NEXTorch: A Design and Bayesian Optimization Toolkit for Chemical Sciences and Engineering

Yifan Wang,[1,2†] Tai-Ying Chen,[1,2†] and Dionisios G. Vlachos[1,2*]

[1]Department of Chemical and Biomolecular Engineering, 150 Academy St., University of Delaware, Newark, Delaware 19716, United States
[2]Catalysis Center for Energy Innovation, RAPID Manufacturing Institute, and Delaware Energy Institute (DEI), 221 Academy St., University of Delaware, Newark, Delaware 19716, United States

[†]These authors contributed equally.

*Corresponding author: vlachos@udel.edu

Abstract
Automation and optimization of chemical systems require well-inform decisions on what experiments to run to reduce time, materials, and/or computations. Data-driven active learning algorithms have emerged as valuable tools to solve such tasks. Bayesian optimization, a sequential global optimization approach, is a popular active-learning framework. Past studies have demonstrated its efficiency in solving chemistry and engineering problems. We introduce NEXTorch, a library in Python/PyTorch, to facilitate laboratory or computational design using Bayesian optimization. NEXTorch offers fast predictive modeling, flexible optimization loops, visualization capabilities, easy interfacing with legacy software, and multiple types of parameters and data type conversions. It provides GPU acceleration, parallelization, and state-of-the-art Bayesian Optimization algorithms and supports both automated and human-in-the-loop optimization. The comprehensive online documentation introduces Bayesian optimization theory and several examples from catalyst synthesis, reaction condition optimization, parameter estimation, and reactor geometry optimization. NEXTorch is open-source and available on GitHub.

Keywords
Design of experiments, Bayesian optimization, response surface, statistical learning, active learning, adaptive experimentation

Introduction

Data generation in chemical sciences can be expensive and time-consuming when performing computations or experiments. Chemical systems are often complex, multidimensional, and include various interacting parameters. These traits make the autonomous discovery,[1] predictive modeling, and optimization challenging. Design of experiments (DOE) has been employed to determine the relationships between input parameters and output responses. However, the traditional DOE is difficult to scale to high dimensional problems where the cost of fine-tuning parameters and locating optima is prohibitive. Active learning refers to the idea of a machine learning algorithm "learning" from data, proposing next experiments or calculations, and improving prediction accuracy with fewer training data or lower cost.[2] Bayesian Optimization (BO), an active learning framework, often used to tune hyperparameters in machine learning models, has seen a rise in its applications to various chemical science fields, including parameter tuning for density functional theory (DFT) calculations,[3] catalyst synthesis,[4,5] high throughput reactions,[6] and computational material discovery.[7–9] Its close variant, kriging,[10] originating in geostatistics, has also been widely applied in process engineering.[11,12] Fundamentally, BO is a sequential global optimization approach consisting of two essential parts: (1) a surrogate model (often a Gaussian process[13]) to approximate the system response and (2) an acquisition function to suggest new experiments to run. The method is designed to balance the exploration of uncertainty and exploitation of current knowledge in the parameter space. Previously, researchers have designed similar global optimization methods for chemistry applications, such as the Stable Noisy Optimization by Branch and Fit (SNOBFIT),[14,15] a genetic algorithm (GA),[16,17] and reinforcement learning.[18] Some methods resemble GP-based BO with different surrogate models (e.g., neural networks) and customized acquisition functions.[19,20] Here, we focus our discussion on GP-based BO because its simple architecture scales well for medium-sized systems with less than 20 dimensions,[21] requires less data and training time, and supports native uncertainty quantification. GP-based BO often outperforms experts or other algorithms in locating optima and producing accurate surrogate models with minimal customization effort.[6] There is no doubt that BO could supercharge the field towards faster adoption of automation.

The machine learning community has developed several BO software tools, with most of them interfaced in Python. We curate a list of open-source BO packages and provide further discussion in Table S1. Spearmint[22] and GyOpt[23] are among the early works that make BO accessible to end-users. Recently, some packages are built on popular machine learning frameworks, such as PyTorch and TensorFlow, to benefit from fast matrix operations, parallelization, and GPU acceleration. Examples include BoTorch[24] and GPflowOpt[25]. BoTorch stands out since it naturally supports parallel optimization, Monte Carlo acquisition functions, and advanced multi-task and multi-objective optimization. The PyTorch backend also makes it suitable for easy experimentation and fast prototyping. However, most tools are designed for AI researchers or software engineers, often requiring a steep learning curve. The workflow can also be less transparent to end-users. Occasionally, design choices are made that keep humans out of the optimization loop.[26,27] The above reasons make these tools difficult to extend to chemistry or engineering problems, where domain knowledge is essential. We have seen such an attempt by the authors of edbo[6] (a Bayesian reaction optimization package). They performed extensive testing and benchmarking to showcase the effectiveness of the method.[6] However, the software is still based on command-line scripts, and clear documentation is lacking. Edbo lacks hardware acceleration or the latest state-of-the-art BO methods.

From a practical perspective, we believe a BO tool should be scalable, flexible, and accessible to the end-users, i.e., chemists and engineers. Hence, we build NEXTorch (Next EXperiment toolkit in PyTorch), extending the capabilities of BoTorch, to democratize the use of BO in chemical sciences. NEXTorch is unique for several reasons. First, it benefits from the modern architecture and a variety of models and functions offered by BoTorch. Second, going beyond BoTorch, NEXTorch provides connections to real-

world problems, including automatic parameter scaling, data type conversions, and visualization. These features allow human-in-the-loop design where decision-making on generating future data is aided by domain knowledge. Utilizing these features, NEXTorch can assist not only chemical synthesis in laboratory experiments[4] but also the multiscale computational tasks from molecular-scale design, such as heterogeneous catalyst calculations[17] and homogeneous (ligand) catalyst discovery, to reactor-scale optimization, i.e., automatic reactor optimization with computational fluid dynamics (CFD).[28] Third, NEXTorch is modular, making it easy to extend to other frameworks. It also serves as a library for learning the theory and implementing BO. We provide clear documentation and guided examples at https://nextorch.readthedocs.io/en/latest/. We believe its easy use could serve the community, including experimentalists with little or no programming background.

In the first section of this paper, we review the theoretical foundations of NEXTorch. In the second, we describe the design of the software and the data structures. In the third, we demonstrate how NEXTorch can optimize a plug flow reactor (PFR) performance.

Theory

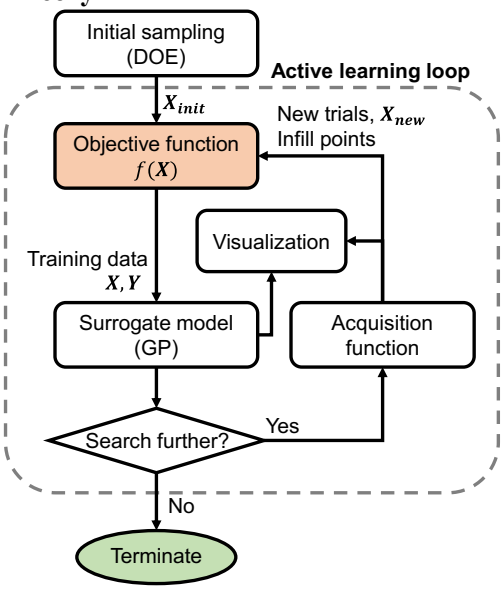

Figure 1. Overview of the active learning framework.

Overview. Our framework (Figure 1) integrates DOE, BO, and surrogate modeling. The goal is to optimize the function of interest, i.e., objective function $f(\boldsymbol{X})$. $\boldsymbol{X}$ denotes the input variables (parameters). Here, $\boldsymbol{X} = \boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \ldots \boldsymbol{x_d}$; each of $\boldsymbol{x_i}$ is a vector $\boldsymbol{x_i} = (x_{1i}, x_{2i}, \ldots, x_{ni})^T$; n is the number of sample points, and d is the dimensionality of the input. Each parameter is bounded, and the resulting multidimensional space is called the design space A, i.e., $\boldsymbol{X} \in \mathrm{A} \in \mathbb{R}^d$. The objective function is evaluated using complex computer simulations or experiments. The responses, $\boldsymbol{Y}$, are usually expensive, time-consuming, or difficult to calculate or measure. A set of initial sampling $\boldsymbol{X_{init}}$ data is generated using DOE. These sampling points are passed to evaluate f. One collects the data $D = (X, Y)$ and use it to train a cheap-to-evaluate surrogate model $\hat{f}$ (e.g., a Gaussian process). Next, an acquisition function gives the new sampling points (i.e., infill points, $\boldsymbol{X_{new}}$) for achieving an optimization goal. At this stage, one could choose to visualize the response

surfaces using the surrogate model or the infill point locations in the design space. A new dataset is collected by evaluating f at $\boldsymbol{X_{new}}$ and surrogate model $\hat{f}$ is updated. This process is repeated until the accuracy of $\hat{f}$ is satisfactory or the optimum location $\boldsymbol{x}^* = \underset{\boldsymbol{x}\in A}{\operatorname{argmax}} f(\boldsymbol{x})$ is found.

Design of experiments (DOE). Standard methods include full factorial design, completely randomized design, and Latin hypercube sampling (LHS), described in statistics books.[29] Here, we use LHS heavily due to its near-random design and efficient space-filling abilities.[30]

Gaussian process (GP). A GP model constructs a joint probability distribution over the variables, assuming a multivariate Gaussian distribution. A GP is determined from a mean function $\mu(\boldsymbol{X})$ and a covariance kernel function $\sum(\boldsymbol{X}, \boldsymbol{X}')$.[13] A Matern covariance function is typically used as the kernel function,[21]

$$C_\nu(\mathrm{d}) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2}\nu \frac{d}{\rho})^\nu K_\nu (\sqrt{2}\nu \frac{d}{\rho}) \quad (1)$$

where d represents the distance between two points, σ represents the standard deviation, Γ is the gamma function, K_ν is the modified Bessel function, and ν and ρ are non-negative parameters.

Acquisition function. The acquisition function is applied to obtain the new sampling point $\boldsymbol{X_{new}}$. It measures the value of evaluating the objective function at $\boldsymbol{X_{new}}$, based on the current posterior distribution over $\hat{f}$. The most commonly used acquisition function is the expected improvement (EI). The EI is the expectation taken under the posterior distribution $\hat{f}$ of the improvement of the new observation at $\boldsymbol{X}$ over the current best observation f^*:

$$EI(\boldsymbol{X}) = \mathbb{E}[max(\hat{f}(\boldsymbol{X}) - f^*, 0)] \quad (2)$$

Aside from the EI, there are also other acquisition functions for a single objective BO available in NEXTorch, including probability of improvement (PI), upper confidence bound (UCB), and their Monte Carlo variants (qEI, qPI, qUCB).

Parameter types. Parameters and the associated data are generally continuous, categorical, and ordinal. Continuous parameters are numerical and take any real value in a range. Categorical are non-numeric and are denoted by words, text, or symbols. Ordinal parameters take ordered discrete values. Traditionally, BO is designed for systems with all continuous parameters. However, depending on the problem, the resulting design space can be discrete or mixed (continuous-discrete). If the number of discrete combinations is low, one possible solution is to enumerate the values and optimize the continuous parameters for each. Without loss of generality, we use a continuous relaxation, where the acquisition function is optimized and rounded to the available values. For ordinal parameters, these values are the ordered discrete values. For categorical parameters, we encode the categories with integers from 0 to $n_{category} - 1$ and then perform continuous relaxation in the encoding space. Since a parameter can be approximated as continuous, given a high order discretization, this approach usually works well for problems with many discrete combinations.

Multi-objective optimization (MOO). MOO involves minimizing (maximizing) the multi-objective function. The optimal is not a single point but a set of solutions defining the best tradeoff between competing objectives. The goodness of the solution is determined by the dominance, where $\boldsymbol{X_1}$ dominates $\boldsymbol{X_2}$ when $\boldsymbol{X_1}$ is not worse than $\boldsymbol{X_2}$ in all objectives, and $\boldsymbol{X_1}$ is strictly better than $\boldsymbol{X_2}$ in at least one objective. A Pareto optimal set defines points not dominated by each other, where the boundary is the Pareto front.

The weighte-sum method is a classic MOO method that scalarizes a set of objectives $\{\boldsymbol{Y_1}, \boldsymbol{Y_2}, \ldots \boldsymbol{Y_i}, \ldots \boldsymbol{Y_M}\}$ into a new objective $\boldsymbol{Y_{mod}}$ by multiplying each with user-defined weights $\{w_1, w_2, \ldots w_i, \ldots w_M\}$, where $\boldsymbol{Y_{mod}} = \sum_{i=1}^{M} w_i \boldsymbol{Y_i}$. The MOO becomes a single-objective optimization

An alternative is to use the expected hypervolume improvement (EHVI) as the acquisition function. A hypervolume indicator (HV) approximates the Pareto set and EHVI evaluates its EI. In NEXTorch, one can use either the weighted-sum method or the Monte Carlo EHVI (qEHVI) method as an acquisition function. The methods are detailed in the Supporting Information (SI).

Software Design

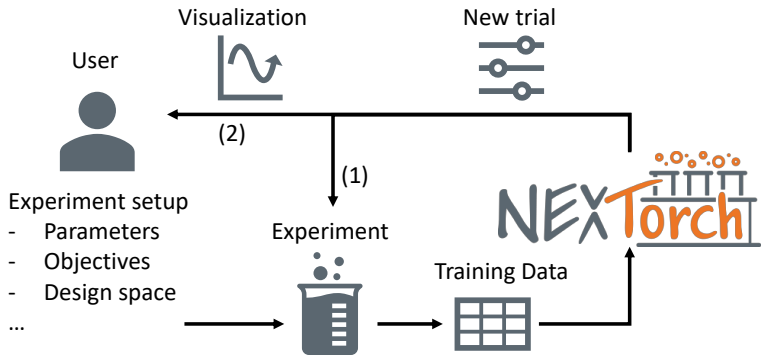

Figure 2. High-level workflow in NEXTorch. (1) Automated optimization; (2) Human-in-the-loop optimization.

The NEXTorch software package is structured in a similar way to the active learning framework. Figure 2 shows the high-level workflow. Initially, users identify the parameters and objectives and frame the optimization problem. The critical information includes the ranges and types (categorical, ordinal, continuous, or mixed) of each parameter. It is helpful to know the sensitivity of the parameters by performing exploratory data analysis. Depending on the availability of the objective function, NEXTorch supports two types of optimization: (1) automated optimization, where the analytical form of the objective function is known and provided to the software (in the form of a Python object). This is often the case in modeling and simulations; however, closed-form objective functions may also not be available in complex models, a situation that falls under the next type. And (2) human-in-the-loop optimization, where the objective function is unknown, as happens in laboratory experiments. We call the action of generating data from the objective function an "experiment," which is also the name of the core class in NEXTorch, irrespective of whether this is done in the laboratory or on a computer. In (1), data is passed through the loop, and new experiments are conducted as suggested by the acquisition function automatically. In (2), visualization could help the users decide whether to carry on the experiments or adjust the experimental setup. The users are left to perform new trials, i.e., conduct further experiments and supply additional data. NEXTorch reads CSV or Excel files and exports the data in the same format.

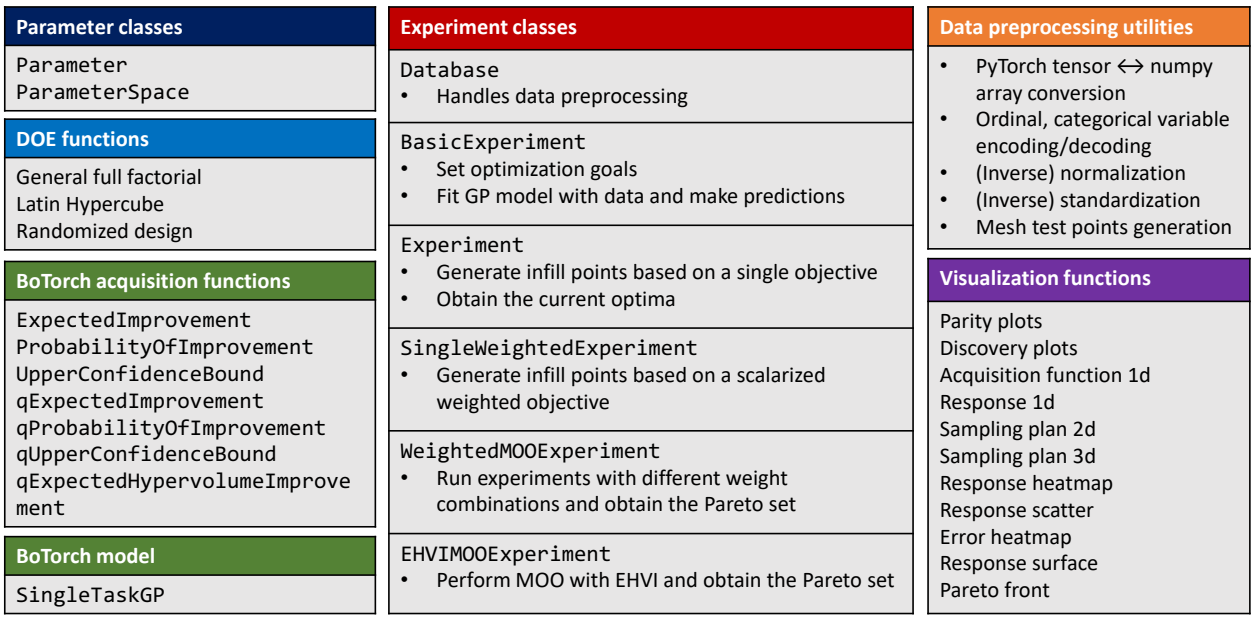

Figure 3. Main classes and functions in NEXTorch.

NEXTorch is implemented in a modular fashion. Figure 3 summarizes its available classes, modules, and functions. The *Parameter* class stores the range and type of each parameter. The *ParameterSpace* class consists of all *Parameter* classes. In the beginning, users can choose a DOE function to construct a sampling plan and get the initial responses. Next, the data is passed to an *Experiment* class. We integrate most acquisition functions and GP models from the upstream BoTorch with the *Experiment* classes. The *Experiment* classes are the core of NEXTorch, where data is preprocessed, GP models are trained and used for prediction, and the optima are located. There are several variants of the *Experiment* class depending on the application. At the higher level, users interact with the *Experiment* class, *WeightedMOOExperiment* class, and *EHVIMOOExperiment* for sequential single-objective optimization, MOO with a weighted sum method, and MOO with EHVI. Moreover, NEXTorch also offers various visualization functions for the sampling plans, response surfaces/heatmaps, acquisition functions, Pareto fronts, etc., up to three dimensions.

Applications

To demonstrate the easy implementation and modularity of NEXTorch, we provide comprehensive examples on the documentation page (https://nextorch.readthedocs.io/en/latest/user/examples.html). The examples cover various types of experiments, designs, BO methods, and parameter types from reaction engineering and catalysis synthesis. A full list of examples and their description is in Table S2.

Here, we briefly demonstrate the overall optimization pipeline for two specific cases with code snippets (shown in the SI) on single-objective (example 5) and multi-objective optimization (examples 7 and 11), respectively. The examples are built on prior kinetics studies of fructose conversion to 5-hydroxymethylfurfural (HMF) in a PFR.[31,32] Fructose, produced from biomass, can be converted to valuable fuels and chemicals through a chain of reactions. HMF is an essential intermediate in this supply chain derived from fructose through acid-catalyzed dehydration.[33,34] Besides, side reactions produce byproducts,

such as humins, formic acid (FA), and levulinic acid (LA). A schematic of the PFR is shown in Figure 4a. In this example and similar reaction problems, it is essential to maximize the HMF yield while maintaining a high selectivity of HMF to reduce downstream separation costs. The objective functions are derived from a kinetic model consisting of a set of ordinary differential equations (ODEs) that compute the concentrations of each species at a given time in a batch reactor or location in a PFR (as is the case here).[1] The input parameters ($\boldsymbol{X}$), include the reaction temperature (T), pH, and final residence time (tf). Their ranges are 140-200 °C, 0-1, and 0.01 to 100 min, respectively.

Case 1 – Renewable Platform Chemical (HMF) Yield Optimization

In this case, the goal is to maximize a single objective, the HMF yield (Y), in the three-dimensional continuous space. The objective function (PFR_yield) is a Python function object. The first steps involve importing NEXTorch modules, defining the parameter spac and a DOE sampling plan to obtain an initial set of responses. Code example S1 illustrates these steps. Next, we initialize an *Experiment* object, input the initial data, and set the goal as maximization (Code example S2). 54 additional BO trials using the default acquisition function (EI), where one infill point is generated in each iteration, are obtained (Code example S3). An explicit (human-in-the-loop) loop allows users to access the values of parameters (X_new_real) and responses (Y_new_real) in real units. The final set of optimal values can also be extracted and further exported to standard data storage files.

NEXTorch also offers a variety of visualization options (Figure 4). We compare the aforementioned LHS+BO approach with two other sampling plans with the same total number of 64 points: (1) full factorial design with four levels in each dimension and (2) completely random sampling. In each approach, we construct an *Experiment* object and train a GP model based on all points. Interestingly, BO samples more in the low pH and low tf region where the potential optimum is (Figure 4g). A two-dimensional plane at pH=0.7 is shown in the three-dimensional space. On this plane, the response surfaces (Figure 4b, e, h) and surface plots (Figure 4c, f, i) suggest that the surrogate model predictions from LHS+BO are generally closer to the ground truth. Next, we plot the best yield observed sequentially (Figure S1). The method also locates a higher optimal value compared to others. This example showcases that LHS+BO efficiently converges to the optimum and produces accurate surrogate models at an affordable computational cost. The runtime of core BO functions (GP training and acquisition function evaluation) completes in seconds per iteration on a laptop CPU, negligible compared to the cost of the objective function evaluation. Even though we use a simple PFR model here with a relatively small reaction network of overall reactions, the computational cost escalates if one uses instead complex parameter estimations, computational fluid dynamics (CFD) simulations and/or complex reaction networks of thousands and millions of elementary reactions. The central idea and workflow remain the same. We recently demonstrated such an example of optimization in five dimensions (reactor geometry and operating conditions) of a microwave reactor using CFD.[28]

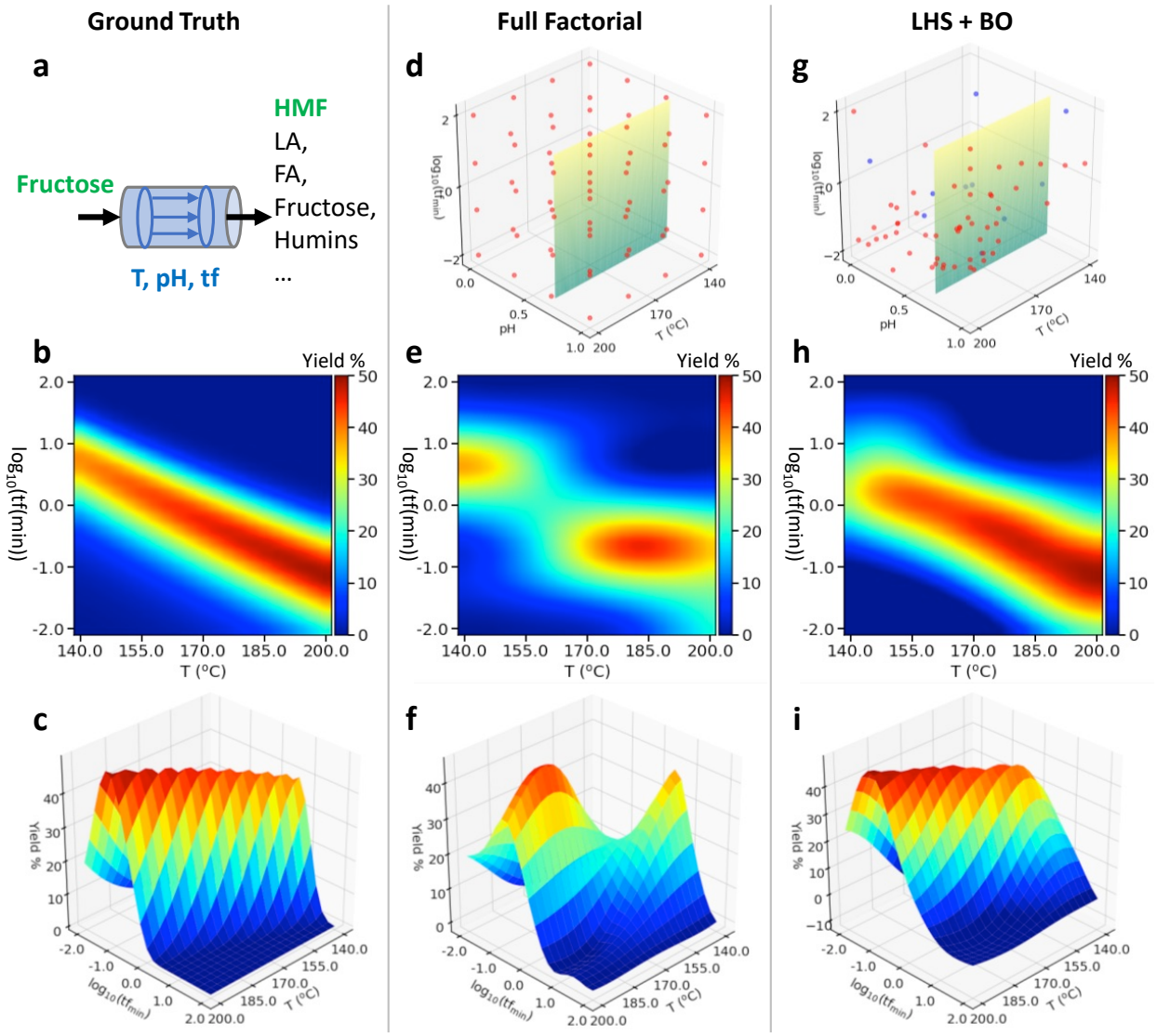

Figure 4. Optimization of HMF yield. (a) Scheme of a PFR that converts fructose to HMF via acid-catalyzed dehydration. (b) Response surface, and (c) surface plot for the ground truth model at pH = 0.7. (d) Sampling plan, (e) response surface, and (f) surface plot at pH = 0.7 for the full factorial approach. (g) Sampling plan, (h) response surface, and (i) surface plot at pH = 0.7 for the LHS+BO approach. In (d) and (g), the sampling points are visualized in three dimensions, and the two-dimensional plan at pH=0.7 is shown in green. In (g), the initial points from LHS are shown in blue, and the infill points in red.

Case 2 – Simultaneous Two Variable- (HMF Yield and Selectivity) Optimization

While aiming to maximize the HMF yield, similar to Case 1, it is crucial to maintain a high HMF selectivity to reduce the separation costs. In this regard, two objectives, yield and selectivity, need to be optimized. These two variables may not be maximized simultaneously, indicating that an increase in yield may lead to a decrease in selectivity and vice versa. Therefore, we perform the multi-objective optimization to find out a Pareto front.

As in Case 1, the optimization is performed in the three-dimensional continuous space, and the objective function (PFR_yield) is a Python function object, which returns the yield and selectivity of HMF. We initialize an *EHVIMOOExperiment* object, which uses the Monte Carlo EHVI as the acquisition function, and set the reference point. The reference point defines slightly worse values than the lower bound of the objective values that are acceptable for each objective (Code example S4). Alternatively, we can initialize a *WeightedMOOExperiment* object, which handles multiple weight combinations to perform multi-objective optimizations using the weighted-sum method automatically (Code example S5).

Then, we perform multi-objective optimization using either method, as in Code example S6. We perform 20 trials using the Monte Carlo EHVI acquisition function, and the loop is explicitly defined, where the values of parameters (X_new_real) and responses (Y_new_real) are fed explicitly at each iteration. On the other hand, the NEXTorch also provides automatic optimization, as shown in Code example 6 for the weighted-sum method, where the loop is defined implicitly in the software. Either implicitly or explicitly defined optimization loops can be used in both methods. The final set of optimal values can be extracted and further exported to standard data storage files as in the single-objective optimization.

The Pareto optimal can be visualized using the tools of NEXTorch (Figure 5). We compare the Pareto front from the two methods, where both sets of optimal lay in the region that HMF selectivity is greater or equal to the HMF yield and share a similar Pareto front. Interestingly, the Pareto front obtained from the Monte Carlo EHVI method covers a larger range than the weighted sum method. This is attributed to the limitation of the pre-assigned weights to each objective in the weighted-sum method, which are not guaranteed to include the desired region of the design space. Moreover, the computational time of the weighted sum method (12.9 min) is orders of magnitude higher than that of the Monte Carlo EHVI method (0.2 min) because the objective function is called many more times ($n\_trials * n\_exp = 420$). This example showcases that the Monte Carlo EHVI method efficiently finds out the Pareto optimum and produces an accurate surrogate model at a lower cost.

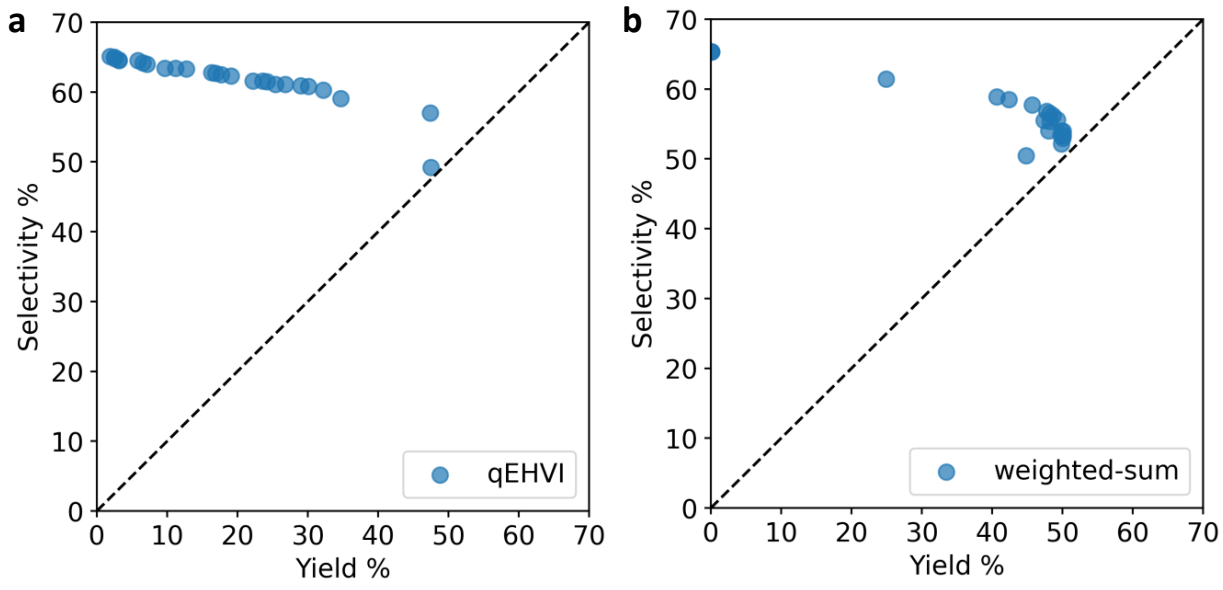

Figure 5. Pareto optimum of the HMF yield and selectivity using (a) the Monte Carlo EHVI and (b) weighted sum method. The diagonal line is used for visualization reasons and indicates the locus of 100% conversion conditions. As the points approach the diagonal, the conversion increases. The selectivity drops only slightly as the conversion increases until high conversions are reached.

Conclusions

Optimization and predictive modeling are ubiquitous in chemical sciences. Bayesian Optimization-based active learning is a powerful approach for such tasks. The NEXTorch library was created to enable streamlined and efficient Bayesian Optimization and surrogate modeling for chemical sciences and engineering applications. Its backend from BoTorch/PyTorch enables GPU acceleration, parallelization, and state-of-the-art Bayesian Optimization algorithms. The modular and flexible design of NEXTorch expands its capabilities and connects with real-world problems. Specifically, it can deal with mixed types of parameters and data type conversions, supports both automated and human-in-the-loop optimization, and offers various visualization options. It can be used in chemical synthesis, molecular modeling, reaction condition optimization, parameter estimation, and reactor geometry optimization, to mention a few examples. Such tasks can easily be performed without extensive programming effort so that the user can focus on domain-specific questions. Moreover, NEXTorch enables interface with the commonly used simulation tools in the reaction engineering field, such as CFD and multiphysics simulations, for automatic optimization. Including surrogate models, such as random forest and multi-fidelity models, extending the acquisition function list, and adding a graphical user interface, are important future directions. We believe the adoption of Bayesian Optimization in daily laboratory or computing practices could advance the field towards a more efficient and automated future.

ASSOCIATED CONTENT

Supporting Information

The supporting information is available free of charge. (1) Comparison of existing Python-based Bayesian Optimization or kriging packages; (2) Details of multi-objective optimization methods; (3) List of examples in the NEXTorch online documentation; (4) Comparison of the best yield value attained using different sample methods; and (5) Code examples for case 1 and 2.

Software Availability

NEXTorch is publicly available free of charge at https://github.com/VlachosGroup/nextorch. It can be installed using pip, the standard package-management system in Python. The online documentation is available at https://nextorch.readthedocs.io/en/latest.

Conflicts of interest

There are no conflicts to declare.

Acknowledgments

Funding from the RAPID manufacturing institute, supported by the Department of Energy (DOE) Advanced Manufacturing Office (AMO), award numbers DE-EE0007888-9.5 is gratefully acknowledged. The Delaware Energy Institute gratefully acknowledges the support and partnership of the State of Delaware in furthering the essential scientific research being conducted through the RAPID projects. The authors thank Jaynell Keely for assistance with the graphics.

References

(1) Coley, C. W.; Eyke, N. S.; Jensen, K. F. Autonomous Discovery in the Chemical Sciences Part II: Outlook. *Angew. Chem., Int. Ed.* **2020**, *59*, 23414–23436.

(2) Settles, B. *Active Learning Literature Survey*; 2009.

(3) Yu, M.; Yang, S.; Wu, C.; Marom, N. Machine Learning the Hubbard U Parameter in DFT+U Using Bayesian Optimization. *npj Comput. Mater.* **2020**, *6*.

(4) Ebikade, E. O.; Wang, Y.; Samulewicz, N.; Hasa, B.; Vlachos, D. Active Learning-Driven Quantitative Synthesis–Structure–Property Relations for Improving Performance and Revealing Active Sites of Nitrogen-Doped Carbon for the Hydrogen Evolution Reaction. *React. Chem. Eng.* **2020**.

(5) Burger, B.; Maffettone, P. M.; Gusev, V. V.; Aitchison, C. M.; Bai, Y.; Wang, X.; Li, X.; Alston, B. M.; Li, B.; Clowes, R.; Rankin, N.; Harris, B.; Sprick, R. S.; Cooper, A. I. A Mobile Robotic Chemist. *Nature* **2020**, *583*, 237–241.

(6) Shields, B. J.; Stevens, J.; Li, J.; Parasram, M.; Damani, F.; Alvarado, J. I. M.; Janey, J. M.; Adams, R. P.; Doyle, A. G. Bayesian Reaction Optimization as a Tool for Chemical Synthesis. *Nature* **2021**, *590*, 89–96.

(7) del Rosario, Z.; Rupp, M.; Kim, Y.; Antono, E.; Ling, J. Assessing the Frontier: Active Learning, Model Accuracy, and Multi-Objective Materials Discovery and Optimization. **2019**, *024112*.

(8) Tran, A.; Tranchida, J.; Wildey, T.; Thompson, A. P. Multi-Fidelity Machine-Learning with Uncertainty Quantification and Bayesian Optimization for Materials Design: Application to Ternary Random Alloys. *J. Chem. Phys.* **2020**, *153*.

(9) Montoya, J. H.; Winther, K. T.; Flores, R. A.; Bligaard, T.; Hummelshøj, J. S.; Aykol, M. Autonomous Intelligent Agents for Accelerated Materials Discovery. *Chem. Sci.* **2020**, *11*, 8517–8532.

(10) Forrester, A. I. J.; Sóbester, A.; Keane, A. J. *Engineering Design via Surrogate Modelling*; Wiley, 2008.

(11) Jones, D. R.; Schonlau, M.; W. J. Welch. Efficient Global Optimization of Expensive Black-Box Functions," , Vol. 13, No. 4, Pp. 455-492, 1998. *J. Glob. Optim.* **1998**, *13*, 455–492.

(12) Boukouvala, F.; Muzzio, F. J.; Ierapetritou, M. G. Dynamic Data-Driven Modeling of Pharmaceutical Processes. *Ind. Eng. Chem. Res.* **2011**, *50*, 6743–6754.

(13) Rasmussen, C. E.; Williams, C. K. I. *Gaussian Processes for Machine Learning*; 2000; Vol. 7.

(14) Bédard, A.; Adamo, A.; Aroh, K. C.; Russell, M. G.; Bedermann, A. A.; Torosian, J.; Yue, B.; Jensen, K. F.; Jamison, T. F. Reconfigurable System for Automated Optimization of Diverse Chemical Reactions. *Science* **2018**, *361*, 1220–1225.

(15) Mateos, C.; Nieves-Remacha, M. J.; Rincón, J. A. Automated Platforms for Reaction Self-Optimization in Flow. *React. Chem. Eng.* **2019**, *4*, 1536–1544.

(16) Berardo, E.; Turcani, L.; Miklitz, M.; Jelfs, K. E. An Evolutionary Algorithm for the Discovery of Porous Organic Cages. *Chem. Sci.* **2018**, *9*, 8513–8527.

(17) Wang, Y.; Su, Y.-Q.; Hensen, E. J. M.; Vlachos, D. G. Finite-Temperature Structures of Supported Subnanometer Catalysts Inferred via Statistical Learning and Genetic Algorithm-Based Optimization. *ACS Nano* **2020**, *14*, 13995–14007.

(18) Zhou, Z.; Li, X.; Zare, R. N. Optimizing Chemical Reactions with Deep Reinforcement Learning. *ACS Cent. Sci.* **2017**, *3*, 1337–1344.

(19) Häse, F.; Roch, L. M.; Kreisbeck, C.; Aspuru-Guzik, A. Phoenics: A Bayesian Optimizer for Chemistry. *ACS Cent. Sci.* **2018**, *4*, 1134–1145.

(20) Janet, J. P.; Ramesh, S.; Duan, C.; Kulik, H. J. Accurate Multiobjective Design in a Space of Millions of Transition Metal Complexes with Neural-Network-Driven Efficient Global Optimization. *ACS Cent. Sci.* **2020**.

(21) Frazier, P. I. A Tutorial on Bayesian Optimization. **2018**.

(22) Snoek, J.; Larochelle, H.; Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. **2012**, 1–12.

(23) GPyOpt. GPyOpt: A Bayesian Optimization framework in python http://github.com/SheffieldML/GPyOpt.

(24) Balandat, M.; Karrer, B.; Jiang, D. R.; Daulton, S.; Letham, B.; Wilson, A. G.; Bakshy, E. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. **2019**.

(25) Knudde, N.; Van Der Herten, J.; Dhaene, T.; Couckuyt, I. GPflowOpt: A Bayesian Optimization Library Using TensorFlow. *arXiv* **2017**, 0–1.

(26) Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; De Freitas, N. Taking the Human out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* **2016**, *104*, 148–175.

(27) Kandasamy, K.; Vysyaraju, K. R.; Neiswanger, W.; Paria, B.; Collins, C. R.; Schneider, J.; Poczos, B.; Xing, E. P. Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly. **2019**.

(28) Chen, T.-Y.; Baker-Fales, M.; Vlachos, D. G. Operation and Optimization of Microwave-Heated Continuous-Flow Microfluidics. *Ind. Eng. Chem. Res.* **2020**, *59*, 10418–10427.

(29) Ogunnaike, B. A. (Babatunde A. *Random Phenomena : Fundamentals of Probability and Statistics for Engineers*; CRC Press: Boca Raton, 2010.

(30) McKay, M. D.; Beckman, R. J.; Conover, W. J. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* **1979**, *21*, 239.

(31) Swift, T. D.; Bagia, C.; Choudhary, V.; Peklaris, G.; Nikolakis, V.; Vlachos, D. G. Kinetics of Homogeneous Brønsted Acid Catalyzed Fructose Dehydration and 5-Hydroxymethyl Furfural Rehydration: A Combined Experimental and Computational Study. *ACS Catal.* **2014**, *4*, 259–267.

(32) Desir, P.; Saha, B.; Vlachos, D. G. Ultrafast Flow Chemistry for the Acid-Catalyzed Conversion of Fructose. *Energy Environ. Sci.* **2019**, *12*, 2463–2475.

(33) Chen, T. Y.; Desir, P.; Bracconi, M.; Saha, B.; Maestri, M.; Vlachos, D. G. Liquid-Liquid Microfluidic Flows for Ultrafast 5-Hydroxymethyl Furfural Extraction. *Ind. Eng. Chem. Res.* **2021**.

(34) Chen, T. Y.; Cheng, Z.; Desir, P.; Saha, B.; Vlachos, D. G. Fast Microflow Kinetics and Acid Catalyst Deactivation in Glucose Conversion to 5-Hydroxymethylfurfural. *React. Chem. Eng.* **2021**, *6*, 152–164.