BoFire: Bayesian Optimization Framework Intended for Real Experiments

Johannes P. Dürholt [1] **Thomas S. Asche** [1] **Johanna Kleinekorte** [1] **Gabriel Mancino-Ball** [1] **Benjamin Schiller** [1] **Simon Sung** [1] **Julian Keupp** [2] **Aaron Osburg** [3] **Toby Boyne** [4] **Ruth Misener** [4] **Rosona Eldred** [5] **Chrysoula Kappatou** [5] **Robert M. Lee** [5] **Dominik Linzner** [5] **Wagner Steuer Costa** [5] **David Walz** [5] **Niklas Wulkow** [5] **Behrang Shafei** [5]

Abstract

Our open-source Python package BoFire combines Bayesian Optimization (BO) with other design of experiments (DoE) strategies focusing on developing and optimizing new chemistry. Previous BO implementations, for example as they exist in the literature or software, require substantial adaptation for effective real-world deployment in chemical industry. BoFire provides a rich feature-set with extensive configurability and realizes our vision of fast-tracking research contributions into industrial use via maintainable open-source software. Owing to quality-of-life features like JSON-serializability of problem formulations, BoFire enables seamless integration of BO into RESTful APIs, a common architecture component for both self-driving laboratories and human-in-the-loop setups. This paper discusses the differences between BoFire and other BO implementations and outlines ways that BO research needs to be adapted for real-world use in a chemistry setting.

1. Introduction

Once a chemist has outlined a possible reaction for creating a new chemical, or proposed a formulation or process for a new product, the focus in industrial chemistry shifts towards optimization. There are lots of questions that need to be answered. For example: *How, by changing the temperature and pressure of the reaction, can we maximize the yield and the purity of the desired chemical? How, by changing the chemical formulation, can we minimize environmental impact and maximize safety? Given a set of thousands of candidate molecules, which should be tested in the laboratory when only limited resources are available?*

To answer these questions, the most common approach in industry is still human intuition, trial-and-error, or expensive mechanistic models. However, Bayesian optimization (BO) and design of experiments (DoE) offer great possibilities to the chemical industry: treating chemical experiments as black-box functions and optimizing them in the most efficient manner or uncovering the sources of variation under relevant conditions, respectively (Coley et al., 2017; Hase et al., 2018; Shields et al., 2021; Thebelt et al., 2022; Frazier, 2018).

Software tools have been introduced to enhance the application of BO, for instance Ax (Bakshy et al., 2018) and BayBE (Fitzner et al., 2022), building on foundational machine learning software like BoTorch (Balandat et al., 2020). Other BO tools include Dragonfly (Kandasamy et al., 2020), NEXTorch (Wang et al., 2021), and SMAC3 (Lindauer et al., 2022). The BO tools are complemented by software with cheminformatics capabilities, for example providing representations of molecules, such as SMILES (Landrum, 2006; Moriwaki et al., 2018; Griffiths et al., 2023).

However, in industrial chemistry, existing BO and active learning software require substantial adaptation prior to deployment. Further, as experiments grow in scale and complexity, coordinating between lab components becomes challenging: inconsistent data handling makes implementing standalone software into a larger pipeline infeasible. Following the needs in chemical industry, we have developed (and continue developing) the open-source software package *Bayesian Optimization Framework Intended for Real Experiments* or `BoFire`[1] . Our companies deploy `BoFire` in both self-driving labs and human-in-the-loop applications. `BoFire` also supports serialization, whereby all of its components can be translated into a RESTful format, providing an API out of the box and simplifying implementation in existing systems. By making the algo-

[1]Evonik Operations GmbH, DE [2]Boehringer Ingelheim Pharma GmbH & Co. KG, DE [3]Heidelberg University, DE [4]Imperial College London, UK [5]BASF SE, DE. Correspondence to: Johannes P. Dürholt <johannespeter.duerholt@evonik.com>.

Proceedings of the ICML 2025 Workshop on Championing Open-source Development in Machine Learning (CODEML '25).

[1]`https://github.com/experimental-design/bofire`

rithmic component of our software open-source, we seek to give machine learning researchers a path towards fast-tracking their research ideas into practice and to provide an easy to use tool for industrial practitioners.

1.1. Practical uptake.

There has been substantial BoFire uptake at the three companies (BASF, Boehringer Ingelheim, and Evonik) represented in our author list, for example hundreds of employees at both BASF and Evonik use BoFire. Four additional companies have contributed employee time towards BoFire: Agilent Technologies (link), Bayer (link), Radical AI (link), and SOLVE (link).

Dr Jose Folch of SOLVE explains: *At SOLVE we are looking for ways of making experimentation as efficient, reproducible, and automated as possible. This has led us to contribute to BoFire as a tool that will be important for providing efficient experimentation while being able to save JSON method files for each experiment.*

Dr Lukas Hebing of Bayer writes: *Bayesian Optimization has emerged as a crucial method at Bayer across various application fields, including chemistry, biotechnology, and formulation technology. Scientists in our laboratories utilize our in-house tool, which is built on BoFire. BoFire offers an intuitive interface and effective solutions for our optimization challenges.*

1.2. Comparison to related work.

Our key strengths over similar frameworks are serialization, DoE strategies, and chemistry-specific utilities. Combined, these enable `BoFire` to be deployed in real-world labs - information can be easily transferred between the systems within a self-driving lab, supporting powerful data validation.

The frameworks most similar to `BoFire` are Ax (Bakshy et al., 2018) and BayBE (Fitzner et al., 2022). Compared to Ax, `BoFire` offers chemoinformatics capabilities, classical DoE approaches and serialization via Pydantic (Colvin, 2024), which enables easy FastAPI integration compared to unstructured JSON. Compared to BayBE, `BoFire` offers DoE strategies, serialization via Pydantic, output constraints including path-based constraints (Paulson et al., 2023) and categorical outputs, and other application-relevant features such as outlier detection and hyper parameter optimization. Compared to GAUCHE, `BoFire` offers BO and DoE capabilities out-of-the-box, with an opinionated API to make deployment of such algorithms more streamlined. We developed `BoFire` to meet the BO and DoE needs of industrial chemists in a single package.

2. Integrating experimental design into real-world labs

We take an experimentalist-first approach to the software architecture, implementing features that are industrially useful and focusing on easy user deployment. A real-world example motivates this section (with corresponding code in Listing 1 in the appendix and visualization in Figure 1) and our GitHub repository features other examples in Jupyter notebooks, as well as documentation listing the available classes for each component.

Domains. In `BoFire`, a `Domain` consists of `Inputs`, `Outputs`, and optionally `Constraints`. `BoFire` allows the user to define an input space $\mathcal{X} = x_1 \otimes x_2 \ldots \otimes x_D$ where the input features x_i can be continuous, discrete, molecular or categorical.

`BoFire` supports the following constraints: (non)linear (in)equality, NChooseK, and interpoint equality. The package also provides support for learning black-box inequality constraints.

> A chemist designs a paint using a selection of 20 different compounds, each of which has a continuously-varying concentration. They use an `NChooseK` constraint to limit each test-paint mixture to at most 5 compounds. For a batch of multiple mixtures, all paints are tested at the same temperature, requiring an `InterpointEquality` constraint which keeps the temperature fixed during the batch of experiments.

Objectives. In `BoFire`, objectives are defined separately from the outputs on which they operate. This allows us to define outputs in a physically meaningful way. Here, minimization, maximization, close-to-target and sigmoid-type objectives are supported.

For multi-objective optimization, `BoFire` supports two schemes: a linearization approach, in which the user specifies an additive or multiplicative weighting of each objective; and a Pareto front approach, where the optimizer approximates the Pareto front of all optimal compromises for subsequent decision-making. The latter is implemented via `qParEGO` (Knowles, 2006) and `q(log)(N)EHVI` strategies (Daulton et al., 2020; 2021; Ament et al., 2023). Both can be used in combination with black box constraints.

> The chemist wants to achieve a target viscosity, while maximising hydrophobicity. They define the measurements as `Outputs`, and use the `CloseToTargetObjective` and `MaximizeObjective` respectively to drive the optimization.

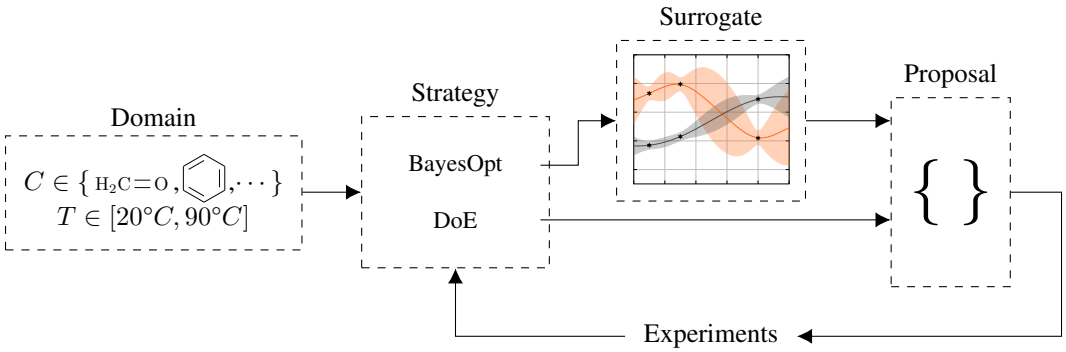

Figure 1. `BoFire` enables defining and solving optimization problems in the lab. All objects in the loop (candidates, strategies, surrogates, and proposals) are serializable.

Strategies. Given a `Domain`, the user selects a `Strategy` to generate experimental proposals. Classical DoE based strategies can generate (fractional)-factorial, space-filling (via sobol-, uniform- or latin-hypercube sampling), and D-,E-,A-,G-, or K-optimal designs. Compared to commercial software (e.g. Modde, JMP), `BoFire` supports designs over constrained mixed-type input spaces.

Alternatively, predictive strategies use `Surrogates` to model the data-generating process and perform BO. Many of these strategies are built on BoTorch (Balandat et al., 2020) and provide numerous acquisition functions. They are easily extendable and allow users to define custom strategies and surrogates, for instance as we did with ENTMOOT (Thebelt et al., 2021).

> The initial paint experiments should be selected using a `SpaceFillingDesign`, then use a `PredictiveStrategy` to suggest optimal experiments. The chemist uses the `StepwiseStrategy` interface to seamlessly transition between strategies.

3. Library Philosophy

3.1. Fully serializable.

`BoFire` is industry-ready for self-driving labs. In this setting, communication is key: many systems pass data and information between each other, and data integrity is essential.

`BoFire` is natively usable with a RESTful Application Programming Interface (API) and structured JSON-based, document-oriented databases, via the use of the popular data-validation library Pydantic allowing for seamless integration into FastAPI (Ramírez, 2024). We separate all `Strategies` and `Surrogates` into data models, and functional components. Data models are fully JSON-(de)serializable classes built on Pydantic, which hold complete information regarding the search space, surrogates and strategies.

This clear distinction allows for a minimal `BoFire` installation consisting only of the data models. This is especially useful in scenarios where a process orchestration layer (POL) is involved as the middle layer between a centrally deployed planner using `BoFire`, and closed-loop equipment. One can then communicate between these subsystems using an API; we support FastAPI since it is both fast and automatically validates data via the Pydantic data models. [2]

3.2. Modularization.

`BoFire` is both easy to use and highly customizable with respect to its strategies and surrogates. Each component of `BoFire` is modular - problem definitions are independent of the strategies used to solve them, which are in turn independent of the surrogates used to model the observed data. This separation of responsibility enables a 'plug-and-play' approach. By building `BoFire` using the BoTorch library, we can leverage the wide range of software written in the BoTorch ecosystem.

4. Discussion & Conclusion

This paper has presented `BoFire`, our open-source BO and DoE python package. Representing several companies in the chemical industry, we deploy `BoFire` daily to bring BO and DoE into our companies. Each individual contributing company could have easily developed their own bespoke package, but we joined forces to create `BoFire` because of our vision of catalyzing machine learning research. `BoFire` exemplifies our collaboration goals with researchers, for example those working in academia, for

[2] For an example, see `https://github.com/experimental-design/bofire-candidates-api`

example current work on practical multi-fidelity modeling (Bonilla et al., 2007; Folch et al., 2023) and tree kernels (Boyne et al., 2025). Through `BoFire`, we offer the possibility for researchers to use our platform to translate new strategies and surrogates into practice.

References

Ament, S., Daulton, S., Eriksson, D., Balandat, M., and Bakshy, E. Unexpected improvements to expected improvement for Bayesian optimization. In *NeurIPS*, volume 36, pp. 20577–20612, 2023.

Bakshy, E., Dworkin, L., Karrer, B., Kashin, K., Letham, B., Murthy, A., and Singh, S. Ae: A domain-agnostic platform for adaptive experimentation. In *NeurIPS Systems for ML Workshop*, 2018. URL http://learningsys.org/nips18/assets/papers/87CameraReadySubmissionAE%20-%20NeurIPS%202018.pdf.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *NeurIPS*, 2020.

Bonilla, E. V., Chai, K., and Williams, C. Multi-task Gaussian process prediction. *NIPS*, 20, 2007.

Boyne, T., Folch, J. P., Lee, R. M., Shafei, B., and Misener, R. BARK: A fully Bayesian tree kernel for black-box optimization. *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Coley, C. W., Barzilay, R., Jaakkola, T. S., Green, W. H., and Jensen, K. F. Prediction of organic reaction outcomes using machine learning. *ACS Central Science*, 3(5):434–443, 2017.

Colvin, S. Pydantic, June 2024. URL https://github.com/pydantic/pydantic.

Daulton, S., Balandat, M., and Bakshy, E. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *NeurIPS*, volume 33, pp. 9851–9864, 2020.

Daulton, S., Balandat, M., and Bakshy, E. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *NeurIPS*, volume 34, pp. 2187–2200, 2021.

Fitzner, M., Šoši'c, A., Hopp, A., and Lee, A. BayBE – a Bayesian back end for design of experiments, 2022. URL https://github.com/emdgroup/baybe. Accessed: 2024-02-22.

Folch, J. P., Lee, R. M., Shafei, B., Walz, D., Tsay, C., van der Wilk, M., and Misener, R. Combining multi-fidelity modelling and asynchronous batch Bayesian optimization. *Computers & Chemical Engineering*, 172: 108194, 2023.

Frazier, P. I. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Griffiths, R.-R., Klarner, L., Moss, H., Ravuri, A., Truong, S. T., Du, Y., Stanton, S. D., Tom, G., Ranković, B., Jamasb, A. R., Deshwal, A., Schwartz, J., Tripp, A., Kell, G., Frieder, S., Bourached, A., Chan, A. J., Moss, J., Guo, C., Dürholt, J. P., Chaurasia, S., Park, J. W., Strieth-Kalthoff, F., Lee, A., Cheng, B., Aspuru-Guzik, A., Schwaller, P., and Tang, J. GAUCHE: A library for Gaussian processes in chemistry. In *NeurIPS*, 2023.

Hase, F., Roch, L. M., Kreisbeck, C., and Aspuru-Guzik, A. Phoenics: a Bayesian optimizer for chemistry. *ACS Central Science*, 4(9):1134–1145, 2018.

Kandasamy, K., Vysyaraju, K. R., Neiswanger, W., Paria, B., Collins, C. R., Schneider, J., Poczos, B., and Xing, E. P. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *Journal of Machine Learning Research*, 21(81): 1–27, 2020.

Knowles, J. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE transactions on evolutionary computation*, 10(1):50–66, 2006.

Landrum, G. RDKit: Open-source cheminformatics, 2006. URL https://www.rdkit.org.

Lindauer, M., Eggensperger, K., Feurer, M., Biedenkapp, A., Deng, D., Benjamins, C., Ruhkopf, T., Sass, R., and Hutter, F. Smac3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

Moriwaki, H., Tian, Y.-S., Kawashita, N., and Takagi, T. Mordred: a molecular descriptor calculator. *Journal of cheminformatics*, 10(1):1–14, 2018.

Paulson, J. A., Sorouifar, F., Laughman, C. R., and Chakrabarty, A. LSR-BO: Local search region constrained Bayesian optimization for performance optimization of vapor compression systems. In *2023 American Control Conference (ACC)*, pp. 576–582. IEEE, 2023.

Ramírez. Fastapi, June 2024. URL 'https://github.com/tiangolo/fastapi'.

Shields, B. J., Stevens, J., Li, J., Parasram, M., Damani, F., Alvarado, J. I. M., Janey, J. M., Adams, R. P., and Doyle, A. G. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021.

Thebelt, A., Kronqvist, J., Mistry, M., Lee, R. M., Sudermann-Merx, N., and Misener, R. ENTMOOT: A framework for optimization over ensemble tree models. *Computers & Chemical Engineering*, 151:107343, 2021.

Thebelt, A., Wiebe, J., Kronqvist, J., Tsay, C., and Misener, R. Maximizing information from chemical engineering data sets: Applications to machine learning. *Chemical Engineering Science*, 252:117469, 2022.

Wang, Y., Chen, T.-Y., and Vlachos, D. G. NEXTorch: a design and Bayesian optimization toolkit for chemical sciences and engineering. *Journal of Chemical Information and Modeling*, 61(11):5312–5319, 2021.

Listing 1 Defining the domain of the paint problem in Section 2.

```
compounds = [f"compound_{i}" for i in range(20)]
inputs = [
    ContinuousInput(key="temp", bounds=[20, 90], unit="°C"),
    *(ContinuousInput(key=comp, bounds=[0, 1]) for comp in compounds)
]

outputs = [
    ContinuousOutput(
        key="viscosity",
        objective=CloseToTargetObjective(target_value=0.5, exponent=2)
    ),
    ContinuousOutput(
        key="hydrophobicity",
        objective=MaximizeObjective()
    )
]

constraints = [
    NChooseKConstraint(
        features=compounds, min_count=1, max_count=5,
    ),
    InterpointEqualityConstraint(feature="temp")
]

domain = Domain.from_lists(
   inputs, outputs, constraints
)
```