Chair of Scientific Computing in Computer Science
School of Computation, Information and Technology
Technical University of Munich

Guided Research Report: Bayesian Optimization of Material Synthesis Parameters with Gaussian Processes

Forschungsarbeit unter Anleitung: Bayes'sche Optimierung von Materialsyntheseparametern mit Gauß-Prozessen

Josef Mayr

Examiner:
Dr. Felix Dietrich

Supervisor:
Dr. Felix Dietrich

Submitted:
Munich, 21.04.2023

I hereby declare that this guided research is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 21.04.2023 Josef Mayr

Abstract

Metal-organic frameworks (MOFs) are a class of porous crystalline materials that are used in various tasks such as catalysis, gas storage, gas separation. In the chemical synthesis of MOFs one of the objectives is to find reaction conditions under which the MOF formed during the reaction process fulfills certain criteria that can be formulated as optimization objectives. As the reaction process is time-consuming and costly, the number of failed experiments should be minimized along with the other optimization objectives.

Bayesian optimization has been found to be a sample-efficient strategy applicable to the optimization of an unknown black-box function. It iteratively suggests new points at which an improvement may occur. As such, it is a promising technique regarding the optimization of reaction conditions.

The objective of this work is to establish a framework that makes state-of-the-art Bayesian optimization accessible to chemists trying to optimize a MOF. The MOF that guided the design of the system is UiO-66, but the rich feature set provided by the framework is also applicable to other MOFs. In addition to single- and multi-objective Bayesian optimization, data processing capabilities, reaction kinetics, and human-in-the-loop functionality have been implemented in an straightforward-to-use manner.

Contents

1 Introduction

In chemistry, metal-organic frameworks (MOFs) are a class of materials used, among other, in catalysis, gas storage, gas separation, and membranes [17]. In the design of MOFs, finding reaction conditions at which a certain outcome is achieved is often a costly and time-consuming process. As such, one of the aims is to have to perform as few failing experiment iterations as possible until successful reaction conditions are found. This problem has proven to be non-trivial, and various optimization techniques have been applied to tackle it.

One of the techniques applicable to such a scenario is Bayesian optimization (BO). It has been found to provide a sample-efficient strategy for optimizing an unknown black-box function in an iterative manner. The inputs to this black-box function, in the case of reaction optimization, are the reaction conditions, the black-box function is the experiment performed in reality which then yields the output of some evaluation criteria that rate the outcome produced for the given reaction conditions. Bayesian optimization iteratively suggests new points that are likely to yield an improvement or that give information relevant for subsequent iterations.

In a work prior to this, Ortner had applied Bayesian optimization to the reaction process in her bachelor's thesis [14]. The objective of this work is to develop the approach further by providing a modularized framework that offers an accessible set of features useful for further research and the MOF design process. Among the features is Bayesian optimization, human-in-the-loop functionality that can be coupled with it, and the framework furthermore formulates a foundation for the integration of reaction kinetics into the optimization process. The main design objective was to implement features relevant for Bayesian optimization in a way that their application is straightforward to use for chemists to enable independent research while remaining efficient, modular, and extensible. Another objective was to lay a foundation for further development.

The main question at hand is whether Bayesian optimization is a suitable optimization approach, to compare different approaches – also among the various Bayesian optimization approaches available – and to make further research tractable and lessen the time and cost for further research.

The project was performed in cooperation with Lena Pilz and Dr. Manuel Tsotsalas from the Karlsruhe Institute of Technology (KIT). Both gave valuable insight into MOFs, the chemistry and theoretical background involved in them, and the current optimization procedure. In the prior work by Ortner, Lena Pilz had conducted experiments and provided data reused for this work.

The paper is organized in the following manner: Section 2.1 gives a short overview of the chemical background, section 2.2 explains Gaussian processes and Bayesian optimization that employs them. Section 2.3 gives a short overview of related work. Section 3 then describes the current state of the framework and its core components.

2 Background

2.1 Chemistry Background

2.1.1 Metal-organic Frameworks

The substances of particular interest in this work are *metal-organic frameworks* (MOFs). They form a class of crystalline, yet porous materials that exhibit properties applicable in, for example, gas storage, gas separation, or catalysis. [17]

MOFs are constituted of *metal clusters* which are also referred to as *secondary building units* (SBUs), and *organic linkers*. A MOF is built by metal clusters that are connected by the organic linkers to form a porous crystal (see figure 1). This formation is not a reaction but an aggregation. This type of bonding is a reversible process.

SBUs, organic linkers, and the crystal structure formed by them can be viewed as graphs where the nodes are the atoms connected by edges, the bonds. An important property about these SBUs is the number of *points of extension* (POE) at which an SBU allows bonding with other SBUs via the organic

linkers. Schoedel and Rajeh [17] give an overview of different types classified by the number of POE. By the overall SBU topology, certain molecular symmetries are established that influence the layout of the MOF resulting after reaction.

The MOF of particular interest to this work is UiO-66 (the UiO stands for 'University of Oslo') which has four POE (see figure 2).

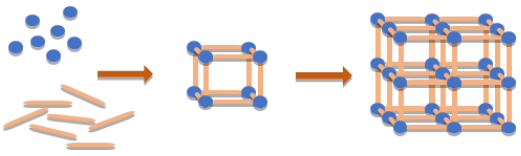
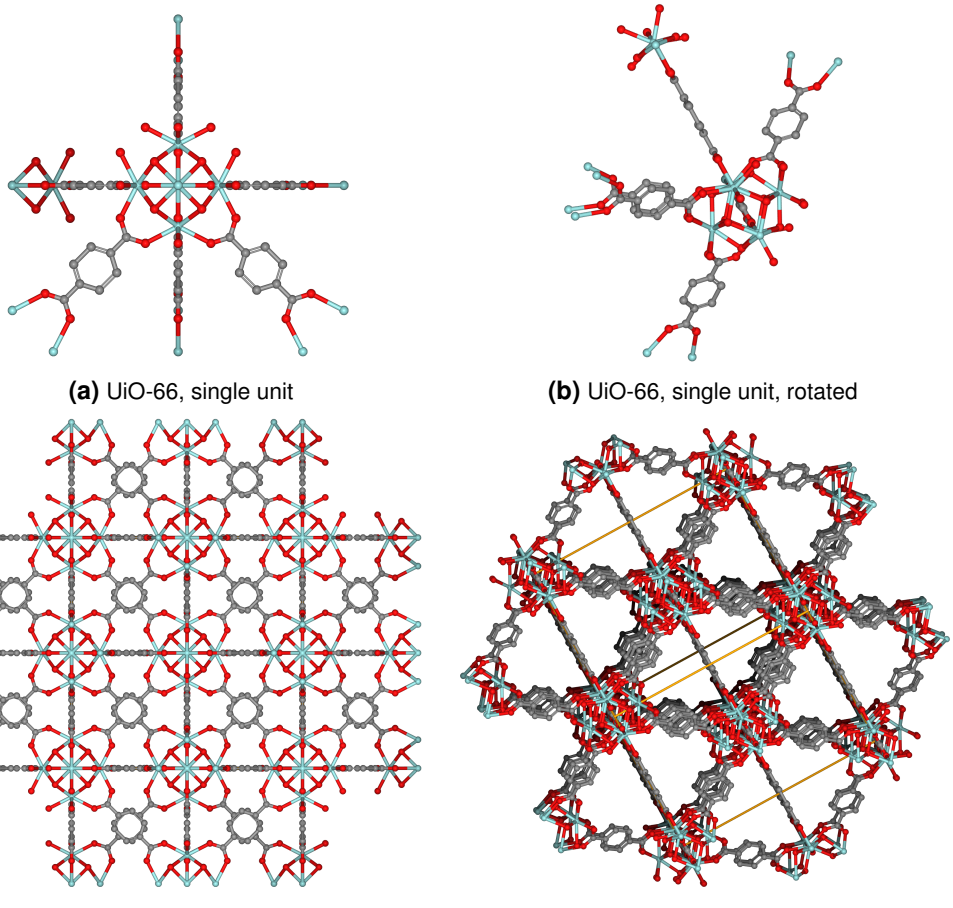


Figure 1 Crystal formation: The metal clusters (blue) and organic linkers (orange) form a smallest unit that then aggregates into a larger structure. Image kindly provided by Lena Pilz (KIT)

However, not only the topology of the SBU leads to different outcomes but also the different reaction conditions under which an aggregation happens. After a reaction process is performed with a certain configuration of reaction conditions, the MOF gets characterized. In the case of this work, this was performed via two criteria: *phase identity* and *crystallinity*. Phase identity is a binary value determined via *x-ray diffraction* (XRD). It indicates whether the intended substance emerged in synthesis. *Crystallinity* describes the ratio from crystalline to amorphous material – the objective is to obtain a high value for crystallinity. It is a continuous percentage value. The evaluation was performed by the KIT using the software DIFFRAC.EVA by Bruker [2] that computes crystallinity as a percentage value c and *amorphousness* which is $100\% - c$.



(a) UiO-66, single unit

(b) UiO-66, single unit, rotated

(c) UiO-66, crystal

(d) UiO-66, crystal, rotated

Figure 2 Four images of UiO-66. 2a and 2b show a single unit from two sides, 2c and 2d show a crystal comprised of multiple such units connected at the points of extension. Plot generated by [11] (identifier: *RUBTAK01*, the reader is encouraged to have a look at this 3D view of UiO-66).

The reaction conditions to optimize over are the temperature value at which to perform the reaction, the time for which to perform the reaction, the metal rate, and the modulator rate. Each are constrained

independently to a specific region. The specific value ranges are not seen as important for this work since the actual experiments will be performed in a follow-up work.

The overall objective however will be to predict reaction conditions such that crystallinity is maximized and phase identity established.

2.1.2 Chemical Kinetics of Metal-organic Frameworks and UiO-66

This section's aim is to shortly introduce chemical kinetics (also referred to as reaction kinetics) as required by this work. For a more in-depth introduction the user is referred to, for example, [19]. In the context of MOFs, ideas from polymerization have been applied [4].

The field of chemical kinetics is concerned with the rate at which reactions take place. These rates of reactions are based also on the reaction conditions discussed in section 2.1.1 that influence quantity and quality of the MOF. Chemical kinetics will however not be able to provide a qualitative insight as described by crystallinity and amorphousness – the objective rather is to attain quantitative insights by achieving a certain crystal size.

For the MOF reactions described in section 2.1.1, the initial reaction can be denoted as follows [4]:

$$mM + lL \xrightarrow{k_{\text{init}}} P_1$$

M and L are the metal clusters and organic linker, respectively, m and l are referred to as their respective stoichiometric coefficients. So, m units of metal M and l units of organic linker L bond together to form one unit of P_1, the smallest growth unit. From P_1, larger *oligomers* P_h, built from h smallest growth units, form up to a certain maximum size N. For $i, j \in \mathbb{N}, i + j \leq N, j \leq i$:

$$P_i + P_j \xrightarrow{k_{ij}} P_{i+j}$$

k_{init} is the temperature-dependent reaction rate constant for the initial reaction, k_{ij} is the reaction rate constant of the reaction between P_i and P_j. Dighe et al. [4] used the Arrhenius equation to estimate the reaction rate constants for reactions at new temperatures from a set of performed experiments. They furthermore argue for Flory's approximation, an approximation from polymerization that chooses $k_{ij} = \kappa$ for all pairs of oligomers P_i, P_j in the above set of reactions. κ was obtained from experiments.

Let $[M](t), [L](t), [P_1](t), ..., [P_N](t)$ denote the concentrations of metal, organic linker, and the respective oligomers over time. The rate of change of the concentration $\frac{\mathrm{d}[Q]}{\mathrm{d}t}(t)$ of a substance Q at time t is determined by the reaction rates of all reactions in the system involving Q, and the amount of Q being consumed or produced in each reaction as indicated by the respective stoichiometric coefficient. In the setting at hand, $[M](0)$ and $[L](0)$ are initialized to a certain concentration, and the oligomers P_h each initially have concentration $[P_h](0) = 0$

Therein, the rate of the initial reaction r_{init} and the reaction rates r_{ij} between P_i and P_j are described in terms of the reaction rate constants k_{init} and k_{ij} respectively and the concentrations of the substances involved in each reaction.

Upadhyay [19] describes the reaction rate for a reaction $aA + bB \xrightarrow{k} \ldots$ with reaction rate constant k as

$$r = k \cdot [A]^x \cdot [B]^y$$

where $[A], [B]$ are the concentrations of the respective substances.

However, for UiO-66 Dighe et al. [4] suggest that the reaction rate is affected by the build-up of oligomers P_h from latter reactions due to a phenomenon termed *autocatalysis* – a form of catalysis where a catalyst is formed as a product in the reaction chain [19]. The catalyst's presence increases the reaction rate which then additionally depends on its concentration. Dighe et al. determined that the initial reaction rate r_{init} of metal and organic linker is affected by such an autocatalytic behavior (see [4] and its supplementary information for detailed information and choices for the parameters of the reactions).

The idea is that knowing the concentration of the oligomers may make it possible to predict a distribution over the grain sizes of the crystals for certain reaction times. Having an analytical formulation of part of the problem also allows the augmentation of the the Bayesian optimization or solving inverse problems.

2.2 Mathematical Background

2.2.1 Gaussian Processes

In many typical scenarios, only a finite number of points are observed but the function of interest is actually defined for an infinite set of points. There are a lot of approaches to deal with such a problem, the idea of Gaussian processes is one such approach and based on a probabilistic model [20].

A *Gaussian process* (GP) is defined via two functions: the *mean function* $m(\mathbf{x})$, and the *covariance function* $\kappa(\mathbf{x}, \mathbf{x}')$. Both are defined for a function $f(\mathbf{x})$ of interest and are choices for the assumptions the GP makes. The mean function and covariance function are defined as follows:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ \kappa(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - \mu(\mathbf{x})) \cdot (f(\mathbf{x}') - \mu(\mathbf{x}'))] \end{aligned}$$

The Gaussian process is then denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}'))$$

The Gaussian process defines a prior over functions [13]. Often, the covariance function is directly modeled via a *kernel* κ over a pair $\mathbf{x}, \mathbf{x}'$, and the mean function is set to 0. Due to the use of the probabilistic framework, the choice for the mean function will not obstruct the predictive capabilities of the model. An overview over common kernel functions is given in [13] and [8]. The kernels aim to compute the covariance over the inputs $\mathbf{x}, \mathbf{x}'$ rather than relying on their corresponding values $f(\mathbf{x}), f(\mathbf{x}')$ in the co-domain. Many kernels come with hyperparameters that can either be defined beforehand or trained for as more observations become available.

The idea of Gaussian processes is that a finite set $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ of random variables defines a joint probability distribution over the function values $y_1, ..., y_n$ corresponding to each input, respectively. To model this joint distribution, the mean and kernel function are used to form a normal distribution:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is described via the covariance function κ and the mean $\boldsymbol{\mu} \in \mathbb{R}^n$ via the mean function m from the definition of the Gaussian process:

$$\begin{aligned} \Sigma_{ij} &= \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \mu_i &= m(\mathbf{x}_i) \end{aligned}$$

So, the parameters of the normal distribution are computed over the inputs in X and define the distribution over the function values.

From this joint distribution other distributions can be derived such as the posterior distribution that allows the incorporation of observations, or the posterior predictive distribution that allows prediction for a new data point that has no observation.

A remarkable feature of Gaussian processes is that they inherently provide a measure for uncertainty due to the reliance on probability distributions. In the following sections this property will play a significant role.

A more in-depth introduction to Gaussian processes is given in [20].

2.2.2 Bayesian Optimization

Bayesian optimization (BO) is an approach based on a wider concept referred to as surrogate modeling [8]. In surrogate modeling one performs the optimization of an expensive black-box function $f : \mathbb{R}^n \to \mathbb{R}$ by employing a surrogate model that is, in comparison, cheap to optimize.

The expensive black-box function can, for example, be experiments to be performed by a chemist that gets the reaction conditions and returns some evaluation criteria. The goal would be to optimize these criteria.

The general approach is typically to first perform a couple of experiments by employing some space-filling approach to create data the surrogate model is based on. The surrogate model is then used to obtain new points (or one new point) at which to evaluate the expensive black-box function.

In the context of *Bayesian optimization* (BO), Gaussian process regression (see section 2.2.1) is typically used as a probabilistic surrogate model, and an *acquisition function* is used to obtain the new experiments [8].

The Bayesian optimization step is iterated, for example, by a fixed number of iterations:

- Optional: Optimize acquisition function hyperparameters
- Maximize the acquisition function $\alpha(\mathbf{x})$ to obtain a batch $\mathbf{x}'_1, ..., \mathbf{x}'_q$ of batch size q of new candidates at which to evaluate
- Evaluate the expensive black-box objective $f(\mathbf{x})$ at these new candidates to obtain $\mathbf{y}'_1, ..., \mathbf{y}'_q$
- Add these data points to the dataset. This causes the posterior of the surrogate model to update.

As more data is acquired, the posterior of the underlying model will change based on the data. Therefore, the underlying Gaussian process learns with each iteration via the definition of its posterior. The acquisition function makes use of this posterior distribution to yield new candidates. If the batch size q is greater than 1, multiple candidates that should be performed in parallel can be determined. These experiments are performed without including the evaluations obtained from other values from the same batch on f. If q is chosen as 1 the candidates will include the results from the prior step. So, a batch size of 1 would be the typical choice if only a single evaluation can be performed at the same time.

The acquisition function itself is a heuristic that assigns a utility to each point in the space. This utility is to be maximized. Therefore, an optimization problem is solved over the acquisition function – which in return is based on the cheap surrogate model – in each iteration. The acquisition function performs a trade-off between exploration and exploitation: It aims to explore points where the uncertainty of the underlying model is high (exploration) or to optimize found optima even further (exploitation). There are, also depending on the scenario, different choices for acquisition functions that may weight points of the input space differently according to the two criteria, but there are also so-called non-myopic acquisition functions that do not only look at the utility assigned to the current step but rather also decide what may be a good choice of candidates regarding follow-up iterations [21]. Acquisition functions themselves can have further parameters that can be optimized – even during the execution of the algorithm as described in the optional step.

An influence neglected so far is that caused by noisy observations. If only noisy observations of $f(\mathbf{x})$ are observed it may be necessary to include this noise in the model of the problem. Such statistical noise can, for example, occur due to errors in the sensors, limitations on the number of observations, etc.

A more in-depth introduction to Bayesian optimization is given in [8].

2.2.3 Multi-objective Bayesian Optimization

So far, only a single objective $f : \mathbb{R}^n \to \mathbb{R}$ has been taken into consideration. In a multi-objective scenario, the codomain of $\mathbf{f}$ has a dimensionality $m > 1$, i.e., $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. In the following, the component functions of $\mathbf{f}(\mathbf{x})$ are going to be denoted as $f_1(\mathbf{x}), ..., f_m(\mathbf{x})$.

The objectives may be contradictory in the sense that choosing a new point $\mathbf{x}'$ that increases one objective's value $f_i(\mathbf{x}')$ may worsen another objective's value $f_j(\mathbf{x}')$. The points where none of the objectives can be improved without worsening another is called *Pareto frontier*. Figure 3 shows an example of what such a Pareto frontier may look like.

A solution $\mathbf{f}$ that cannot be improved in any component without worsening another component is called *Pareto optimal*. Typically, there is a whole set of such Pareto optimal points. This set is called *Pareto frontier*. The points not on the Pareto frontier are said to be *dominated* by some points that are on it.

In multi-objective Bayesian optimization, the goal is to apply the idea of Bayesian optimization in the multi-objective scenario.

The approach this work uses is called *noisy expected hypervolume improvement* (NEHVI) [3]. The approach it takes is to model each objective as a separate Gaussian process. The acquisition function NEHVI aims to maximize the expected hypervolume spanned by the estimated Pareto frontier and a user-specified reference point that must be dominated by all points on the frontier. In simpler terms its objective is to make the expected hypervolume dominated by the Pareto frontier as large as possible. This causes more values that are dominated by the actual Pareto frontier to be also dominated in the approximate Pareto frontier. As stated before, the surrogates for the objectives are modeled as Gaussian processes. So, an expected hypervolume is being computed using the observations that the Gaussian processes' posteriors take into consideration. NEHVI furthermore assumes that the observations of the objective values themselves are noisy.

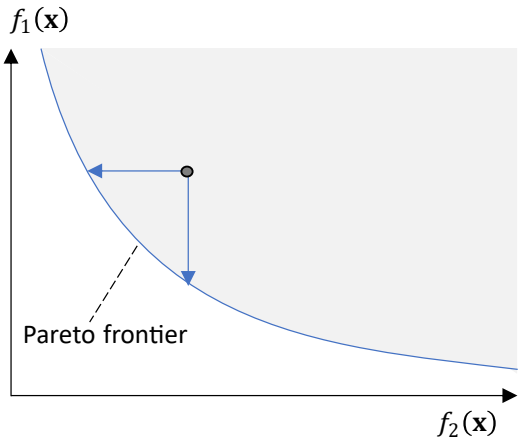


Figure 3 An example for a 2-objective minimization problem. The gray area and blue curve are the values $\mathbf{f}(\mathbf{x})$ plotted for all $\mathbf{x}$ in the input domain. The blue curve shows the Pareto frontier of all the non-dominated objective values $\mathbf{f}(\mathbf{x})$, i.e., the inputs $\mathbf{x}$ corresponding to the points $\mathbf{f}(\mathbf{x})$ on the curve are Pareto-optimal for $\mathbf{f}$. The gray area consists of points that are not Pareto-optimal: An $\mathbf{x}'$ can be found for any point $\mathbf{x}$ in it such that $f_i(\mathbf{x}') < f_i(\mathbf{x})$ for some objective but none of the other objective values decrease for $\mathbf{x}'$. For example, a Pareto improvement for the gray point would be to find points such that their objective values are moved along the blue arrows towards the frontier.

2.3 Related Work

In the design of experiments (DOE) many techniques from machine learning have been applied to the field of reaction condition optimization. In this section, a short overview over machine learning approaches in the field of reaction optimization is given with special focus on Bayesian optimization and metal-organic frameworks.

Two major types of machine learning approaches have been found in the literature: predicting the unknown reaction conditions for a new reaction from a set of reaction conditions known to be successful, and learning from a sequence of failed experiments until success. In the following, the main focus will be on the latter since learning from failed experiments is the aim of this paper.

Raccuglia et al. [15] built a kernel-based support vector machine (SVM) over the reactants' properties to predict experimental results and derived a decision tree from that. Raccuglia et al. compared various models and determined that such SVMs yield the highest accuracy among the compared models. Compared to the other approaches, there is no active learning strategy being followed.

Reker et al. [16] employed an adaptive random forest approach that allows an iterative adaptation to new data. They built a reaction optimization framework with the objective to minimize the number of experiments to be performed and found that the performance is comparable to that of human experts in their performed experiments.

Moosavi et al. [12] used a robot to reconstruct a series of failed experiments for the MOF HKUST-1 and used a genetic algorithm to predict the next batch of experiments to perform. After the sequence of experiments, an analysis was performed using random forests to describe the relative impact of the reaction conditions to the criteria used to evaluate the outcome, crystallinity and phase purity. The overall objective of the random forest approach was to learn chemical intuition that was then transferred to a similar material. Domingues et al. [6] extended on the work by Moosavi et al. [12] and used a genetic algorithm to optimize the reaction conditions for the MOF Al-PMOF.

Felton et al. [7] performed a benchmark of various techniques on multi-objective reaction optimization. While unrelated to MOFs, the paper may give evidence about the performance of the different approaches. It concluded that multi-objective Bayesian optimization outperformed the other algorithms in the test setup.

Greenhill et al. [10] give an overview on the usage of Bayesian optimization with respect to experimental design and, among other, give suggestions on how to incorporate prior knowledge.

Shields et al. [18] used Gaussian process-based single-objective Bayesian optimization for reaction optimization. They compared various techniques and in the end employed reaction descriptors obtained from density functional theory and the parallel expected improvement acquisition function. They found that Bayesian optimization on average outperformed human experts in a game and consistently achieved high results.

Xie et al. [22] used single-objective Bayesian optimization over a random forest together with a robotic platform to optimize a MOF. The decision to use random forests and not Gaussian processes was made to deal with categorical variables that were part of their model.

In the bachelor's thesis preceding this work, Ortner [14] applied single- and multi-objective Bayesian optimization with a focus on the MOF ZiF-8. In her work, kernel optimization was performed for the Gaussian process used by the single-objective Bayesian optimization approach and a second Gaussian process was used to approximate an outcome if a chemist were to do it. Due to the sparsity of the available data the predictive capability of this external Gaussian process was however deemed low. Both, the single- and multi-objective approach successfully predicted a reaction condition on the Pareto frontier. However, a comparison of the two approaches was found to be not possible.

To the best of our knowledge, Bayesian optimization has not been augmented by simulation data in the field of design of experiments regarding MOFs. However, reaction kinetics has been used to characterize the reaction process by Dighe et al. [4, 5]. Raccuglia et al. [15] shortly describe that the prediction of crystal structures is a complicated problem and emphasized that the use of physical models characterizing the crystal formed by a chemical reaction is not tractable so far.

3 Framework

The objective of this work is to provide a Python-based framework that can easily be extended by computer scientists yet remains straightforward to use by chemists and integrate into their workflow. Furthermore, it should be applicable to different MOFs or chemical processes and serve as an accessible assistant during reaction optimization.

The framework is comprised of the following core components:

- Performing single-objective or multi-objective Bayesian optimization
- Handling reactions and corresponding differential equations
- Human-in-the-loop functionality and user interaction

The main focus was on Bayesian optimization which was realized in BoTorch [1], a framework that uses the Gaussian process implementation of GPyTorch [9]. Both, BoTorch and GPyTorch, are based on PyTorch[1] and are thus capable of being executed on both, CPU or GPU.

The experimentation for UiO-66 is located in the *project.guided_research* package that can be run via the *main.py* script located in the project root.

In the following, an overview of the provided functionality is provided. For a more in-depth report, the reader is referred to the code documentation that also explains class members and method parameters left out for legibility. The following sections provide an overview of the framework. A more in-depth treatment can be found in the software and its documentation.

[1] PyTorch: `https://pytorch.org/`

3.1 Data Loading and Preparation

The data used in the framework was provided in the form of an Excel-document. To integrate well into the workflow, functionality to load documents of this type and further process the provided data was implemented. An abstraction away from the file structure was performed since it is expected that optimization will be performed over different reaction conditions and results are evaluated with different evaluation criteria. The data package provides generic data management for a Pandas[2]-data frame based data organization. The data, as in Pandas's data frames, is organized in columns and rows. The columns indicates the category of data being stored in the rows. So, each row has data for each column. For each such data table, metadata can be indicated that is equal for all entries in it.

For example, each performed experiment is described in a row. There are columns for each reaction condition parameter and for each evaluated criterion. The reactants being used can also be stored in the table. If the same reactants were used in each experiment, it is good practice to treat them as metadata, as they are not an experiment-specific property.

Data can, for example, be grouped by the general setup (these could be, e.g., organic linker type, metal type, MOF type), columns can be removed, rows from separate datasets can be concatenated, etc. At the same time, access to the underlying data remained straightforward and, to the outside, is provided as a NumPy[3] array.

While this is rather a wrapper for the functionality provided by Pandas, it is a suitable adjustment to the requirements of the project by providing frequently used operations in an organized and straightforward-to-use manner and integrating well into the experimentation pipeline. It is furthermore a standalone package not strictly required by the other parts of the framework but allows unified access and transformation of data, and is a helpful addition for chemists to organize the data.

3.2 Optimization Functionality

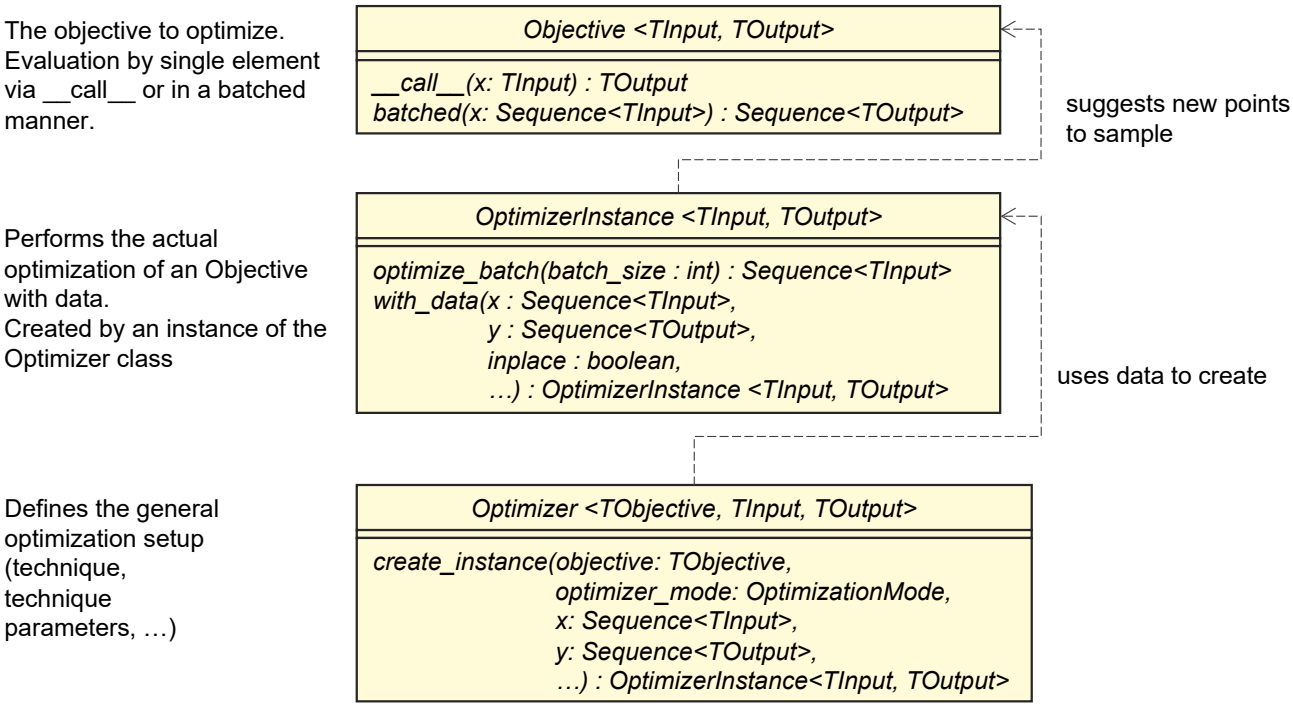


Figure 4 Class diagram of the core base classes used by the optimization framework, adjusted for the purpose of showing the most relevant components. A description can be found in the text.

[2]Pandas: `https://pandas.pydata.org/`
[3]NumPy: https://numpy.org/

Figure 4 shows the relationship between the various classes.

The optimization package is organized around the class `Objective` which, derived in a base class, represents a function to evaluate. This could, for instance, be a function that waits for a chemist performing the evaluation. An objective takes a certain input and produces its corresponding output – both types are indicated by the generic arguments they get. Objectives can be added, subtracted, multiplied, divided, or transformed in various ways.

The goal in optimization is to optimize such an objective, either via minimization or via maximization. Different optimization approaches exist even in the Bayesian optimization family. For example, Gaussian process-based single-objective and multi-objective optimization can be adjusted by parameters such as kernels or the used acquisition function. To cope with the wide variety of possible optimizers, the `Optimizer` class provides a base class that defines the used technique and its parameters. On an abstract level, it describes everything needed to create an `OptimizerInstance` (by the `create_instance`-function) that applies the approach to a concrete dataset. An existing `OptimizerInstance` furthermore provides functionality to add further data obtained from subsequent evaluations of the objective over which optimization is performed (by the `with_data`-function that, by default, returns a new `OptimizerInstance` unless `inplace` is set to `True`).

All three classes are abstract, i.e., they provide only the definitions that any actual optimizer should implement. An actual optimizer can be swapped as each such optimizer provides the same basic functionality, only the approach varies. `Optimizer`s provide the `create_instance`-function that creates an `OptimizerInstance` from the objective to optimize (the class of objectives supported depends on the optimizer), an optimization mode (maximization or minimization), and some sequences of input $x_1, x_2, ..., x_n$ and output data $y_1, y_2, ..., y_n$.

3.2.1 Usage

The optimizers implemented so far are both Bayesian optimization-based: the single-objective Bayesian optimizer `SingleTaskGPOptimizerInstance` in package *optimization.bo.sobo.single_task* and the qNEHVI-based `qNEHVIOptimizer` in package *optimization.bo.mobo.nehvi*.

The `SingleTaskGPOptimizerInstance` is to be used if only a single objective should be optimized, the `qNEHVIOptimizer` if there are multiple conflicting objectives.

To use an optimizer, an instance must first be created. Depending on the optimizer, various parameters are provided that affect the way, the optimizer instance performs its task (optional parameters are in italics):

- ***input_bounds***: The hyperrectangle in which the input data lies.
- ***output_bounds***: The hyperrectangle in which the output data lies.
- ***fit_upon_new_data***: Set to `True` to fit the parameters of the underlying model(s) as new data is acquired.

In addition to that, the `qNEHVIOptimizer` provides the **reference_point**-parameter that is a point that should be dominated by any point on the Pareto frontier (see section 2.2.3).

If hyperrectangles are defined for input or output bounds, the respective input or output data is scaled by the provided bounds before being used. The goal is to provide bounds such that each feature lies in the range $[0, 1]$. Note that while optional, this choice affects the performance of the Bayesian optimization as well as the statistical properties of the input and output data.

The following example shows an example of the usage of the optimization functionality. The optimizer is given some input boundaries, output boundaries, and a reference point as explained above. The `optimizer.create_instance` method receives the optimization mode (maximization/minimization), and input and output data.

In line 9, the new optimum candidates are returned. For the Bayesian optimization, the candidates are the points at which to sample next as obtained via the internally used acquisition function.

```
# Initialize the optimizer with the input boundaries, output boundaries, and the reference
# point dominated by all points on the frontier
optimizer = qNEHVIOptimizer(input_bounds, output_bounds, reference_point)

# create the maximizing optimizer instance from inputs x and observations y
optimizer_instance = optimizer.create_instance(OptimizationMode.maximize, x, y)

# get a batch of candidates at which to sample next
candidates = optimizer_instance.optimize_batch(batch_size)
```

3.2.2 Remarks

An approach that was chosen by Ortner in her bachelor's thesis was the use of an external Gaussian process that was used to simulate the real world [14]. The optimization procedure would then suggest points that could be evaluated on the external Gaussian process instead of being realized by a chemist performing the actual experiment. With the framework, this idea can be realized by implementing a respective objective that acts as a surrogate for the chemist (not to be confused with the surrogate model used by Bayesian optimization!) and internally uses a Gaussian process as, for example, provided by GPyTorch.

3.3 Reactions and Differential Equations

During the decision process it was hypothesized that simulating the reaction process via reaction kinetics would proof beneficial as an augmentation to the Bayesian optimization. As pointed out in section 2.1.2, the concentration changes over time can be described as a differential equation system – more specifically as an ordinary differential equation system in the case at hand. As such, a package was implemented that would allow the definition of reactions and the computation of differential equations from it.

The output of the simulation may be used to augment the Bayesian optimization or to narrow the space explored by Bayesian optimization if it is already known that a reaction will not yield the desired results.

It should be noted that, at this stage, solving the differential equations symbolically is not yet possible as the initial assumption that SymPy[4] could solve them did not hold (see below).

An alternative to this framework would be the use of ChemPy[5] which does however not offer symbolic solutions either which was one of the reasons that the development of a custom framework was made. If a symbolic solution were possible, computations would be highly efficient and accurate thus reducing the amount of time of the Bayesian optimization. The numerical solution of the ordinary differential equation for the vast number of reactions was found to be very inefficient.

3.3.1 Setting up Reaction Systems

The package for describing reactions is located in the *chem.reactions* package. Figure 5 gives an overview over the most important classes used in the reaction modeling. A substance is seen as a basic unit. Substances can be elements, aliases, compounds, or any other representation. They are used for the identification of equal substances in reactions. The `StoichiometricSubstance` then represents a multiple of a substance. For example, in some hypothetical reaction $3M + 2L \rightarrow P$, the substances M, L, and P have the respective stoichiometric coefficients 3, 2, and an implicit 1.

The `StoichiometricSubstance` instances are then used in `ReactionPart`s that are composed of multiple such substances (and due to their simplicity omitted in the diagram). In the above example, $3M + 2L$ and P are the reaction parts that form the overall reaction. A set of such `Reaction`s is then represented by a `ReactionSystem`.

In the case of MOFs, the reaction system would contain the initial reaction and its subsequent reactions.

An example for setting up a reaction system would be the following:

[4] SymPy: `https://www.sympy.org/`
[5] ChemPy: https://pypi.org/project/chempy/

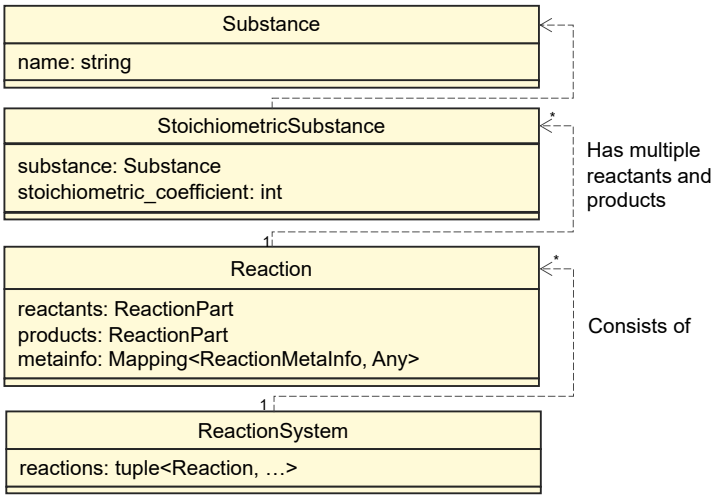


Figure 5 Class diagram of the most important reaction modeling classes, adjusted for the purpose of showing the most relevant components. A description can be found in the text.

```
1 import chem.reactions as react

3 M, L, P = react.substances('M', 'L', 'P')
4 reactions = []

6 reactions.append(3 * M + 2 * L >> P)

8 system = react.ReactionSystem(*reactions)
```

First, the package *chem.reactions* is imported by with an alias `react`. Then, the substances `M`, `L`, `P` are created as in the above example. `reactions` is a list to which reactions can be appended. In line 5, the example reaction is then added to this list of reactions and in line 7, the reaction system is created.

In the *mof.reactions*-package a convenience method `create_mof_reaction_system` is provided that creates a whole system of reactions for MOFs such as the UiO-66.

3.3.2 Differential Equations

The *chemmath.solver.symbolic* and *chemmath.solver.numerical* packages contain code for the symbolic and numerical modeling and solution of the differential equations. The *chem.simulation* package contains the `to_symbolic`-function that is intended to convert a `ReactionSystem` into a form that can be used to set up the equations of the symbolic solver.

A `ReactionSystem` can also be directly used by the numerical reaction solver, `ReactionSolver`, in package *chemmath.solver.numerical*. For the large number of reactions typical to MOFs, this approach was however found to be rather inefficient – a problem an analytical solution would not have.

An overview of the most important classes of the symbolic solver is given in Figure 6. The solver's setup is very similar to that of the reaction modeling. The `WeightedSymbol` is a product between a coefficient and a SymPy-Symbol. Contrary to the reaction modeling, the coefficient now can be an arbitrary SymPy-expression and the result of a computation, but constant values such as `int` and `float` are also possible values.

The `SymbolicReaction`-class again consists of reactants and products as tuples of arbitrary length, and the `SymbolicReactionSystem` collects such reactions. The most important function defined on it is the `setup_differential_equation_system`-function that returns an instance of the `ReactionDifferentialEquationSystem`-class. Therein, the overall symbolic differential equation system has been created ready to be solved by SymPy's `dsolve`-function. However, contrary to what prior experiments seemed to have shown, SymPy was not able to symbolically solve the given equation systems.

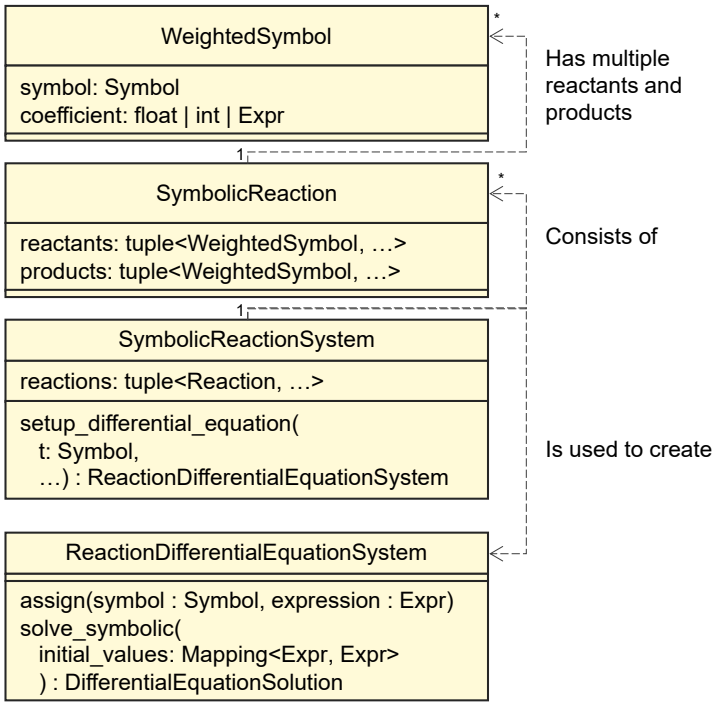


Figure 6 Class diagram of the most important simulation classes, adjusted for the purpose of showing the most relevant components. A description can be found in the text.

The initial idea was to support both, symbolic and numerical solution. For that reason, the `WeightedSymbol` would also support non-constant expressions so that parameters could be optimized over the solution of the differential equation system. By that, models or parameters used to predict inputs to the Bayesian optimization could independently be optimized while being straightforward to define.

A workflow applicable in a scenario where an analytical solution is unavailable would be to compile the symbolic equations from Python-code as provided by SymPy's `lambdify`-function and hand that function to a numerical solver. However, the code generated by `lambdify` was found to be rather inefficient. A solution to the translation of SymPy's generated expressions to Python as well as numerical solution based on it would require further investigation and work.

Due to its code length, an example would be beyond scope for this report and is provided in the tutorials of the application.

3.4 Other

The framework provides functionality for user interaction to perform a human-in-the-loop or, in the future, robot-in-the-loop workflow. This kind of interaction loop is necessary to inform the human or robot chemist about an experiment to perform and to await the experiment's outcome.

While the chemist performs their task, the application is expected to be shut down. This requires, however, that the previous state is recovered when the application is next executed. A foundation has been implemented in the *stateful.persistence* package that allows file-bound object persistence in the form that after a sequence of changes is performed on an object or objects contained in it, the object is saved to a file and reloaded the next time it is created. The *stateful* package further provides a class for tracking and persisting tasks to be performed.

An example experiment-suggestion application was implemented in the *project.guided_research* package. The NEHVI-based Bayesian optimization and, in parts, persistence functionality have been applied in this package.

A more in-depth description is beyond the depth this report is to provide. It is provided in the documentation.

4 Conclusion

In this report the current state of the framework developed for MOF reaction condition optimization has been described. The current main features are single- and multi-objective Bayesian optimization using BoTorch, a modern Bayesian optimization-framework, human-in-the-loop functionality and object persistence, as well as a foundation for symbolic or numerical solution of differential equations.

While the Bayesian optimization framework is in an operable state and was tested, it has not been applied to a real-world problem, yet. The human-in-the-loop functionality requires further integration and testing.

The solutions from the (symbolic or numerical) differential equation solver still must be combined with the Bayesian optimization modules. As argued in section 3.3.2, symbolic solutions may provide fast and accurate results but their solution has not been achieved, yet. Autocatalysis has, so far, not been implemented and requires further investigation and development. The framework design should make integrating it a straightforward procedure. Maybe an even more sophisticated simulation mechanism would furthermore be beneficial to further bound the space Bayesian optimization explores.

If such a simulation system were to provide a sufficiently efficient and accurate simulation for reality, it could furthermore act as a benchmark for the optimization procedure enabling testing algorithms without the expensive feedback procedure of algorithm and chemist interacting or as a rejection criterion for the Bayesian optimization's suggestions.

Due to the characteristics it comes with, Bayesian optimization promises to be a good tool for the design of experiments. In the case of metal-organic frameworks, vast numbers of experiments have been conducted to perform optimization tasks – a number that reflects also the number of failed experiments and the cost incurred for them. Making optimization procedures accessible to chemists that do not have a strong background in application development, and providing tools that integrate well into their workflow are promising assets to decrease the cost of attaining a successful experiment and the time needed to complete it. As such, implementing a powerful realistic framework that makes the experimentation process more accessible or even automated is expected to proof beneficial to the successful discovery of new materials. From a computer science perspective, the evaluation of Bayesian optimization on an empirical basis should also be possible if the framework were to be applied to a larger number of MOFs.

References

[1] 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*, 2020.

[2] Bruker AXS GmbH. *DIFFRAC.SUITE User Manual*. Bruker AXS GmbH, 2019.

[3] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2187–2200. Curran Associates, Inc., 2021.

[4] Anish V. Dighe, Luke Huelsenbeck, Rajan R. Bhawnani, Prince Verma, Kevin H. Stone, Meenesh R. Singh, and Gaurav Giri. Autocatalysis and oriented attachment direct the synthesis of a metal–organic framework. *JACS Au*, 2(2):453–462, 2022.

[5] Anish V. Dighe, Roshan Y. Nemade, and Meenesh R. Singh. Modeling and simulation of crystallization of metal–organic frameworks. *Processes*, 7(8):527, 2019.

[6] Nency P. Domingues, Seyed M. Moosavi, Leopold Talirz, Kevin Maik Jablonka, Christopher P. Ireland, Fatmah M. Ebrahim, and Berend Smit. Using genetic algorithms to systematically improve the synthesis conditions of al-PMOF. *Communications Chemistry*, 5(1), 2022.

[7] Kobi C. Felton, Jan G. Rittig, and Alexei A. Lapkin. Summit: Benchmarking machine learning methods for reaction optimisation. *Chemistry–Methods*, 1(2):116–122, 2021.

[8] Peter I. Frazier. A tutorial on Bayesian optimization. 2018.

[9] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, 2018.

[10] Stewart Greenhill, Santu Rana, Sunil Gupta, Pratibha Vellanki, and Svetha Venkatesh. Bayesian optimization for adaptive experimental design: A review. *IEEE Access*, 8:13937–13948, 2020.

[11] V. Guillerm, S. Gross, C. Serre, T. Devic, M. Bauer, and G. Ferey. Ccdc 752051: Experimental crystal structure determination, 2010.

[12] Seyed Mohamad Moosavi, Arunraj Chidambaram, Leopold Talirz, Maciej Haranczyk, Kyriakos C. Stylianou, and Berend Smit. Capturing chemical intuition in synthesis of metal-organic frameworks. *Nature Communications*, 10(1), 2019.

[13] Kevin P. Murphy. *Machine Learning*, chapter Gaussian proccesses, pages 515–542. MIT Press Ltd, 2012.

[14] Luisa Ortner. Bayesian optimization of material synthesis parameters. Bachelor's thesis, Technical University of Munich, 2023.

[15] Paul Raccuglia, Katherine C. Elbert, Philip D. F. Adler, Casey Falk, Malia B. Wenny, Aurelio Mollo, Matthias Zeller, Sorelle A. Friedler, Joshua Schrier, and Alexander J. Norquist. Machine-learning-assisted materials discovery using failed experiments. *Nature*, 533(7601):73–76, 2016.

[16] Daniel Reker, Emily A. Hoyt, Gonçalo J. L. Bernardes, and Tiago Rodrigues. Adaptive optimization of chemical reactions with minimal experimental information. *Cell Reports Physical Science*, 1(11):100247, 2020.

[17] Alexander Schoedel and Sahar Rajeh. *Metal-Organic Framework*, chapter Why Design Matters: From Decorated Metal Oxide Clusters to Functional Metal–Organic Frameworks, pages 1–55. Springer International Publishing, 2020.

[18] 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.

[19] Santosh K. Upadhyay. *Chemical Kinetics and Reaction Dynamics*. Springer Dordrecht, 2006.

[20] Carl E. Rasmussen; Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press Ltd, 2005.

[21] James T. Wilson, Frank Hutter, and Marc Peter Deisenroth. Maximizing acquisition functions for Bayesian optimization. 2018.

[22] Yunchao Xie, Chi Zhang, Heng Deng, Bujingda Zheng, Jheng-Wun Su, Kenyon Shutt, and Jian Lin. Accelerate synthesis of metal–organic frameworks by a robotic platform and Bayesian optimization. *ACS Applied Materials & Interfaces*, 13(45):53485–53491, 2021.