BO as Assistant: Using Bayesian Optimization for Asynchronously Generating Design Suggestions

Yuki Koyama
koyama.y@aist.go.jp
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan

Masataka Goto
m.goto@aist.go.jp
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba, Ibaraki, Japan

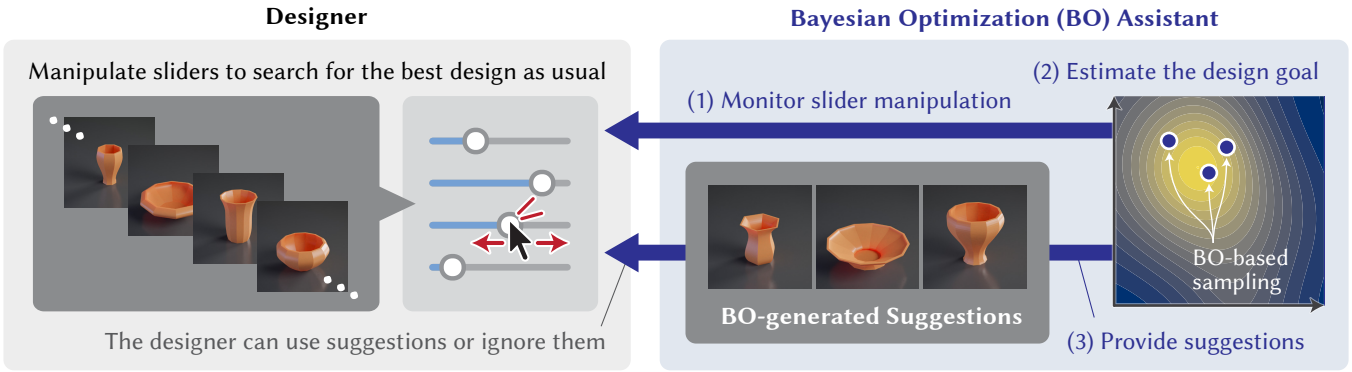

Figure 1: Concept of *BO as Assistant*, a framework for assisting designers in finding appropriate design parameter values by using Bayesian optimization (BO) techniques. The system (1) monitors the designer's slider manipulation, (2) automatically estimates the designer's design goal, and (3) asynchronously provides suggestions sampled using BO's strategy. The designer can choose to use suggestions or ignore them.

ABSTRACT

Many design tasks involve parameter adjustment, and designers often struggle to find desirable parameter value combinations by manipulating sliders back and forth. For such a multi-dimensional search problem, *Bayesian optimization* (BO) is a promising technique because of its intelligent sampling strategy; in each iteration, BO samples the most effective points considering both exploration (*i.e.*, prioritizing unexplored regions) and exploitation (*i.e.*, prioritizing promising regions), enabling efficient searches. However, existing BO-based design frameworks take the initiative in the design process and thus are not flexible enough for designers to freely explore the design space using their domain knowledge. In this paper, we propose a novel design framework, *BO as Assistant*, which enables designers to take the initiative in the design process while also benefiting from BO's sampling strategy. The designer can manipulate sliders as usual; the system monitors the slider manipulation to automatically estimate the design goal on the fly and then asynchronously provides unexplored-yet-promising suggestions using BO's sampling strategy. The designer can choose to use the suggestions at any time. This framework uses a novel technique to automatically extract the necessary information to run BO by observing slider manipulation without requesting additional inputs. Our framework is domain-agnostic, demonstrated by applying it to photo color enhancement, 3D shape design for personal fabrication, and procedural material design in computer graphics.

CCS CONCEPTS

• **Computing methodologies** → *Graphics systems and interfaces*; • **Human-centered computing** → *Human computer interaction (HCI)*.

KEYWORDS

Bayesian optimization, visual design, suggestive interface

ACM Reference Format:
Yuki Koyama and Masataka Goto. 2022. BO as Assistant: Using Bayesian Optimization for Asynchronously Generating Design Suggestions. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3526113.3545664

UIST '22, October 29-November 2, 2022, Bend, OR, USA

ACM ISBN 978-1-4503-9320-1/22/10.
https://doi.org/10.1145/3526113.3545664

1 INTRODUCTION

1.1 Background

Many design tasks involve parameter adjustment, *i.e.*, searching for a parameter value combination that produces a desirable design outcome [26]. For example, a photo color enhancement task involves multiple parameters such as brightness, contrast, and saturation, which in combination create interesting color effects to the original

photograph [21]. Similarly, digital content creation (DCC) tools [3, 44, 50] and computer-aided design (CAD) tools [10, 35] provide parametric functionality to procedurally generate and edit contents. Also, machine learning techniques [17, 31, 36] have enabled designers to generate various high-quality content by providing parameters called latent codes. While such parametric design features enable designers to try different design variations easily, designers are required to make an effort to find a desirable parameter combination by manipulating sliders back and forth many times. This task is considered a high-dimensional optimization problem, in which the objective is the design goal that the designer has in mind. Designers manually solve this problem in practice. In this work, we want to facilitate designers' manual optimization tasks.

Computational optimization is a promising technique to tackle such high-dimensional design problems because optimization algorithms [29] use mathematically reasonable sampling strategies to navigate high-dimensional search spaces. With this in mind, researchers have investigated *human-in-the-loop* optimization frameworks [7, 8, 12, 18, 22, 23, 34, 48, 56, 57] for design problems that need human evaluators to evaluate the optimization objective. Among various optimization algorithms, *Bayesian optimization* (BO) [38] is particularly attractive for human-in-the-loop settings [8, 12, 18, 22, 23, 57] due to its intelligent sampling strategy; in each iteration, BO searches for the most effective sampling points considering both *exploration* (*i.e.*, prioritizing unexplored regions in the design space) and *exploitation* (*i.e.*, prioritizing promising regions in the design space), enabling efficient searches. BO iteratively asks human evaluators (*i.e.*, the designer in our case) to provide feedback about the sampled points.

However, existing designer-in-the-loop optimization frameworks (*i.e.*, human-in-the-loop optimization frameworks in which a designer plays the role of the evaluator), regardless of using BO, are insufficiently flexible for designers to freely explore the design space using their domain knowledge. For example, suppose a designer is adjusting parameters for photo color enhancement using a designer-in-the-loop optimization framework. At a certain iteration step, the system may propose a nice color enhancement, but the designer may wish for the image to be brighter. Thus, this designer may want to increase the "brightness" parameter value according to the designer's domain knowledge. However, such an operation is not possible during the optimization since the optimizer determines what and how parameters are changed (*i.e.*, all the designer can do is evaluate the provided designs). In this way, the designer-in-the-loop workflow does not enable designers to take the initiative in determining how the design process progresses, unlike the traditional slider-based workflow. As a result, it may reduce the designers' sense of agency, ownership, and creativity [6].

1.2 Contributions

We propose a novel design framework, *BO as Assistant*, which enables designers to take the full initiative in the design process while also benefiting from BO's intelligent sampling strategy. Figure 1 illustrates this framework. This framework does not require the designer to provide any additional input to run the optimization; the designer can manipulate sliders as usual. The system monitors the slider manipulation to progressively and automatically estimate the design goal in a Bayesian manner, and then it asynchronously provides suggestions. Thus, this framework has no explicit optimization loop unlike previous frameworks. Suggestions are sampled from unexplored-yet-promising regions in the multi-dimensional design space using BO techniques. The designer can choose to use the suggested designs or just ignore them if they are unsatisfactory or the designer wants to focus on manual exploration. This "optimization as a design assistant" approach has been investigated previously [2, 30, 51]. Our framework is unique in that it dynamically estimates the designer's design goal (*i.e.*, the intention to determine parameter sets of interest) from the behavior of the design exploration (*i.e.*, how the designer explores the design space), instead of assuming domain-specific pre-defined design goals; thus, our framework is general and domain-agnostic.

To develop this framework, we propose a novel technique to automatically extract the necessary information to run BO only from slider manipulations. Standard BO [38] requires to observe a numerical value of the objective function at each sampling point (*i.e.*, in our case, a score representing how subjectively good the corresponding design is). This information cannot be observed unless the designer explicitly provides an absolute score for every visited parameter set, which is impractical; thus, standard BO is unsuitable for our purpose. Instead, we use *preferential Bayesian optimization* (PBO) [5, 22, 23], a variant of BO that runs with *relative* preferential information (*i.e.*, which parameter set is preferred over other parameter sets) instead of absolute scores. Our technique enables to extract such relative preferential information by observing the behavior of the designer's slider manipulation, and then runs PBO using the acquired data to generate reasonable suggestions.

To demonstrate the proposed framework and its generality, we apply it to diverse design domains, including photo color enhancement, 3D shape design for personal fabrication, and procedural material design in computer graphics. The framework could work in these different scenarios in a unified manner and automatically provide reasonable suggestions without interrupting the original slider-based workflow.

2 RELATED WORK

2.1 Interface for Design Parameter Tweaking

Various user interfaces have been proposed for adjusting design parameters. For example, gallery-based interfaces [22, 26, 28, 39] enable designers to select from visually presented options, by which the user can concentrate on the look of the design rather than raw parameter values. In this work, we focus on using sliders as the means of design exploration since sliders are very common in practical scenarios. Our framework considers sliders as the main tool and adds asynchronous suggestions into the slider-based workflow.

Researchers have extended slider interfaces in various ways [11, 20, 21, 41, 43, 49], and some of the systems estimate design goals either by explicitly asking human evaluators for feedback [11, 20, 41] or by implicitly gathering data by monitoring multiple design sessions [21]. However, none of them focus on monitoring the trajectory of the designer's exploration and estimating the designer's design goal from the slider trajectory.

2.2 Design Tools with Optimization Assistant

Optimization techniques have been utilized as assistants in design tools [2, 30, 51]. DesignScape [30] and Sketchplore [51] assist designers in designing graphic layouts by asynchronously providing suggestions using optimization. Our framework has similarities to these systems in that designers are not required to provide extra input to run optimization but are expected to operate tools as usual and that the generated suggestions neither automatically overwrite the current design nor interrupt ongoing tool operations (*i.e.*, the designer has full initiative). However, our framework is unique in that it dynamically estimates the design goal on the fly in a Bayesian manner by monitoring tool operations (more specifically, slider manipulations), and that it does not rely on domain-specific, pre-defined design goals; DesignScape and Sketchplore use pre-defined objectives specialized to layout design tasks (*e.g.*, avoiding visual clutter, harmonizing colors).

In addition, our suggestions are generated considering not only exploitation (*i.e.*, respecting the estimated design goal) but also exploration (*i.e.*, trying not-yet-explored designs).

2.3 Human-in-the-Loop Bayesian Optimization

Human-in-the-loop optimization is a computational approach to solve parameter optimization problems, where human evaluators are involved in its iterative algorithm. This approach is effective in design problems where the objective (*i.e.*, how well the design goal is achieved) needs to be evaluated by human evaluators (*e.g.*, evaluated subjectively [7, 22, 23, 34, 48, 56], and evaluated by human performance measurement [12, 16, 18]). The optimizer asks human evaluators to provide feedback iteratively to obtain information about the objective and then proceeds to search for the optimal parameters.

BO is a black-box optimization algorithm [38], known for its *sample efficiency*: it can find good solutions within a small number of iterations due to its intelligent sampling strategy, which considers both exploration and exploitation. Thus, BO has been applied to expensive-to-evaluate problems such as hyperparameter tuning in deep learning [1]. Since humans are also expensive to query, BO is an attractive choice for human-in-the-loop optimization [8, 12, 18, 22, 23, 57]. For example, BO has been successfully applied to the optimization of user interface design [12], game level design [18], and font design [16], since these design tasks need expensive human performance measurements to evaluate their objective functions.

When the design goal is defined subjectively (*e.g.*, on the basis of the designer's preference), it is in general considered to be better to request a *relative* assessment (*e.g.*, which design is better between two options), instead of an *absolute* assessment (*e.g.*, how good the design is) [4, 52]. To enable BO to run with such relative information, Brochu *et al.* [5] proposed a new variant of BO that runs with *pairwise comparison* queries: the human evaluator is asked to select one of the two options sampled by BO techniques. Following [22], we call BO variants that run with relative information PBO[1]. Recently, Koyama *et al.* [22, 23] proposed even more sample-efficient PBO methods, where the human evaluator is asked to manipulate a slider [23] or to select the best option from a design gallery [22]. PBO has been applied to specific domains such as animation [4], GAN-based image generation [8], and melody composition [57].

Our framework differs from human-in-the-loop optimization frameworks in interaction design. There is no explicit optimization loop; instead, BO implicitly learns the design goal by observing the design exploration behavior, and then it provides sampled points as asynchronous suggestions. Our work is the first to enable BO to run without requiring an explicit feedback loop and to use it as a suggestive assistant. The key idea is to utilize BO's intelligent sampling strategy to generate suggestions.

3 FRAMEWORK AND INTERACTION

3.1 Framework Overview

Our framework, *BO as Assistant*, uses BO as a design suggestive assistant. That is, it uses BO techniques for assisting designers in adjusting design parameters by providing suggestions without requesting any explicit inputs for the assistance. The designer can interact with the suggestions at any time or ignore them if the designer finds them unattractive or wants to concentrate on slider manipulation. This enables the designer to take the full initiative in the design process, unlike designer-in-the-loop optimization frameworks [5, 7, 22]. Figure 1 illustrates this concept, and Figure 2 shows the interfaces of our proof-of-concept systems.

Benefits of BO-based suggestions. Suggestions are carefully generated by sampling from the multi-dimensional parameter space by simultaneously considering exploration and exploitation. The exploration aspect is useful for avoiding getting stuck in local optima; for example, the designer might be reluctant to try different parameters by manipulating sliders after finding a satisfactory design, but there might be better, unseen designs of which the designer is unaware. The exploitation aspect is useful for making suggestions respectful to the designer's design goal; without considering the design goal, the system may continue to sample suggestions from the regions that are intentionally unexplored because the designer knows they are useless. Together, our suggestions are expected to provide additional inspiration during the slider-based design process. Note that, even when the suggested designs are unattractive, they can still assist familiarizing the designer with the design space (*i.e.*, what designs are possible), and thus enabling the designer to be more confident about their current design. Another important benefit of providing reasonable suggestions is to enable efficient navigation in the parameter space; if the designer finds a better design in the suggestions, the designer can navigate the high-dimensional space directly towards the suggested point without manipulating multiple sliders individually.

Technical components. The framework runs the following steps automatically in the background.

- **Monitoring the slider manipulation (Section 6)** The system monitors how the designer manipulates the sliders. This includes not only the slider values that the designer eventually determines but also the entire back-and-forth trajectories. Then, the system extracts the information to run BO.
- **Estimating the design goal (Section 5)** Using the extracted information, the system estimates the designer's design goal by determining which regions in the parameter space are

[1] Gonzalez *et al.* [13] also use the term PBO for a slightly different scope; we use the term in a broader sense.

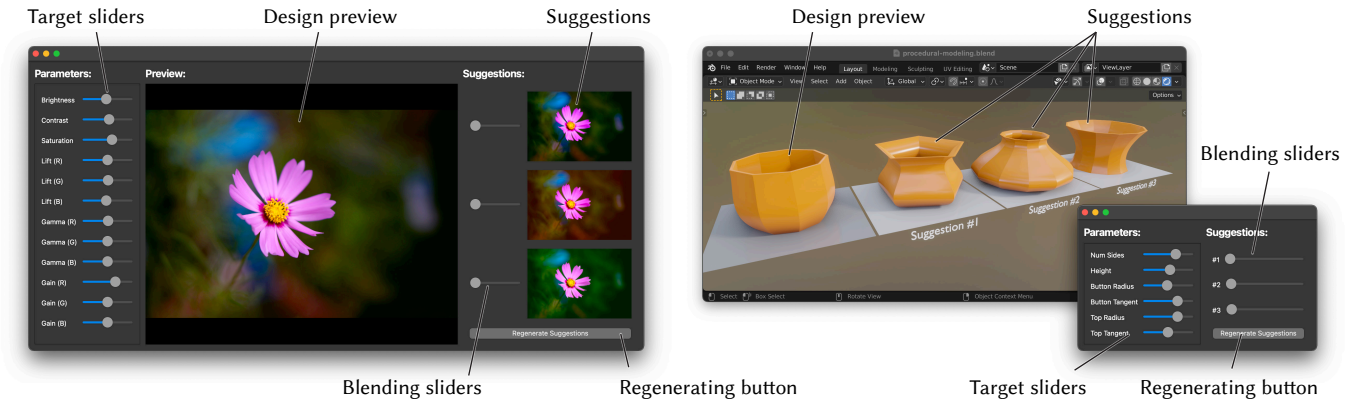

Figure 2: Interfaces of our proof-of-concept systems. The designer primarily interacts with the target sliders while seeing the design preview. The system asynchronously provides several suggestions (three suggestions in these cases). The designer can blend a suggested design with the current design using the associated blending slider. The designer can also request regenerating suggestions to see more variations. (Left) A photo color enhancement system, implemented as a standalone system. (Right) A procedural modeling system, implemented as an addon for Blender [3].

preferred and which are not. More precisely, it constructs a predictive model that predicts how good the design is under a set of parameter values.

Providing suggestions (Section 5) The system determines what to provide as suggestions using the estimated design goal. For this purpose, the system solves optimization subproblems where both exploration and exploitation are maximized simultaneously.

3.2 Task Assumptions

Our framework targets parametric design tasks where a designer adjusts multiple design parameter values via sliders and finds the best parameter value combination. We put several assumptions as follows.

Parameter types. Each parameter value needs to be mapped to a continuous finite value range (*e.g.*, [0, 1]) so that it can be manipulated by a slider. Integer parameters are within this scope since they can be mapped to sliders by quantization. Nominal parameters (*e.g.*, parameters usually selected by dropdown lists) are out of scope.

Design goal. We assume that the design goal is defined by the designer's subjective preference and can be evaluated visually. Also, we assume that the goodness function does not change over time. For example, given a preferable design, the designer consistently considers it to be preferable, regardless of the progression of the design session. If this assumption is broken, the suggestions may become unreasonable. Note that, even in this case, the system does not bother the designer since the designer can ignore the suggestions.

Number of parameters. We assume that the number of target parameters is two to around twenty, considering typical settings in DCC tools. The case that more parameters need to be adjusted simultaneously, such as content generation using deep generative models [7] (*e.g.*, 512 parameters [17]), is out of scope.

Real-time preview. We assume that the design preview can be generated in real time while the designer manipulates sliders. Thus, the case that heavy computation is necessary to generate design previews given design parameter values (*e.g.*, offline photorealistic rendering, high-resolution physical simulation, geometry synthesis using topology optimization [27]) is out of scope.

Continuous change. We assume that the design (and thus its goodness) continuously changes when a parameter value changes. This assumption is necessary for the estimation of the design goal. Note that most practical scenarios satisfy this assumption. One exception is the random seed parameter, which often plays an important role in procedural modeling with stochastic rules [45] in computer graphics. This is out of scope since the design changes discontinuously along with the random seed.

Inter-parameter effects. We are interested in the case that parameters have inter-parameter effects and are not perfectly orthogonal; that is, their values cannot be determined independently but need to be determined in combination. Thus, the designer needs to manipulate each slider several times; if a slider value has changed, other slider values need to change accordingly. For example, in photo color enhancement, the brightness and contrast parameters define the photo look in combination, so they need to be adjusted together. Note that if a subset of the parameters is known to be perfectly orthogonal to the rest, we can divide the task into two independent subtasks beforehand, and doing so is more reasonable.

3.3 Interaction with Suggestions

3.3.1 Blending suggestions. The most straightforward approach to using suggestions may be to let the designer select one of the suggestions and then replace the current design with the selected one. While this approach is fine, we adopt a different approach: let the designer blend the current design with the selected suggestion. This blending approach is inspired by previous work [23], which demonstrated that blending could be very effective in searching for optimal parameters compared with using only discrete selection [5].

More specifically, we provide a "blending" slider along with each suggestion (see Figure 2) and let the designer interactively specify a

blending weight whose default and minimum value is zero (*i.e.*, no blend with the suggested design) and maximum value is one (*i.e.*, full overwrite by the suggested design). While the designer moves the blending slider, the original sliders are automatically moved in conjunction, and the design preview is updated accordingly in real time. This helps the designer understand how each parameter value changes. Once the blending slider is released, the current design is replaced with the blended design. Then, the designer can either go back to slider manipulation or blend the new design with another suggested design.

3.3.2 Regenerating suggestions. We also provide an optional interaction: the designer can request the system to discard all the current suggestions and generate new ones. For the designer, this interaction is useful for exploring design variations quickly without manipulating sliders. For the system, this request is helpful for better estimating the designer's design goal; it interprets that the discarded suggested designs are not preferred over the current design, and this information is used to refine the estimation and thus provide more sophisticated suggestions.

4 PROBLEM FORMULATION

Let n be the number of target design parameters, and let x_i ($i = 1, \ldots, N$) be their values. We assume that the range of each parameter is normalized into $[0, 1]$ without loss of generality. During the design process, the designer manipulates these values via sliders. We denote these values altogether by $\mathbf{x} = [x_1 \ \cdots \ x_N]^\top \in \mathcal{X}$, where $\mathcal{X} = [0, 1]^N$ is the search space. The goal of the design task is to determine the optimal parameter set, $\mathbf{x}^* \in \mathcal{X}$, defined as

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in \mathcal{X}} g(\mathbf{x}), \tag{1}$$

where the objective function $g : \mathcal{X} \to \mathbb{R}$ is called a *goodness function* [23] and represents the subjective design goal (*i.e.*, how good the design is). Under the assumptions described in Section 3.2, the function g does not change during the design process.

The goodness function value cannot be directly evaluated (*i.e.*, the system never requests the designer to provide a score for a given design), and even the designer does not know the function shape (*e.g.*, which regions in $\mathcal{X}$ provide large goodness values). Instead, the system can observe the designer's behavior that is based on the goodness function; in general, the designer is expected to manipulate sliders such that the goodness function value becomes larger.

Technical goal. Our goal is to always provide the designer with K suggestion points, $\mathbf{x}_1^{\text{suggest}}, \ldots, \mathbf{x}_K^{\text{suggest}}$, and asynchronously update these points during the design process. Our implementation uses $K = 3$ considering the balance between the chance of including good suggestions (*i.e.*, large K is desirable) and that of overwhelming the designer with the number of suggestions available at once (*i.e.*, small K is desirable).

Baselines and our proposition. A possible naïve approach is to use random sampling from the search space $\mathcal{X}$ to provide random suggestions. However, this approach is not ideal because it may generate samples from already explored regions and regions in which the designer is not interested. Another possible, more sophisticated approach is to generate samples from not-yet-explored regions. However, this approach still ignores the designer's design goal, and it may stick to the regions that the designer is not interested in and thus has not explored intentionally. We propose using BO to sample suggestions by considering both exploration (*i.e.*, prioritizing not-yet-explored regions) and exploitation (*i.e.*, prioritizing the regions that are expected to align with the design goal). For this purpose, we need a technique to run BO by only observing slider manipulation behaviors.

5 PREFERENTIAL BAYESIAN OPTIMIZATION

Before describing our technique to extract the necessary information to run BO from slider manipulation, we explain how BO [38] works. In particular, we explain PBO [5, 22, 23], a variant of BO that runs with relative preferential data. Readers who are familiar with PBO can skip this section; we include this section for completeness.

Preferential data modeling. In PBO, the observable data are not absolute function values (*e.g.*, $g(\mathbf{x}_A) = 0.1, g(\mathbf{x}_B) = 0.2$) but relative comparison information (*e.g.*, $g(\mathbf{x}_A) < g(\mathbf{x}_B)$). More specifically, we consider the following observation as a preferential data entry: "given L options, $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)}$ ($L \geq 2$), the i-th option $\mathbf{x}^{(i)}$ ($1 \leq i \leq L$) is preferred." We denote this observation by

$$d = [\mathbf{x}^{(i)} \succ \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(i-1)}, \mathbf{x}^{(i+1)}, \ldots, \mathbf{x}^{(L)}\}]. \tag{2}$$

Let us denote their (latent) goodness values by $g^{(i)} = g(\mathbf{x}^{(i)})$ for $i = 1, \ldots, L$, and $\mathbf{g} = [g^{(1)} \ \cdots \ g^{(L)}]^\top$. The likelihood of the aforementioned preferential data entry can be modeled by the Bradley–Terry–Luce model [52]:

$$p(d \mid \mathbf{g}) = \frac{\exp(g^{(i)})}{\sum_{j=1}^{L} \exp(g^{(j)})}. \tag{3}$$

When we have multiple preferential data entries, we denote them by $\mathcal{D} = \{d_1, d_2, \ldots\}$. The overall data likelihood is $p(\mathcal{D} \mid \mathbf{g}) = \prod_i p(d_i \mid \mathbf{g})$. In human-in-the-loop PBO systems, such preferential data is obtained by explicitly asking human evaluators for feedback repeatedly. In our case, the goal is to extract such preferential data by observing slider manipulation (Section 6).

Goodness value estimation. We assume that the goodness function follows a *Gaussian process* (GP) [33]. Suppose that we have observed M data points in total, $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)} \in \mathcal{X}$ in the preferential data $\mathcal{D}$. Since we cannot directly observe the goodness values, we need to estimate them from the preferential data. For this purpose, we use the *maximum a posteriori* (MAP) estimation [23]; that is, we obtain the estimate of the goodness values by

$$\mathbf{g}^{\text{MAP}} = \arg\max_{\mathbf{g} \in \mathbb{R}^M} p(\mathbf{g} \mid \mathcal{D}) = \arg\max_{\mathbf{g} \in \mathbb{R}^M} p(\mathcal{D} \mid \mathbf{g})\, p(\mathbf{g}), \tag{4}$$

where $p(\mathbf{g})$ is the prior distribution of the goodness values, which is simply a Gaussian distribution by the GP assumption. Once the goodness values are estimated, we can calculate the predictive distribution at any unseen data point $\mathbf{x}$ as

$$g(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})). \tag{5}$$

See Section A.1 for the details of the mean μ and the variance σ^2.

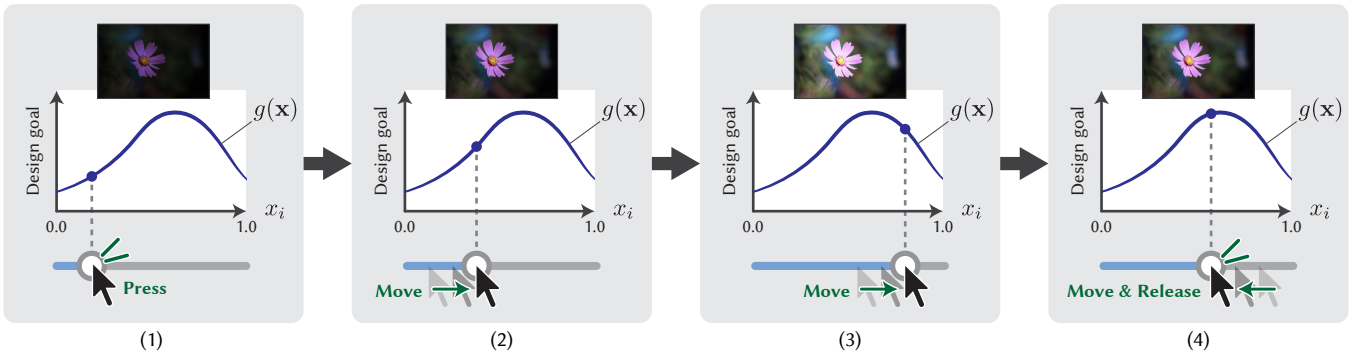

Figure 3: Illustration of a slider manipulation session. The designer moves the slider back and forth to adjust the brightness of the image in this example. (1) To increase the parameter value, the designer presses and moves the slider knob to the right to increase it. (2) The designer can observe the changes in the design preview while increasing the value. So far, it looks good (*i.e.*, the goodness value g is getting larger). (3) When the quality of the image starts to look worse (*i.e.*, the goodness value g is getting smaller), it becomes clear that the value has become too high, and therefore, the slider has been moved too much. (4) The designer moves the slider knob back to the left to a suitable position and releases it. The final point of the knob is assumed to be around the maximum of the goodness function g in this one-dimensional subspace.

Sampling. BO uses an *acquisition function* to determine the next sampling points (in our case, the suggestions to provide to the designer). Given the predictive distribution (Equation 5), an acquisition function, $a : \mathcal{X} \to \mathbb{R}$, estimates the "effectiveness" of a point, $\mathbf{x}$, if the point $\mathbf{x}$ is observed next. Thus, the point with the highest acquisition function value is considered the most effective point to observe next. To determine such a point, a maximization problem:

$$\mathbf{x}^{\text{sample}} = \arg\max_{\mathbf{x} \in \mathcal{X}} a(\mathbf{x}) \tag{6}$$

is solved. Commonly used acquisition functions (such as GP-UCB and EI; see Section A.3 for more details) are designed to balance exploration and exploitation automatically. To generate multiple samples at once, we can use *batch* BO techniques [14, 37]. Specifically, we use a method proposed by Schonlau *et al.* [37]; that is, we solve Equation 6 sequentially K times while adding the newly sampled point in the calculation of the variance of the predictive distribution each time (see Section A.4).

6 TECHNIQUE TO EXTRACT DATA FOR PREFERENTIAL BAYESIAN OPTIMIZATION

This section describes our technique to extract the necessary information to run BO without explicitly requesting the designer perform additional tasks. More specifically, our technique extracts preferential data in the format of Equation 2 by observing slider manipulation (Section 6.1). We also gather preferential data when the designer interacts with suggestions (Section 6.2 and Section 6.3) to better estimate the designer's design goal (*i.e.*, the goodness function).

6.1 Extraction from Slider Manipulation

6.1.1 Slider Manipulation Session. Although there are multiple sliders to manipulate, the designer can manipulate only one slider at once. Typically, the following steps occur repeatedly: the knob of a slider is pressed by the mouse cursor, moved back and forth, and then released at a point that provides a good design. We call each sequence of these mouse interactions (*i.e.*, mouse press, mouse move, and mouse release) a slider manipulation session. Our technique extracts one preferential data entry from each slider manipulation session.

Our key idea is that, in each slider manipulation session, the designer is expected to manipulate the target slider to search for a better point within the one-dimensional search space. Figure 3 illustrates a slider manipulation session and how this idea can be interpreted. On the basis of this idea, we can come up with several strategies to interpret the observed data into a preferential data entry as described in Section 6.1.2.

Suppose that the designer adjusts the i-th slider among the N sliders in a slider manipulation session. The system records the trajectory of all the slider values. Let s be the list of all the recorded points in this slider session, which we denote by

$$s = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_s)}), \tag{7}$$

where $n_s = |s|$. Since only the i-th slider is manipulated, the i-th dimensional components of these vectors vary over time, and the other components are static.

6.1.2 Strategies. It is a reasonable assumption that the last point, $\mathbf{x}^{(n_s)}$, is a relatively good choice among the list s. However, it is not trivial to define the set of non-preferred points. Several strategies are possible as follows.

Initial Point This strategy uses only the initial point for the set of non-preferred points. The preferential data entry is composed as

$$d = [\mathbf{x}^{(n_s)} \succ \{\mathbf{x}^{(1)}\}]. \tag{8}$$

This strategy provides only minimal information to PBO.

All Points This strategy uses all the recorded points except for the last point. The preferential data entry is composed as

$$d = [\mathbf{x}^{(n_s)} \succ \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_s-1)}\}] \tag{9}$$

This strategy, however, has a risk of mistakenly encoding the designer's preference. The designer may release the slider knob without fine tuning to find the exact best position but roughly around the best position. In this case, there may be a better point in $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_s-1)}$ than $\mathbf{x}^{(n_s)}$, by which PBO can estimate the preference wrongly.

Turning Points This strategy uses all the "turning" points (*i.e.*, the points at which the slider movement turns from back to forth, or from forth to back) for the set of non-preferred points. Turning points are important because these points are where the designer intentionally changes the direction. Using mathematical notations, the set of turning points in s is described as

$$\mathcal{T}_s = \{\mathbf{x}^{(j)} \mid T_s(j), j \in \{2, \ldots, n_s - 1\}\}, \tag{10}$$

where T_s is a boolean function to discriminate whether the j-th point is a turning point or not, defined as

$$T_s(j) = \left[(x_i^{(j+1)} - x_i^{(j)})(x_i^{(j)} - x_i^{(j-1)}) < 0\right]. \tag{11}$$

Note that it is reasonable to include the initial point, $\mathbf{x}^{(1)}$, to the set of non-preferred points. Thus, the preferential data entry is composed as

$$d = [\mathbf{x}^{(n_s)} \succ \{\mathbf{x}^{(1)}\} \cup \mathcal{T}_s]. \tag{12}$$

6.1.3 Example. For discussion purpose, here we introduce an illustrative slider manipulation data (Figure 4), where the number of target parameters is two ($N = 2$), and the designer performed five slider manipulation sessions ($s_1, \ldots, s_5$). The data is around 25-second long.

Figure 5 visualizes how data points (orange dots) are accumulated, how BO's internal models (*i.e.*, the mean of the predictive distribution $\mu(\mathbf{x})$, the standard deviation of the predictive distribution $\sigma(\mathbf{x})$, and the acquisition function, $a(\mathbf{x})$) evolve, and how the suggestions (blue dots) are updated through these sessions using the Turning Points strategy. The number of suggestions is three ($K = 3$). In the beginning, since the standard deviation $\sigma(\mathbf{x})$ (*i.e.*, the uncertainty of the estimate) is large in most regions, the suggestions are sampled at distant locations from the existing data points (*i.e.*, the exploration is dominant). After several sessions, since the standard deviation becomes smaller in most regions, and so the suggestions are sampled from both unexplored (*i.e.*, high $\sigma(\mathbf{x})$) and promising (*i.e.*, high $\mu(\mathbf{x})$) regions.

The other strategies can also work and generate similar suggestions. However, we can observe that the Initial Point strategy estimates the design goal (*i.e.*, the distribution of $\mu(\mathbf{x})$) less reasonably, and that it is more likely to generate extereme suggestions sampled from the boundary of the search space, compared to the Turning Points strategy; see Figure 6 for an example. The All Points strategy generates suggestions similar to the Turning Points strategy (see Figure 7). However, with the All Point strategy, the number of data points, M, increases very quickly, and it is concerned that the computational cost becomes intractable; note that the complexity is $O(M^3)$ to calculate the predictive distribution (see Section A.1). Thus, we recommend using the Turning Points strategy, and we will use it in the rest of the paper.

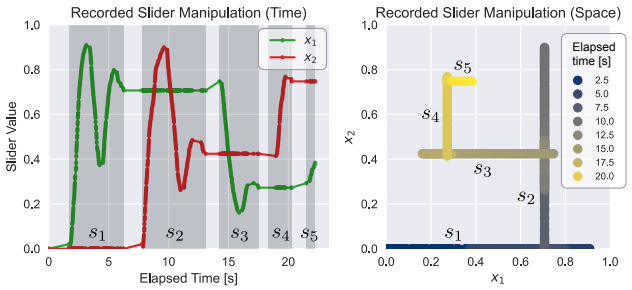

Figure 4: Illustrative slider manipulation data in a two-dimensional problem. Two sliders (corresponding to x_1 and x_2, respectively) are manipulated alternately. It consists of five slider manipulation sessions ($s_1, \ldots, s_5$). (Left) A time view. (Right) A two-dimensional space view.

6.2 Extraction from Suggestion Blending

The designer can perform linear interpolation between the current design and one of the suggestions. Suppose that the designer selects the i-th suggestion. The blended design is calculated by

$$\mathbf{x}^{\text{blend}} = (1 - t)\mathbf{x}^{\text{current}} + t\mathbf{x}_i^{\text{suggest}}, \tag{13}$$

where $t \in [0, 1]$ is the blending weight that the designer specifies by manipulating the blending slider. Once the blending is done, the system adds a new preferential data entry,

$$d = [\mathbf{x}^{\text{blend}} \succ \{\mathbf{x}^{\text{current}}, \mathbf{x}_1^{\text{suggest}}, \ldots, \mathbf{x}_K^{\text{suggest}}\}], \tag{14}$$

and then updates the internal models and generates new suggestions.

6.3 Extraction from Suggestion Regeneration

When the designer requests the "Regenerate Suggestions" functionality, the system can add a preferential data entry by assuming that the current design is preferred over all the current suggestions. That is, a preferential data entry:

$$d = [\mathbf{x}^{\text{current}} \succ \{\mathbf{x}_1^{\text{suggest}}, \ldots, \mathbf{x}_K^{\text{suggest}}\}], \tag{15}$$

is composed, and then the system refines the internal model and generates new suggestions. For the designer, this is useful not only for seeking unseen good designs but also for explicitly training the system to obtain more personalized suggestions. Figure 8 visualizes how the internal models are updated by applying this functionality.

7 GENERALITY DEMONSTRATION

To demonstrate the generality of our framework, we show three applications in diverse parametric design scenarios. We recommend readers watch the supplemental video for the entire screen recordings; in this section, we only offer several representative screenshots from the recordings. Note that the two interaction techniques (*i.e.*, blending and regenerating suggestions) are effectively used in the demonstration.

Photo color enhancement. Photo color enhancement is the task of adjusting the colors of a target image by manipulating parameters such as brightness and contrast [21]. This task is performed by various people, from casual end-users (*e.g.*, using Instagram) to

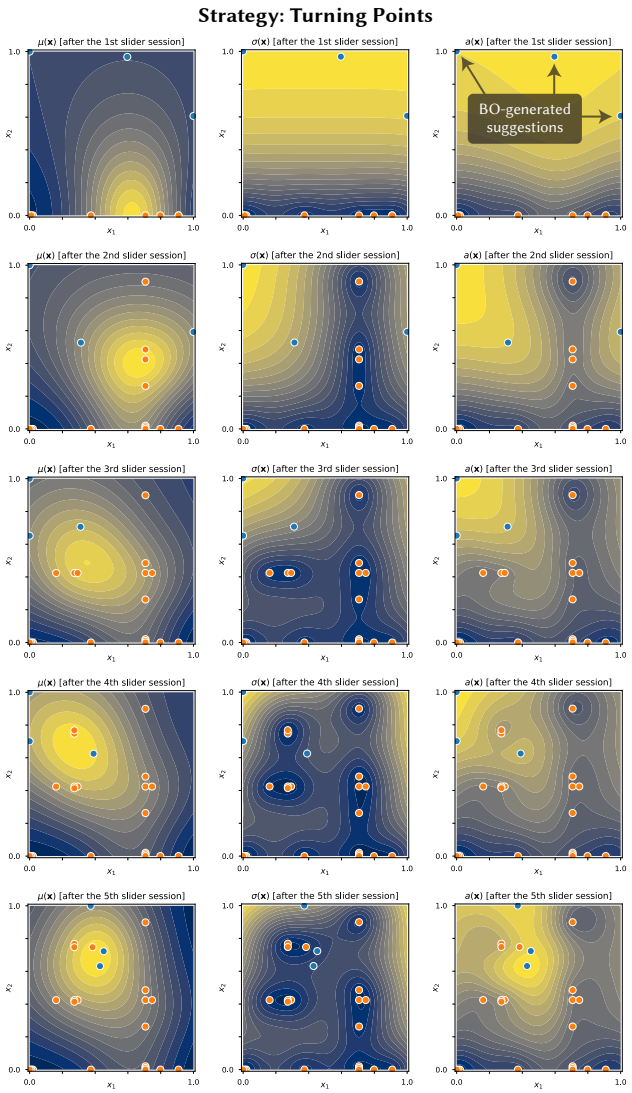

Figure 5: Visualization of BO's internal models (*i.e.*, $\mu(\mathbf{x})$, $\sigma(\mathbf{x})$, and $a(\mathbf{x})$) derived from the illustrative slider manipulation data (Figure 4) using the Turning Points strategy. Orange dots represent the points used by BO. Blue dots represent the suggestions generated by BO. (Left) The mean of the predictive distribution, $\mu(\mathbf{x})$, representing the estimated design goal. (Center) The standard deviation of the predictive distribution, $\sigma(\mathbf{x})$, representing uncertainty. (Right) The acquisition function, $a(\mathbf{x})$, which is used for suggestion sampling.

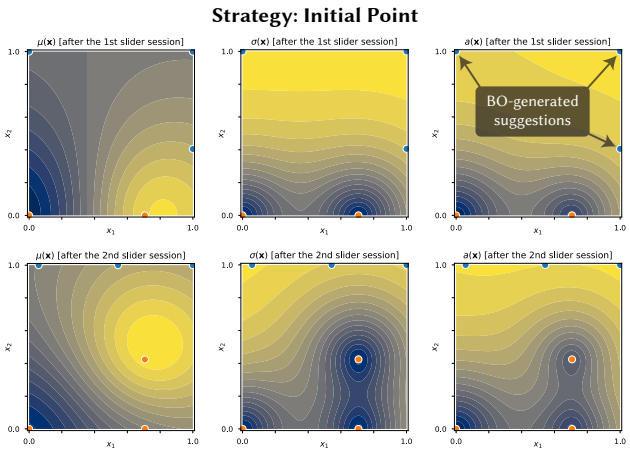

Figure 6: Visualization of BO's internal models derived from the illustrative slider manipulation data (Figure 4) using the Initial Point strategy. It shows the visualizations after the first and second slider manipulation sessions. Compared to the results using the Turning Points strategy (Figure 5), the points used by BO are sparse, and extreme suggestions are more likely to be sampled.

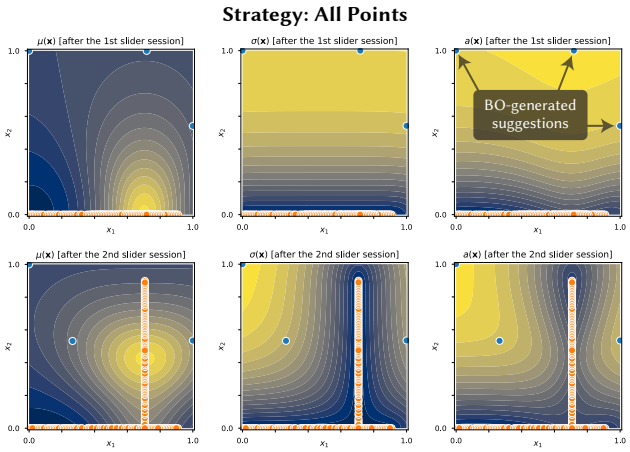

Figure 7: Visualization of BO's internal models derived from the illustrative slider manipulation data (Figure 4) using the All Points strategy. It shows the visualizations after the first and second slider manipulation sessions. It generates suggestions similar to the Turning Points strategy (Figure 5), but uses more points for BO.

expert photographers (*e.g.*, using Photoshop). For demonstration, our implementation uses a 12-dimensional setting (*i.e.*, brightness, contrast, saturation, lift (RGB), gamma (RGB), and gain (RGB)) following previous work [22]. This system runs as a standalone application. Figure 9 shows two enhancement sequences with different photographs. Note that different parameters are eventually applied to these two photographs, indicating different design goals.

Procedural modeling. Procedural modeling [45, 55] has become common in DCC tools [3, 44], and CAD tools [10, 19, 35]. In addition to visual art and engineering, it is also popular in the context of personal fabrication; for example, in Thingiverse [25], end-users share "customizable" parametric models with each other. For demonstration, we implemented a Blender addon, which provides an additional slider window on top of the Blender window. We prepared a

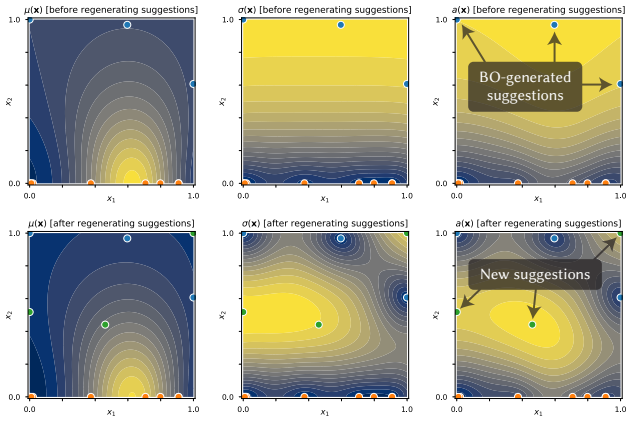

Figure 8: Comparison between before and after applying the "regenerate suggestions" functionality. Green dots represent the newly generated suggestions. In addition to generating new suggestions, this functionality reduces the estimation uncertainty; see the before and after of $\sigma(\mathbf{x})$.

fabrication-oriented procedural shape model, which has 6 parameters in total and is controlled by a cubic Bézier curve. We can preview it in real time with the Eevee renderer[2] in the Blender's 3D viewport. We show suggestions next to the current design in the 3D viewport. Figure 10 shows three modeling sequences with different design goals: modeling a vase, a plate, and a pen holder, respectively. Since our framework does not assume pre-defined design goals but learns the goals on the fly, it can handle these different cases in a unified manner and provide context-aware suggestions.

Procedural material. Procedural material is a method to create materials for 3D objects procedurally rather than relying on static texture images. It has been popular in many 3D graphics tools [3, 9, 53]. Designers need to adjust many non-intuitive parameters (such as the ones for controlling Perlin noise [32]) to create desired materials. Note that, while inverse procedural material methods have been proposed [15, 40], they require preparing reference photographs that precisely specify the designer's design goal, which is not practical in many cases. We prepared a procedural rusted metal material with peeled painting[3], controlled by 8 parameters. As in the procedural modeling demonstration, we implemented a Blender addon that provides a slider window and uses the real-time 3D viewport preview with the Eevee renderer. Figure 11 shows a material designing sequence, suggesting that our framework could successfully provide useful suggestions.

8 DISCUSSIONS AND FUTURE WORK

Assumption on slider manipulation. Our data extraction strategies (Section 6) assume that the designer always tries to find a better design in each individual slider session. It is our limitation that we have not evaluated how much and when this assumption is valid in real-world scenarios. Although this is a reasonable assumption, the designer may sometimes break this assumption (*i.e.*, releasing the slider knob at a location where the design is worse than the one before manipulating the slider). In this case, a wrong preferential data entry may be derived and added to the data for BO, decreasing the accuracy of the design goal estimation. Nonetheless, this is not critical to our framework because our data interpretation is probabilistic (Equation 3); the estimation can be improved once appropriate data entries are accumulated in the following slider manipulation. Also, even wrong estimation is not negative in our case (in contrast to human-in-the-loop systems); the designer can always ignore the assistance when finding it useless.

Target dimensionality. As described in Section 3.2, our framework assumes that the target dimensionality is around twenty at most. This is because DCC tools typically expose less than twenty sliders at once as a slider group. Also, it is known that BO does not work very effectively in higher-dimensional problems [7, 54]. Nonetheless, in case of applying our framework to higher-dimensional problems, one possible remedy is to provide a "freeze" option for each slider; only sliders with the "freeze" option unchecked are considered active dimensions. For the problems with no semantics in each slider (*e.g.*, searching for a latent code for deep generative models), it is effective to apply dimensionality reduction techniques and then use BO in the reduced space [57].

Assumption on preference. We assume that the designer has a consistent preference (Section 3.2). This does not require the designer to have a final look in their mind from the beginning; instead, it requires just picking a better option during slider manipulation according to their preference, which could be feasible in broader contexts. We also assume that the preference (*i.e.*, the goodness function) does not change over time. Nonetheless, even if the preference changes during a design session, the model often quickly adapts to the new preference as more slider manipulation sessions are observed. Also, we consider that supporting the change of preference could be achieved by providing a "discard history" button to allow the designer to explicitly indicate preference change or decaying the influence of observed data to prioritize newer observations.

Computational cost. Our framework could run so fast that it could provide new suggestions immediately once a slider session is done. For example, we observed in a typical design process that it could run in 18 ms (for the 1st–10th slider sessions), 65 ms (for 11th–20th slider sessions), and 122 ms (for the 21st–30th slider sessions) on average using MacBook Pro with M1 Max. Note that the numbers of slider sessions in our demonstrations (Section 7) were 12 at most. Note also that the user does not need to wait; suggestions can be asynchronously displayed.

Implementation to existing tools. Since many DCC tools (*e.g.*, Blender) can be augmented by developing editor extension plugins, implementing our framework as a third-party plugin for those tools is straightforward. Such a plugin needs three components: (1) a PBO engine, (2) a slider widget, and (3) a design preview widget. If the target tool's API is flexible enough, (2) and (3) are implemented by directly overriding the tool's native sliders and preview interfaces. If not, they can be implemented as an independent window. Our Blender demos (Figure 10 and Figure 11) use an independent window for sliders and use Blender's native preview interface.

[2] https://docs.blender.org/manual/en/3.1/render/eevee/index.html
[3] https://www.youtube.com/watch?v=5LYF4sJ3tBo

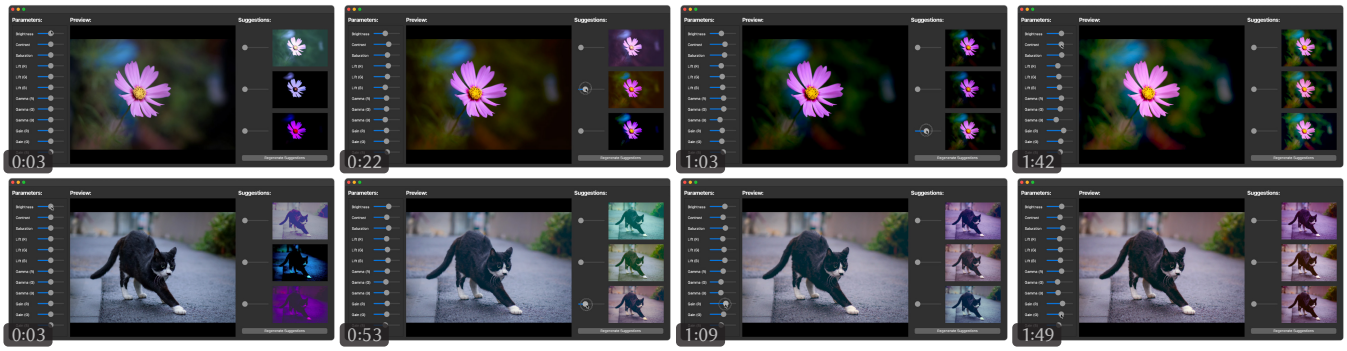

Figure 9: Screenshots of two design sessions (Top and Bottom) for photo color enhancement (12 parameters). Each photograph needs different parameters. Our system could estimate the design goal from slider manipulation on the fly and provide suggestions adaptively. Elapsed time is displayed on the lower left corner of each screenshot. See the supplemental material for the entire screen recordings.

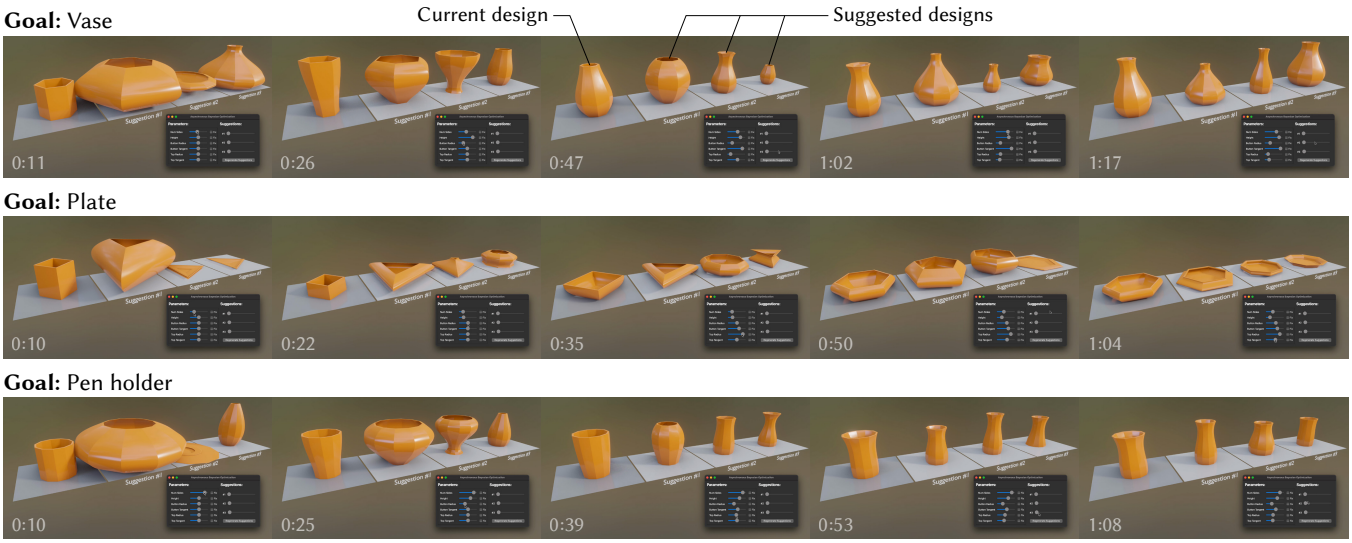

Figure 10: Screenshots of three design sessions (Top, Middle, and Bottom) for procedural shape modeling (6 parameters). Each design session uses the same procedural model but has a different design goal; creating a vase, a plate, and a pen holder, respectively. Our system could estimate the design goal from slider manipulation on the fly and provide suggestions adaptively. Elapsed time is displayed on the lower left corner of each screenshot. See the supplemental material for the entire screen recordings.

Sense of agency. It is an interesting question how the sense of agency in our framework differs from that in designer-in-the-loop optimization systems [6]. While this research question is beyond the scope, it is important in the human-AI collaboration viewpoint and is an interesting future research direction.

Usability and usefulness. We have not evaluated the usability and usefulness of our framework because our focus is on *enabling* a novel computational interaction concept (*i.e.*, BO as Assistant), and a formal evaluation of such interaction is considered beyond the scope of this paper. Nonetheless, evaluating these aspects is important to understand the efficacy of our framework. Note that these aspects should be highly dependent on contexts (*e.g.*, the designer's domain knowledge and design goal, the target domain, *etc.*), and it is not easy to discuss general usability and usefulness. It is interesting future work to focus on specific domains and users and then evaluate the usability and usefulness to obtain insights for better interactions.

Control of suggestion diversity. Our framework automatically adjusts the diversity of suggestions by computationally balancing exploration and exploitation using BO techniques. Note that it has been suggested that allowing users to control the balance would improve user engagement in human-in-the-loop systems [58]. Adding such control and evaluating the experience in the context of suggestive assistance would be interesting future work. Also, similar to previous systems [30, 51], it could be useful to generate exploration-dominated and exploitation-dominated suggestions separately.

Figure 11: Screenshots of a design session of procedural material editing (8 parameters). The design goal is to create a nice-looking rusted metal with peeled painting. Elapsed time is displayed on the upper left corner of each screenshot. See the supplemental material for the entire screen recording. The 3D model is provided by Bastien Genbrugge under CC BY 4.0 at https://skfb.ly/6pNQ6.

Design stages. We expect BO's exploration and exploitation aspects can support various stages of the whole design process. The former aspect (*i.e.*, sampling unexplored designs) can be useful to the stage of trying diverse options, and the latter (*i.e.*, sampling likely-preferred designs) can be useful to the stage of approaching a goal. Our framework would naturally support the seamless transition between such stages since BO typically tends to emphasize exploration at the beginning (because of a shortage of observed data) and then emphasize exploitation later. Explicit control of the exploration-exploitation balance [58] may help the seamless transition further, which is worth investigating in future work.

Creativity support. We hope that our framework inspires researchers in the Creativity Support Tools domain [42]. It is often discussed [24] that creative ideas are linked to originality and usefulness. We can see an analogy with exploration and exploitation in BO. With this in mind, new research questions arise: are BO-generated suggestions creative? Can BO support creativity? Future investigation is necessary to answer these questions.

9 CONCLUSION

In summary, our contributions are (1) the first framework where BO plays the role of a suggestive assistant, (2) the first technique to extract preferential data from slider manipulation and run BO using it, and (3) the demonstrations with diverse scenarios, validating its generality. We believe that our work is an important step toward drawing BO's unrevealed potential and the human-centered use of Bayesian methods in broader contexts.

ACKNOWLEDGMENTS

This work was supported in part by JST CREST Grant Number JPMJCR20D4, Japan. The 3D model used in Figure 11 was provided by Bastien Genbrugge under CC BY 4.0 at https://skfb.ly/6pNQ6, and we modified its material to demonstrate the proposed framework.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *Proc. KDD '19*. 2623–2631. https://doi.org/10.1145/3292500.3330701

[2] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. MenuOptimizer: Interactive Optimization of Menu Systems. In *Proc. UIST '13*. 331–342. https://doi.org/10.1145/2501988.2502024

[3] Blender Foundation. 2022. blender.org - Home of the Blender project - Free and Open 3D Creation Software. Retrieved March 24, 2022 from https://www.blender.org/.

[4] Eric Brochu, Tyson Brochu, and Nando de Freitas. 2010. A Bayesian Interactive Optimization Approach to Procedural Animation Design. In *Proc. SCA '10*. 103–112. https://doi.org/10.2312/SCA/SCA10/103-112

[5] Eric Brochu, Nando de Freitas, and Abhijeet Ghosh. 2007. Active Preference Learning with Discrete Choice Data. In *Proc. NIPS '07*. 409–416. http://papers.nips.cc/paper/3219-active-preference-learning-with-discrete-choice-data

[6] Liwei Chan, Yi-Chi Liao, George B Mo, John J Dudley, Chun-Lien Cheng, Per Ola Kristensson, and Antti Oulasvirta. 2022. Investigating Positive and Negative Qualities of Human-in-the-Loop Optimization for Designing Interaction Techniques. In *Proc. CHI '22*. 112:1–112:14. https://doi.org/10.1145/3491102.3501850

[7] Chia-Hsing Chiu, Yuki Koyama, Yu-Chi Lai, Takeo Igarashi, and Yonghao Yue. 2020. Human-in-the-Loop Differential Subspace Search in High-Dimensional Latent Space. *ACM Trans. Graph.* 39, 4 (July 2020), 85:1–85:15. https://doi.org/10.1145/3386569.3392409

[8] Toby Chong, I-Chao Shen, Issei Sato, and Takeo Igarashi. 2021. Interactive Optimization of Generative Image Modelling using Sequential Subspace Search and Content-based Guidance. *Comput. Graph. Forum* 40, 1 (February 2021), 279–292. https://doi.org/10.1111/cgf.14188

[9] Dassault Systèmes SolidWorks Corporation. 2022. Adobe. Retrieved March 24, 2022 from https://www.adobe.com/products/substance3d-designer.html.

[10] Dassault Systèmes SolidWorks Corporation. 2022. Design/Engineering | SOLIDWORKS. Retrieved March 24, 2022 from https://www.solidworks.com/domain/design-engineering.

[11] Ruta Desai, Fraser Anderson, Justin Matejka, Stelian Coros, James McCann, George Fitzmaurice, and Tovi Grossman. 2019. Geppetto: Enabling Semantic Design of Expressive Robot Behaviors. In *Proc. CHI '19*. 369:1–369:14. https://doi.org/10.1145/3290605.3300599

[12] John J. Dudley, Jason T. Jacques, and Per Ola Kristensson. 2019. Crowdsourcing Interface Feature Design with Bayesian Optimization. In *Proc. CHI '19*. 252:1–252:12. https://doi.org/10.1145/3290605.3300482

[13] Javier González, Zhenwen Dai, Andreas C. Damianou, and Neil D. Lawrence. 2017. Preferential Bayesian Optimization. In *Proc. ICML '17*. 1282–1291. http://proceedings.mlr.press/v70/gonzalez17a.html

[14] Javier Gonzalez, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. 2016. Batch Bayesian Optimization via Local Penalization. In *Proc. AISTATS '16*. 648–657. https://proceedings.mlr.press/v51/gonzalez16a.html

[15] Yiwei Hu, Julie Dorsey, and Holly Rushmeier. 2019. A Novel Framework for Inverse Procedural Texture Modeling. *ACM Trans. Graph.* 38, 6 (November 2019), 186:1–186:14. https://doi.org/10.1145/3355089.3356516

[16] Florian Kadner, Yannik Keller, and Constantin Rothkopf. 2021. AdaptiFont: Increasing Individuals' Reading Speed with a Generative Font Model and Bayesian Optimization. In *Proc. CHI '21*. 585:1–585:11. https://doi.org/10.1145/3411764.3445140

[17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *Proc. ICLR 2018*. https://openreview.net/forum?id=Hk99zCeAb

[18] Mohammad M. Khajah, Brett D. Roads, Robert V. Lindsey, Yun-En Liu, and Michael C. Mozer. 2016. Designing Engaging Games Using Bayesian Optimization. In *Proc. CHI '16*. 5571–5582. https://doi.org/10.1145/2858036.2858253

[19] Marius Kintel. 2022. OpenSCAD - The Programmers Solid 3D CAD Modeller. Retrieved March 24, 2022 from https://openscad.org/.

[20] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2014. Crowd-Powered Parameter Analysis for Visual Design Exploration. In *Proc. UIST '14*. 65–74. https://doi.org/10.1145/2642918.2647386

[21] Yuki Koyama, Daisuke Sakamoto, and Takeo Igarashi. 2016. SelPh: Progressive Learning and Support of Manual Photo Color Enhancement. In *Proc. CHI '16*. 2520–2532. https://doi.org/10.1145/2858036.2858111

[22] Yuki Koyama, Issei Sato, and Masataka Goto. 2020. Sequential Gallery for Interactive Visual Design Optimization. *ACM Trans. Graph.* 39, 4 (July 2020), 88:1–88:12. https://doi.org/10.1145/3386569.3392444

[23] Yuki Koyama, Issei Sato, Daisuke Sakamoto, and Takeo Igarashi. 2017. Sequential Line Search for Efficient Visual Design Optimization by Crowds. *ACM Trans. Graph.* 36, 4 (July 2017), 48:1–48:11. https://doi.org/10.1145/3072959.3073598

[24] Aaron Kozbelt, Ronald A. Beghetto, and Mark A. Runco. 2010. Theories of Creativity. In *The Cambridge Handbook of Creativity*, James C. Kaufman and Robert J. Sternberg (Eds.). Cambridge University Press, Chapter 2, 20–47. https://doi.org/10.1017/CBO9780511763205.004

[25] MakerBot Industries, LLC. 2022. Thingiverse - Digital Designs for Physical Objects. Retrieved March 24, 2022 from https://www.thingiverse.com/.

[26] Joe Marks, Brad Andalman, Paul A. Beardsley, William T. Freeman, Sarah F. Gibson, Jessica K. Hodgins, Thomas Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, Kathy Ryall, Joshua E. Seims, and Stuart M. Shieber. 1997. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proc. SIGGRAPH '97*. 389–400. https://doi.org/10.1145/258734.258887

[27] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets. In *Proc. CHI '18*. 369:1–369:12. https://doi.org/10.1145/3173574.3173943

[28] Addy Ngan, Frédo Durand, and Wojciech Matusik. 2006. Image-driven Navigation of Analytical BRDF Models. In *Proc. EGSR '06*. 399–407. https://doi.org/10.2312/EGWR/EGSR06/399-407

[29] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer Science+Business Media. https://doi.org/10.1007/978-0-387-40065-5

[30] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. DesignScape: Design with Interactive Layout Suggestions. In *Proc. CHI '15*. 1221–1224. https://doi.org/10.1145/2702123.2702149

[31] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proc. CVPR '19*. 165–174. https://doi.org/10.1109/CVPR.2019.00025

[32] Ken Perlin. 2002. Improving Noise. In *Proc. SIGGRAPH '02*. 681–682. https://doi.org/10.1145/566570.566636

[33] Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning*. The MIT Press. http://www.gaussianprocess.org/gpml/

[34] Yi Ren and Panos Y. Papalambros. 2011. A Design Preference Elicitation Query as an Optimization Process. *Journal of Mechanical Design* 133, 11 (November 2011), 111004:1–111004:11. https://doi.org/10.1115/1.4005104

[35] Robert McNeel & Associates. 2022. Rhino - Rhinoceros 3D. Retrieved March 24, 2022 from https://www.rhino3d.com/.

[36] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2018. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. In *Proc. ICML '18*. 4364–4373.

[37] Matthias Schonlau, William J. Welch, and Donald R. Jones. 1998. Global versus local search in constrained optimization of computer models. *IMS Lecture Notes—Monograph Series* 34, 1 (January 1998), 11–25. https://doi.org/10.1214/lnms/1215456182

[38] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (January 2016), 148–175. https://doi.org/10.1109/JPROC.2015.2494218

[39] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. 2009. Image Appearance Exploration by Model-Based Navigation. *Comput. Graph. Forum* 28, 2 (2009), 629–638. https://doi.org/10.1111/j.1467-8659.2009.01403.x

[40] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. 2020. Match: Differentiable Material Graphs for Procedural Material Capture. *ACM Trans. Graph.* 39, 6 (November 2020), 196:1–196:15. https://doi.org/10.1145/3414685.3417781

[41] Evan Shimizu, Matthew Fisher, Sylvain Paris, James McCann, and Kayvon Fatahalian. 2020. Design Adjectives: A Framework for Interactive Model-Guided Exploration of Parameterized Design Spaces. In *Proc. UIST '20*. 261–278. https://doi.org/10.1145/3379337.3415866

[42] Ben Shneiderman. 2007. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM* 50, 12 (Dec. 2007), 20–32. https://doi.org/10.1145/1323688.1323689

[43] Maria Shugrina, Ariel Shamir, and Wojciech Matusik. 2015. Fab Forms: Customizable Objects for Fabrication with Validity and Geometry Caching. *ACM Trans. Graph.* 34, 4 (July 2015), 100:1–100:12. https://doi.org/10.1145/2766994

[44] SideFX. 2022. Houdini | 3D Procedural Software for Film, TV & Gamedev | SideFX. Retrieved March 24, 2022 from https://www.sidefx.com/products/houdini/.

[45] Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A Survey on Procedural Modelling for Virtual Worlds. *Comput. Graph. Forum* 33, 6 (2014), 31–50. https://doi.org/10.1111/cgf.12276

[46] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proc. NIPS '12*. 2951–2959. https://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms

[47] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2012. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Trans. Inf. Theory* 58, 5 (May 2012), 3250–3265. https://doi.org/10.1109/TIT.2011.2182033

[48] Hideyuki Takagi. 2001. Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation. *Proc. IEEE* 89, 9 (Sep. 2001), 1275–1296. https://doi.org/10.1109/5.949485

[49] Michael Terry and Elizabeth D. Mynatt. 2002. Side Views: Persistent, On-demand Previews for Open-ended Tasks. In *Proc. UIST '02*. 71–80. https://doi.org/10.1145/571985.571996

[50] The Foundry Visionmongers Ltd. 2022. Nuke | VFX and Film Editing Software. Retrieved April 7, 2022 from https://www.foundry.com/products/nuke-family/nuke.

[51] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and Explore with a Layout Optimiser. In *Proc. DIS '16*. 543–555. https://doi.org/10.1145/2901790.2901817

[52] Kristi Tsukida and Maya R. Gupta. 2011. *How to Analyze Paired Comparison Data*. Technical Report UWEETR-2011-0004. University of Washington, Department of Electrical Engineering. https://vannevar.ece.uw.edu/techsite/papers/refer/UWEETR-2011-0004.html

[53] Unity Technologies. 2022. Unity Real-Time Development Platform | 3D, 2D VR & AR Engine. Retrieved March 24, 2022 from https://unity.com/.

[54] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando De Freitas. 2016. Bayesian Optimization in a Billion Dimensions via Random Embeddings. *J. Artif. Intell. Res.* 55 (February 2016), 361–387. https://doi.org/10.1613/jair.4806

[55] Mehmet Ersin Yumer, Paul Asente, Radomir Mech, and Levent Burak Kara. 2015. Procedural Modeling Using Autoencoder Networks. In *Proc. UIST '15*. 109–118. https://doi.org/10.1145/2807442.2807448

[56] Mingyuan Zhong, Gang Li, and Yang Li. 2021. Spacewalker: Rapid UI Design Exploration Using Lightweight Markup Enhancement and Crowd Genetic Programming. In *Proc. CHI '21*. 315:1–315:11. https://doi.org/10.1145/3411764.3445326

[57] Yijun Zhou, Yuki Koyama, Masataka Goto, and Takeo Igarashi. 2020. Generative Melody Composition with Human-in-the-Loop Bayesian Optimization. In *Proc. CSMC-MuMe '20*. 21:1–21:10. https://boblsturm.github.io/aimusic2020/papers/CSMC__MuMe_2020_paper_21.pdf

[58] Yijun Zhou, Yuki Koyama, Masataka Goto, and Takeo Igarashi. 2021. Interactive Exploration-Exploitation Balancing for Generative Melody Composition. In *Proc. IUI '21*. 43–47. https://doi.org/10.1145/3397481.3450663

A DETAILS OF BAYESIAN OPTIMIZATION IMPLEMENTATION

In this appendix section, we describe the details of our Bayesian optimization implementation for completeness and reproducibility. The implementation is available at https://koyama.xyz/project/bo-as-assistant/.

A.1 Gaussian Process

A GP model is characterized by its prior mean function $\mu_0 : \mathcal{X} \to \mathbb{R}$ and its kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ [33]. In this work, a constant prior mean function, $\mu_0(\mathbf{x}) = 0$, is assumed. The kernel is explained in Section A.2.

Suppose that we have M pairs of a data point and its goodness function value, $\{(\mathbf{x}_i, g_i)\}_{i=1}^{M}$. Under the GP prior, it is known [33] that the predictive distribution of the goodness function value at an unseen data point $\mathbf{x}$ is a Gaussian distribution; that is,

$$g(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x})). \tag{16}$$

The mean and variance of this Gaussian distribution can be written in closed forms as

$$\mu(\mathbf{x}) = \mathbf{k}^\top (\mathbf{K} + \theta_{\text{noise}}\mathbf{I})^{-1}\mathbf{g}, \tag{17}$$

$$\sigma^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^\top (\mathbf{K} + \theta_{\text{noise}}\mathbf{I})^{-1}\mathbf{k}, \tag{18}$$

where

$$\mathbf{g} = \begin{bmatrix} g_1 & \cdots & g_M \end{bmatrix}^\top, \tag{19}$$

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) & \cdots & k(\mathbf{x}, \mathbf{x}_M) \end{bmatrix}^\top, \tag{20}$$

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_M) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_M, \mathbf{x}_1) & \cdots & k(\mathbf{x}_M, \mathbf{x}_M) \end{bmatrix}, \tag{21}$$

and θ_{noise} is a hyperparameter to represent the noise level in observed function values, and our implementation sets $\theta_{\text{noise}} = 0.005$ by consulting the prior work [23].

A.2 Kernel

Following Snoek *et al.* [46], our implementation uses the Matérn 5/2 kernel:

$$k(\mathbf{x}_A, \mathbf{x}_B) = \theta_{\text{signal}} \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp\left(-\sqrt{5}r\right), \tag{22}$$

where $r = \theta_{\text{length}}^{-1} \|\mathbf{x}_A - \mathbf{x}_B\|$. The parameters, $\theta_{\text{signal}} > 0$ and $\theta_{\text{length}} > 0$, are the kernel hyperparameters, and we set $\theta_{\text{signal}} = 0.5$ and $\theta_{\text{length}} = 0.5$ throughout the paper.

Note that the kernel hyperparameters can be adaptively set via *maximum a posteriori* (MAP) estimation [23] rather than fixed. We tested MAP estimated hyperparameters, but we observed that the generated suggestions were similar to those in the case with fixed values. To avoid unnecessary complexity, we fixed the hyperparameter values.

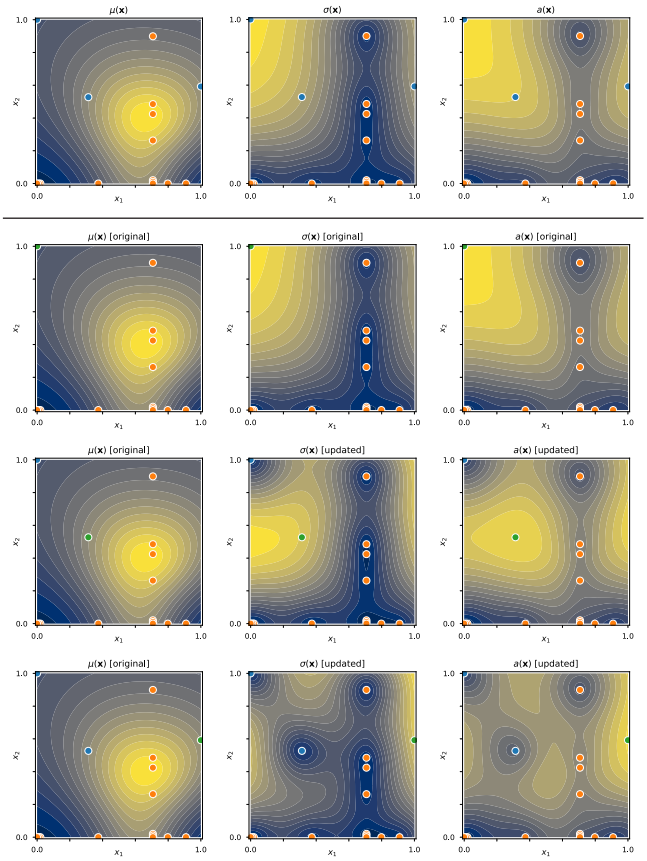

Figure 12: Breakdown of a process of sampling multiple points using a batch BO technique [37]. The first row shows three sampled points after the first and second slider sessions (see the second row of Figure 5). The second to fourth rows show how these three points are sampled sequentially. Every time a new point is sampled, the variance (or the standard deviation) is temporarily updated, and the acquisition function is updated accordingly.

A.3 Acquisition Function

For the acquisition function, our implementation uses the *Gaussian process upper confidence bound* (GP-UCB) [47], defined by

$$a^{\text{GP-UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + \beta\sigma(\mathbf{x}), \tag{23}$$

where $\beta \geq 0$ is a hyperparameter controlling the balance between exploration (*i.e.*, the effect of $\sigma(\mathbf{x})$) and exploitation (*i.e.*, the effect of $\mu(\mathbf{x})$). Our implementation uses a fixed value, $\beta = 0.5$, for simplicity. Note that it is possible to automatically change this value on the basis of the context [47] or let the designer directly control this value [58].

Another commonly used choice for the acquisition function is the *expected improvement* (EI) [46], calculated by

$$a^{\text{EI}}(\mathbf{x}) = (g^+ - \mu(\mathbf{x}))\Phi(\gamma(\mathbf{x})) + \sigma(\mathbf{x})\mathcal{N}(\gamma(\mathbf{x}); 0, 1), \tag{24}$$

where g^+ is the largest predicted function value at the so-far visited points, $\gamma(\mathbf{x}) = (g^+ - \mu(\mathbf{x}))/\sigma(\mathbf{x})$, and Φ is the cumulative distribution

function of the standard normal. We tested EI and observed that it worked as expected. We chose GP-UCB instead of EI simply because GP-UCB is more interpretable in terms of the balance between exploration and exploitation.

A.4 Sampling Multiple Points

Standard BO samples a single point in each step by maximizing the acquisition function (Equation 6). To generate multiple samples at once, we use a *batch* BO technique proposed by Schonlau *et al.* [37]. This technique greedily samples K points ($K = 3$ in our case) as follows. The first point is sampled as usual by maximizing the acquisition function (Equation 6). Since we do not know its goodness function value at this moment, we cannot update the mean function, μ (Equation 17), with this new point. Instead, we can update the variance function, σ^2 (Equation 18), with this new point since its calculation does not require knowing the goodness function value. Consequently, we can calculate an updated acquisition function using the original mean function μ and the updated variance function σ^2. By maximizing this updated acquisition function, the second point is sampled. The rest of the points (*i.e.*, the third point in our case) is sampled sequentially in the same way by maximizing updated acquisition functions. Figure 12 shows a breakdown of our batch BO process.