the right side of the figures (Figure S3 **A** & **B**). The top ten element compositions are also visualized for comparison marked in color gradients (Figure S3 **C** & **D**).

Figure S4 shows the general schematic for implementing the deep reinforcement learning method in our proposed framework. The MEGNet material evaluations are treated as the environment and the agent (represented as a DNN) is trained to learn the
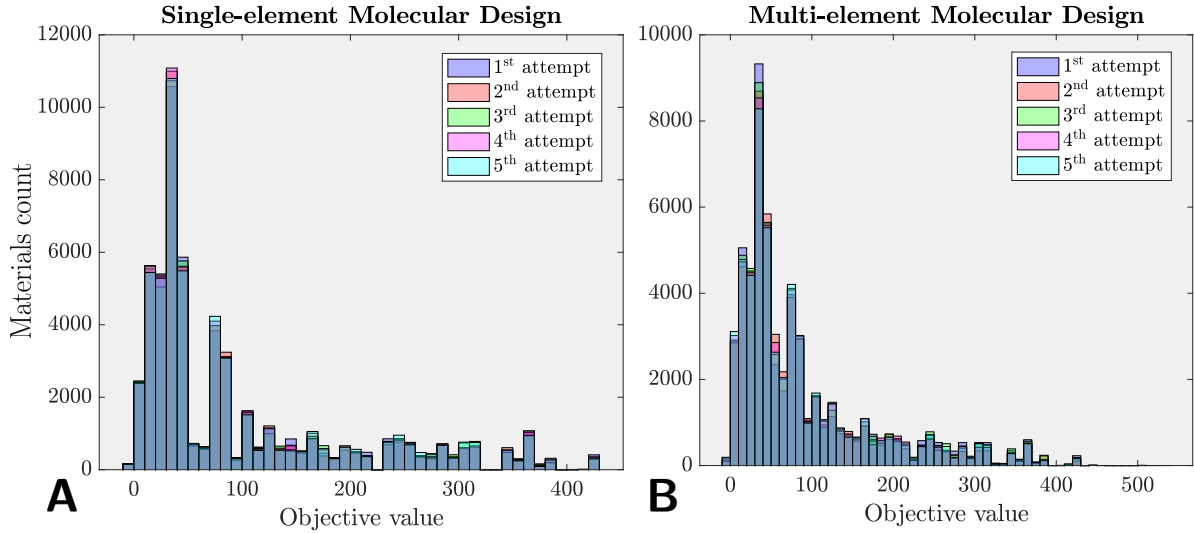

FIGURE S2. The distribution of the materials count over different objective values for the five different experiments, in which the single-element composition design experiments are shown in **A** and multi-element case in **B**.

action for selecting the best element composition from the materials project database. Detailed formulations are elaborated in the following section. Figure S5 showcases some preliminary results from using DRL to design molecular materials, where the single-element composition design processes are shown in Figure S5 **A** and the multi-element composition design processes are shown in Figure S5 **B**. Correspondingly, the material evaluations indicated in Figure 4 **A** are projected into 2-dimensional space and shown in Figure S6.

To better visualize our efforts in estimating the design processes through the mean density scan methods, we present some example density scan evaluations using BO and RUN in Figure S7. It can be seen that BO tends to evaluate a broad range of different element compositions with shallow gradients distributed among a material region, whereas RUN tends to repeatedly evaluate the detected "optimal" element composition multiple times, with sharp gradient points discretely distributed among the objectives space. This direct observation cross-validated our observations and proposed understandings of the searching strategies of BO and RUN in Sections 3 & 4.

Extension on Problem Formulation

Based on our problem formulation in Equation (1), we can further expand the materials parametrization function Ω. To explain, this is a highly nonconvex problem where the design variables are not related to the objective values at all. The ideal scenario, as described in the article, is to directly tune the graph structure representing the element composition to achieve direct modifications of its corresponding materials' properties. However, the difficulty that arises in this matter is it is knotty to represent the graph as the same dimensional input for perturbing the variables.

If assume the graph structures of the element composition can be written in the form of $\mathscr{G}(\mathbf{u}, \mathbf{e}, \mathbf{v})$. The relation between the design variables and the objective values can be written as

$$
\begin{aligned}
K &= MEGNet_K(\mathscr{G}(\mathbf{u}, \mathbf{e}, \mathbf{v})) \xleftarrow{\mathscr{G}(\mathbf{u}, \mathbf{e}, \mathbf{v}) = \Omega(n_{atom}, \xi_n, \eta)} n_{atom}, \xi_n, \eta \\
E_{\mathrm{Fermi}} &= MEGNet_{E_{\mathrm{Fermi}}}(\mathscr{G}(\mathbf{u}, \mathbf{e}, \mathbf{v})) \xleftarrow{\mathscr{G}(\mathbf{u}, \mathbf{e}, \mathbf{v}) = \Omega(n_{atom}, \xi_n, \eta)} n_{atom}, \xi_n, \eta
\end{aligned}
\tag{S1}
$$

where $MEGNet_K$ and $MEGNet_{E_{\mathrm{Fermi}}}$ represent the two pretrained models for predicting the properties of bulk modulus K and Fermi energy E_{Fermi}. In our formulation (Equation (1)), the molecular graph is generated from the design variables via a parameterization function Ω. This parameterization process relaxes the dimensional constraint of the molecular graphs for
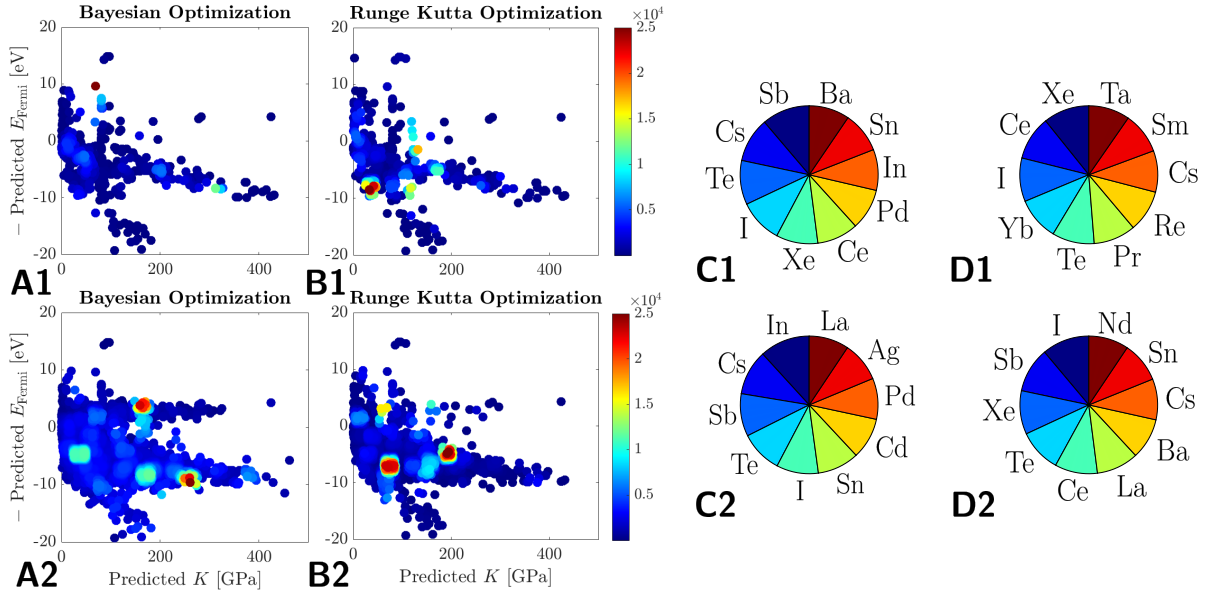
FIGURE S3. Some example numerical experiments showcasing results from Bayesian optimization and Runge Kutta optimization. Subfigures **A1** & **B1** are the objective space evaluations for the single-element composition design for BO and RUN; Subfigures **A2** & **B2** are the objective space evaluations for the multi-element composition design for BO and RUN. Subfigure **C1** & **D1** are the top ten element compositions extracted from single-element composition discovery for the single-element composition design. Subfigure **C2** & **D2** are the top ten element compositions extracted from single-element composition discovery for the multi-element composition design.

different types of compounds. The function, Ω, can be extended to

$$\mathscr{G}_{mat}(\mathbf{u}, \mathbf{e}, \mathbf{v}) = \Omega(n_{atom}, \xi_n, \eta):$$

$$n_{atom} = \texttt{rand}_{\Theta_1} \in [n_{atom}^{low}, n_{atom}^{high}],$$
$$\xi_n = \texttt{rand}_{\Theta_2} \in [0, 1],$$
$$\eta = \texttt{rand}_{\Theta_3} \in [0, 1],$$
$$C_{mat} = C_{mat}(n_{atom}, \xi_n),$$
$$\mathbb{L}_{mat} = \mathbb{L}_{mat}(C_{mat}) \in [MP],$$
$$\mathscr{T}_{mat} = \mathscr{T}_{mat}(\mathbb{L}_{mat}, \eta),$$
$$\mathscr{G}_{mat}(\mathbf{u}, \mathbf{e}, \mathbf{v}) = MP(\mathscr{T}_{mat})$$

(S2)

where $\mathbf{u}, \mathbf{e}, \mathbf{v}$ are state, bond, and atom attributes, following the notation of Ref.[26]. n_{atom}^{low} and n_{atom}^{high} are readjusted for the two design optimization scenarios. $C_{mat}, \mathbb{L}_{mat}$, and $\mathscr{T}_{mat}$ are different functions relating the design variables to the graph structure through hierarchical algorithmic steps. $[MP]$ simply stands for the material database stored in the *Materials Project*.

Estimation Metrics

Density Scan

We use density scans to estimate the repeated evaluations of the same material in the objectives space. First, the material evaluation in the objectives space (Figure 4 **A** & Figure S7) is first projected on a 100×100 grids as a density scan to describe the repeated evaluations of the same points. We then eliminate all the zero-value scan points and compute the mean values of the non-zero density scans. The equation describing the density scan can be written as

$$\bar{\rho} = \texttt{mean}(\mathbb{N}(n_{atom}, \xi_n, \eta) | \mathbb{N} \neq 0)$$

(S3)

where $\mathbb{N}$ is the density scan, which is a function of the design variables $\mathbb{N} = \mathbb{N}(n_{atom}, \xi_n, \eta)$.
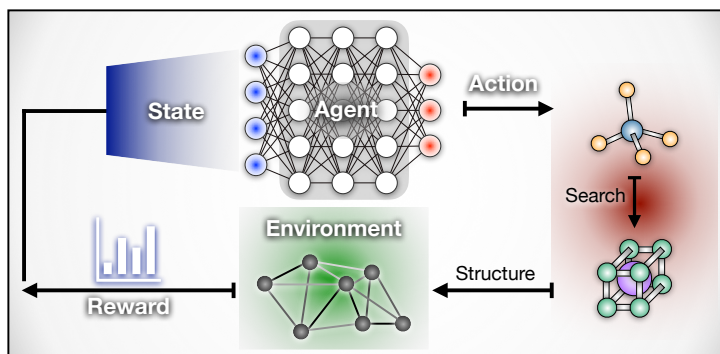
FIGURE S4. Schematic for materials discovery using deep reinforcement learning. The optimization is enabled by training a deep neural network as the "agent" to make optimal decisions, i.e., selecting the best materials. The action, and output of the network, screen the materials database and find the corresponding structure as input to the GNN as the environment. The environment then feeds back rewards (from predicted material properties) to the agent for making the best decisions.
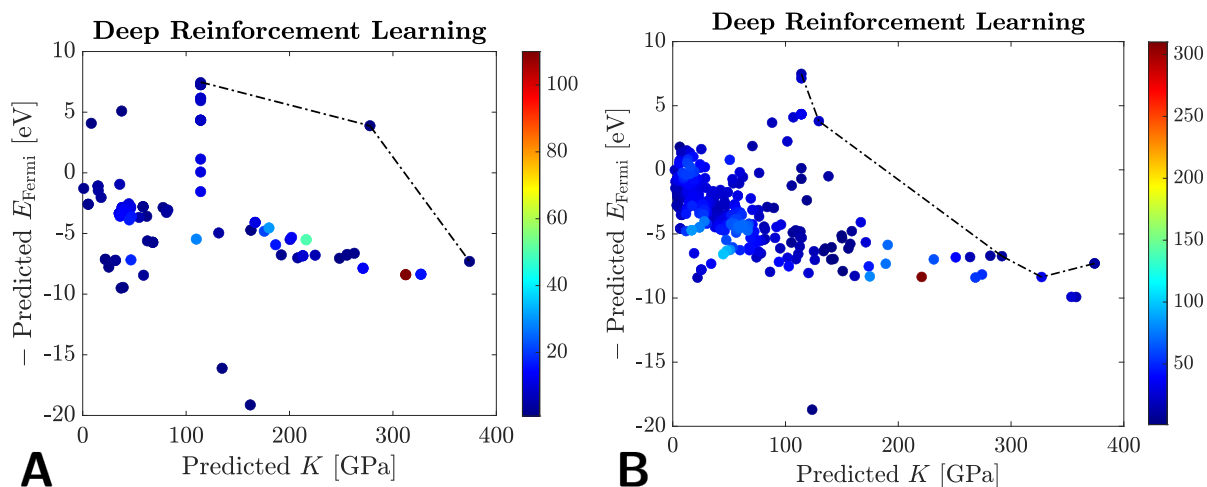
FIGURE S5. Example cases of molecular discovery using the deep reinforcement learning methods. (**A**) Single-element composition design case with 110 molecular evaluations. (**B**) Multi-element composition design case with 300 molecular evaluations.
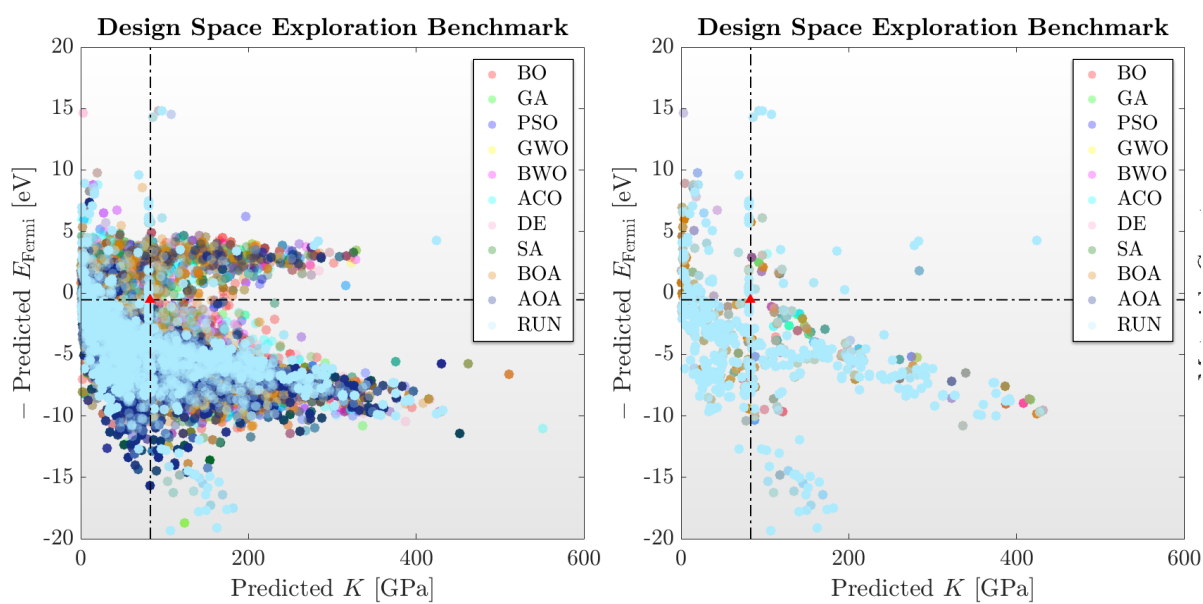
FIGURE S6. The projection in 2-dimensional objective spaces of different optimization algorithms corresponding to Figure 4 **A**. The left subfigure is the single-element composition design case and the right subfigure is the multi-element composition design case.
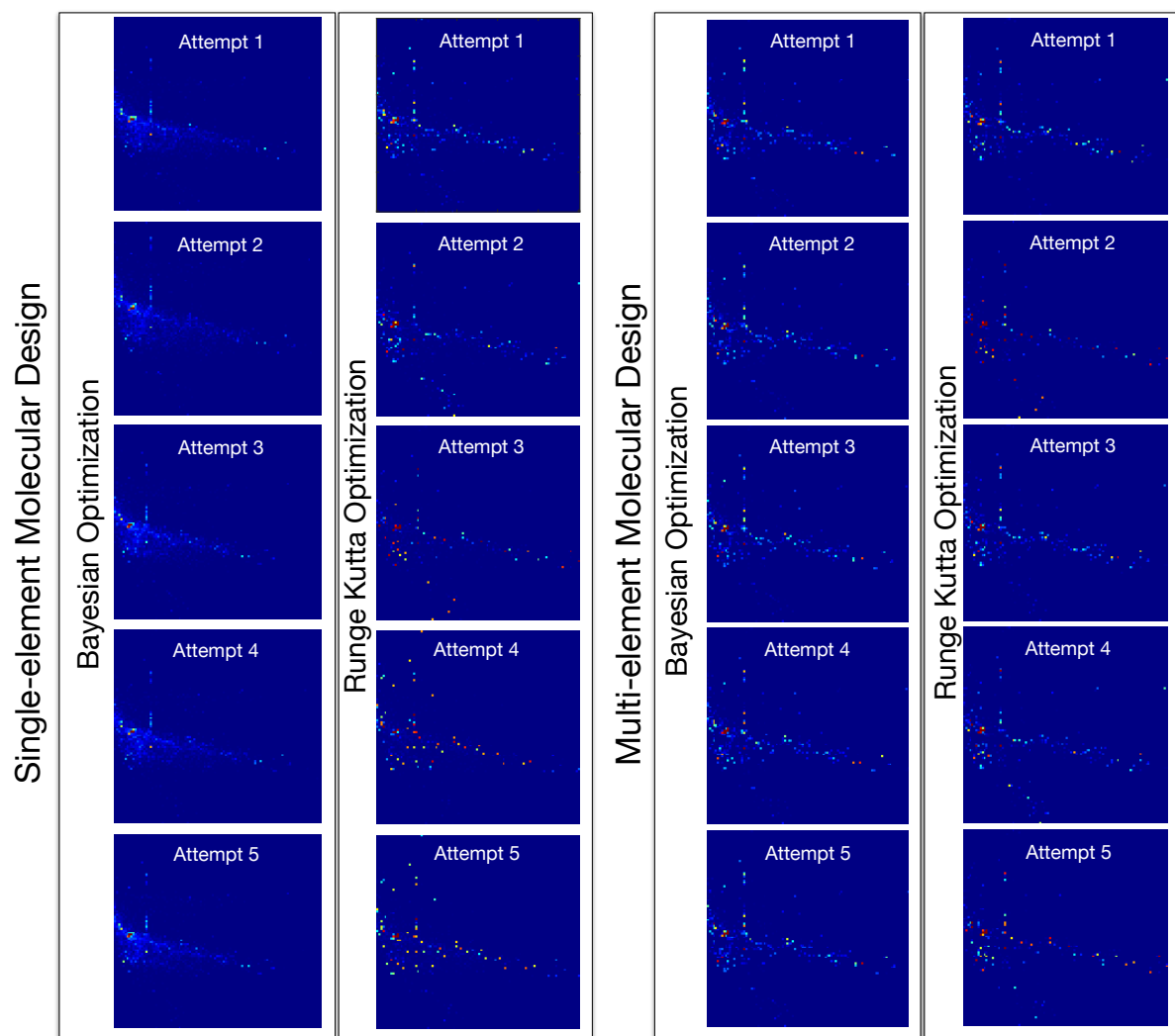
FIGURE S7. Example density scan map corresponding to the mean density scan evaluations in Figure 4 **C** for BO and RUN. Note that the blue area corresponds to the zero molecular evaluation region.

Machine Learning Based Algorithms

Deep Reinforcement Learning

For approach (2), one trains a deep neural network (DNN) as the agent to learn the policy between the environment ($\Theta \rightarrow \mathscr{J}$) and action ($a$). Here, the environment would be the material evaluator, i.e., MEGNet. The action a would be the change of the design variables Θ. The process of agent tuning the design variables can be represented as

$$\Theta_{next} \overset{a}{\leftarrow} DNN(\mathscr{J}, r; \mathbf{p}_{DNN}) \tag{S1}$$

where r is the rewards from the evaluated material properties, which the DNN receives as input. Similar to GPR, $\mathbf{p}_{DNN}$ is the hyperparameters involved in training the DNN. The agent receives the reward r based on the properties of the evaluated material and takes the corresponding actions a to tune the design variables Θ.

Here, we employ the basic deep Q-learning. The update of the Q-learning schemes is solved from the Bellman's equation[28]:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r + \gamma \mathbb{E}\left[Q^{\pi}(s',a')\right]\right] \tag{S2}$$

where Q^{π} is the optimal action-value function, γ is the discount-rate parameter. s' & a' represents the update for the action and states. The update for the agent's new action requires a *policy*, in which we employ the Boltzmann Q policy. In the DRL implementation, the optimization objective max $\mathscr{J}$ is expected to be learned as policies via agent-environment interactions by feeding rewards. Here, we implement the basic Deep Q Network (DQN) by employing the Q-learning strategy. The agent numerically approaches the strategy for selecting the best Θ to either maximize or minimize $\mathscr{J}$.

In our DRL implementations, the agent is a deep neural network (DNN):

$$[a_1, a_2, a_3] = \left(L^5 \circ \sigma_{ReLU} \circ L^4_{64} \circ \sigma_{ReLU} \circ L^3_{64} \circ \sigma_{ReLU} \circ L^3_{64} \circ \sigma_{ReLU} \circ L^1_{32}\right) s,$$
$$\text{where } s \equiv \mathscr{J} = K - E_{\text{Fermi}} \tag{S3}$$

It can be seen that our agent has 3 hidden layers with 64 neurons per layer activated by the ReLU function. s is the state, which we defined as the objective value. Here, a_1, a_2, a_3 correspond to changing the design variables, n_{atom}, ξ_n, η. The rewards are crafted based on the state as

$$r = \begin{cases} 10, & s \geq 350 \\ 5, & 250 \leq s \leq 350 \\ 1, & 150 \leq s \leq 250 \\ -5, & 50 \leq s \leq 150 \\ -10, & s \leq 50 \end{cases} \tag{S4}$$

The reward differences for different materials' properties intervals are relatively small. Our goal here is to pose a more challenging learning task for the agent to approximate the correct policy for maximizing the policy for design. The reward feeds into the total scores per episode to train the agent.

The standard reinforcement learning setup is considered, consisting of an agent interacting with an environment E in discrete timesteps. The action-value function is used in many reinforcement learning algorithms. It describes the expected return after taking an action at in state st and thereafter following policy π: $Q^{\pi}(s,a) = \mathbb{E}[R|s,a]$. The optimization goal can be achieved via solving the Bellman equation, where the Q^{π} values are updated recursively[44]:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r + \gamma \mathbb{E}\left[Q^{\pi}(s',a')\right]\right] \tag{S5}$$

The Q-learning is achieved by minimizing the loss by considering the function approximators parameterized by $\mathbf{p}^Q$:

$$\mathscr{L}(\mathbf{p}^Q) = \mathbb{E}\left[\left(Q(s,a|\mathbf{p}^Q) - \mathscr{Y}\right)^2\right] \tag{S6}$$

where $\mathscr{Y} = r(s,a) + \gamma Q(s', \mu(s')|\mathbf{p}^Q)$. The optimization is achieved via feedback-action loops through the rewards. Randomized

$$a \xrightarrow{MEGNet} s(a) \xrightarrow{DNN} r$$ Here, we employ the Boltzmann Q policy for the Q^{π} values:
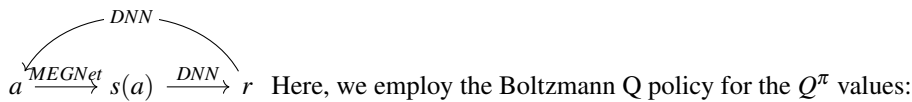
$$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_{a'} e^{Q(s,a')/\tau}} \tag{S7}$$

where $\pi(a|s)$ is the probability of taking action a in state s; $Q(s,a)$ is the estimated Q-value of taking action a in state s; τ is the temperature parameter that controls the exploration-exploitation trade-off. In our implementation, The algorithm undergoes an exploration phase consisting of 30 episodes, during which 5 elemental compositions are evaluated per episode. After this, the algorithm is trained for 60 steps, with 1 material evaluation per step. Finally, the algorithm is tested over 20 episodes, with 5 material evaluations per episode. This setup allows for the algorithm to learn and optimize material designs, with the testing phase providing a measure of its effectiveness.

Bayesian Optimization

Gaussian Process Regression

Gaussian process regression (GPR) is a Bayesian statistical approach to approximate and model function(s). Considering our optimization problem, if the function is denoted as $y = \hat{DS}(\mathbf{x}, \mathbf{p})$, where $\hat{DS}$ is evaluated at a collection of different sets of points: $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k \in \mathbb{R}^d$, we can obtain the vector $[f(\mathbf{x}_1), ..., f(\mathbf{x}_k)]$ to construct a surrogate model for the design parameters with the correlated objectives. The vector is randomly drawn from a prior probability distribution, where GPR takes this prior distribution to be a multivariate normal with a particular mean vector and covariance matrix. Here, the mean vector and covariance matrix are constructed by evaluating the mean function μ_0 and the covariance function Σ_0 at each pair of points x_i, x_j. The resulting prior distribution on the vector $[f(x_1), ..., f(x_k)]$ is represented in the form of a normal distribution to construct the surrogate model

$$\hat{DS}(\mathbf{x}_{1:k}) \sim \mathcal{N}\left(\mu_0(\mathbf{x}_{1:k}), \Sigma_0(\mathbf{x}_{1:k}, \mathbf{x}_{1:k})\right) \tag{S8}$$

where $\mathcal{N}(\cdot)$ denotes the normal distribution. The collection of input points is represented in compact notation: $1:k$ represents the range of $1, 2, ..., k$. The surrogate model $f(\mathbf{x})$ on $1:k$ is represented as a probability distribution given in Equation (S8). To update the model with new observations, such as after inferring the value of $f(\mathbf{x})$ at a new point $\mathbf{x}$, we let $k = l+1$ and $\mathbf{x}_k = \mathbf{x}$. The conditional distribution of $f(\mathbf{x})$ given observations $\mathbf{x}_{1:l}$ using Bayes' rule is

$$\begin{aligned}
\hat{DS}(\mathbf{x})|\hat{DS}(\mathbf{x}_{1:l}) &\sim \mathcal{N}(\mu_l(\mathbf{x}), \sigma_l^2(\mathbf{x})) \\
\mu_l(\mathbf{x}) &= \Sigma_0(\mathbf{x}, \mathbf{x}_{1:l})\Sigma_0(\mathbf{x}_{1:l}, \mathbf{x}_{1:l})^{-1}\left(f(\mathbf{x}_{1:l}) - \mu_0(\mathbf{x}_{1:l}) + \mu_0(\mathbf{x})\right) \\
\sigma_l^2 &= \Sigma_0(\mathbf{x}, \mathbf{x}) - \Sigma_0(\mathbf{x}, \mathbf{x}_{1:l})\Sigma_0(\mathbf{x}_{1:l}, \mathbf{x}_{1:l})^{-1}\Sigma_0(\mathbf{x}_{1:l}, \mathbf{x})
\end{aligned} \tag{S9}$$

where the posterior mean $\mu_l(\mathbf{x})$ is a weighted average between the prior $\mu_0(\mathbf{x})$ and the estimation from $f(\mathbf{x}_{1:l})$, where the weight applied depends on the kernel used. We used the Matérn function in the implementation.

Acquisition Function

Given the training data $[y_k, \mathbf{x}_k]$, Equation (S8) gives us the prior distribution $y_l \sim \mathcal{N}(\mu_0, \Sigma_0)$ as the surrogate. This prior and the given dataset induce a posterior: the acquisition function denoted as $\mathscr{A} : \mathscr{X} \to \mathbb{R}^+$, determines the point in $\mathscr{X}$ to be evaluated through the proxy optimization $\mathbf{x}_{best} = \arg\max_x \mathscr{A}(\mathbf{x})$. The acquisition function depends on the previous observations, which can be represented as $\mathscr{A} = \mathscr{A}(\mathbf{x}; (\mathbf{x}_l, y_l), \theta)$. Taking our previous notation, the new observation is probed through the acquisition[17]

$$\mathbf{x}_k = \mathbf{x}_{l+1} = \underset{x \in \frac{\mathscr{X}}{\mathscr{X}_l}}{\arg\max}\, \mathscr{A}\left(\mathbf{x}; (\mathbf{x}_l, y_l), \theta_m\right) \tag{S10}$$

where the input space contains the evaluation of design variables at n points: $\mathscr{X}_l := (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_l)$. In our case, $\mathscr{X}$ is acquired through running n numbers of material evaluations. We pick the GP Upper Confidence Bound (GP-UCB) as the acquisition function, exploiting the lower confidence bounds (in the case of minimizing the objective function) to construct the acquisition and minimize the regret. GP-UCB takes the form

$$\mathscr{A}\left(\mathbf{x}; (\mathbf{x}_l, y_l), \theta_m\right) := \mu_l\left(\mathbf{x}; (\mathbf{x}_l, y_l), \theta_m\right) + \kappa\sigma\left(\mathbf{x}; (\mathbf{x}_l, y_l), \theta_m\right) \tag{S11}$$

where κ is a tunable parameter balancing exploitation and exploration when constructing the surrogate model. We take $\kappa = 2.5$ as a default value in the model. Combining GPR and the acquisition function, the surrogate model can be constructed to approximate the minimum value in the design space. In our case, such BO methods are applied to obtain active surface typologies with minimal residual bacterial cells. For both material optimizations, we randomly explored the design space for 5000 material evaluations and repeated 5 times of different random seeds.

METAHEURISTIC OPTIMIZATION

Methodology Generalization

Although different methods may employ totally different notations and problem representations varied by their different optimization techniques, the general metaheuristic methods can be generalized to a "heuristic sampling" technique to sample the design space, in a non-rigorous sense. To symbolize our thoughts, assuming we have a design space map our design variables (recall we parametrize the elemental graph in Equation (1)) Θ to our objective values, one can write

$$DS : \Theta \to \mathscr{J}, \quad \Theta \in [\Theta_d^{low}, \Theta_d^{high}] \tag{S12}$$

where Θ_d^{low} and Θ_d^{high} represent the lower and upper bounds of the design variables and d indicate that the problem is d-dimensional.

We generalize metaheuristic optimizations as sampling a set of points (or particles) with d dimension, of which locations are represented as $X_1^d, X_2^d, ..., X_i^d, ..., X_m^d$ if we initialize the search with m particles. This is how we perceive the metaheuristic optimization process shown in Equation (5), where we use $\mathscr{R}$ to denote the update strategy differed by different metaheuristic methods employing different prior information. Note that we here just propose a way to simplify the representation, but not state we will use this notation for all the further methods.

Here, we roughly classified our employed metaheuristic optimization methods into two categories, Nature-inspired algorithms, and mathematics-inspired algorithms. Of course, one may further categorize simulated annealing as physics-inspired (or materials-inspired), and genetic algorithm, particle swarm optimization, etc., as biology-inspired algorithms. For the sake of simplicity and clarity, we do not further proceed. In most implementations, we pick the default hyperparameters from the library or previous literature hoping to eliminate the bias towards hyperparameters.

Nature Inspired Algorithms

Genetic Algorithm

Genetic algorithms have been recently applied to elemental composition design with rapid growth. The genetic algorithm was originally proposed inspired by the mutation and reproduction of genes. The algorithm seeks to mimic the fundamental mathematical rule to be treated as an optimization process. The process involves representing the design variables, denoted as θ, as a set of "chromosomes," denoted as $\mathbb{C}$. The chromosomes can be written as $\mathbb{C} = [n_{atom}, \xi_n, \eta]$, or $\mathbb{C} \equiv [\Theta]$.

The population, denoted as N_{pop}, is selected as a randomly chosen assortment of chromosomes, such that $N_{pop} = [\mathbb{C}_1, \mathbb{C}_2, ...]$. For instance, if the population contains five chromosomes, it can be represented as [10011]. The population then undergoes "crossover" and "mutation" operations to generate the next generation of chromosomes, to maximize or minimize a given "fitness function," denoted as $\mathscr{J}$.

The probability of selecting a chromosome for reproduction is determined by its fitness value. Specifically, the probability of choosing the ij^{th} chromosome, denoted as $\mathbb{C}_{ij}$, is given by:

$$P(\mathbb{C}_{ij}) = \left| \frac{f(\mathbb{C}_{ij})}{\sum_{k=1}^{N_{pop}} f(\mathbb{C}_k)} \right|, \tag{S13}$$

where $f(\mathbb{C}_{ij})$ represents the fitness value of the chromosome, and N_{pop} represents the total number of chromosomes in the population. Here, $f(\mathbb{C}_{ij}) \equiv MEGNet(\Theta)$, where the MEGNet material property evaluations are defined as the fitness function(s). In the context of our generalization, one can treat each chromosome as a particle X_i^D, and N_{pop} as the initial set of particles. The mutation and selection of chromosomes are analogous to the update of these particles and the hyperparameters in the update together with the probability determine the $\mathscr{R}$.

By iteratively applying crossover and mutation operations on the population and selecting chromosomes based on their fitness values, the genetic algorithm is able to identify the most promising set of design variables that maximize or minimize the fitness function. For the hyperparameters, for the single and multi-element cases, we pick the maximum iteration number as 25 and 75, respectively, and population size as 10, mutation probability as 0.3, crossover probability as 0.5, parents portion as 0.5, and uniform crossover type.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is another evolutionary optimization algorithm that can be used for materials design. PSO starts with a set of randomized design variables (similar to chromosomes in GA) and propagates these points according

to certain rules. The velocities and positions of these points in the dth dimension of the ith particle are denoted as V and X respectively. The update of these variables is given by:

$$V_i^d \leftarrow V_i^d + c_1 \texttt{rand1}_i^d (\texttt{pbest}_i^d - X_i^d) + c_2 \texttt{rand2}_i^d (\texttt{gbest}^d - X_i^d)$$
$$X_i^d \leftarrow X_i^d + V_i^d$$
(S14)

Here, $\texttt{rand1}_i^d$ and $\texttt{rand2}_i^d$ are random numbers between 0 and 1, c_1 and c_2 are acceleration coefficients, and $\texttt{pbest}_i^d$ and $\texttt{gbest}^d$ are the best positions of the ith particle and the swarm respectively in the dth dimension.

Liang et al.[30] proposed a new formulation to improve traditional PSO by changing the update of velocities. The new formulation is given by:

$$V_i^d \leftarrow w V_i^d + c_{local} \texttt{rand}_i^d (\texttt{pbest}_{fi(d)}^d - X_i^d)$$
(S15)

Here, $\mathbf{fi} = [f_i(1), f_i(2), f_i(3)]$ defines which particles' $\texttt{pbest}$ the particle i should follow. Additionally, c_{local} is a local acceleration coefficient, w is the inertia weight, and $flag_i$ represents the number of generations that the ith particle has not improved over its $\texttt{pbest}$ position. In our PSO implementations, for the single and multi-element design cases, the population size is 10, c_{local} equals 1.2, the parameter $flag_i$ equals 7, the range of the weight is $w \in [0.4, 0.9]$, the portion of parents is selected as 0.5.

Hybrid Grey Wolf Optimization

Grey wolf optimization (GWO) is a swarm intelligence algorithm that is inspired by the dynamics of hunting grey wolfs originally proposed by Mirjalili et al.[45]. To explore the design space, GWO starts with a group of particles representing initial designs, denoted as $X_i = [n_{atom}, \xi_n, \eta]$ for the ith design, with the position represented by X (Equation (5)). The GWO algorithm updates the population inspired by the hunting behavior of grey wolves. The algorithm comprises a pack of grey wolves, each representing a solution to the optimization problem. The original GWO algorithm updates the position of the ith wolf as follows:

$$X_i^d \leftarrow \frac{\left(X_\alpha^d - A_1 D_\alpha^d\right) + \left(X_\beta^d - A_2 D_\beta^d\right) + \left(X_\delta^d - A_3 D_\delta^d\right)}{3}$$
(S16)

where

$$D_\alpha = \left|C_1 X_\alpha^d - X_i^d\right|, \quad D_\beta = \left|C_2 X_\beta^d - X_i^d\right|, \quad D_\delta = \left|C_3 X_\delta^d - X_i^d\right|$$
(S17)

where X_i is the position of the ith wolf, X_α, x_β and X_δ are the positions of the α, β, and δ wolves respectively, and A_1, A_2, A_3 are random coefficients. Here, both the coefficients $\mathbf{A} = [A_1, A_2, A_3]$ and $\mathbf{C} = [C_1, C_2, C_3]$ are calculated as

$$\mathbf{A} = 2\mathbf{a}\mathbf{r}_1 - \mathbf{a}$$
$$\mathbf{C} = 2\mathbf{r}_2$$
(S18)

where components of $\mathbf{a}$ are linearly decreased from 2 to 0 during the material evaluations and $\mathbf{r}_1$ and $\mathbf{r}_2$ are random vectors in $[0,1]$. These randomization settings make the GWO a pseudo-hyper-parameter-free optimization method, as one does not need to set hyperparameters to update evaluations.

However, while the original GWO effectively explores potential areas in a large search space, its exploitation capability can be improved. To address this, Obadina et al.[31] proposed combining GWO with the whale optimization algorithm (WOA) to use the logarithmic spiral equation from WOA to enhance GWO's exploitation capability. This hybrid algorithm, denoted as GWO-WOA, updates the position of the ith wolf as the same as the original GWO, where the form of the two parameters D_α and $\mathbf{a}$ are different. For parameters D_α of the alpha wolf:

$$D_\alpha = \begin{cases} \mathscr{Y}\left|C_1 X_\alpha^d - X_i^d\right|, & \text{if } p_1 < 0.5 \\ \rho e^{bl} \cos(2\pi l)\left|C_1 X_\alpha^d - X_i^d\right|, & \text{if } p_1 \geq 0.5 \end{cases}$$
(S19)

where p_1, $\mathscr{Y}$, and ρ are random numbers within $[0,1]$. l is a random number in $[-1,1]$. b is a constant that defines the logarithmic spiral shape in WOA, which in our cases equals 1. As we mentioned in the vanilla GWO settings, $\mathbf{a}$ linearly

decreased from 2 to 0, which can be written as $\mathbf{a} = 2 - \frac{2\mathtt{iter}}{\mathtt{iter}_{max}}$, where $\mathtt{iter}$ is the iteration number and $\mathtt{iter}_{max}$ is the maximum iterations. In the hybrid GWO (or GWO-WOA), this $\mathbf{a}$ is modified to

$$\mathbf{a} = 2\left(1 - \frac{\mathtt{iter}}{2\mathtt{iter}_{max}}\right) \tag{S20}$$

where for now $\mathbf{a}$ decreases from 2 to 1. This hybrid GWO method, given a fixed $b = 1$, can be considered a hyperparameter-free method. In our implementation, we set the population size as 10 and pick epochs as 10.

Hybrid-Improved Whale Optimization Algorithm

Following our previous introduction to GWO, the Hybrid-Improved Whale Optimization Algorithm (HIWOA) is another optimization method that imitates the hunting behavior of blue whales searching for global optima. In the original WOA, the update depends on a random number p and involves a linearly decreasing parameter $\mathbf{a}$ from 2 to 0 (similar to Hybrid GWO). The location of the whale (or particle) closest to the prey influences other whales' motion, in which the update writes:

$$\begin{aligned} \text{if } p < 0.5: \quad & X_i^d \leftarrow X^* - A\left|CX^* - X_i^d\right| \\ \text{where}: \quad & A = \mathbf{a}(2\mathtt{rand}_1 - 1) \\ & C = 2\mathtt{rand}_1 \end{aligned} \tag{S21}$$

where X^* represents the whale (or particle) that is closest to its prey. This update scheme is considered as the "encircling prey". $\mathtt{rand}_1$ is a random vector of $[0,1]$. When $p \geq 0.5$, the whale position update is considered as "attacking prey", where the update writes:

$$\begin{aligned} \text{if } p \geq 0.5: \quad & X_i^d \leftarrow X^* - D_1 e^{bl}\cos(2\pi l) \\ \text{where}: \quad & D_1 = \left|X^* - X_i^d\right| \end{aligned} \tag{S22}$$

where b and l holds the same definition as those of GWO. To improve global searchability, the whales surround prey when $|A| \leq 1$ or randomly select a whale position for reference when $|A| \geq 1$. This is usually referred as "searching prey behavior", for which the update of the whale position writes:

$$\begin{aligned} & X_i^d \leftarrow X_{\mathtt{rand}} - AD_2 \\ \text{where}: \quad & D_2 = \left|CX_{\mathtt{rand}} - X_i^d\right| \end{aligned} \tag{S23}$$

where $X_{\mathtt{rand}}$ is a random whale location.

In the vanilla WOA, the update of the particle positions is strongly dependent on $\mathbf{a}$, where irrational randomized $\mathbf{a}$ values may lead to the failure of optimization. To overcome this, Tang et al.[32] proposed the following update rules for HIWOA, in which they modify $\mathbf{a}$ as:

$$\mathbf{a} = 2 + 2\cos\left(\frac{\pi}{2} \times \left(1 + \frac{\mathtt{iter}}{\mathtt{iter}_{max}}\right)\right) \tag{S24}$$

Here, w represents the inertia weight, modified by Tang et al.[32] to the form: $w = 0.5 + 0.5\left(\frac{\mathtt{iter}}{\mathtt{iter}_{max}}\right)^2$, which is a function of the iteration number. As the iteration number increases, the weight w becomes larger, leading to a stronger dependence of the population on the best-so-far position X^*. Compared to the original WOA, HIWOA can improve the local search ability and achieve better optimization performance, in theory. In this implementation for materials discovery, the population size is set as 5, and the maximum feedback is also set to 5.

Ant Colony Optimization

Ant Colony Optimization (ACO) is similar to other population-based optimization algorithms such as PSO and WOA. It employs a population of virtual ants to search for the optimal solution, where each ant represents a candidate solution. The movement of each ant is determined by the difference between its position and the position of a randomly chosen ant, which is controlled by a step length parameter. Most importantly, the algorithm presents the dynamics (or motion) of the ant by mimicking the ant's foraging behavior by detecting the intensity of the pheromone in searching the food. Here, the food sources

represent our design variables Θ, where the ants explore the food sources to readjust the design variables to further tune the material properties. If assume there is a finite set of available solution components $\mathbf{C} = \{c_{ij}\}$, representing the candidate solution that the ant is currently evaluating. Associated with c_{ij} is τ_{ij}, is the pheromone trail intensity connecting the two ants i and j. The probabilistic choice of solution components takes the form:

$$p(c_{ij}|s^p) = \frac{\tau_{ij}^{\alpha} \mathscr{W}(c_{ij})^{\beta}}{\sum_{c_{ij} \in N^p} \tau_{il}^{\alpha} \mathscr{W}(c_{il})^{\beta}} \tag{S25}$$

where $\mathscr{W}(\cdot)$ is a weighting function assigns at each construction step a heuristic value to each solution candidate c_{ij}. α and β are positive parameters determining the relation between pheromone and heuristic information. N^p represents the feasible solution set, which can be interpreted as the analogy of the population in the genetic algorithm.

In addition to the movement of ants, ACO also updates a pheromone trail to facilitate the search process. The pheromone trail represents the quality of the solution at each position in the solution space. A higher concentration of pheromone indicates a better solution quality. The pheromone trail is updated based on the following equation:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \rho\Delta\tau, & \text{if } \tau_{ij} \in s_{good} \\ (1-\rho)\tau_{ij}, & \text{otherwise} \end{cases} \tag{S26}$$

where the pheromone evaporation rate is denoted by $\rho \in (0,1]$. s_{good} is a chosen good solution, and $\Delta\tau$ is an associated interval to update τ. ACO updates the pheromone to determine the optimal value between the defined "food source" and "colony", in which one can perceive the candidate solution c_{ij} as the connection between the ant X_i^d (in our formulation in Equation (5)) and the design variables Θ in an analogous sense, and the pheromone τ_{ij} relate to the candidate solution c_{ij} from the probabilistic choice (Equation (S25)). Here, the optimizations are initialized by counting 5 samples with a population size of 10.

Differential Evolution

Differential evolution (DE) is very similar to GA, initializing the optimization set with a set of populations, and then conducting mutation, and selection, for iterative updates. The proposed algorithm commences with an initialization step, which involves generating the initial population of solutions. Recall our formulated problem in Equation (5), given a design optimization of d dimensions, the ith particle's j^{th} dimension ($j \in d$) can be initialized as:

$$X_{i,j}^0 = X_{j,L} + \texttt{rand}(X_{j,U} - X_{j,L}) \tag{S27}$$

where $X_{j,L}$ and $X_{j,U}$ are the lower and upper bounds, respectively, for the j-th variable of the problem. $\texttt{rand}$ returns to a random number in $[0,1]$.

After the initialization step, the mutation operation is performed on the population. The mutation is defined via a mutation vector that is constructed in various ways. One possible way is to use the following equation[34]:

$$v_i^G = X_{r_1}^G + F(X_{r_2}^G - X_{r_3}^G), \quad r_1 \neq r_2 \neq r_3 \neq i \tag{S28}$$

where r_1, r_2, and r_3 are randomly selected indices within the population (in the same concept as GA and ACO), distinct from each other and i, and F is a scaling factor controls the amplification of the difference vector $(X_{r_2}^G - X_{r_3}^G)$.

The next step in the algorithm is the crossover operation, which is carried out via the binomial crossover. In this operation, the target vector is combined with the mutated vector to produce the trial vector, given by:

$$u_{j,i}^G = \begin{cases} v_{j,i}^G, & \text{if } (\texttt{rand} < Cr \text{ or } j = j_{rand}) \\ X_{j,i}^G, & \text{otherwise} \end{cases} \tag{S29}$$

where j is the index of the current variable, $\texttt{rand}$ is a random number between 0 and 1, Cr is the crossover rate, and j_{rand} is a randomly selected index.

Finally, the selection process is performed, in which the newly generated trial vector is compared with the parent vector of the same index, and the best of the two vectors is chosen as the parent for the next generation. This process can be expressed mathematically as:

$$X_i^{G+1} = \begin{cases} u_i^G, & MEGNet(u_i^G) \leq MEGNet(x_i^G) \\ X_i^G, & \text{otherwise} \end{cases} \tag{S30}$$

where $MEGNet(\cdot)$ is the fitness evaluation function that measures the quality of each solution in the population, and G is the generation number. In our DE implementation, we pick the weighting (scaling) factor F as 0.7, the crossover rate is 0.9, and initialized with a population of 10.

Simulated Annealing

Simulated annealing (SA) is an optimization mimicking the annealing process of metals by defining a cooling process (i.e., decreasing of temperature) to perturb design variables by constructing a discrete-time inhomogeneous Markov chain to select the "change" of design variables and further determine the updates. The "discrete-time" characteristic can be represented by "state" (defined by epoch in the algorithm), symboled as t. Given our objective function $\mathscr{J}$ defined on a set S, and let $S^* \subset S$ be the best global optimum of $\mathscr{J}$. Additionally, we define a positive coefficient q_{ij} ($j \in S(i)$) for every i, such that $\sum_{j \in S} q_{ij} = 1$[35].

To begin the simulated annealing process, we need an initial state $X(0) \in S$. Here, $X \equiv \Theta$ represents the vector of the design variables. The update of X can be written as[35]

$$
\begin{aligned}
&\text{if } \mathscr{J}(j) \leq \mathscr{J}(i): \quad X(t) = j \\
&\text{if } \mathscr{J}(j) > \mathscr{J}(i): \quad X(t) = j, \text{ with probability } \exp\left(-\frac{\mathscr{J}(j) - \mathscr{J}(i)}{T}\right) \\
&\qquad\qquad\qquad\quad X(t) = i, \text{ otherwise}
\end{aligned}
\tag{S31}
$$

Formally, the chosen of X can be expressed as

$$
P[X(t+1) = j | X(t) = i] = \begin{cases} q_{ij}\exp\left[-\frac{1}{T(t)}\max\{0, \mathscr{J}(j) - \mathscr{J}(i)\}\right], \text{ if } j \neq i, j \in S(i) \\ 0, \text{ if } j \neq i, j \notin S(i) \end{cases}
\tag{S32}
$$

This can be interpreted as a homogeneous Markov chain $X_T(t)$, where the temperature T is held constant. The invariant probability distribution of the Markov chain X_T is given by

$$
\pi_T(i) = \frac{1}{Z_T}\exp\left(-\frac{\mathscr{J}(i)}{T(t)}\right), \quad i \in S
\tag{S33}
$$

where Z_T is a normalizing constant. For the convergence of the algorithm, it can be shown that for any schedule $T(t) = \frac{d}{\log t}$ for all t:

$$
\max_{X(0)} P[X(t) \notin S^* | X(0)] \geq \frac{A}{t^\alpha}
\tag{S34}
$$

where A and α are positive constants depending on the function $\mathscr{J}$. A comprehensive detailed derivative can be found in Ref.[35]. In our implementations, we use the following hyperparameters: the population size is 10, the maximum number of sub-iteration is 1, the mutation rate is 0.1, the mutation step size is 0.1 and the damping factor for mutation step size is 0.99. For single and multi-element material optimization cases, the move counts per individual solution are 2 and 3, the epochs are 5 and 10, and the temperature changes are $500 \rightarrow 1$ and $1000 \rightarrow 1$, respectively.

Bees Optimization Algorithm

The Bees Optimization Algorithm (BOA) is a bio-inspired optimization algorithm that shares similarities with other swarm intelligence algorithms such as PSO and GWO. It is inspired by the motion of bees. The group of artificial bees contains three sets: employed bees, onlookers, and scouts. the employed and onlooker bees are first played on "food sources" in the memory initially, and the scouts were sent to search for the new "food sources" for updates, where the food source can be treated as different sets of design variables Θ. The update rule for BOA can be expressed as follows[36]:

$$
X_i^d \leftarrow X_i^d + \phi_{rand}(X_j - X_k)
\tag{S35}
$$

where X_i, X_j, and X_k are the current positions of three randomly selected bees from the population, and ϕ_{rand} is a random number between -1 and 1.

The bees in BOA select a food source based on the probability values of the food source, which are determined by:

$$
P_i = \frac{MEGNet_i}{\sum_{n=1}^{N} MEGNet_n}
\tag{S36}
$$

Here, $MEGNet_i$ is used as a fitness evaluation surrogate, which in the context of BOA represents the measure of the quality of the food source i and N is the total number of food sources in the population. In our BOA implementations, we use the following hyperparameters: the selected site ratio is 0.5, the elite site ratio is 0.2, the selected site bees ratio is 0.1, the elite site bees ratio is 1.2, the dancing radius is 0.1 and the dancing reduction is 0.99 with a population size of 10. We select 10 and 30 for single and elemental composition optimization cases, respectively.

Mathematically Inspired Algorithms

Arithmetic Optimization Algorithm

The AOA algorithm is an optimization technique inspired by basic mathematical operations. The algorithm calculates two values, MOA and MOP, denote the math optimizer accelerated function and math optimizer probability, respectively, based on the current iteration and maximum iteration values. The MOA value is calculated as the minimum value plus the current iteration times the range of the search space divided by the maximum iteration. The MOP value is calculated as one minus the ratio of the current iteration raised to the power of 1 over a user-defined parameter α, divided by the maximum iteration raised to the power of 1 over α.

The update equations in the exploration and exploitation phases are as follows. MOA is a coefficient in the exploration phase to condition the use of different defined operators and MOP is a coefficient used in the update of the solution:

$$\text{MOA}(\text{iter}) = \text{MOA}_{min} + \text{iter} \left(\frac{\text{MOA}_{max} - \text{MOA}_{min}}{\text{iter}_{max}} \right)$$

$$\text{MOP}(\text{iter}) = 1 - \frac{(\text{iter})^{1/\alpha}}{(\text{iter}_{max})^{1/\alpha}}$$

(S37)

where iter is the current iteration and iter_{max} is the maximum iteration. MOA_{min} and MOA_{max} denote the minimum and maximum values of the accelerated function, respectively. α is a sensitivity parameter and is fixed to 5 in the implementation. The update of the solution, namely the exploration and exploitation stages, writes:

$$X_{i,j}(\text{iter}+1) = \begin{cases} best(X_j) \div (\text{MOP} + \varepsilon) \times ((UB_j - LB_j) \times \mu + LB_j), & r_2 < 0.5 \\ best(X_j) \times \text{MOP} \times ((UB_j - LB_j) \times \mu + LB_j), & otherwise \end{cases}$$

$$X_{i,j}(\text{iter}+1) = \begin{cases} best(X_j) - \text{MOP} \times ((UB_j - LB_j) \times \mu + LB_j), & r_3 < 0.5 \\ best(X_j) + \text{MOP} \times ((UB_j - LB_j) \times \mu + LB_j), & otherwise \end{cases}$$

(S38)

where $X_{i,j}(\text{iter}+1)$ denotes the update of the jth position (in the population) of the ith solution. $best(X_j)$ is the best obtained solution of jth position. UB_j and LB_j are the upper and lower bound value of the jth position. μ is a control parameter in the search process and is fixed to 0.5. These equations update the positions of particles in the search space to improve the fitness value of the objective function being optimized. For the hyperparameters, we set $\alpha = 5$ and $\mu = 0.5$. Here, MOA_{max} and MOA_{min} equals 0.9 and 0.2, respectively. The population size is 10. The epochs for single and multi-element composition optimizations are 10 and 30, respectively.

Runge Kutta Optimization

The Runge Kutta Optimization (RUN) is an optimization method inspired by the Runge Kutta method for solving differential equations. The method employs the discretization strategy to search for the optimal points in a function distribution by employing the parameters k_i to explore possible elemental compositions in our case. To begin with, recall first-order ODE with initial value problem (IVP):

$$\frac{dy}{dt} = f(x,y) \quad y(x_0) = y_0$$

(S39)

We can expand this equation on $d_t y$ (or can also be denoted as $\dot{y}$) based on Taylor series:

$$y(x + \Delta x) = y(x) + \dot{y}(x)\Delta x + \ddot{y}(x)\frac{(\Delta x)^2}{2!} + ...$$

(S40)

where Δx is the interval. Equation (S40) can be discretized using various methods. The discretization for the differentiation based on the 4th order Runge Kutta method (RK4) is written as:

$$f(x + \Delta x) = y(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta x$$

(S41)

in which the four weighted factors (k_1, k_2, k_3, k_4) are given by:

$$k_1 = \dot{y}(x) = f(x, y)$$
$$k_2 = f\left(x + \frac{\Delta x}{2}, y + k_1 \frac{\Delta x}{2}\right)$$
$$k_3 = f\left(x + \frac{\Delta x}{2}, y + k_2 \frac{\Delta x}{2}\right) \tag{S42}$$
$$k_4 = f(x + \Delta x, y + k_3 \Delta x)$$

where k_1, k_2, k_3, k_4 are the four increments that determine the slope of the intervals of different orders. This is the core ideology of Runge Kutta's discretization for numerically solving the ODE in Equation (S39).

Recall our formulated problem and analogy of using particles to sample the design space in Equation (5). Under the optimization, and inspired by the Runge Kutta discretization, the RUN optimization update of particle i (i in population N^p) with d^{th} dimension follows:

$$X_i^d = L^d + \texttt{rand}(U^d - L^d) \tag{S43}$$

where $\texttt{rand}$ is a random number between $[0, 1]$; L^d and U^d are the lower and upper bounds of the dth variable of the problem ($d = 1, 2, ..., D$) and here D can be equal 2 or 3. U^d and L^d are thence set correspondingly given our discussion about Equation (1).

One of the key innovations and strengths, discussed in the main article, is that the RUN algorithm is hyperparameter-free, i.e., independent of the setting of hyperparameters. This is enabled by randomized parameters within the optimization process. The optimization algorithm can be initiated first by randomizing the four weighted factors parameters:

$$k_1 = \frac{1}{2\Delta X}(\texttt{rand} \times X_w - u \times X_b), \quad u = \texttt{round}(1 + \texttt{rand}) \times (1 - \texttt{rand})$$
$$k_2 = \frac{1}{2\Delta X}(\texttt{rand}(X_w + \texttt{rand}_1 k_1 \Delta x) - (uX_b + \texttt{rand}_2 k_1 \Delta X))$$
$$k_3 = \frac{1}{2\Delta X}\left(\texttt{rand}\left(X_w + \texttt{rand}_1\left(\frac{1}{2}k_2\right)\Delta X\right) - \left(uX_b + \texttt{rand}_2\left(\frac{1}{2}k_2\right)\Delta X\right)\right) \tag{S44}$$
$$k_4 = \frac{1}{\Delta X}(\texttt{rand}(X_w + \texttt{rand}_1 k_3 \Delta X) - (uX_b + \texttt{rand}_2 k_3 \Delta X))$$

where u is a random parameter to increase the importance of the best solution X_b. X_w and X_b are the worst and best solutions determined based on three random solutions $(X_{r_1}, X_{r_2}, X_{r_3})$ from the population, are determined from

$$\begin{cases} X_b = X_i, X_w = X_{bi}, \ MEGNet(X_n) < MEGNet(X_{bi}) \\ X_b = X_{bi}, X_w = X_i, \ MEGNet(X_n) \geq MEGNet(X_{bi}) \end{cases} \tag{S45}$$

where X_{bi} is the best random solution in the population. The update schemes follow the rule

$$X_{i+1} \leftarrow \begin{cases} X_c + SF \times SM + \mu \times X_s, \ \texttt{rand} < 0.5 \\ X_m + SF \times SM + \mu \times X_s, \ \texttt{rand} \geq 0.5 \end{cases} \tag{S46}$$

where $\mu = 0.5 + 0.1 \times \mathcal{N}(\texttt{rand})$, in which $\mathcal{N}(\texttt{rand})$ is a random number with a normal distribution. The formulation for X_s and X_s' are written as:

$$X_s = \mathcal{N}(\texttt{rand})(X_m - X_c), \quad X_s' = \mathcal{N}(\texttt{rand})(X_{r_1} - X_{r_2}) \tag{S47}$$

and in which X_m and X_c are calculated as

$$X_c = \phi \times X_n + (1 - \phi) \times X_{r_1}$$
$$X_m = \phi \times X_{best} + (1 - \phi) \times X_{lbest} \tag{S48}$$

Here, SF and SM are given by

$$SM = \frac{1}{6}(x_{RK})\Delta x, \text{ where } x_{RK} = k_1 + 2 \times k_2 + 2 \times k_3 + k_4$$
$$SF = 2(0.5 - \texttt{rand}) \times \mathscr{F}, \text{ where } \mathscr{F} = a \times e^{-b \times \texttt{rand} \times \left(\frac{i}{max_i}\right)} \tag{S49}$$

where a and b are two constant numbers. Herein is only a simplified expression, the full derivation can be checked in Ref.[38]. Here, we treated this problem as a hyperparameter-free scenario.