Electronic Supplementary Material (ESI) for Lab on a Chip.

Meta optimization based on real-time benchmarking of multiple surrogate models for autonomous flow synthesis

Authors: Amirreza Mottafegh+, Dr. Gwang-Noh Ahn+, Prof. Dong Pyo Kim*

Center for Intelligent Microprocess of Pharmaceutical Synthesis,
Department of Chemical Engineering,
Pohang University of Science and Technology (POSTECH),
Pohang 790-784,
Republic of Korea
E-mail: dpkim@postech.ac.kr

+ These authors contributed equally to this work

Supporting information for this article is given via a link at the end of the document.

Supporting Information

Table of contents:

1. Bayesian optimization

1.1. Various surrogate models for Bayesian optimization

Experimental design via Bayesian optimization (EDBO) for optimization of reaction conditions begins by collecting initial reaction result data (synthetic yield or conversion) via a design of experiments (DOE) or drawing from existing outcomes for a given search space. These initial data are employed to train a probabilistic surrogate model which is built by conditioning a prior over functions, capturing our assumptions about the reaction's response surface (smoothness, experimental noise, etc.) and making it attainable to infer the position of the global optimum. The most basic needs of an effective surrogate model are the capability of making predictions and estimating variance. Thus, in general, Bayesian optimization (BO) can be implemented with many different models depending on the problem. For example, for optimizations over discrete or categorical domains, the response surface may be best represented by a random forest (RF) model. RF is an ensemble of decision trees representing a set of piecewise functions trained by randomly subsampling the data. While RF models do not provide an estimate of variance, in practice, one can use the empirical variance across the trees in the ensemble. RF models tend to show good prediction performance close to the training data but are poor extrapolators similar to other tree-based regressors. Thus, far away from the training data, the individual trees may give similar predictions resulting in the underestimation of variance. This can hamper exploration and result in the optimizer entrapped in local maxima. For continuous domains, it is common to assume the unknown function is sampled from a Gaussian process (GP). GP uses a distribution over functions defined by its prior mean and covariance kernel. In GP regression, the selection of kernel specifies the shape of functions in the distribution. Importantly, for GPs,

predictions and variance estimates can be calculated analytically. Gradient Boosting (GB) regressor is a generalization of boosting to arbitrary differentiable loss functions. GB is an accurate and effective procedure that can be used for regression and classification problems in diverse areas. GB supports several different loss functions for regression which can be specified via the argument loss; the default loss function for regression is squared error. Neural Network Ensemble (NNE)[1] are custom-written regressors that are specifically implemented using the sklearn regressor template to be compatible with the skopt optimization algorithm used in the optimization process. These regressors are designed to implement the regular fit and predict API, with the added functionality that the predict function can return the standard deviation of the predictions if wanted. An ensemble of NNs is preferred to a single NN because a single NN is deterministic after being trained.

1. **Gaussian Process:** *f(x)* is a collection of variables with a joint distribution, and can be completely described by the mean *m(x)* and covariance *k(x,x')*

$$m(x) = E[f(x)]$$
$$k(x,x') = E[(f(x) - m(x))(f(x') - m(x')]$$

where $E[f(x)]$ is the expected value of the function $f(x)$. When using GP for machine learning regression, it is essential to select the covariance function or kernel function $k(x,x')$ to define and smooth the prior distribution for the function that maps the inputs (reaction conditions) to the output (objective values: synthetic yield or conversion).

The posterior distribution is then calculated using the observed values from the dataset by the regression algorithm. Obviously, the choice of the kernel function with its corresponding hyperparameters will have a great effect on the

performance of the GP algorithm. In the presented research, the radial basis function (RBF) kernel which is a popular kernel function used in various kernelized learning algorithms was selected as the kernel function.

The RBF kernel on two samples $x \in R^k$ and x', represented as feature vectors in some input space, is defined as:

$$k(x.x') = \exp\left(\frac{\|x - x'\|}{2\sigma^2}\right)$$

$\|x - x'\|^2$ is recognized as the squared Euclidean distance between the two feature vectors while σ is a free parameter. An equivalent definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$. As for hyperparameters of this anisotropic kernel, length scale was set tunable in the range (0.000001,100000).

2. **Random Forest:** In RF, each tree in the ensemble is built from a sample drawn with a replacement (i.e., a bootstrap sample) from the training set. Likewise, when splitting each node during a tree's construction, the best split is found either from all input features or a random subset. The purpose of the origins of randomness is to diminish the variance of the forest estimator. Undoubtedly, individual decision trees typically reveal high variance and tend to overfit. The insinuated randomness in forests delivers decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. RF reaches a reduced variance by integrating diverse trees, sometimes at the cost of a slight increase in bias. In practice, the variance reduction is often noteworthy, hence delivering an overall better model.

3. **Gradient Boosting:** Gradient Boosting **(**GB) regressors are additive models whose prediction y_i for a x_i given input is of the following form:

$$y_i = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i)$$

where h_m are estimators called weak learners in the context of boosting. GB uses decision tree regressors of fixed size for weak learners. The constant M corresponds to the number of estimators parameter. Similar to other boosting algorithms, GB is built in a greedy fashion:

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

where the newly added tree h_m is fitted in order to minimize a sum of losses L_m, given the previous ensemble F_{m-1}:

$$h_m = \arg\min_h L_m = \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$$

Where $l(y_i, F(x_i)$ is defined by the loss parameter.

4. **Neural Network Ensemble:** In Neural Network Ensemble (NNE), the output of a hidden (intermediate) layer from a single perceptron in the network is given by**:**

$$Z_m = \sigma\left(\alpha_{m0} + \sum_{p=1}^{P} \alpha_{mp} X_p\right), m = 1,\ldots,M$$

where Z_m is the output of the m^{th} node of a hidden layer with M nodes, X_p are the inputs to this hidden layer, α_{mp} are the weights connecting the previous layer with this layer, and σ is the activation function which we have chosen to be the Rectified Linear Unit (ReLU) for hidden layers, and linear activation function for output layer, to handle nonlinear relationships between the inputs and outputs

$$\sigma(x) = max(0,x)$$

The reason for differentiation between the output layer and the hidden layer's activation function is that for regression tasks, using a linear activation function in the output layer is generally a good choice because it allows the network to predict a wide range of continuous values. This is in contrast to activation functions like ReLU (rectified linear unit), which have a bounded output range of 0 to infinity. After training, the weights α_{mp} of the NN become fixed; therefore, it will always genera the same output when given a certain input. However, if the NN is trained again on the same data, the weights will become slightly different, affected by random reinitialization of the weights, and the new prediction will be different. Therefore, our first implementation of the Neural Ensemble Regressor is simply to retrain the NN several times (the ensemble size is a hyperparameter that can be tuned) and then output the mean and standard deviation of the predictions. This NN is implemented using the multilayer perceptron regressor (MLPRegressor) in sklearn as well as the Sequential network using TensorFlow package.

1.2. Various acquisition functions for Bayesian optimization

Acquisition function selection was carried through entirely on an experimental basis by approximating BO performance on the four development objectives. Over the span of our studies, we examined popular acquisition functions from the BO literature expected improvement *(EI)*, probability of improvement *(PI)*, *GP Hedge* (GPH, based on probability of cumulative gains), and lower confidence bound *(LCB)* with $\kappa = 1.96$ as hyperparameter Controlling how much of the variance in the predicted values should be taken into account. Also, for minimizing the acquisition function, 'sampling' and 'lbfgs' methods were configured automatically on the basis of the base estimator (optimizers'

regressor), and the space searched over. These functions applied to all the studied datasets, covering all the concentrated surrogate models.

2. Meta optimization

2.1. Meta optimizer for real-time benchmarking

Real-time benchmarking is performed for four different surrogate model building methods (GP, RF, GB, and NNE) implemented via the skopt library package in Python. The built-in optimization functions ('*dummy_minimize*', '*forest_minimize*', '*gbrt_minimize*', and '*gp_minimize*') of skopt library are used to run the optimization using random sampling (dummy), RF, GB, and GP as their respective surrogate models, respectively. The optimization function for GP implementation can accept a user-defined GP regressor, which we have defined as one with a radial basis function (RBF) Kernel. Our custom neural ensemble and neural dropout surrogate models are implemented using the '*base_minimize*' function with the '*base_estimator*' specified as a neural ensemble regressor. For making each step segregated, we used '*ask*' and '*tell*' functions implemented in the optimizer class in skopt. After running some (3) initial iterations without the interference of the surrogate model, the real-time algorithm performs '*ask*' function from all the surrogate models. The output will be four suggested conditions from all the surrogate models (4 surrogate models). Then comes the core of the real-time benchmarking concept, where we have to sort all the suggested conditions and select the one which maximizes the objective function more than the other algorithms. Each suggested condition is given to the regressor associated with each surrogate model (optimizer) for ranking the suggested points. In another word, the regressor within the surrogate model, which is already trained with the previous conditions, predicts the outcome of the suggested point by the same surrogate model. Then the predicted values from

each surrogate model are aggregated into an array in order of magnitude, and then the surrogate model with the biggest predicted value is taken place. After finding the condition offered by the selected surrogate models, we will use '*tell*' function to update all of the surrogate models, and this loop continues until we are satisfied with the convergence of the objective value. To guarantee the reliability of the outcome optimization results, the calculations will be repeated multiple times (in this case 10 times), called as rounds. The following pseudocode in **Algorithm 1** shows the whole real-time benchmarking process.

Algorithm 1: Real-Time Benchmarking Process

```
input optimizers [optimizer1,optimizer2,optimizer3,optimizer4,optimizer5]
for number of rounds to repeat the calculations: (10)
input initial points to start the bayesian optimization
observed conditions <- initial points
for number of experiment runs to perform: (30)
I: train ML models within each optimizer's surrogate model on observed
points
II: ask all the optimizers to suggest a new condition
III: use trained ML models on (I) to predict the corresponding condition on
(II)
compare all the predicted objective values on III
IV: select the condition from (II) that the predicted objective is best
(maximum)
add selected condition on (IV) to observed conditions
go to (I)
```

2.2. Performance evaluation for BOs and MO

For the manifestation of performance of MO and BOs based on various surrogate model builders (GP, RF, NNE, and GB), 5 descriptive quantified values were defined and calculated. Furthermore, these were illustrated in a radar graph. Before proceeding with explaining all of the descriptive values, the structure of the whole extracted data must be scrutinized.

Given, *n* as the number of iterations in which BOs must be taken into account for optimizing the reaction, we performed this *n* iteration optimization process

multiple times (10 rounds), for each surrogate model, and we called each of these iterations, one round. Thus, we simulated optimization of a specific reaction (given the dataset) for *n* iteration and repeated this process according to the number of rounds (fixed at 10). Furthermore, for the projection of the real performance of each model in each ensemble, only maximum values obtained in the process of optimization were taken into consideration, and the objective values got updated only when any improvement occurred. Eventually, the averaged value of all the rounds for each iteration was calculated as an indicator of the performance of each optimizer (surrogate model).

Then the following notation can be used for derivation of the descriptive values as below.

N: total number of rounds

n: number of experiment iteration

X_i : Array of explored objective values within ith rounds

x_{ij} : individual objective values for jth experiment iteration in ith round

μ_j : averaged value of x_{ij} for jth experiment iteration over all the rounds where $\mu_j = \frac{\sum_{i=1}^{N} x_{ij}}{N}$. In this work we conducted the optimization of the virtual reactions through 30 iterations and repeated this whole process 10 different rounds with different random initial values. The μ_j value indicates the objective value achieved from optimization for a specific iteration of experiment, averaged over all rounds of calculation.

Accordingly, 5 selected values for describing each model, were calculated based on the aforementioned data structure.

1. **Standard Deviation:** this value was the indicator of the standard deviation of proposed objective values for a specific experiment through all rounds.

$$S_j = \frac{\sum_{i=1}^{N} (x_{ij} - \mu_j)}{N - 1}$$

Thus, for each experiment iteration we will have a standard deviation value which is calculated over all the ensembles. Then for quantifying this value for whole process, this value was calculated for all the experiment iterations, and then averaged value were the quantity as reported as std of each optimizer.

$$std = S_j = \frac{\sum_{j=1}^{N} S_j}{n}$$

As a result, for better demonstration in the provided radar graph, the normalized reciprocal of calculated std value, similar to a concept named as signal-to-noise ratio (SNR), was used.

$$SNR = \frac{1}{std}$$

2. **Acceleration Factor (AF):** This value shows how much faster an optimizer could reach a specific value compared to the baseline random sampling model. To calculate this value, a fixed reached objective value is considered as basis of the calculation, and the ratio of number of iterations needed to reach this objective value from BO model and number of iterations needed to reach this value for baseline (random sampling) model is calculated. Then this calculations is repeated for all the experiment iterations (30 iterations in here) and the average value is represented as the total AF value of an specific optimizer. The final achieved results then are normalized for better demonstration of performance.

$$AF = \frac{i_{BO}}{i_{Random}}$$

3. **Maximum Value:** this value was simply selected as the average of all the maximum values proposed by a specific model in each ensemble.

$$MV = \frac{\sum_{i=1}^{N} max(X_i)}{N}$$

4. **Average Value:** this value projects the averaged performance of each surrogate model over total number of ensembles and experiment iterations.

$$AV = \frac{\sum_{i=1}^{N}\sum_{j=1}^{n} x_{ij}}{N \times n}$$

5. **Enhancement Factor (EF):** the concept for calculating this value is almost similar to the dummy distance value, except the ratio of the model proposed value should be replaced by numerical distance.

$$EF = \frac{\sum_{j=1}^{n}\left(\frac{\mu_j^{model}}{\mu_j^{dummy}}\right)}{n}$$

3. Supporting Tables

3.1. Input feature space for organic chemistry datasets

Table S1. Chemoselective[2] metalation dataset input feature space. The dataset has size 111.

Parameter	Kind	Range	Description
C	Discrete	[0.1,0.4]	Concentration of Xylene [M]
S	Categorial	[THF,2-MeTHF]	Type of Solvent
L	Categorial	[n-BuLi, t-BuLi]	Type of Lithium
E	Discrete	[1.05,1.5]	Equivalent of Base
t	Continuous	[0.25,87.9]	Residence Time [s]
T	Discrete	[-78,25]	Reaction Temperature [^{o}C]

Table S2. Thioquinazolinone formation[3] reaction dataset input feature space. The dataset has size 96.

Parameter	Kind	Range	Description
F	Discrete	[6,24]	Flowrate [ml/min]
V	Discrete	[2,787]	Reaction Volume [ml]
t	Discrete	[-20, 20]	Residence Time [s]

Table S3. Aldol condensation[4] reaction dataset input feature space. The dataset has size 131.

Parameter	Kind	Range	Description
M2	Continuous	[1,43.57]	Concentration of Acetone [M]
M3	Continuous	[0.02,0.2]	Concentration of NaOH [M]
T	Continuous	[30, 70]	Reaction Temperature [^{o}C]

t	Continuous	[5,15]	Residence Time [s]

Table S4. S_NAr[5] reaction dataset input feature space. The dataset has size 273.

Parameter	Kind	Range	Description
r	Continuous	[0.5,3]	Ratio of reagent
C	Continuous	[0.2,0.4]	Reagent Concentration [M]
t	Continuous	[2.5, 6]	Residence Time [s]
T	Continuous	[60,160]	Reaction Temperature [^oC]

3.2. Hyperparameters selected for implemented ML models

Hyperparameters for critical ML models used for the construction of the predictive experiment emulator, as well as for the surrogate model regressors, were tuned using K-fold cross-validation through grid search method.

Table S5. Hyperparameters investigated for NNE showcasing the rank between total searched grid parameters, the architecture of the network (number of hidden layers and neurons), activation function used for all layers, and number of networks employed in the ensemble.

	Rank	Architecture	Activation	Ensemble Size	RMSE (stdev)
Chemoselective metalation	1	(200,200)	tanh	10	1.87 (0.77)
	3	(20,100)	tanh	10	2.36 (1.02)
	10	(100,120)	tanh	10	2.57 (0.53)
	14	(20,100)	RELU	10	2.63 (0.94)
	16	(90,120,100)	tanh	5	2.69 (1.23)
	19	(120,20,30)	RELU	10	2.75 (1.64)
	70	(5,10)	Linear	5	4.12 (1.51)

Thioquinazolinone formation	1	(20,100)	tanh	5	2.07 (1.11)
	3	(60,100,10)	RELU	5	2.13 (0.88)
	9	(100,20)	RELU	10	2.25 (1.34)
	13	(50,50,50)	RELU	10	2.41 (0.94)
	18	(100,120)	tanh	10	2.44 (1.39)
	19	(10,20,10)	RELU	5	2.45 (0.86)
	70	(10,20,10)	Linear	10	3.76 (2.98)
S_NAr	**1**	(200,120)	tanh	10	2.07 (0.96)
	2	(120,100)	tanh	10	2.11 (0.23)
	8	(120,100)	tanh	10	2.16 (1.42)
	13	(20,100)	tanh	10	2.81 (2.19)
	18	(120,100)	RELU	10	3.17 (1.36)
	21	(100,20)	RELU	10	3.20 (2.02)
	30	(100,20)	Linear	10	3.49 (2.19)
Aldol condensation	**1**	(180,120)	tanh	10	1.42 (0.67)
	4	(100,100)	tanh	10	1.49 (1.03)
	6	(100,20)	tanh	5	1.57 (1.01)
	11	(100,20)	tanh	10	1.81 (1.05)
	15	(100,100)	Linear	10	1.97 (1.98)
	17	(120,100)	RELU	10	2.06 (1.02)
	20	(100,100)	RELU	10	2.31 (1.40)

Table S6. Hyperparameters investigated for GP, showcasing the utilized kernel and the kernel length scale range.

	Kernel	Length Scale	RMSE (stdev)
Chemoselective metalation	RBF	(0.000001,100000)	16.12 (3.29)
	Matérn52	(0.000001,100000)	19.22 (3.86)
	Matérn32	(0.000001,100000)	18.76 (2.72)
	Matérn12	(0.000001,100000)	22.12 (4.09)
Thioquinazolinone formation	RBF	(0.000001,100000)	24.44 (5.94)
	Matérn52	(0.000001,100000)	34.63 (4.11)
	Matérn32	(0.000001,100000)	17.54 (4.15)
	Matérn12	(0.000001,100000)	29.08 (6.82)
S_NAr	RBF	(0.000001,100000)	9.87 (2.64)
	Matérn52	(0.000001,100000)	21.87 (2.52)
	Matérn32	(0.000001,100000)	15.72 (3.75)
	Matérn12	(0.000001,100000)	21.68 (2.89)
Aldol condensation	RBF	(0.000001,100000)	18.19 (3.48)
	Matérn52	(0.000001,100000)	25.18 (6.59)
	Matérn32	(0.000001,100000)	18.22 (2.61)
	Matérn12	(0.000001,100000)	21.91 (4.34)

Table S7. Hyperparameters investigated for RF, showcasing the maximum depth of the tree, the number of trees in forest, and the minimum number of samples required to split an internal node.

Dataset	Max Depth	Estimator	Min Sample Split	RMSE (stdev)
Chemoselective metalation	90	100	8	1.88 (0.57)

Thioquinazolinone formation	110	100	8	1.69 (0.88)
S_NAr	90	100	8	1.92 (0.61)
Aldol condensation	110	100	8	2.22 (1.04)

Table S8. Hyperparameters investigated for GB, showcasing the maximum depth of the individual regression estimators, the number of boosting stages to perform, learning rate, and the fraction of samples to be used for fitting the individual base learners.

Dataset	Max Depth	Estimator	Learning Rate	Subsample	RMSE (stdev)
Chemoselective metalation	4	1000	0.01	0.5	2.02 (1.17)
Thioquinazolinone formation	4	2000	0.01	0.5	2.56 (1.25)
S_NAr	2	1000	0.1	1	1.44 (0.84)
Aldol condensation	4	2000	0.01	0.5	2.39 (1.16)

Table S9. Hyperparameters investigated for XGB, showcasing the Maximum depth of a tree, learning rate, L1 regularization term on weights (Alpha), and the algorithm's conservation index (gamma).

Dataset	Max Depth	Alpha	Learning Rate	Gamma	RMSE (stdev)
Chemoselective metalation	3	1	0.01	3	1.48 (0.62)
Thioquinazolinone formation	6	1	0.01	5	1.72 (1.16)
S_NAr	3	3	0.1	1	1.15 (0.71)
Aldol condensation	6	2	0.01	1	1.49 (0.88)

4. Supporting Figures

4.1 Evaluation of ML models used for predictive emulators

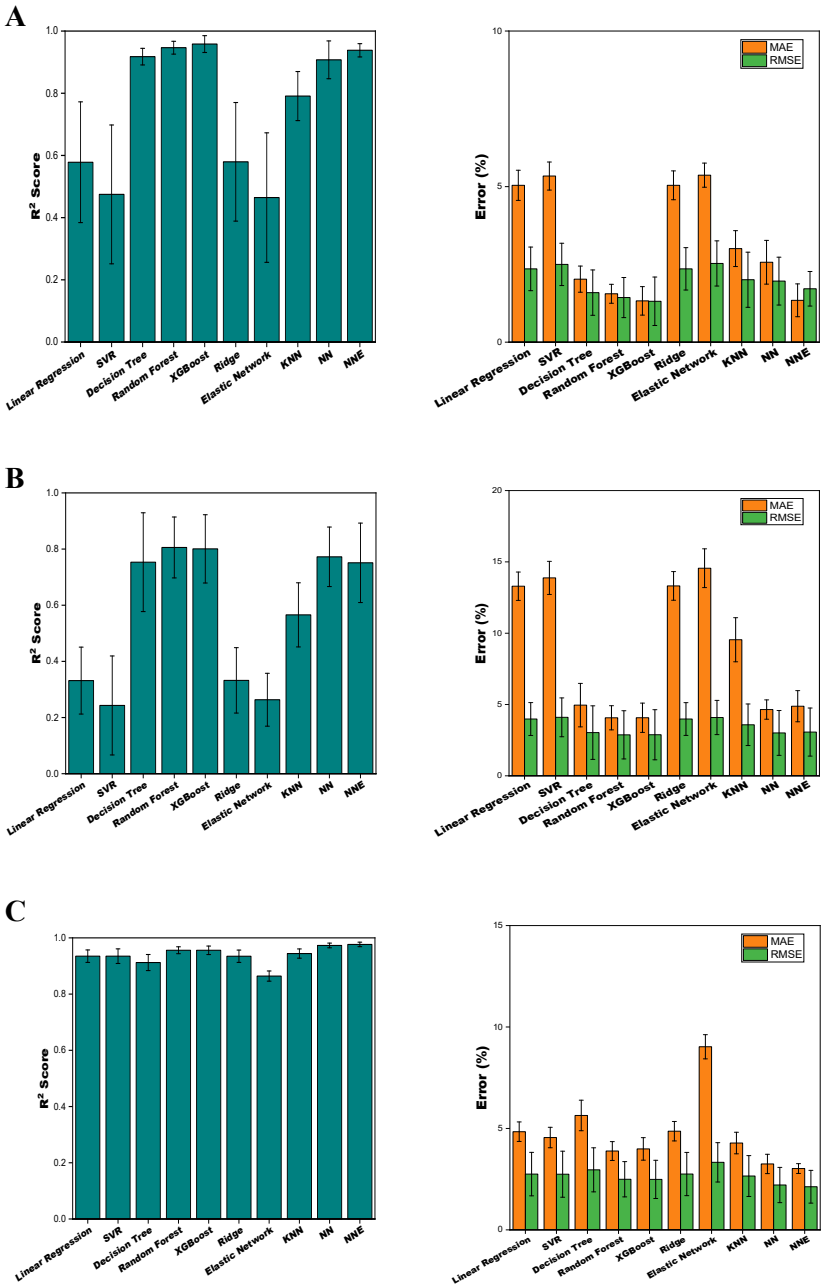


Figure S1: Mean absolute error (MAE), round mean squared error (RMSE), and R^2 score values of multiple machine learning algorithms for **(A)** Thioquinazolinone formation reactions, **(B)** aldol condensation, and **(C)** S_NAr associated with multiple machine learning algorithms.

4.2 Radar graphs with respect to each acquisition function

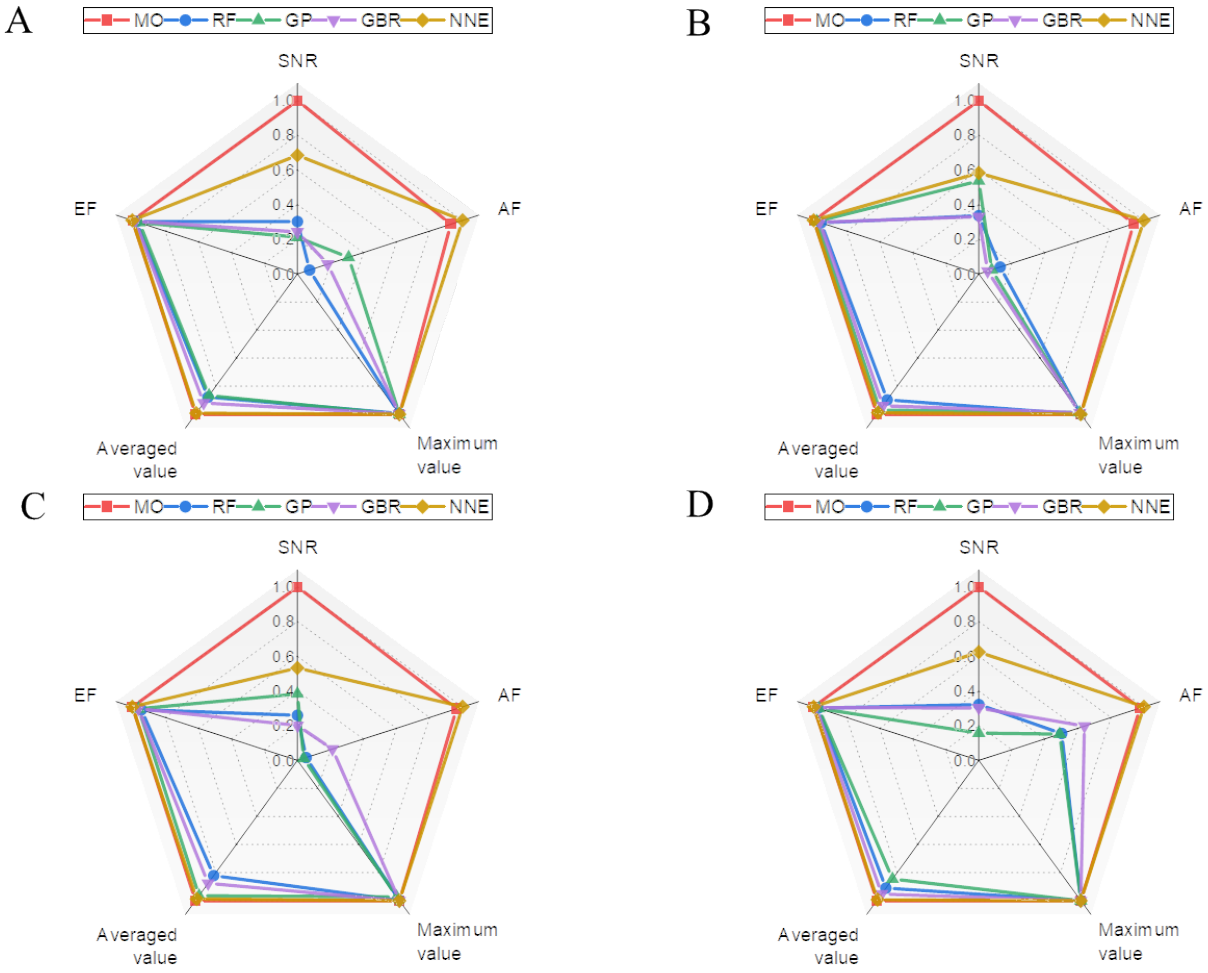


Figure S2: Radar graphs for chemoselective metalation reaction for **(A)** *EI*, **(B)** *GP Hedge*, **(C)** *LCB*, and **(D)** *PI* acquisition functions.

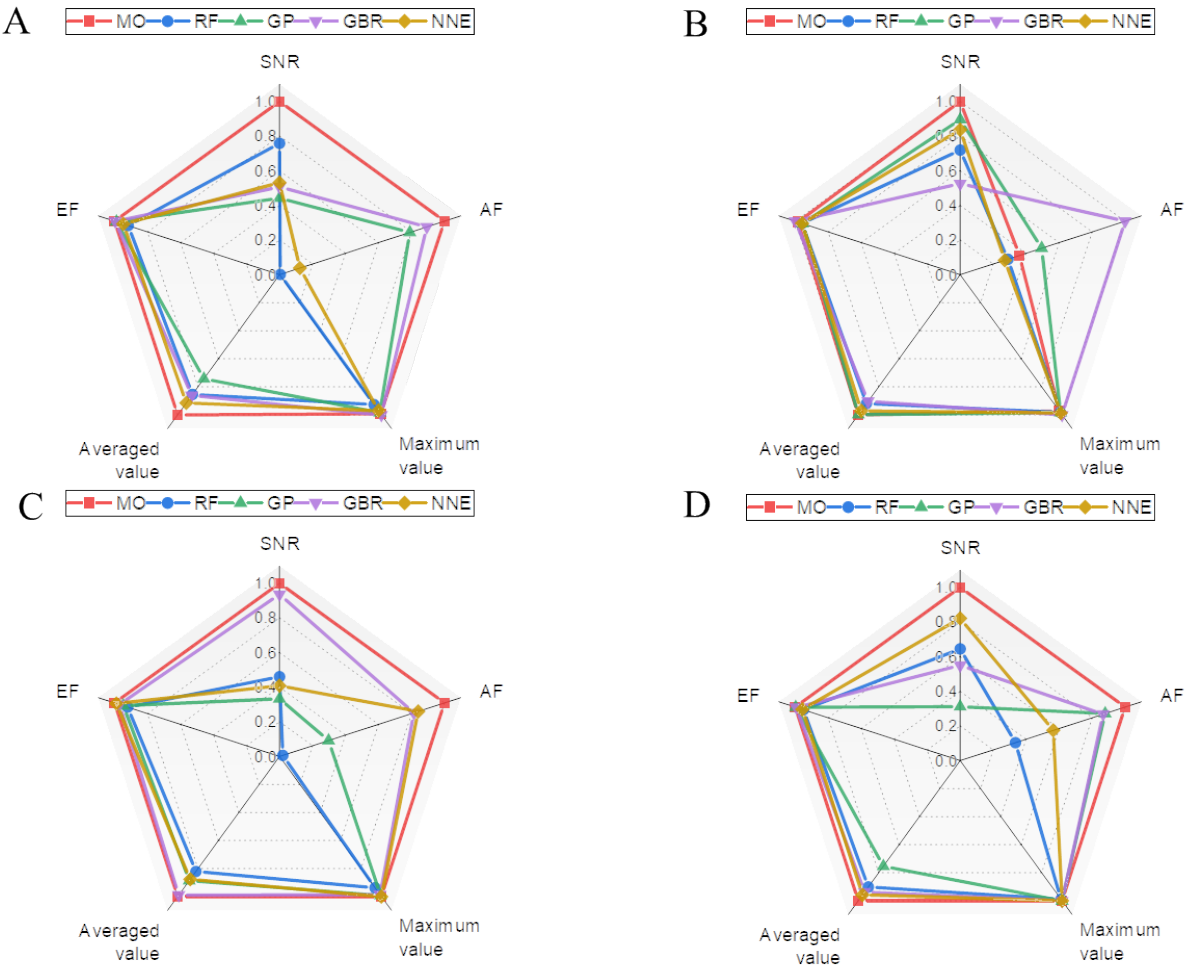


Figure S3: Radar graphs for thioquinazolinone formation reaction for **(A)** *EI*, **(B)** *GP Hedge*, **(C)** *LCB*, and **(D)** *PI* acquisition functions.

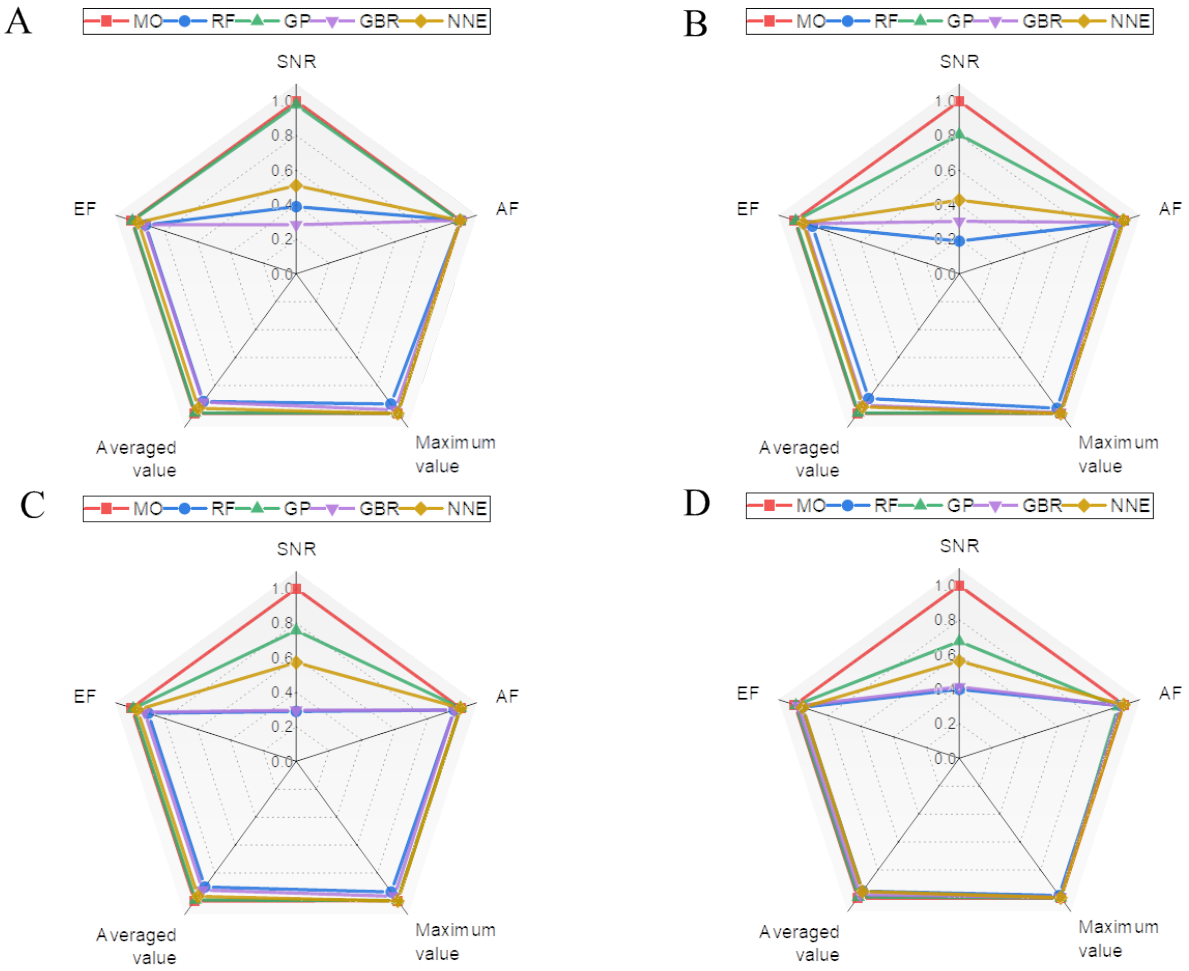


Figure S4: Radar graphs for aldol condensation reaction for **(A)** *EI*, **(B)** *GP Hedge*, **(C)** *LCB*, and **(D)** *PI* acquisition functions.

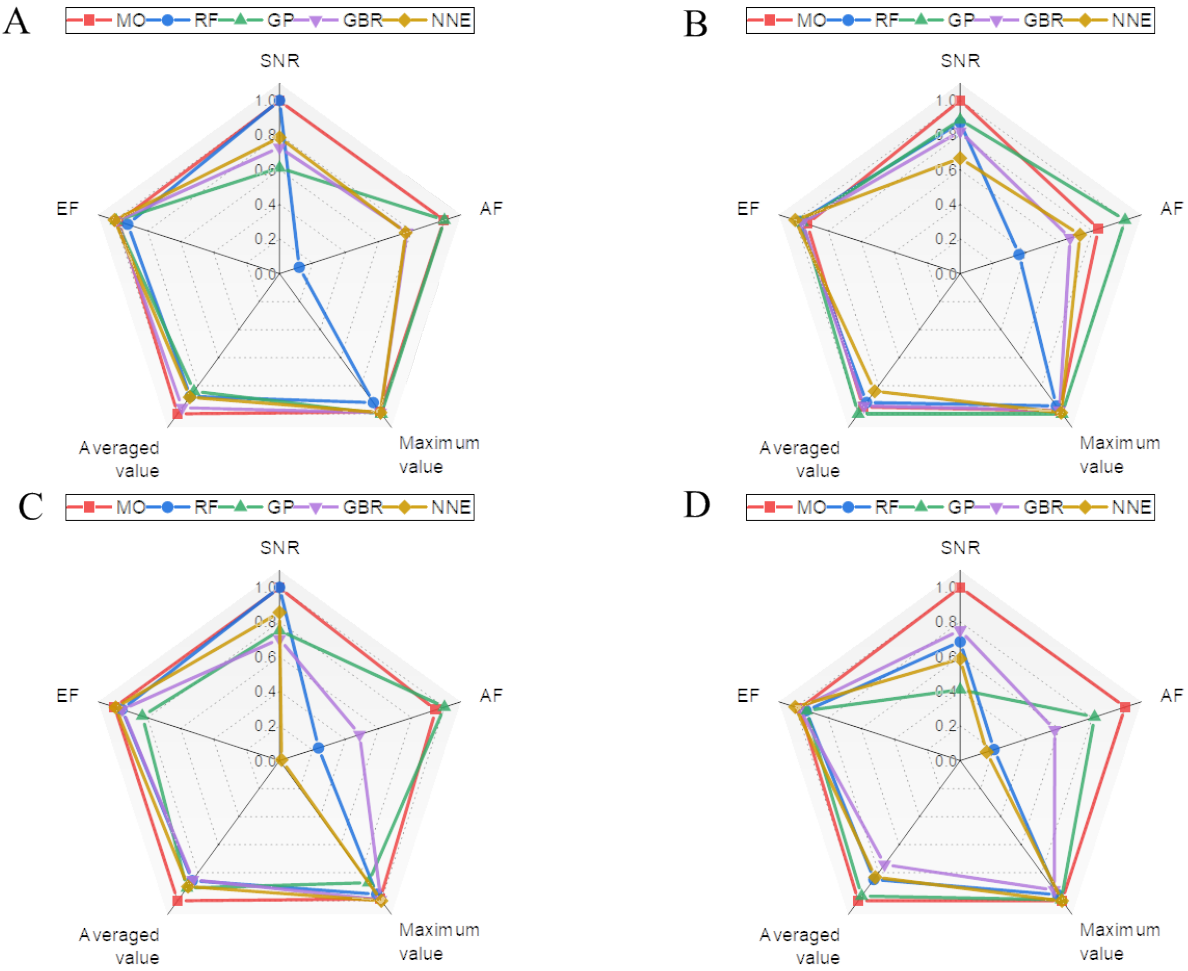


Figure S5: Radar graphs for S_NAr for **(A)** *EI*, **(B)** *GP Hedge*, **(C)** *LCB*, and **(D)** *PI* acquisition functions.

4.3 Comparative performance of MO and Selection policies

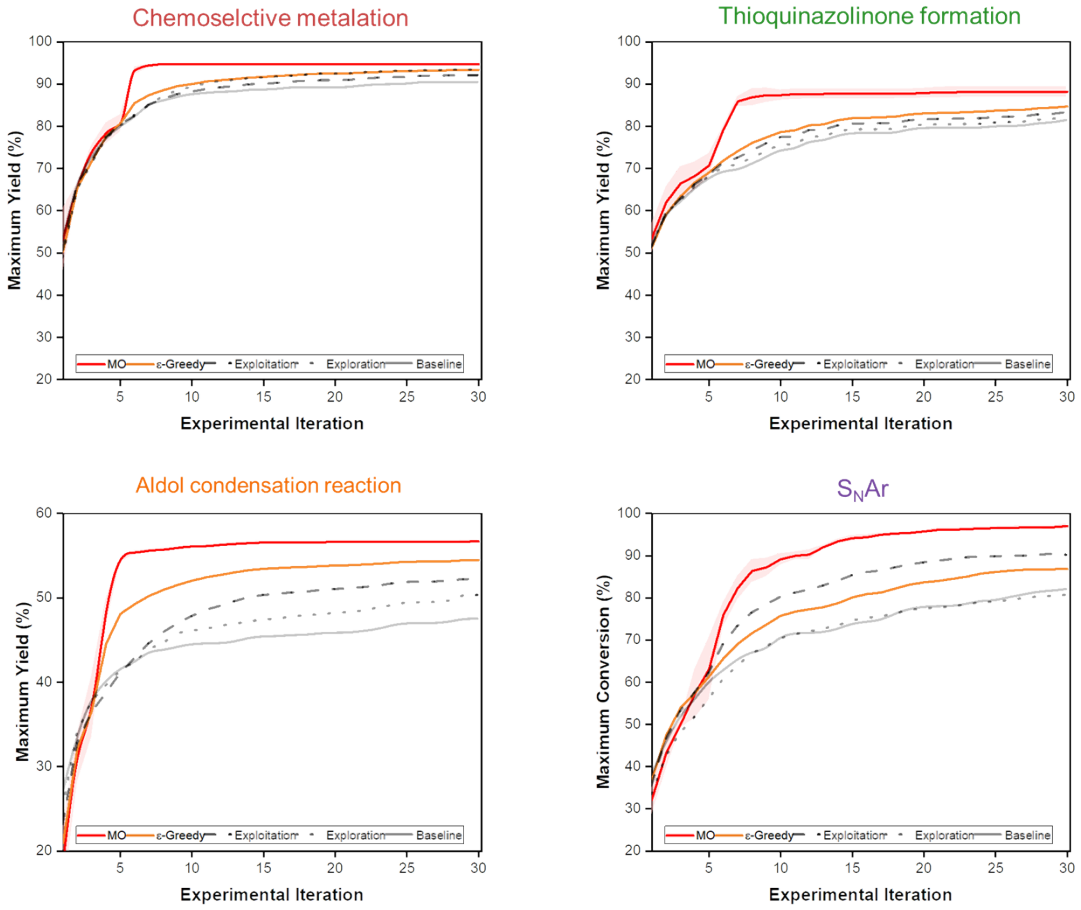
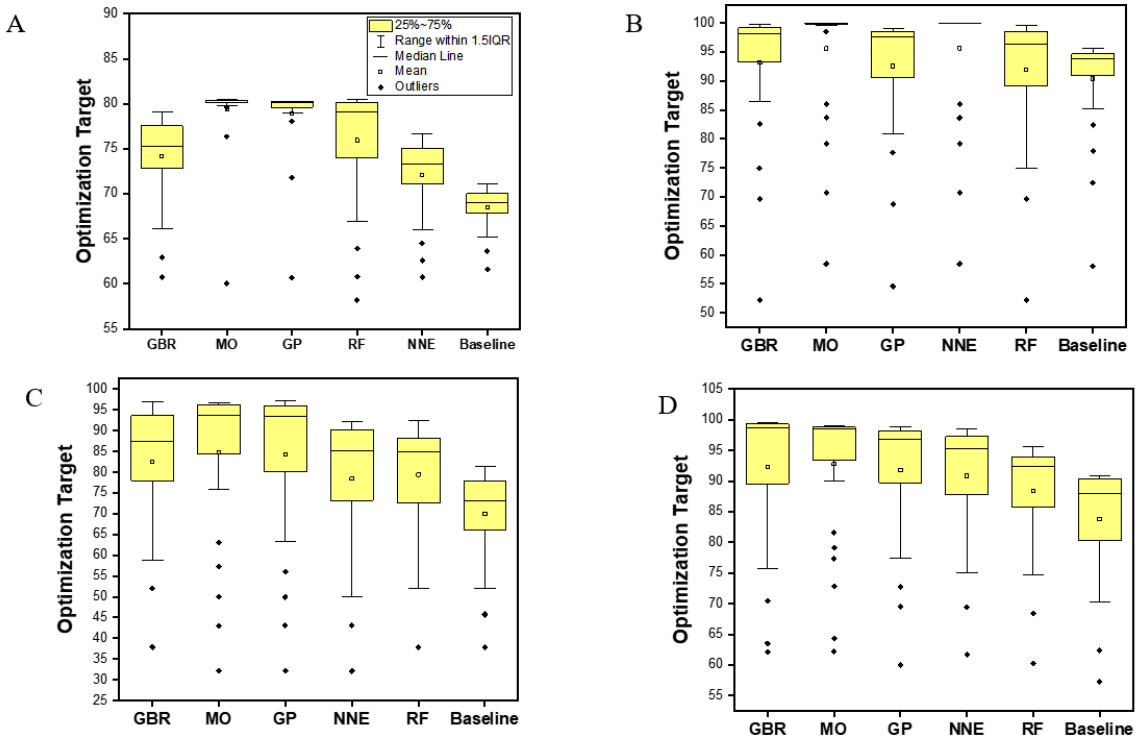


Figure S6: Statistical performance of MO and three different selection policies including purely exploitative (mean maximization), purely exploratory (variance maximization), and balanced exploration and exploitation via random selection of the next point with probability ε-Greedy with ε=0.1), for all studied reactions. For the case of MO, the averaged maximum objectives in terms of four different acquisition functions (*EI*, *PI*, *GP Hedge*, and *LCB*) are depicted in the solid line, and the distribution according to the difference in the acquisition functions is indicated by shaded regions.



4.4 Optimization outcome Box plots

Figure S7: Box plots as a method for demonstrating the distribution of achieved objective functions for **(A)** aldol condensation, **(B)** chemoselective metalation, **(C)** S_NAr, and **(D)** Thioquinazolinone formation reaction. The smaller range of achieved objective yield is interpreted as better performance of the optimizer.

References

[1] Y.-F. Lim, C. K. Ng, U. S. Vaitesswar, K. Hippalgaonkar, *Adv. Intell. Syst.* **2021**, *3*, 2100101.

[2] H. J. Lee, H. Kim, D. P. Kim, *Chem. - A Eur. J.* **2019**, *25*, 11641–11645.

[3] H. Kim, H. J. Lee, D. P. Kim, *Angew. Chemie - Int. Ed.* **2015**, *54*, 1877–1880.

[4] L. A. Jeraal M, Sung S, *Chemistry–Methods* **2021**, *1*, 71–77.

[5] A. M. Schweidtmann, A. D. Clayton, N. Holmes, E. Bradford, R. A. Bourne, A. A. Lapkin, *Chem. Eng. J.* **2018**, *352*, 277–282.