AutODEx: Automated Optimal Design of Experiments Platform with Data- and Time-Efficient Multi-Objective Optimization

Yunsheng Tian YUNSHENG@CSAIL.MIT.EDU
MIT CSAIL, Cambridge, MA 02139, USA

Pavle Vanja Konaković PAVLE.KONAKOVIC@GMAIL.COM
Independent Researcher

Beichen Li BEICHEN@CSAIL.MIT.EDU

Ane Zuniga ANEZU@CSAIL.MIT.EDU

Michael Foshey MFOSHEY@CSAIL.MIT.EDU

Timothy Erps TERPS@CSAIL.MIT.EDU

Wojciech Matusik WOJCIECH@CSAIL.MIT.EDU

Mina Konaković Luković MINA@CSAIL.MIT.EDU
MIT CSAIL, Cambridge, MA 02139, USA

Abstract

We introduce AutODEx[1], an automated machine learning platform for optimal design of experiments to expedite solution discovery with optimal objective trade-offs. We implement state-of-the-art multi-objective Bayesian optimization (MOBO) algorithms in a unified and flexible framework for optimal design of experiments, along with efficient asynchronous batch strategies extended to MOBO to harness experiment parallelization. For users with little or no experience with coding or machine learning, we provide an intuitive graphical user interface (GUI) to help quickly visualize and guide the experiment design. For experienced researchers, our modular code structure serves as a testbed to quickly customize, develop, and evaluate their own MOBO algorithms. Extensive benchmark experiments against other MOBO packages demonstrate AutODEx's competitive and stable performance. Furthermore, we showcase AutODEx's real-world utility by autonomously guiding hardware experiments with minimal human involvement.

1. Introduction

Optimal design of experiments in science and engineering often involves optimizing conflicting objectives to discover a set of Pareto optimal solutions. Furthermore, the objectives are typically black-box functions whose evaluations are slow and costly (e.g., lab experiments or numerical simulation). Hence, the limited evaluation budget requires an efficient strategy for guiding experimental design towards Pareto optimal solutions. A machine learning concept that enables such automatic guidance is Bayesian optimization (BO), and its recent advances have improved experimental design in various domains, e.g., chemical design (Shields et al., 2021), material design (Zhang et al., 2020), resource allocation (Wu

1. Project website: autodex.csail.mit.edu

et al., 2013), recommender systems (Chapelle and Li, 2011), and robotics (Martinez-Cantin et al., 2009). While well-studied for single-objective problems, its practical use in multi-objective cases remains limited due to a lack of user-friendly, open-source platform.

We present AutODEx, an open-source platform for efficiently optimizing multi-objective problems with a restricted budget of experiments. The key features of AutODEx include: (1) **Data efficiency**: State-of-the-art MOBO strategies that rapidly advances the Pareto front with a small set of evaluations. (2) **Time efficiency**: Synchronous and asynchronous batch optimization to accelerate the optimization. (3) **Intuitive GUI**: An easy-to-use GUI to directly visualize and guide the optimization progress and facilitate the operation for users with little or no experience with coding or machine learning. (4) **Modular structure**: A highly modular Python codebase enables easy extensions and replacements of MOBO algorithm components. These important features enable practical and straightforward integration of AutODEx into a fully automated design of experiments pipeline.

2. Related Work

Bayesian optimal design of experiments Optimal design of experiments is the process of designing the sequence of experiments to maximize specific objectives efficiently. Therefore, BO is usually applied to find optimal solutions with a minimal number of evaluations. Essentially, BO relies on *surrogate models* like Gaussian processes (GP) to model the experimental process and proposes new experimental designs based on defined *acquisition functions* that trade-off exploration and exploitation. To further speed up when evaluations can be carried out in parallel, asynchronous BO approaches have been developed (Ginsbourger et al., 2010; Kandasamy et al., 2018; Alvi et al., 2019).

Multi-objective Bayesian optimization MOBO is developed to optimize for a set of Pareto optimal solutions with minimal evaluations. One approach solves multi-objective problems by scalarizing them into single-objective ones using random weights (Knowles, 2006) or acquisition functions, e.g., entropy-based or hypervolume-based (Belakaria et al., 2019; Daulton et al., 2020a). Another approach is by defining a separate acquisition function per objective, optimizing using cheap multi-objective solvers (e.g., NSGA-II (Deb et al., 2002)) and finally selecting candidate designs to evaluate next (Bradford et al., 2018; Belakaria and Deshwal, 2020; Konakovic Lukovic et al., 2020). AutODEx implements the latter approach and allows easily changing modules in an unified MOBO framework (see Section 4.2). More discussions between the two approaches can be found in Appendix A.

Bayesian optimization libraries There are many existing Python libraries for BO on general problems including Spearmint (Snoek et al., 2012), HyperOpt (Bergstra et al., 2013), GPyOpt (authors, 2016), GPflowOpt (Knudde et al., 2017), Dragonfly (Kandasamy et al., 2020), Ax (Bakshy et al., 2018), Optuna (Akiba et al., 2019), HyperMapper (Nardi et al., 2019), BoTorch (Balandat et al., 2020a), SMAC3 (Lindauer et al., 2021), and OpenBox (Li et al., 2021). However, they are all targeted for experts in coding due to the lack of an intuitive GUI. In contrast, there are also software platforms that provide GUI but limited to specific applications (Shields et al., 2021; Haan, 2021). Combining powerful BO algorithms and an intuitive GUI for general purposes, AutODEx is designed to be a versatile platform accessible to individuals from various fields with minimal knowledge required.

3. Problem Formulation

We consider optimization over a set of design variables $\mathcal{X} \subset \mathbb{R}^d$, called *design space*. The goal is to minimize $m \geq 2$ objective functions $f_1, ..., f_m : \mathcal{X} \to \mathbb{R}$. Representing all objectives as $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_m(\mathbf{x}))$, the *performance space* is an m-dimensional image $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^m$. Conflicting objectives result in a set of optimal solutions referred to as *Pareto set* $\mathcal{P}_s \subseteq \mathcal{X}$ in the design space, and the corresponding image in the performance space is *Pareto front* $\mathcal{P}_f = \mathbf{f}(\mathcal{P}_s) \subset \mathbb{R}^m$. We use *hypervolume* (Zitzler and Thiele, 1999), the most widely used metric for MOBO (Riquelme et al., 2015), to measure the quality of an approximated Pareto front. The higher the hypervolume, the better $\mathcal{P}_f$ approximates the true Pareto front.

4. The AutODEx Platform

4.1 Overall Workflow

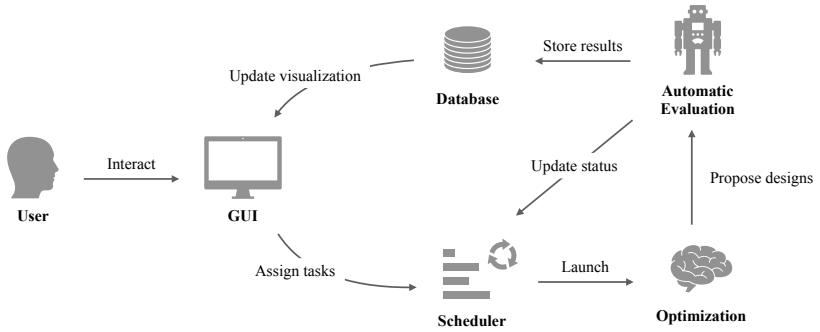


Figure 1: The GUI-directed workflow of automatic design of experiments in AutODEx.

The full-stack workflow of AutODEx is presented in Figure 1, where GUI is the only component that the user directly interacts with. At the beginning, the user configures the optimization problem and algorithm setup in the GUI. After initial configuration, the user can request for optimization with specified stopping criteria. If the evaluation program is available (Python/C/C++/Matlab are supported), the scheduler will automatically repeat the optimization-evaluation cycle either synchronously or asynchronously until one stopping criterion is met. The scheduler automatically starts evaluations after designs are proposed by optimization workers, and restarts optimization after evaluations are done, thus the whole loop of experimental design is executed in an automated way without human involvement. Alternatively, if the evaluation is performed manually (e.g., lab experiments), the user will see the designs proposed by the optimizer from the database part of the GUI and enter the evaluation results for proposed designs in the same interface. Once AutODEx receives the evaluation results, it updates visualizations and statistics to inform users about the latest progress. Please refer to Appendix A for more details of the platform.

4.2 Modular Algorithm Framework

MOBO consists of four core modules in our framework, as shown in Figure 2: (1) an inexpensive *surrogate model* for the black-box objective functions; (2) an *acquisition function* that trades-off exploration and exploitation of the design space; (3) a cheap *multi-objective solver* to approximate the Pareto set and front; (4) a *selection strategy* that proposes exper-

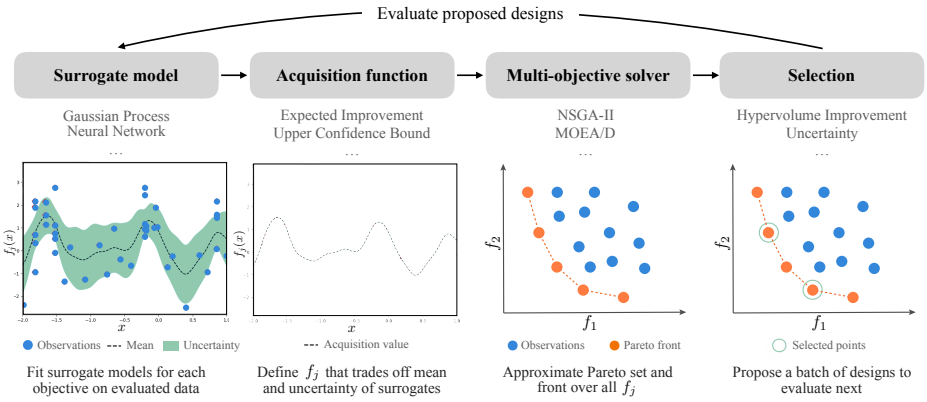


Figure 2: Algorithmic pipeline and core modules of multi-objective Bayesian optimization.

iment candidates to evaluate next. These modules are implemented as decoupled building blocks, making it highly modularized and easy to develop new algorithms. As of now, the following choices are supported for each module: (1) *Surrogate model*: Gaussian process, multi-layer perceptron, Bayesian neural network (Snoek et al., 2015). (2) *Acquisition function*: Expected Improvement, Probability of Improvement, Upper Confidence Bound, Thompson Sampling, identity function. (3) *Multi-objective solver*: NSGA-II, MOEA/D, Schulz et al. (2018). (4) *Selection*: hypervolume improvement, uncertainty, random, etc.

```
class TSEMO(MOBO):
spec = {
    'surrogate': 'gp',
    'acquisition': 'ts',
    'solver': 'nsga2',
    'selection': 'hvi',
}
```

```
class USEMO_EI(MOBO):
spec = {
    'surrogate': 'gp',
    'acquisition': 'ei',
    'solver': 'nsga2',
    'selection': 'uncertainty',
}
```

```
class DGEMO(MOBO):
spec = {
    'surrogate': 'gp',
    'acquisition': 'identity',
    'solver': 'discovery',
    'selection': 'direct',
}
```

Code Example 1: Creating algorithms in AutODEx by simply combining modules.

Based on this framework, we implement several popular and state-of-the-art MOBO methods, including ParEGO (Knowles, 2006), MOEA/D-EGO (Zhang et al., 2009), TSEMO (Bradford et al., 2018), USeMO (Belakaria and Deshwal, 2020), DGEMO (Konakovic Lukovic et al., 2020). The algorithms can be easily composed by choosing modules and inheriting the base class MOBO, see Code Example 1. Users can select an algorithm from this library that best fits their problem, or they can easily create new algorithms by specifying novel combinations of existing modules in just a few lines of code. More discussions on the benefit of this framework can be found in Appendix A. To facilitate asynchronous optimization for a parallel evaluation setup, we extend several asynchronous strategies to the multi-objective scenario including our novel and effective method, please find more details in Appendix B.

4.3 Intuitive Graphical User Interface

The key strength of AutODEx is the user-friendly GUI that significantly lowers the barrier of applying MOBO to real-world problems. Figure 3 shows a few representative screenshots from AutODEx, where users can easily configure problem settings (e.g., design/performance space and initial data), run optimization and inspect progress. For more benefits and screenshots of the GUI, see Appendix A. A video demo of the GUI can be found here. Upon paper acceptance, we commit to publishing the code and activating the online platform.

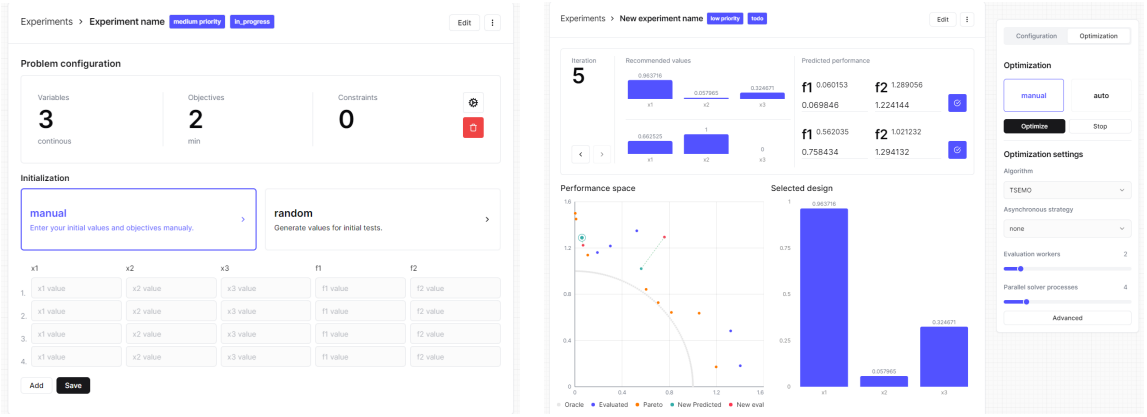


Figure 3: AutODEx's user-friendly GUI (left: problem setup; right: main optimization).

5. Experiments

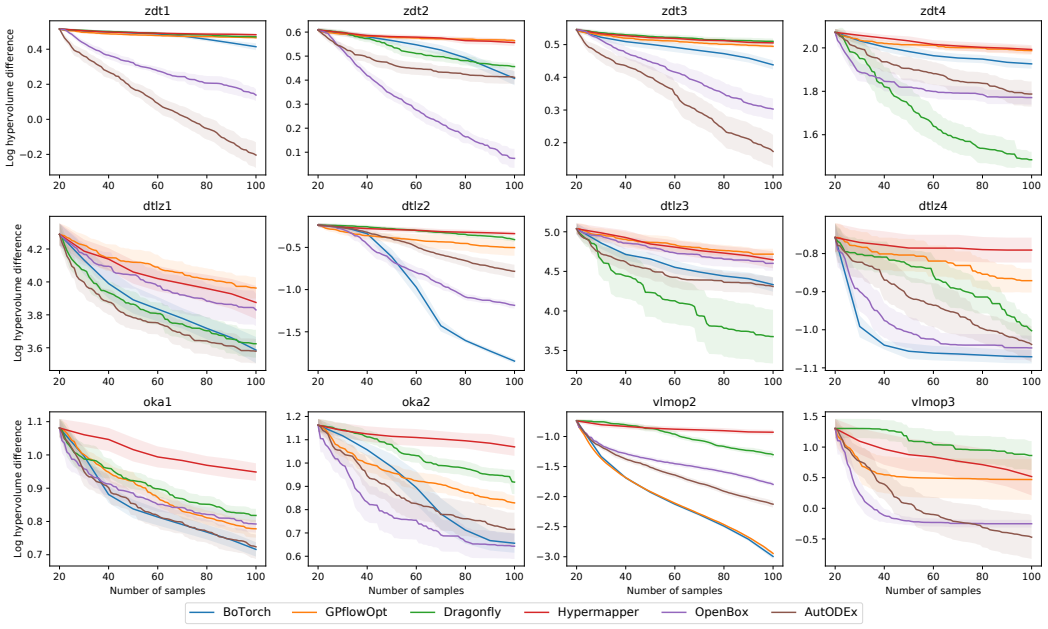


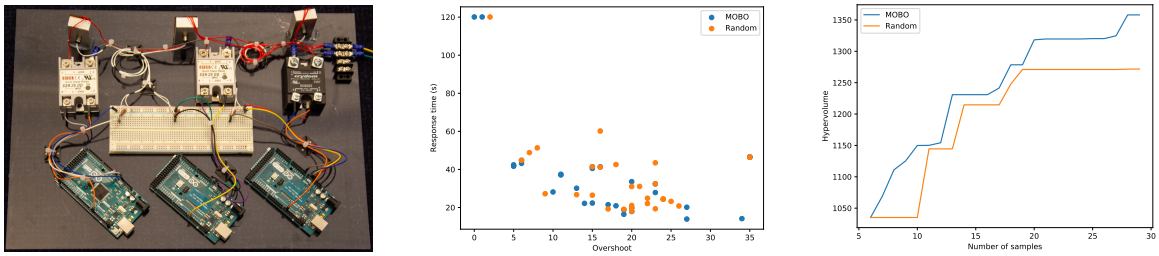
Figure 4: Performance of AutODEx on the benchmarks compared to other BO libraries[2].

Performance Comparison Across Platforms We compare AutODEx against other BO libraries with multi-objective optimization capabilities across 12 standard multi-objective benchmark problems: ZDT1-4 (Zitzler et al., 2000), DTLZ1-4 (Deb et al., 2005), OKA1-2 (Okabe et al., 2004) and VLMOP2-3 (Van Veldhuizen and Lamont, 1999). We run experiments with a budget of 100 samples where the initial 20 samples are generated by Latin hypercube sampling (McKay et al., 1979). We measure as the performance the log of the difference between the hypervolume of the ground-truth Pareto front and hypervolume of the best Pareto front approximation found by the algorithms (thus lower is better). The

2. The result of BoTorch on VLMOP3 is omitted as it fails to complete within 24 hours.

curves are averaged over 20 random seeds and the variance is shown as shaded regions. Problem details and hyperparameters are described in Appendix C. Additional ablation studies are included in Appendix E.

Our baseline implementation follows the default recommended settings in their official documentation. For BoTorch, we use the qEHVI acquisition function. In Figure 4, the performance of AutODEx is generally stable and ends up either the best or comparable on most benchmark problems. Especially, AutODEx takes a major lead in several challenging problems such as ZDT1, ZDT3, and VLMOP3, which shows that our platform handles high-dimensional MOBO problems well. We conduct additional ranked and paired comparisons between AutODEx and all the baseline libraries in Appendix D to further demonstrate AutODEx's robustness and competitive performance.



(a) Experiment setup. (b) Discovered solutions. (c) Hypervolume.

Figure 5: Physical setup and the optimization process of the PID heater control experiment.

Real-World Optimal Design of Experiments We further demonstrate the real-world applicability of AutODEx through applying to optimize a PID heater controller to have optimal response time and minimal overshoot in a fully automated way. Details of the experimental setup can be found in Appendix C.3. To automate the experiment, we simply link the Python evaluation program of this experiment setup to AutODEx through GUI then start the iterative optimization. Finally, the results are exported as shown in Figure 5, where a set of solutions are discovered with optimal trade-offs between minimal response time and minimal overshoot. Using MOBO algorithms provided by AutODEx is able to discover better designs compared to random exploring at the presence of evaluation noise (temperature measurement error, lack of precise initial temperature control, fabrication differences between heater blocks). This example, with all the components that people can easily buy off-the-shelf, serves as a simple proof of concept that AutODEx is applicable to optimize real physical systems. More examples, including optimizing material structure based on FEM simulation and optimizing a physical motor's rotation, can be found in our documentation with detailed instructions on how to interact with GUI and compose the evaluation program for fully automated optimal design of experiments.

6. Conclusion and Future Work

We introduced an open-source platform for automated optimal experimental design of multi-objective problems. Future work includes implementing additional features, e.g., advanced noise handling and incorporating user preferences. We believe AutODEx will effectively reduce the gap between MOBO research and practical applications in experimental design.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

Ahsan Alvi, Binxin Ru, Jan-Peter Calliess, Stephen Roberts, and Michael A Osborne. Asynchronous batch bayesian optimisation with improved local penalisation. In *International Conference on Machine Learning*, pages 253–262. PMLR, 2019.

The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016.

Eytan Bakshy, Lili Dworkin, Brian Karrer, Konstantin Kashin, Benjamin Letham, Ashwin Murthy, and Shaun Singh. Ae: A domain-agnostic platform for adaptive experimentation. 2018.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020a.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020b.

Syrine Belakaria and Aryan Deshwal. Uncertainty-aware search framework for multi-objective bayesian optimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 7823–7833, 2019.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

Eric Bradford, Artur M Schweidtmann, and Alexei Lapkin. Efficient multiobjective optimization employing gaussian processes, spectral sampling and a genetic algorithm. *Journal of global optimization*, 71(2):407–438, 2018.

Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2249–2257. Curran Associates, Inc., 2011.

Ivo Couckuyt, Dirk Deschrijver, and Tom Dhaene. Fast calculation of multiobjective probability of improvement and expected improvement criteria for pareto optimization. *Journal of Global Optimization*, 60(3):575–594, 2014.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9851–9864. Curran Associates, Inc., 2020a.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, 33:9851–9864, 2020b.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. *Advances in Neural Information Processing Systems*, 34:2187–2200, 2021.

Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

Kalyanmoy Deb, Mayank Goyal, et al. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and informatics*, 26:30–45, 1996.

Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*, pages 105–145. Springer, 2005.

Eduardo C Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20–35, 2020.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems*, pages 131–162. Springer, 2010.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pages 648–657. PMLR, 2016.

Sebastian Haan. Geobo: Python package for multi-objective bayesian optimisation and joint inversion in geosciences. *Journal of Open Source Software*, 6(57):2690, 2021.

Himanshu Jain and Kalyanmoy Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling

constraints and extending to an adaptive approach. *IEEE Transactions on evolutionary computation*, 18(4):602–622, 2013.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 133–142. PMLR, 2018.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabas Poczos, and Eric P. Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

Joshua Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. GPflowOpt: A Bayesian Optimization Library using TensorFlow. *arXiv preprint – arXiv:1711.03845*, 2017. URL `https://arxiv.org/abs/1711.03845`.

Mina Konakovic Lukovic, Yunsheng Tian, and Wojciech Matusik. Diversity-guided multi-objective bayesian optimization with batch evaluations. *Advances in Neural Information Processing Systems*, 33, 2020.

Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, and et al. Openbox: A generalized black-box optimization service. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Aug 2021. doi: 10.1145/3447548.3467061.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2021.

Ruben Martinez-Cantin, Nando Freitas, Eric Brochu, Jose Castellanos, and Arnaud Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Auton. Robots*, 27:93–103, 08 2009. doi: 10.1007/s10514-009-9130-2.

Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

Luigi Nardi, Artur Souza, David Koeplinger, and Kunle Olukotun. Hypermapper: a practical design space exploration framework. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 425–426, 2019. doi: 10.1109/MASCOTS.2019.00053.

Tatsuya Okabe, Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. On test functions for evolutionary multi-objective optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 792–802. Springer, 2004.

Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible framework for multi-objective bayesian optimization using random scalarizations. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 766–776. PMLR, 22–25 Jul 2020.

Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015.

Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics (TOG)*, 37 (4):1–14, 2018.

Benjamin J Shields, Jason Stevens, Jun Li, Marvin Parasram, Farhan Damani, Jesus I Martinez Alvarado, Jacob M Janey, Ryan P Adams, and Abigail G Doyle. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM symposium on Applied computing*, pages 351–357, 1999.

J. Wu, W.Y. Zhang, S. Zhang, Y.N. Liu, and X.H. Meng. A matrix-based bayesian approach for manufacturing resource allocation planning in supply chain management. *International Journal of Production Research*, 51(5):1451–1463, 2013.

Qingfu Zhang, Wudong Liu, Edward Tsang, and Botond Virginas. Expensive multiobjective optimization by moea/d with gaussian process model. *IEEE Transactions on Evolutionary Computation*, 14(3):456–474, 2009.

Yichi Zhang, Daniel W Apley, and Wei Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific Reports*, 10(1):1–13, 2020.

Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

Appendix A. Platform Features

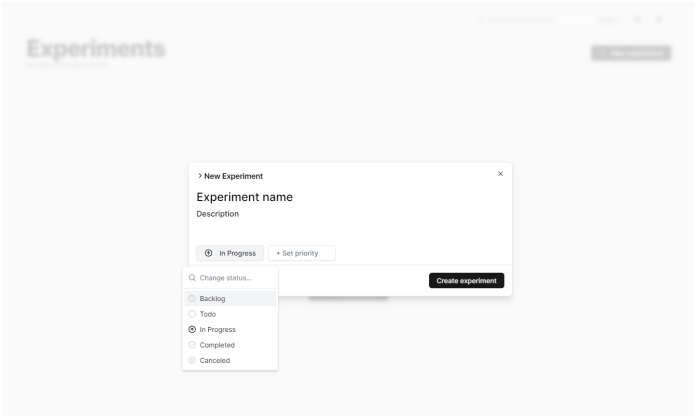
Graphical user interface and visualization The GUI guides the user through a set of simple steps to configure the problem, such as the number of design and performance parameters, the parameter bounds and constraints, parallelization settings, and selection of the optimization algorithm without the need of coding. The GUI also includes a real-time display of the design and performance space which allows users to easily understand the current status of optimization. We also support displaying and exporting the whole optimization history (including database and statistics). All previous libraries do not offer such a convenient GUI and even the visualizations need to be written by the user, except GPyOpt has a built-in tool for plotting the acquisition function and convergence. More interface examples are shown in Figure 6.

Multiple objectives As described in Section 4.2, AutODEx covers a wide range of MOBO algorithms and incorporates them into a more unified modular framework. As a comparison, GPflowOpt implements HVPOI (Couckuyt et al., 2014) and Dragonfly implements MOORS (Paria et al., 2020) without the flexibility of incorporating other algorithms or modules. BoTorch supports MESMO (Belakaria et al., 2019), qParEGO, qEHVI (Daulton et al., 2020b), qNParEGO and qNEHVI (Daulton et al., 2021). BoTorch, along with other existing MOBO libraries, implements the MOBO framework in a similar way to single-objective optimization: the acquisition function is to compute a single objective, e.g., entropy-based (MESMO) or hypervolume-based (EHVI). In their framework, a single-objective solver is applied to solve for the candidate designs to evaluate. In AutODEx, we support a different yet general MOBO framework – we support defining separate acquisition functions for each objective, then applying a multi-objective solver to solve for the candidate designs, and finally using a selection scheme to select the batch of designs to evaluate. Although there is no consensus on which framework is more effective, we found they are performing comparably well empirically. Our framework explicitly encourages diverse and global solutions by using mature multi-objective evolutionary algorithms (MOEA) as solvers, which is desirable for batch experiments. Thanks to the efficiency of MOEA, the algorithms implemented in our framework are much faster with a larger number of objectives (e.g. DGEMO, TSEMO) while EHVI in BoTorch tends to be very slow when the number of objectives increases. Also, many-objective solvers such as NSGA-III (Deb and Jain, 2013; Jain and Deb, 2013) can be applied in our framework for more efficiently exploring the high-dimensional space.

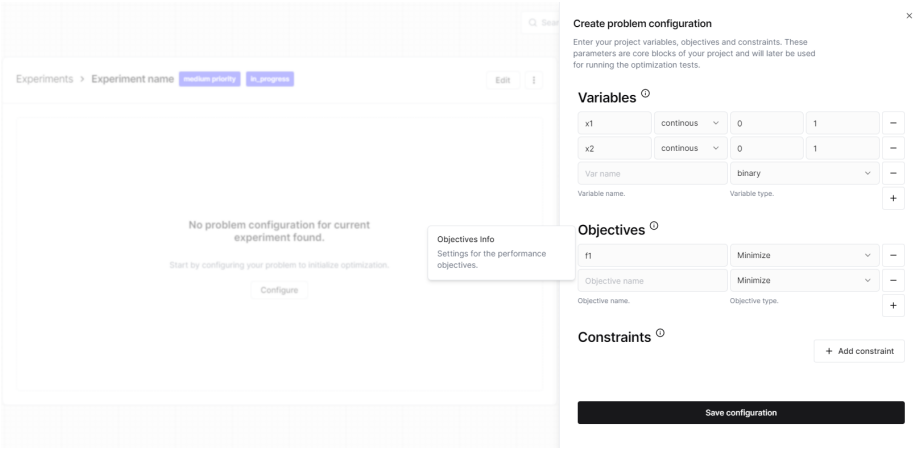
Multiple domains Besides continuous designs, AutODEx supports discrete, binary, categorical designs and a mix of them by applying discrete or one-hot transformation in fitting and evaluating the surrogate model, following the effective approach in Garrido-Merchán and Hernández-Lobato (2020).

Asynchronous optimization As to be discussed in more details in Appendix B, AutODEx supports different asynchronous techniques including Kriging Beliver (KB), Local Penalization (LP) and our novel Believer-Penalizer (BP), while Dragonfly and BoTorch only implements KB and all other libraries do not support asynchronous optimization.

External evaluation There are many real-world experimental design problems where the experimental evaluation must be performed by hand or external lab equipment thus it is



(a) The interface for managing and tagging experiments.



(b) The interface for entering problem details (variables, objectives, constraints).

(c) The interface for showing statistics.

Figure 6: More interface examples of AutODEx's experimental design workflow.

hard to write an analytical objective function in code that integrates with the optimization framework. For existing BO libraries, it means that the user needs to iteratively execute the code, also write code for manually exporting and importing the data between two consecutive rounds of optimization. In contrast, AutODEx allows users to see the suggested designs and directly enter the corresponding evaluation results in our database interface, as described in Section 4.1, without the need of coding data import/export. See Figure 7 for the detailed illustration of this workflow.

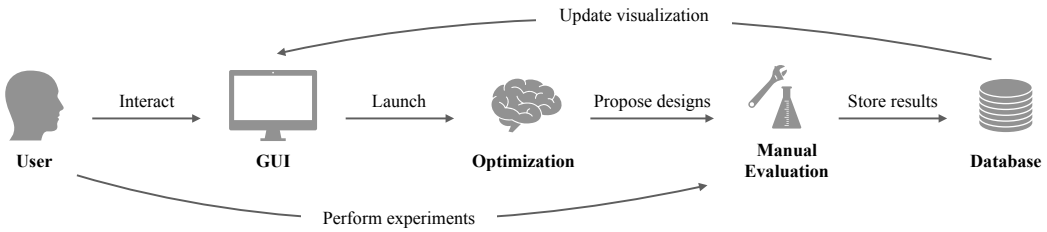


Figure 7: The workflow of AutODEx with manual external evaluations.

Surrogate prediction Users may find AutODEx useful not only in optimization but also in prediction. In addition to the set of optimal solutions, our platform's final product is the learned prediction models of the unknown objectives, which can be used easily from GUI to predict the objectives for a given design from the user. The prediction provides the users with more insights into the potential outcomes of experiments. It helps them better understand the optimization problem to make informed decisions and guide the optimization process towards their preference.

Appendix B. Asynchronous Batch Optimization

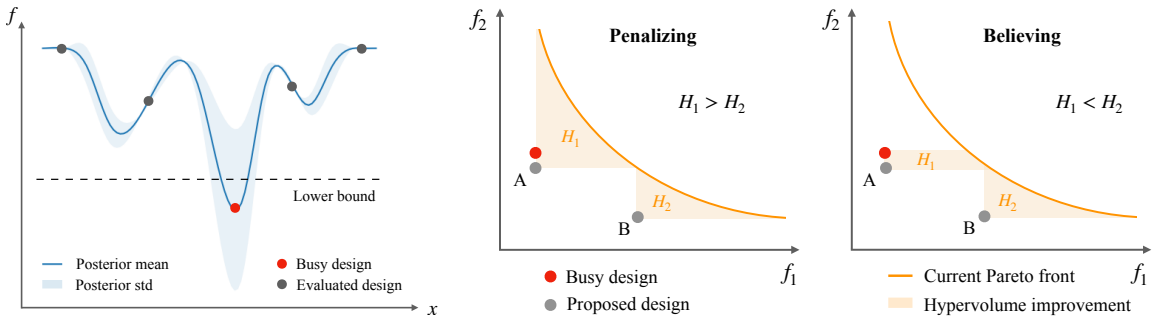
While standard MOBO optimizes for the Pareto front in a data-efficient manner, often, when multiple experiment setups are available, evaluations can be executed in batch by parallel workers to further speed up the whole optimization process. To leverage this speed-up, all the algorithms in AutODEx are implemented to support batch evaluation.

However, if parallel workers evaluate in different speeds, some workers are left idle when they finish evaluations earlier than others. Therefore, asynchronous optimization is desired to maximize the utilization of workers and is able to evaluate many more designs than synchronous optimization in a fixed amount of time, as also illustrated by Kandasamy et al. (2018) and Alvi et al. (2019). Nevertheless, while some designs are being evaluated (i.e., busy designs), how to propose the next design that *(i)* avoids being similar to the busy designs and *(ii)* incorporates knowledge from busy designs to reach better regions in the performance space is the key question that we want to explore.

To develop efficient asynchronous strategy for multi-objective optimization, we borrow ideas from previous literature in the single-objective setting.

B.1 Failures of Existing Strategies

Kriging Believer (KB) (Ginsbourger et al., 2010) is a simple yet effective approach that believes the performance of busy designs is their posterior mean of the surrogate



(a) The failure case of KB when believing overestimated busy designs.

(b) The sub-optimality of LP in multi-objective scenario when believing busy designs affects the selection result.

Figure 8: Analysis of KB and LP strategies for asynchronous optimization.

model when optimizing for new designs. In other words, it treats the mean prediction of the busy designs as their real performance and eliminates their posterior variance to prevent acquisition functions from preferring those regions. However, failure case happens when it believes an overestimated design, it might become difficult to find designs better than this overestimated one and make further improvement, see Figure 8a. Especially, when the posterior mean of the busy design is extremely small and even exceeds the lower bound of the objective, subsequent optimization can hardly find a better solution. In other words, subsequent optimization will only propose more overestimated designs with even lower predicted performance to "make improvement", even though they are even farther from the ground truth and drive the optimization away from the real meaningful regions. This overestimation issue has not been studied in the past literature to the best of our knowledge, though KB is still the strategy used in popular BO packages (Kandasamy et al., 2020; Balandat et al., 2020b).

Local Penalization (LP) (González et al., 2016) is another widely used approach that directly penalizes the nearby region of the busy designs to prevent similar designs from being evaluated next. However, extending this approach to the multi-objective scenario sometimes leads to sub-optimal selection of new designs, as explained in Figure 8b. Intuitively, this sub-optimality comes from the failure of leveraging the accurate predictions from the surrogate model. Consider when selecting the best design to evaluate from a set of candidate designs (A and B) proposed by the multi-objective solver using hypervolume improvement criterion, while a busy design is in evaluation. LP penalizes the nearby regions of the busy design in the design space but has no control over the performance space, which means that designs with similar performance as the busy design could still be selected (design A). Ideally, if the surrogate prediction of the busy design is certain, we can leverage this to avoid proposing designs with little performance gain. For example, simply believing the prediction of the busy design leads to selecting design B that has a higher hypervolume improvement.

B.2 Believer-Penalizer Strategy

In conclusion, we observe that the failure case of KB is due to the trust of uncertain predictions while the sub-optimality of LP comes from not believing the certain prediction.

Therefore, we propose a novel strategy BP that naturally combines KB and LP by applying KB to designs with certain predictions and LP to designs with uncertain predictions. Here, certainty of prediction is simply defined as the posterior standard deviation from the surrogate model which can be Gaussian processes, Bayesian neural networks or other type of model that computes standard deviation of predictions. Though the idea of BP is general and one can use any analytical expression to determine the certainty threshold for applying KB or LP, in practice, we find a simple probabilistic form which works well: $P_i(\mathbf{x}) = \max(1 - 2\sigma_i(\mathbf{x}), 0)$ where P_i is the probability of believing $\mathbf{x}$ for the i-th objective and σ_i is the posterior std of $\mathbf{x}$ from the surrogate model of the i-th objective. Because the objective data is normalized as zero with mean unit variance before fitting the surrogate models, $\sigma_i(\mathbf{x})$ is generally between 0 and 1.

B.3 Performance Comparison

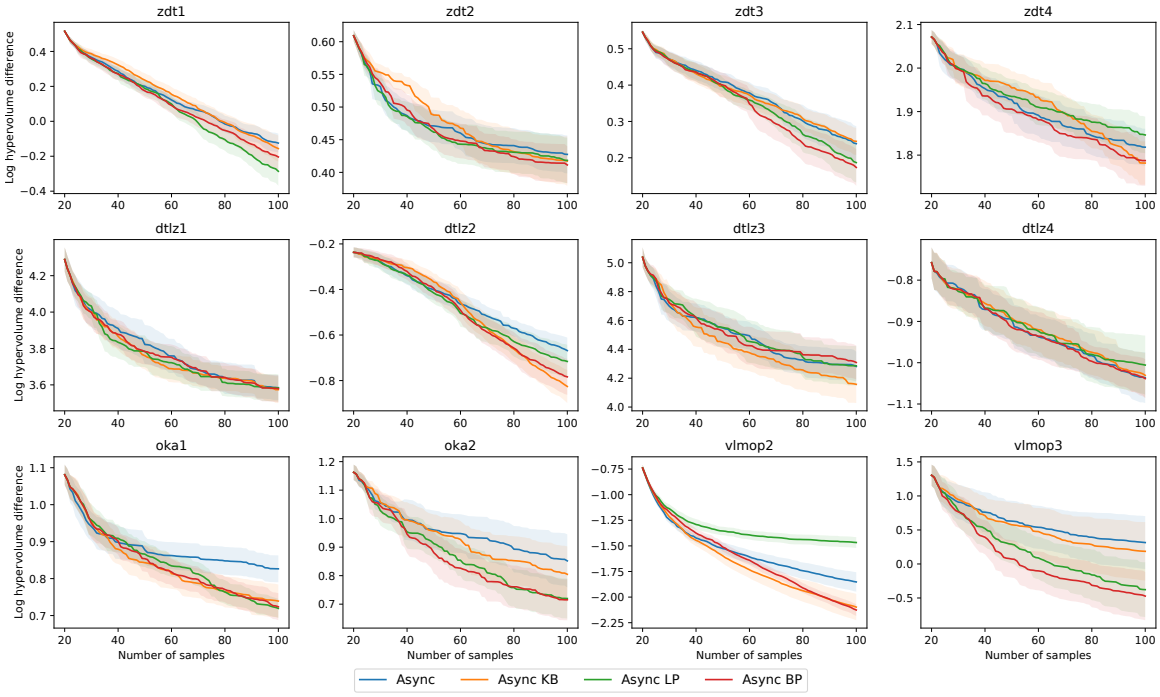


Figure 9: Performance comparison between variants of asynchoronous MOBO algorithms.

To test whether Believer-Penalizer is effective, we compare four asynchronous MOBO algorithms on all benchmark problems. *Async* simply ignores the busy designs while optimizing asynchronously and the remaining algorithms are described in Appendix B. Figure 9 shows that *Async BP* consistently outperforms other methods and follows the best of *Async KB* and *Async LP*.

Appendix C. Experimental Setup

C.1 Benchmark Problems

In this section, we briefly introduce the properties of each benchmark problem, including the dimensions of the design space $\mathcal{X} \subset \mathbb{R}^d$ and performance space $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^m$, and the reference points we use for calculating the hypervolume indicator, which are shown in Table 1. We perform 20 independent test runs with 20 different random seeds for each problem on each algorithm. For each test run of one problem, we use the same initial set of samples for every algorithm, which is generated by Latin hypercube sampling (McKay et al., 1979) using a same random seed. To have a fair comparison, we simply set the reference point $\mathbf{r} \in \mathbb{R}^m$ as a vector containing the maximum value of each objective over the initial set of samples $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$:

$$\mathbf{r} = (\max_{1 \leq i \leq k} f_1(\mathbf{x}_i), ..., \max_{1 \leq i \leq k} f_m(\mathbf{x}_i)).$$

Table 1: Basic descriptions of all the benchmark problems.

Name	d	m	$\mathbf{r}$
ZDT1	30	2	(0.9699, 6.0445)
ZDT2	30	2	(0.9699, 6.9957)
ZDT3	30	2	(0.9699, 6.0236)
ZDT4	10	2	(0.9699, 199.6923)
DTLZ1	6	2	(360.7570, 343.4563)
DTLZ2	6	2	(1.7435, 1,6819)
DTLZ3	6	2	(706.5260, 746.2411)
DTLZ4	6	2	(1.8111, 0.7776)
OKA1	2	2	(7.4051, 4.3608)
OKA2	3	2	(3.1315, 4.6327)
VLMOP2	6	2	(1.0, 1.0)
VLMOP3	2	3	(8.1956, 53.2348, 0.1963)

C.2 Hyperparameters

Here we present all the common hyperparameters that AutODEx uses in the experiments.

Surrogate model We use the same Gaussian process model as the surrogate for all experiments. We use zero mean function and anisotropic Matern 1/2 kernel, which empirically is numerically stable than popular Matern 5/2 kernel in our experiments. The corresponding hyperparameters are specified in Table 2, which are suggested by TSEMO.

Multi-objective evolutionary algorithm The cheap NSGA-II solver employed in AutODEx's MOBO algorithms by default uses simulated binary crossover (Deb et al., 1995) and polynomial mutation (Deb et al., 1996) for finding the Pareto front of acquisition functions. The initial population is obtained from the best current samples determined by non-dominated sort (Deb et al., 2002). The other hyperparameters are specified in Table 3.

Table 2: GP hyperparameters.

parameter name	value
initial l	$(1, ..., 1) \in \mathbb{R}^d$
l range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_f	1
σ_f range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_n	10^{-2}
σ_n range	$(e^{-6}, 1)$

Table 3: NSGA-II hyperparameters.

parameter name	value
population size	100
number of generations	200
crossover η_c	15
mutation η_m	20

C.3 Real-World Experiment Setup

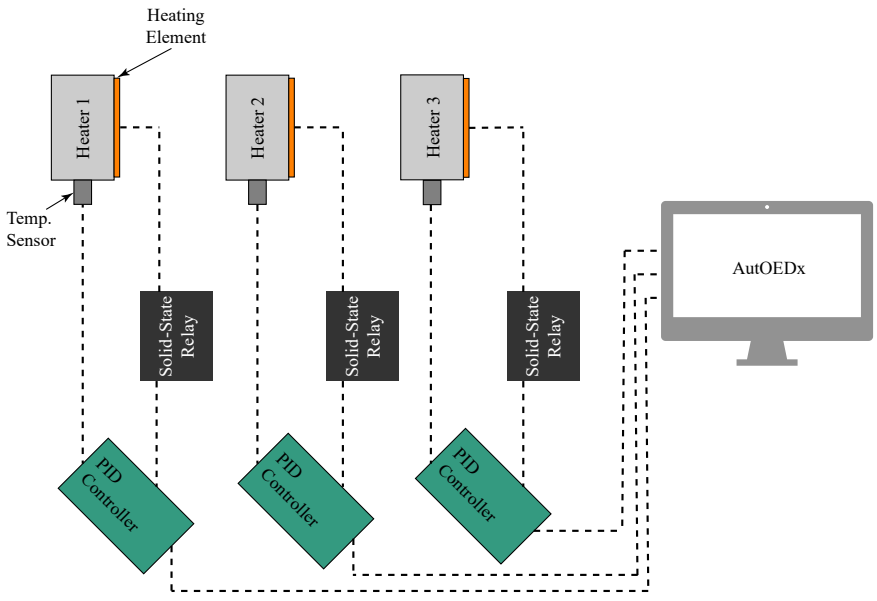


Figure 10: A schematic of the setup used for the real-world experiment.

In our real-world experiment setup, overall, a PID controller is employed to regulate the temperature of the heater block with proportional, integral, and differential constants. To find the optimal set of constants a number of heating cycles are done asynchronously with the controller using AutODEx. The experiment is comprised of setting the PID constants, then heating the block up to a set temperature, and keeping them at the set duration for 2 minutes. During this time the response time and overshoot are measured. After the heating cycle, the heater is then cooled back down to a starting temperature to prepare for another test with new PID constants.

Specifically, the experimental setup is comprised of 3 heaters with identical dimensions and characteristics. Each heater is comprised of a heater block, heating element, temperature sensor, solid-state relay, power supply, and a controller, shown in Figure 10. To run an experiment, AutODEx sends the PID constants to a controller that is free. Next, the PID controller becomes active and starts regulating the temperature of the heater. The temperature sensor measures the temperature of the heater block. Depending on the constants of the PID controller and the temperautre of the heater block, the controller turns the heating element on or off via the solid-state relay. After a period of 2 minutes where the PID controller is active, the controller stops actively regulating the temperature of the heater allowing the heater to cool. Next, the controller starts to monitor the cooling of the heater via the temperature sensor. It monitors it until the heater block cools to a temperature below a threshold. The amount of time it takes to cool the heater block depends on the temperature that the block was heated to during the active period. Once it sufficiently cools, the controller sends the calculated overshoot and response time to AutODEx and notifies that it is ready to run another experiment.

Appendix D. Additional Comparisons

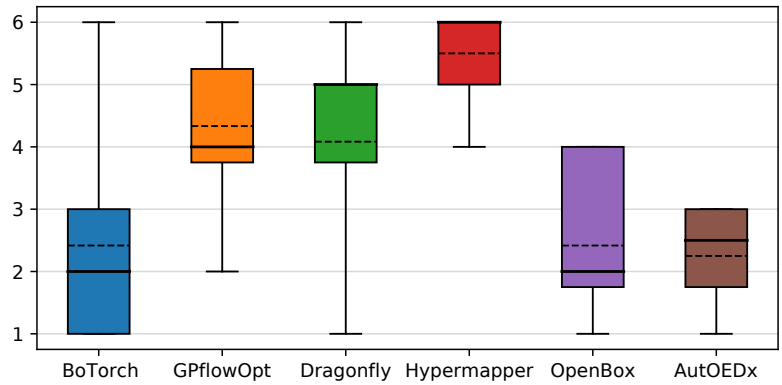


Figure 11: Performance rank of our platform and baseline libraries on the 12 benchmark problems (lower is better). The box extends from the lower to the upper quartile values, with a solid line at the median and a dashed line at the mean. The whiskers that extend the box show the range of the data.

We conduct ranked and paired comparisons between AutODEx and all the baseline libraries based on the 12 benchmark problems, as shown in Figure 11 and Figure 12. The

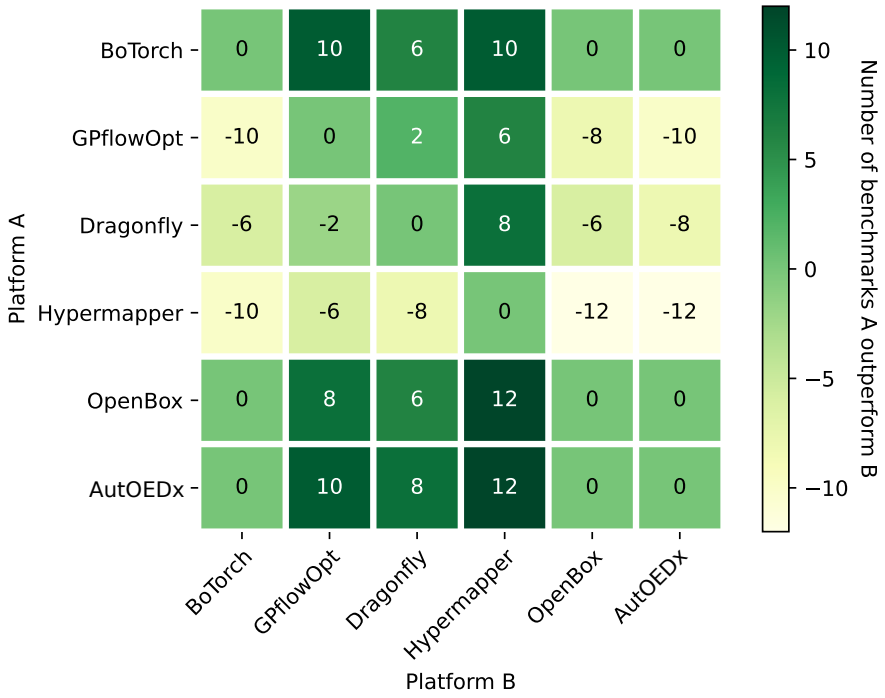


Figure 12: Performance comparison between each pair of platforms on the 12 benchmark problems. Each value in the matrix shows the number of benchmarks that platform A (associated with the row) outperforms platform B (associated with the column), the higher the better.

performance rank comparison in Figure 11 suggests that BoTorch, OpenBox and AutODEx outperform other libraries by a great margin overall. While BoTorch and OpenBox share a better median rank, AutODEx appears to be the stablest that consistently ranks between 1 and 3 on all problems and has a higher lower-bound performance than BoTorch and OpenBox. Figure 12 also suggest that AutODEx has a competitive performance to BoTorch and OpenBox, and outperforms other baselines on a wider range of benchmarks.

Appendix E. Ablation Studies

E.1 Synchronous and Asynchronous MOBO

Following the experiment settings in Section 5, we addtionally compare the performance of synchornous and asynchronous MOBO. As shown in Figure 13, they achieve similar hypervolumes whereas asynchronous MOBO spends less than half of the time of its synchronous counterpart.

E.2 Batch Size

Ablation studies are also conducted on the batch size in asynchronous MOBO. For this category of experiments, we repeat our practice in Section B.3 while changing the batch

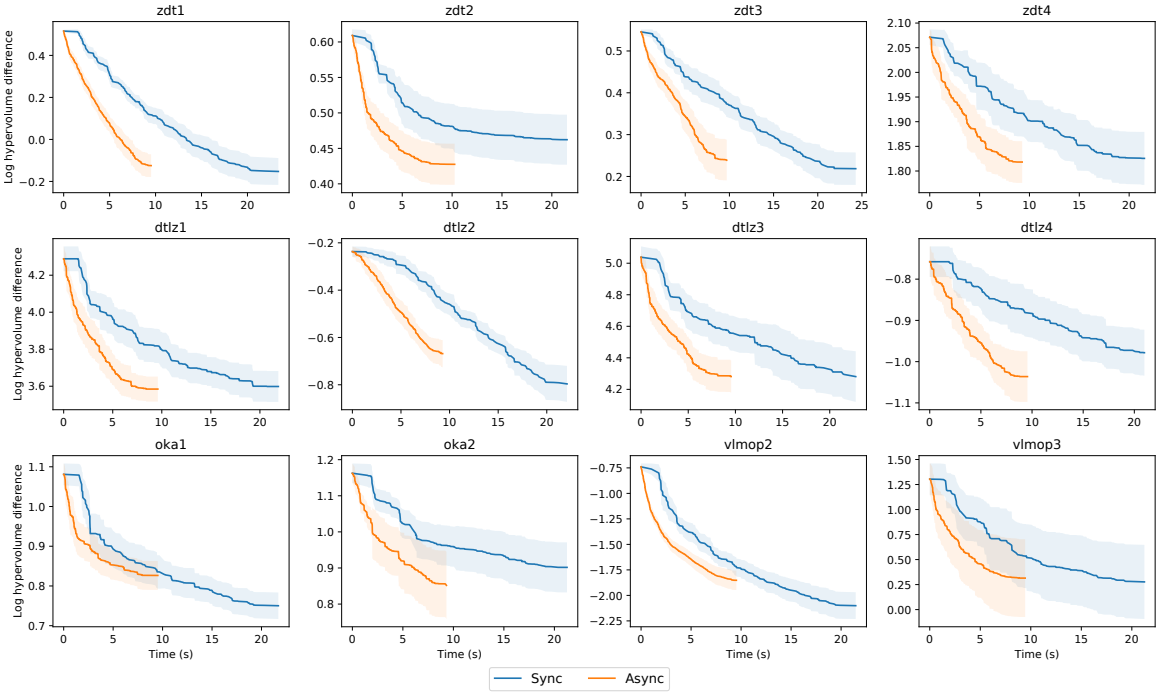


Figure 13: Performance comparison between synchronous and naive asynchronous MOBO algorithms.

size to 4 and 16, respectively. The results are demonstrated in Figure 14 and 15. Our proposed BP strategy maintains its relative lead in VLMOP3 and performs comparably with other variants on the rest of the test problems.

E.3 Acquisition Function

Lastly, we evaluate the asynchronous MOBO variants using the EI acquisition function. Although the change in acquisition function has a clear influence on hypervolume growth, the proposed BP variant still performs favorably in problems such as ZDT2, DTLZ2, and VLMOP2. The performance of BP on the other problems remain comparable to the alternative strategies.

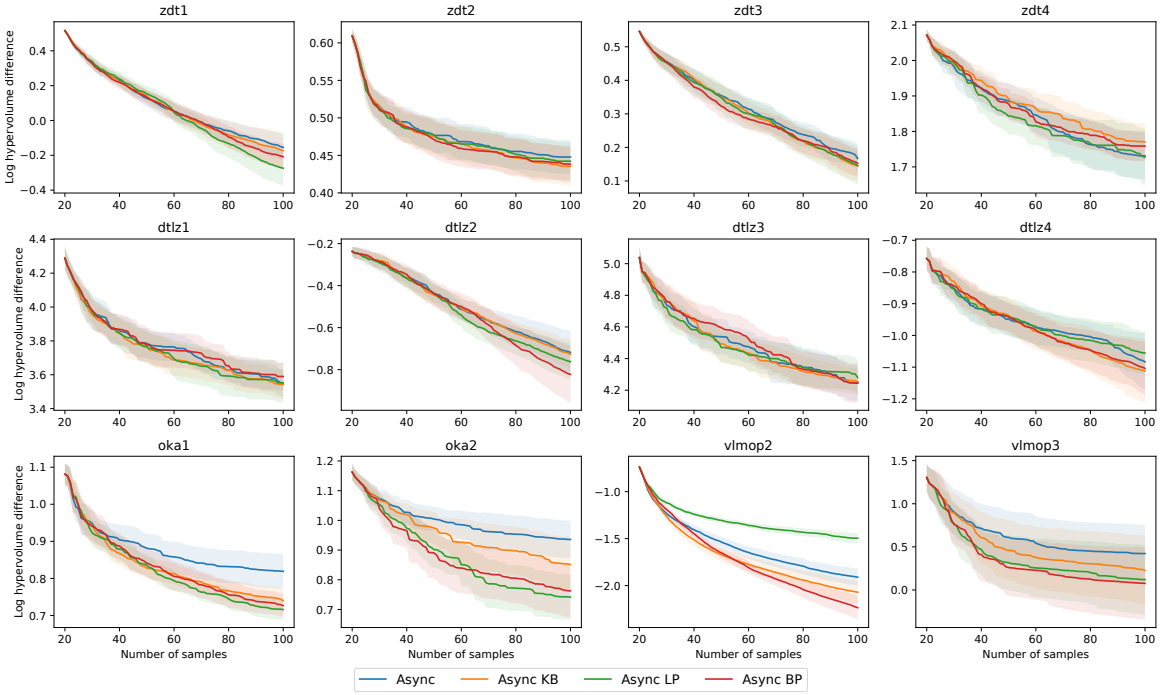


Figure 14: Performance comparison between variants of asynchoronous MOBO algorithms with a batch size of 4.

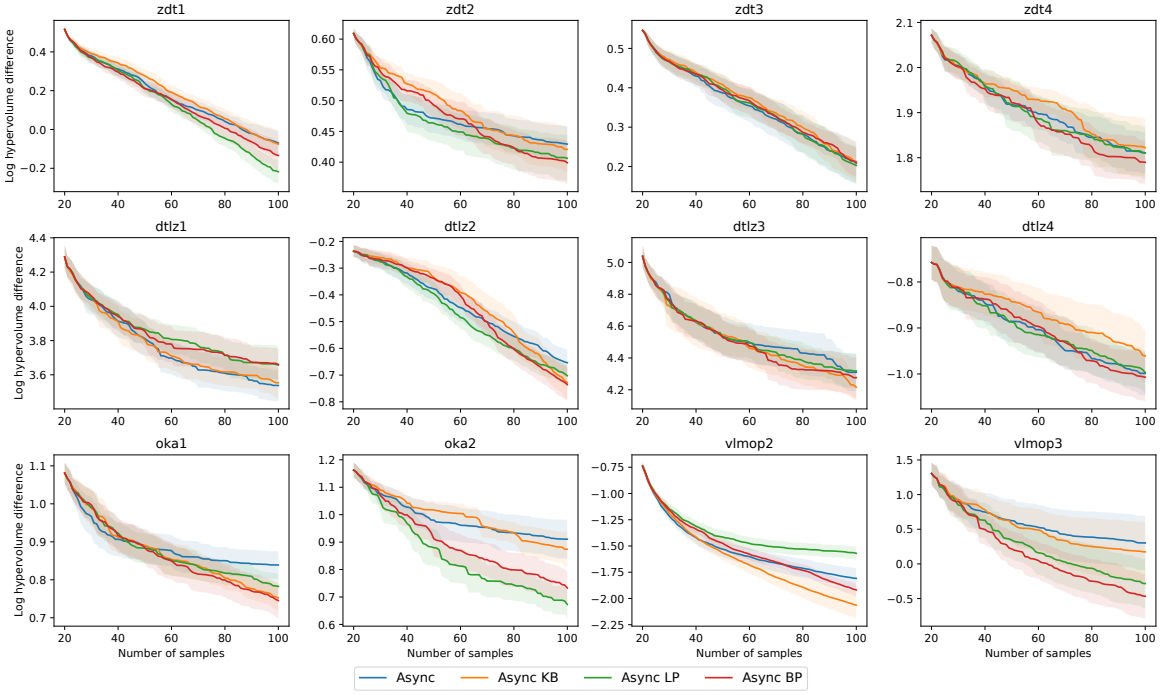


Figure 15: Performance comparison between variants of asynchoronous MOBO algorithms with a batch size of 16.

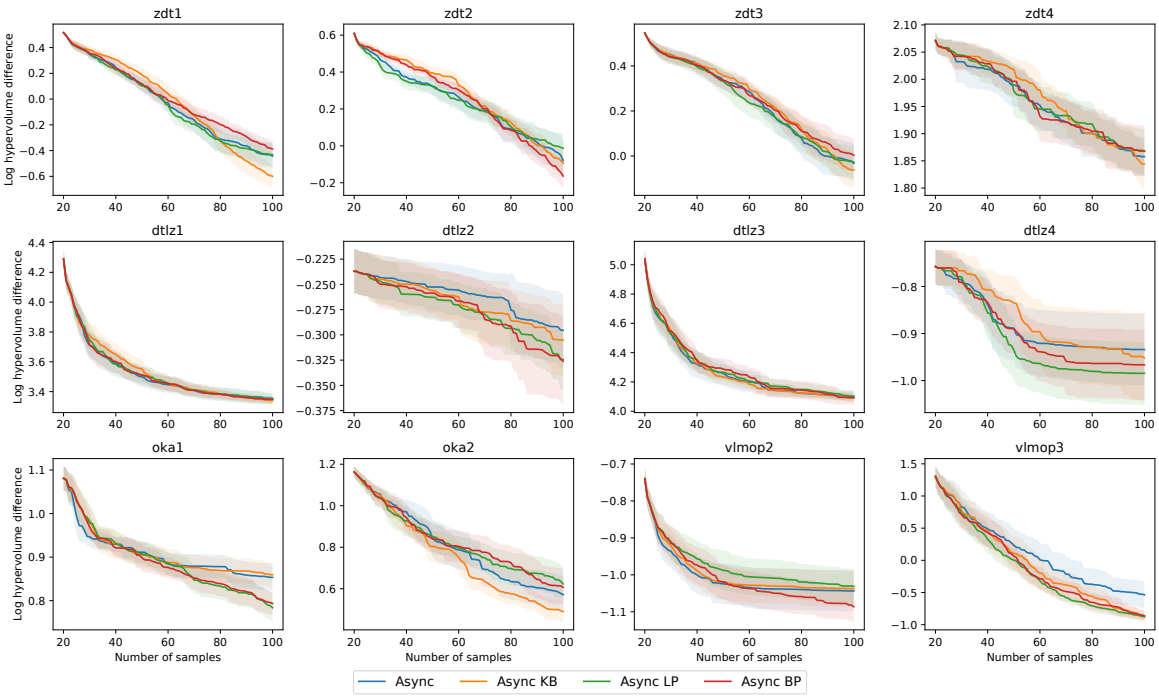


Figure 16: Performance comparison between variants of asynchoronous MOBO algorithms with EI as acquisition function.