Learning Optimal Kernels for Gaussian Process Regression

Abstract

Black box optimization focuses on optimizing unknown functions in high-dimensional spaces. In certain applications, evaluating these functions is expensive, hence the optimization must be performed under tight sample budgets. Past works have approached black-box optimization using Gaussian Process Regression (GPR) and sample efficiency has been improved by utilizing information about the shape/structure of the function.

We propose to discover the structure of the function by learning a GPR kernel. Learning the kernel is achieved via an auxiliary optimization in (latent) kernel space, designed inside a variation autoencoder. The optimal kernel is expected to best "explain" the unknown function, helping lower the sample budget. Results show that our method, *Kernel Optimized Blackbox Optimization* (KOBO), effectively minimizes the black-box function at a substantively lower sample budget. Results hold not only in synthetic black box functions but also in real applications, e.g., where a hearing aid needs to be personalized with limited user queries.

1. Introduction

Many problems involve the optimization of an *unknown* objective function. Examples include personalizing content x to maximize a user's satisfaction $f(x)$, or training deep learning models with hyperparameters x to maximize their performance $f(x)$. Function $f(x)$ is unknown in these cases because it is embedded inside the human brain (for personalization) or too complex to derive (for hyper-parameter tuning). However, for any chosen sample x_i, the value of $f(x_i)$ can be evaluated. For hearing-aid personalization, say, evaluating the function would entail playing audio with some hearing-compensation filter x_i and obtaining the user's satisfaction score $f(x_i)$.

Bayesian methods like Gaussian Process Regression (GPR) are de facto approaches to black-box optimization. Using a set of function evaluations, conventional GPR (6) learns a probabilistic surrogate model $\hat{f}(x)$ for $f(x)$. The optimum is estimated on this surrogate as $\hat{x}^* = \text{argmin} - \hat{f}(x)$.

In most BBO problems, $f(x)$ is expensive to evaluate, hence a strict sample or query budget B is of interest. Techniques that lower this budget have garnered recent attention. One idea is to exploit domain knowledge about the rough shape of $f(x)$, i.e., select a GPR kernel that models this shape. With humans, for example, $f(x)$ may have a staircase structure (Figure 1) as they may not perceive differences in certain neighborhoods of x, but their ratings may change just outside that neighborhood. If GPR's surrogate model $\hat{f}(x)$ captures this staircase structure in its kernel, sample efficiency can improve. The question is, in the absence of domain knowledge, *can the optimal GPR kernel* $\mathbf{K}^*$ *be learnt, using the same sample queries needed to find* x^**?*

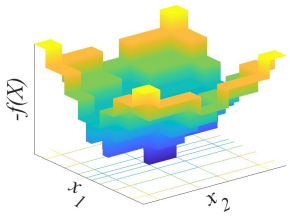


Figure 1. 2D staircase.

A growing body of research (11)(15)(26) is concentrating on kernel learning. One effective approach is Automatic Statistician (AS) (5) where authors compose complex-kernels by combining simple ones, and design a search method over the countably infinite complex-kernels (more in the related work section). Subsequent improvements over AS have used *Hellinger distance* as a measure of kernel similarity (22). This similarity measure guides an optimization-based search over the space of composite kernels. To reduce search complexity, (7) exploits additive structures in the search space and employs MCMC methods to discover the kernel. However, all these search techniques are impeded by the kernel space being discrete, consisting of only categorical compositions of simple kernels.

Our contribution is a kernel-learning technique that first creates a continuous space of kernels, then optimizes on that space to find the optimal kernel $\mathbf{K}^*$ for $\hat{f}(x)$. Briefly, a *Kernel Combiner* generates composite kernels by adding or multiplying simple "basis" kernels like Square Exponential (SE), Periodic (PE), Matérn (MA), etc. The resulting kernel space is discrete and inherits the past challenges of searching for the best kernel. Hence, we propose a generative model – a *Kernel Space Variational Autoencoder (KerVAE)* – that learns a low-dimensional continuous manifold of the discrete kernels. This manifold lives in the latent space

.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

of the VAE but we do not have a function of the kernels that can be optimized. Of course, given any kernel, we can evaluate its effectiveness[1], implying that finding the optimal kernel can also be formulated as a blackbox optimization problem. Thus, kernel learning is achieved through a *Kernel Space GPR (KerGPR)* that outputs an optimal kernel $\mathbf{K}^*$, which is then used to model the surrogate function $\hat{f}(x)$. Figure 2 illustrates the core idea of KOBO, where we learn a GPR's kernel by applying a second GPR on a carefully designed continuous kernel space.

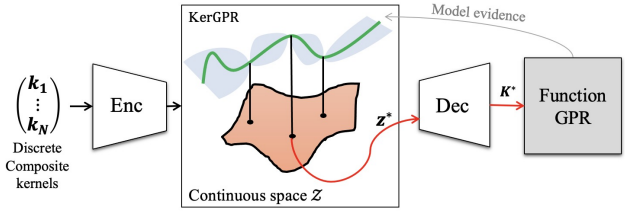


Figure 2. KerGPR in the latent space of a VAE

Results show using kernel $\mathbf{K}^*$ from KOBO reduces the number of function evaluations needed to reach the optimal (in comparison to SOTA methods that use MCMC-based discrete-kernel sampling (7)). Experiments are reported for various synthetic functions, and also from real-world experiments with U=6 users. The users were asked to rate the quality of a limited number of audio clips, and using these ratings, KOBO prescribed a personalized filter, which is applied to audio to maximizes that user's satisfaction. We find that KOBO was able to deliver greater satisfaction to users within a budget $B = 25$ queries, compared to conventional kernels. We believe there is still room for improvement, both in terms of performance as well as in the generalization of KOBO to other applications.

2. Problem Formulation

Consider an *unknown* real-valued function $f : \mathcal{H} \rightarrow \mathbf{R}$ where $\mathcal{H} \subseteq \mathbf{R}^N$, $N \geq 500$. Let x^* be the minimizer of $f(x)$. We want to estimate x^* using a budget of B queries. Thus, the optimization problem is,

$$\begin{aligned} \underset{\hat{x}\in\mathcal{H}}{\text{argmin}} \quad & ||f(\hat{x}) - f(x^*)||_2 \\ \text{s.t.} \quad & Q \leq B \end{aligned} \tag{1}$$

where Q is the number of times the objective function is evaluated/queried, and the sample budget $B \ll N$. Function f may be non-convex, may not have a closed-form expression, and its gradient is unavailable. Blackbox optimization suits such problems, and Bayesian methods like GPR require choosing a kernel to model the function structure – a poor choice incurs more queries for optimization.

[1]Using *model evidence*, described later, that captures how well the kernel fits the shape of the available data

Since queries can be expensive in many applications (e.g., users need to answer many queries, or a NeuralNet needs retraining for each hyper-parameter configuration), lowering Q is of growing interest. Kernel learning aims to address this problem.

3. Background on Bayesian Optimization (BO)

Bayesian optimization (6) broadly consists of two modules: (1) **Gaussian Process Regression (GPR)** that learns a Gaussian posterior distribution of the likely values function f can take at any point of interest x. (2) **Acquisition function**, a sampling strategy that prescribes the point at which f should be evaluated (or observed) next. We briefly discuss GPR to motivate the kernel learning problem.

3.1. Gaussian Process Regression (GPR)

Prior & Posterior: GPR generates a probabilistic surrogate model by defining a Gaussian distribution ($\mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$) over infinite candidate functions. At initialization, i.e., before any function observations, the prior distribution over the candidate functions is defined by $\boldsymbol{\mu} = \mathbf{0}$ and a covariance matrix $\mathbf{K}$. This matrix is computed using a kernel function k as, $\mathbf{K}_{ij}$=$k(x_i, x_j)$. The kernel essentially favors candidate functions that are similar to the kernel's own shape/structure – these candidates are assigned a higher likelihood. An expert with domain knowledge about the structure of $f(x)$ can choose the kernel judiciously, resulting in better surrogate functions $\hat{f}(x)$. Better $\hat{f}(x)$ will ultimately reduce the number of queries needed to optimize the objective $f(x)$.

After initialization, when the function f has been observed for a set of samples $\mathcal{X} = \{x_1, x_2, \ldots, x_K\}$, i.e., we know $\mathcal{F} = \{f(x_1), f(x_2), \ldots, f(x_K)\}$, the prior is updated to form the posterior distribution over the candidate functions. The posterior mean $\boldsymbol{\mu}$ is the most likely surrogate of the function f. Eqn. 2 shows the posterior generated by GPR.

$$P(\mathcal{F}|\mathcal{X}) \sim \mathcal{N}(\mathcal{F}|\boldsymbol{\mu}, \mathbf{K}) \tag{2}$$

where, $\boldsymbol{\mu} = \{\mu(x_1), \mu(x_2), \ldots, \mu(x_K)\}$, $\mathbf{K}_{ij}$=$k(x_i, x_j)$, and k represents a kernel function.

Predictions: To make predictions $\hat{\mathcal{F}} = f(\hat{\mathcal{X}})$ at a set of new points $\hat{\mathcal{X}}$, GPR uses the current posterior $P(\mathcal{F}|\mathcal{X})$ to define the the conditional distribution and hence prediction of $\hat{\mathcal{F}}$ is as shown in Eqn. 3. The proof and explanations of all the above are clearly presented in (28)).

$$P(\hat{\mathcal{F}}|\mathcal{F}, \mathcal{X}, \hat{\mathcal{X}}) \sim \mathcal{N}(\hat{\mathbf{K}}^T\mathbf{K}^{-1}\mathcal{F}, \hat{\hat{\mathbf{K}}} - \hat{\mathbf{K}}^T\mathbf{K}^{-1}\hat{\mathbf{K}}) \tag{3}$$

3.2. Kernel Selection

The kernel selection problem is to determine the optimal kernel $\mathbf{K}^*$ that best describes the structure of the unknown

function for the observations $(\mathcal{X}, \mathcal{F})$. As mentioned earlier, careful kernel selection can be crucial, especially with limited sample budgets B.

Let us consider $\mathcal{K}$ to be the class of all kernels that describes the observations. $\mathcal{K}$ can be thought of as the family of Gaussian Processes (GPs) with different kernel choices. Each kernel in $\mathcal{K}$ represents different structural characteristics of the function based on the observations. For example, Figure 3 illustrates GPs with different kernels. The hyperparameters $\gamma_{\mathbf{K}}$ of a kernel $\mathbf{K}$ distinguish the kernels within the same GP family.

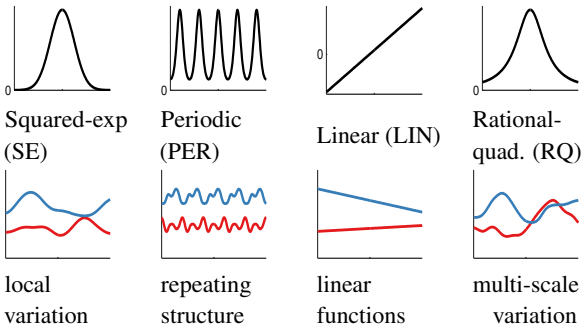


Figure 3. Row 1 shows simple (or basis) kernels (5). Row 2 shows corresponding surrogates drawn from a GP with the above kernel.

The goal of kernel selection is to select one kernel $\mathbf{K}^* \in \mathcal{K}$ that explains the function observations $(\mathcal{X}, \mathcal{F})$ best. Let's denote $\mathcal{L} : \mathcal{K} \to \mathbf{R}$ to be a model evidence that measures how well a kernel $\mathbf{K}$ fits the observations. We assume that evaluating $\mathcal{L}(\mathbf{K})$ for all kernels in $\mathcal{K}$ is too expensive. The kernel selection problem is then,

$$\mathbf{K}^*_\gamma = \underset{\mathbf{K} \in \mathcal{K}}{\operatorname{argmax}}\, \mathcal{L}(\mathbf{K}) \tag{4}$$

This problem is too difficult to be optimized with Bayesian optimization as the kernel space $\mathcal{K}$ is discrete. The key idea is to transform the problem in Eqn. 4 into a problem in continuous space upon which BO can be applied.

In our work, the "model evidence" $\mathcal{L}$ is chosen to be GPR posterior in Eqn. 2 as it generates the surrogate that best describes the observations informed by the chosen kernel.

$$\mathcal{L}(\mathbf{K}) = P(\mathcal{F}|\mathcal{X}, \mathbf{K}) \tag{5}$$

4. Kernel Learning in KOBO

We aim to create a continuous space of kernels and find the optimal kernel $\mathbf{K}^*$. To this end, kernel learning in KOBO is composed of the following 3 modules:

(1) ***Kernel Combiner*** creates composite kernels $\mathbf{K} \in \mathcal{K}$ that form the discrete kernel space $\mathcal{K}$.

(2) ***Kernel space Variational Autoencoder (KerVAE)*** defines the transformation from discrete kernel space $\mathcal{K}$ to low-dimensional continuous space $\mathcal{Z}$. KerVAE is pre-trained on the kernels generated by Kernel Combiner to obtain the corresponding continuous latent space $\mathcal{Z}$.

(3) ***Kernel space GPR (KerGPR)***: Since the kernel space objective $\mathcal{L}(\mathbf{K})$ is also a black-box, a GPR is used to optimize the posterior (from Eqn. 5) on $\mathcal{Z}$; this gives z^* and the corresponding optimal kernel $\mathbf{K}^*$.

Figure 4 connects all the modules to give a complete overview of KOBO. The main objective function from Eqn. 1 is optimized with a GPR that we call **Function GPR (fGPR)**. The kernel for f**GPR** is supplied by the **Kernel space GPR (KerGPR)** running inside **KerVAE**. User satisfaction scores are received in batches – they are used to optimize the kernel in **KerGPR** and to decide the next queries for f**GPR**. The process iterates until the sample budget B is exhausted.

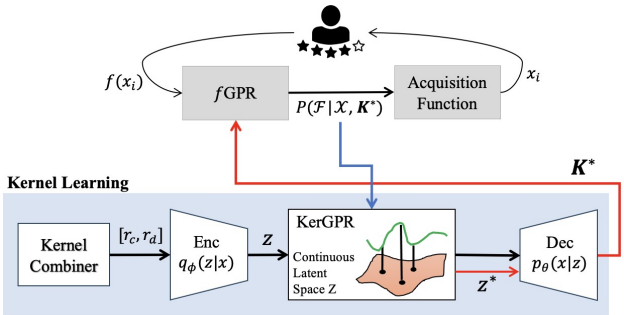


Figure 4. System flow: KOBO iterates across a function GPR (f*GPR*) on top and a kernel GPR (*KerGPR*) below that runs in the latent space of KerVAE. The blue arrow denotes the model evidence input to KerGPR, and the red arrow denotes the optimal kernel $\mathbf{K}^*$ supplied by KerGPR to fGPR.

4.1. Kernel Combiner

Complex kernels can be expressed as operations on the context-free grammar of base kernels (13). Given a set of base kernels $\mathcal{B} = \{A, B, C, D, E\}$, and a set of operators $\mathcal{O} = \{\text{add}, \text{multiply}, \text{end}, ...\}$, the Kernel Combiner generates a composite kernel $\mathbf{k}_\mathcal{C}$ by drawing kernels from $\mathcal{B}$ and operators from $\mathcal{O}$ with probabilities $p_\mathcal{B}, p_\mathcal{O}$. An example $\mathbf{k}_\mathcal{C} = A * C + B * D$.

To form a kernel space $\mathcal{K}$, the Kernel Combiner develops a unique representation for each $\mathbf{k}_\mathcal{C}$. This representation consists of two parts: (1) *Grammar-based representation*, and (2) *Data-based representation*. The Grammar-based representation aims to map each kernel to a point in kernel space $\mathcal{K}$. As discussed next, it tries to preserve the semantic construction of complex kernels $\mathbf{k}_\mathcal{C}$ from simple base kernels $\mathcal{B}$. The Data-based representation encodes the "distances" between different $\mathbf{k}_\mathcal{C}$ to base kernels $\mathcal{B}$, depending on the function samples. This representation provides adequate

information to KerVAE to learn the continuous latent space $\mathcal{Z}$.

Grammar-based representation: Given a composite kernel $\mathbf{k}_\mathcal{C}$, its grammar-based representation is a vector r_c, designed as follows. Let $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{E}$ be five base kernels in $\mathcal{B}$. These are simple kernels like Square-exponential, Periodic, Rational Quadratic, etc. Any composite kernel $\mathbf{k}_\mathcal{C}$ created from the base kernels is expressed in the form of Eqn. 6. The code r_c is then the vector of indices, i.e., $r_c = [a_1, b_1, c_1, d_1, e_1, a_2, b_2, c_2, d_2, e_2, a_3, b_3, c_3, d_3, e_3]$.

$$\begin{aligned} \mathbf{k}_\mathcal{C} = \mathbf{A}^{a_1} * \mathbf{B}^{b_1} * \mathbf{C}^{c_1} * \mathbf{D}^{d_1} * \mathbf{E}^{e_1} \\ + \mathbf{A}^{a_2} * \mathbf{B}^{b_2} * \mathbf{C}^{c_2} * \mathbf{D}^{d_2} * \mathbf{E}^{e_2} \\ + \mathbf{A}^{a_3} * \mathbf{B}^{b_3} * \mathbf{C}^{c_3} * \mathbf{D}^{d_3} * \mathbf{E}^{e_3} \quad \ldots \end{aligned} \tag{6}$$

If a composite kernel is, say, $\mathbf{k}'_\mathcal{C} = \mathbf{A}^2 * \mathbf{E} + \mathbf{C} * \mathbf{D}$, then the corresponding Grammar-based representation would be $r'_c = [2, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]$. Note, the elements of the code vectors can also be fractions.

This encoding scheme has two advantages. Firstly, each code r_c preserves its composition, i.e., given the code vector r_c, the base kernels and the operators used to construct $\mathbf{k}_\mathcal{C}$ can be interpreted. Secondly, unlike one-hot encoding schemes (8)(19), the code space is continuous. For instance, a code $r''_c = [2, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]$ – which is only a flip of the second element in r'_c – results in $\mathbf{k}''_\mathcal{C} = \mathbf{A}^2 * \mathbf{B} * \mathbf{E} + \mathbf{C} * \mathbf{D}$. In general, a small change in the code produces a small modification to the kernel composition.

Data-based representation: We also want to represent the combined kernel $\mathbf{k}_\mathcal{C}$ based on the available function observations $(\mathcal{X}, \mathcal{F})$. For each $\mathbf{k}_\mathcal{C}$, we compute the "distances" between its covariance matrix $M_\mathcal{C}$ and the covariance matrix of each base kernel, $M_{b \in \mathcal{B}}$. The "distance" metric quantifies the similarity in terms of their closeness in kernel space $\mathcal{K}$. We use the Forbenius norm to compute the matrix distances. This representation is denoted as $r_d \in \mathcal{R}^{|\mathcal{B}|}$.

$$r_d = ||M_\mathcal{C} - M_{b \in \mathcal{B}}||_F \tag{7}$$

The final representation of a composite kernel $\mathbf{k}_\mathcal{C}$ is $r = [r_c, r_d]$. This is used to train the KerVAE to generate the continuous low-dimensional kernel space $\mathcal{Z}$.

4.2. Kernel space Variational Autoencoder (KerVAE)

Composite kernels $\mathbf{k}_\mathcal{C}$ form a discrete kernel space $\mathcal{K}$. Brute force search on this space is expensive. Besides, the discrete nature of the space also prohibits optimization techniques – even Bayesian Optimization – to find the optimal $\mathbf{K}^*$.

We turn to a generative model, Kernel space Variational Autoencoder (KerVAE) to learn a continuous low-dimensional latent space $\mathcal{Z}$ of the kernel space $\mathcal{K}$. A Kernel space GPR (KerGPR) in $\mathcal{Z}$ can then be used to find the optimal $\mathbf{K}^*$.

KerVAE has two main components: (1) a probabilistic encoder that models $q_\phi(z|x) \sim p_\theta(x|z)p(z)$ parameterized by ϕ where $p(z)$ is the prior over the latent space, and (2) a decoder that models the likelihood $p_\theta(x|z)$ parameterized by θ. The parameters of $q_\phi(z|x), p_\theta(x|z)$ are optimized by joint maximization of the ELBO loss (16),

$$\mathcal{L}(\phi, \theta, x) = \mathbf{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)] \tag{8}$$

Encoder: KerVAE's encoder learns a Gaussian distribution whose mean and deviation are the outputs of a Multilayer Perceptron (MLP). We define,

$$\begin{aligned} q_\phi(z|x) &= \mathcal{N}(z|\mu_{\text{enc}}, \sigma^2_{\text{enc}}\mathbf{I}) \\ [\mu_{\text{enc}}, \sigma_{\text{enc}}] &= \text{MLP}(x|\phi) \end{aligned} \tag{9}$$

Our MLP is a feed-forward network with three hidden layers and a ReLU activation function parameterized by weights ϕ. $\mu_{\text{enc}}, \sigma^2_{\text{enc}}$ denote the mean and variance of the encoder Gaussian distribution.

Decoder: The output of the KerVAE's decoder is $l = \text{MLP}(z|\theta)$ with the same MLP design as the encoder. θ denotes the corresponding weights that need to be optimized jointly with ϕ during training. The KerVAE Decoder models a Gaussian likelihood $p_\theta(x|z)$ as follows,

$$p_\theta(x|z) \sim \mathcal{N}(x|\mu_{\text{dec}}, \sigma^2_{\text{dec}}\mathbf{I}) \tag{10}$$

where $[\mu_{\text{dec}}, \sigma_{\text{dec}}] = l$ is the MLP's output, and denotes the mean and variance of the decoder's Gaussian distribution.

The KerVAE can be pre-trained independent of the Function GPR (fGPR) and the kernel space GPR (KerGPR). This is because the grammar-based representation r_c does not depend on any function evaluation data $\mathcal{D} = (\mathcal{X}, \mathcal{F})$, and computing the data-representation r_d only uses an initial set of evaluations $\mathcal{D}_0 = (\mathcal{X}_0, \mathcal{F}_0)$.

Once KerVAE is pre-trained, kerGPR is used to determine the optimal kernel $z^* \in \mathcal{Z}$ and the corresponding $\mathbf{K}^* \in \mathcal{K}$.

4.3. Kernel space GPR (KerGPR)

KerVAE's latent space $\mathcal{Z}$ is a (low-dimensional) space of candidate kernels. Optimizing on this kernel space is also a blackbox optimization problem (similar to Eqn. 1) because the objective function is again unknown[2]. We use GPR to find z^* in the latent space and hypothesize that it decodes to the optimal kernel $\mathbf{K}^*$. Therefore, it is possible to search for the best model in continuous space $\mathcal{Z}$ rather than in the original (discrete) space $\mathcal{K}$. Thus our optimization objective is:

$$\mathbf{K}^* = \text{Dec}(\underset{z \in \mathcal{Z}}{\text{argmax}}\, P(\mathcal{F}|\mathcal{X}, \text{Dec}(z))) \tag{11}$$

[2]Observe that for a given z, the objective function can be evaluated by decoding z to a kernel k, and computing the model evidence for this kernel from Eqn. 5.

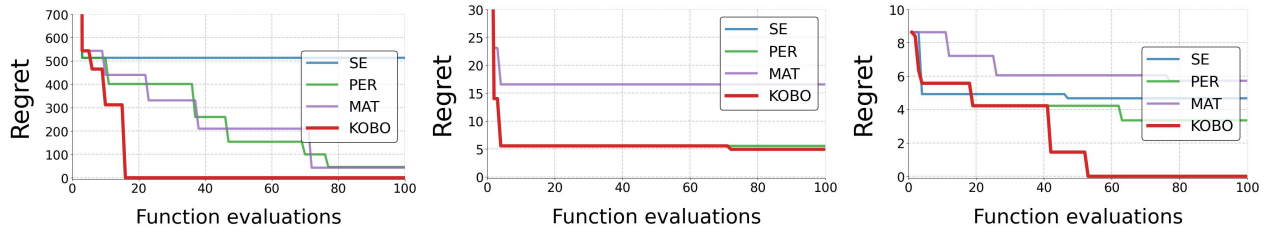


Figure 5. Comparison of KOBO and conventional BO using SE, PER, RQ, and Matérn kernels for (a) Staircase functions, (b) Smooth Branin, and, (c) Periodic Michalewicz

where, Dec(.) denotes the mapping of a point from $\mathcal{Z}$ space to that in $\mathcal{K}$ space through the KerVAE decoder. The optimal kernel $\mathbf{K}^*$ is then used by Function GPR (fGPR) — in the GPR posterior in Eqn. 2 — to generate surrogates that closely model the unknown function structure (Eqn. 1).

5. KOBO: Kernel Optimized BO

Figure 4 shows the complete **Kernel Optimized Blackbox Optimization (KOBO)** model, with two main modules:

(1) ***Function GPR (fGPR)***: This is a Bayesian framework aimed at optimizing the objective function $f(x)$ through a GPR that uses an initial kernel to generate surrogate functions, and then uses an acquisition function to decide the next sample x_i to evaluate. This module forwards the model evidence, $p(\mathcal{F}|\mathcal{X}, \mathbf{K})$ to the second module and receives updated kernel prescriptions from it.

(2) ***Kernel Learning***: This second module learns the continuous kernel space through a kerVAE and applies a GPR on the latent space to estimate z^*. The GPR uses model evidence for the optimization, and returns the optimal kernel $\mathbf{K}^*$ to f**GPR**. The iterations occur until the sample budget B has been expended.

Algorithm 1 presents the interaction between the modules.

Algorithm 1 Kernel Optimized Bayesian Optimization

Create a prior Gaussian with a base kernel in $\mathcal{B}$ in fGPR
Obtain observations $\mathcal{D}_0 = (f(\mathcal{X}_0), \mathcal{X}_0)$ at initial n_0 points: $\mathcal{X}_0$ chosen at random
Generate $\mathbf{k}_\mathcal{C} \in \mathcal{K}$ and compute r_d against $\mathcal{D}_0$
Pre-train KerVAE using r to obtain continuous space $\mathcal{Z}$
Update the fGPR posterior in 2 with observations $\mathcal{D}_0$
Use the fGPR posterior in KerGPR to obtain $\mathbf{K}^*$
Modify the fGPR posterior with the optimal kernel $\mathbf{K}^*$
while $n \leq B$ **do**
  Let x' be the next fGPR sample. Observe $f(x')$
  Update fGPR with observations $(f(\mathcal{X}_n), \mathcal{X}_n = x')$
  Using the updated posterior kerGPR learns new $\mathbf{K}^*$
  Modify the fGPR posterior with new $\mathbf{K}^*$
  $\mathcal{D}_n = \{\mathcal{D}_{n-1} \cup (f(\mathcal{X}_n), \mathcal{X}_n = x')\}, n = n + 1$
end while

6. Evaluation and Results

6.1. Gain from composite kernels over simple kernels

Metric: To quantify the effectiveness of kernels, we use the metric of **Regret**, defined as the difference between the predicted and true minimum, $(f(\hat{x}^*) - f(x^*))$. We compare KOBO's regret against 5 popular base kernels $\mathcal{B} = \{\text{SE}, \text{PER}, \text{RQ}, \text{MAT}, \text{LIN}\}$ which respectively denote Square-Exponential (SE), Periodic (PER), Rational Quadratic (RQ), Matérn (MAT), and Linear (LIN) kernels. The SE base kernel is used as the initialization for all KOBO experiments. All reported results are an average of 10 runs.

Synthetic functions: We report results from experiments conducted on 3 types of synthetic functions $f(x)$, all assumed to be black-boxes:

(1) Staircase functions shaped similar to that of Figure 1 but in $N = 2000$ dimensions; they exhibit non-smooth structures (2).
(2) Smooth benchmark functions such as BRANIN commonly used in Bayesian optimization research (25).
(3) Periodic functions such as MICHALEWICZ that exhibit repetitions in their shape (25). More details on functions and evaluation parameters in the Appendix.

Results: Figures 5(a),(b),(c) report Regret for the Staircase, Smooth (BRANIN), and the MICHALEWICZ objective functions, respectively. For each objective, KOBO is compared to base kernels in $\mathcal{B} = \{\text{SE}, \text{PER}, \text{RQ}, \text{MAT}, \text{LIN}\}$. Kernel learning gains are understandably small when the objective function is smooth (Figure 5(b)). However, when the objective function exhibits complexity like Staircase or MICHALEWICZ in Figure 5(a), (c), KOBO minimizes regret in much fewer function evaluations. When function evaluation is expensive, this is a desirable advantage.

6.2. KOBO versus SOTA's composite kernels

Another Metric: We will again use **Regret** but since we are learning the kernel in the latent space of KerVAE, we use a second metric **Model Evidence**, which

is the normalized probability of generating the observed data $\mathcal{D}$ given a kernel model $\mathbf{K}$, i.e., $\log(P(\mathbf{f}|\mathcal{X},\mathbf{K}))/|\mathcal{D}|$ (22). Computing the exact model evidence is generally intractable in GPs (24)(21). We use the Bayesian Information Criterion (BIC) to approximate the model evidence as $\log(P(\mathbf{f}|\mathcal{X},\mathbf{K})) = -\frac{1}{2}\mathbf{f}^T\mathbf{K}^{-1}\mathbf{f} - \frac{1}{2}\log((2\pi)^N|\mathbf{K}|)$, where N is the dimensions of the input space $\mathcal{H} \subseteq \mathbf{R}^N$.

We will plot `Regret` against the number of "Function Evaluations" (on the X axis), but for `Model Evidence`, we will plot it against the number of "Latent Evaluations". Recall that `Model Evidence` is the metric used in the latent space of KerVAE to find the "best" kernel $\mathbf{K}^*$. Hence "Latent Evaluations" denotes the number of latent space samples z and their corresponding kernels $\text{Dec}(z) = \mathbf{K}$ sampled by KerGPR to find $\mathbf{K}^*$. This reflects the computation overhead of adding the `KOBO` kernel learning module to conventional GPR.

Baseline: The MCMC algorithm (7)(1) is used as the comparison baseline. The number of possible composite kernel models $\mathbf{k}_\mathcal{C}$ is very large. This makes computing the model evidence for each kernel prohibitively expensive. However, computing a small (i.e., not super-exponential) number of model evidence is tractable. Thus, the model evidence is sampled using any MCMC method like Reverse Jump MCMC (RJMCMC) (12) or Metropolis-Hastings (7).

To apply Metropolis-Hastings, the proposal distribution $g(\mathbf{k}'|\mathbf{k})$ is defined as follows; given a current kernel model $\mathbf{k}$, we can either add or multiply a chosen base kernel from $\mathcal{B}$. We construct the proposal distribution by first choosing whether to add or multiply, each with 50% probability, and next, picking a base kernel from $\mathcal{B}$ uniformly at random.

Given the current kernel $\mathbf{k}_j$ (initializing $\mathbf{k}_0$ to the final kernel model found in the previous iteration), we sample a proposed model $\mathbf{k}'$ from the proposal distribution $g(\mathbf{k}'|\mathbf{k}_j)$. Next, we compute the model evidence for $\mathbf{k}'$ and use this to compute the Metropolis-Hastings acceptance probability:

$$A(\mathbf{k}'|\mathbf{k}_j) = \min\left(1, \frac{P(\mathcal{F}|\mathcal{D},\mathbf{K}')g(\mathbf{k}_j|\mathbf{k}')}{P(\mathcal{F}|\mathcal{D},\mathbf{K}_j)g(\mathbf{k}'|\mathbf{k}_j)}\right) \tag{12}$$

Finally, we update the current state of MCMC to $\mathbf{k}'$ with probability $A(\mathbf{k}'|\mathbf{k}_j)$.

Results: Figure 6(a) shows that `KOBO` outperforms MCMC baseline for the staircase objective function in Figure 1. `KOBO` attains the global minimum in about 17 function evaluations in contrast to MCMC, which incurs 28. Figure 6(b) illustrates that `KOBO`'s KerGPR achieves significantly higher "Model Evidence" in the latent space $\mathcal{Z}$ over the MCMC baseline in 20 iterations, i.e., `KOBO`'s "best" kernel $\mathbf{K}^*$ seems to better explain the observed data.

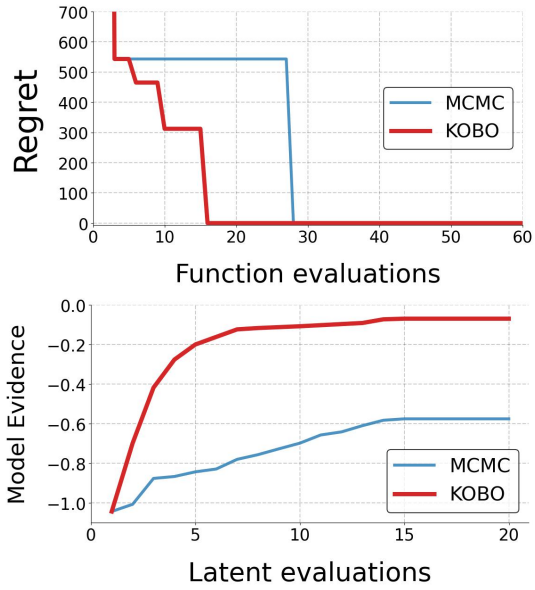


Figure 6. Comparison of `KOBO` and MCMC: (a) `Regret` (b) `Model Evidence`.

6.3. Is K^* indeed learning the structure of $f(x)$?

If we know the objective function $f(x)$, we can verify whether $\mathbf{K}^*$ has learnt its structure. To test this, we sample a function from a GP with a known kernel K^+ and pretend that to be $f(x)$; we check if K^* converges to K^+.

Results from N-dimensional synthetic functions: Table 1 demonstrates `KOBO`'s ability to learn complex kernels. The top row shows the N-dimensional objective functions sampled from a GP using different known kernels. The subsequent rows show KerVAE's learnt kernel after Q observations/queries. With more Q, KerGPR approaches the optimal (known) kernel.

Q	$f_1(x) \sim A*A*B+C$	$f_2(x) \sim C+D$	$f_3(x) \sim D*B+D$	$f_4(x) \sim D*B*A$
5	$A*B$	A	A	B
10	$A*B+C$	$A+D$	$A*B*D+D$	B
15	$A*A*B+D$	$A*C+D$	$A*B+D$	$B*D$
20	$A*B+C*D$	$A*C+D$	$B*D+D$	$B*D*C$
25	$A*A*B+C*D$	$A*C+D$	$B*D+D$	$B*D*C$

Table 1. High-Dimensional Function modeling with `KOBO`, $\{A, B, C, D, E\} = \{\text{SE}, \text{PER}, \text{RQ}, \text{MAT}, \text{LIN}\}$

Learning real-world CO_2 emission data: Figures 7(a,b,c) plot $f(x)$ as the true CO_2 emissions (27) and its corresponding mean fGPR model for increasing function observations. In Figure 7(a), we utilize the first 20% of the available data as observations; `KOBO` learns $\mathbf{K}^* = \text{SE} * \text{PER} + \text{RQ}$. The periodic structure of the function, as evident in the first 20% data, is captured by the KerGPR. When the first 40% of the data is observed, KerGPR captures the downward linear trend of the function resulting in $\mathbf{K}^* = \text{SE} * \text{PER} + \text{PER} + \text{LIN}$. With 60% of the data, $\mathbf{K}^* = \text{SE} * \text{PER} * \text{RQ} + \text{PER} * \text{LIN} + \text{LIN}$ models the interplay between the function's periodic structure and linear

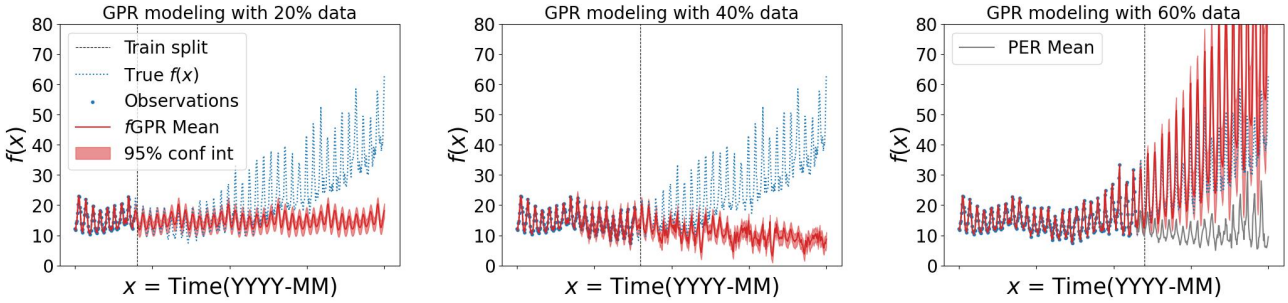


Figure 7. Function structure modeling with KOBO*:* Atmospheric CO_2 emissions (27) is the objective function. The fGPR posterior mean function model using (a) 20% (b) 40% (c) 60% of the data is displayed. All data to the left of the black dotted line are used as observations. The blue dotted line denotes the true function. The red line denotes the fGPR mean and the grey line denotes GPR with Periodic kernel.

Hearing Loss																		
Q	U1			U2			U3			U4			U5			U6		
	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER
5	6	6	6	8	8	8	6	6	6	6	6	6	7	7	7	5	5	5
10	6	8	6	8	8	8	6	7	7	7	6	6	7	9	8	7	9	6
15	6	10	7	8	10	8	7	9	7	8	8	7	7	9	8	7	10	6
20	10	10	10	9	10	9	7	10	10	10	10	9	10	10	9	10	10	9
25	10	10	10	10	10	10	9	10	10	10	10	10	10	10	9	10	10	9
Random Audio Corruption																		
Q	U1			U2			U3			U4			U5			U6		
	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER	SE	KOBO	PER
5	1	1	1	3	3	3	1	1	1	3	3	3	0	0	0	0	0	0
10	2	3	1	4	4	4	2	2	3	5	8	6	3	3	2	7	6	5
15	2	4	5	4	10	4	2	2	3	5	8	7	3	3	2	7	6	5
20	4	10	5	4	10	9	5	8	4	7	9	8	4	10	3	8	10	5
25	8	10	9	8	10	9	10	8	7	7	10	9	4	10	7	10	10	7

Table 2. Audio personalization results with 6 volunteers.

trends. In contrast, a conventional Periodic (PER) kernel (shown in Figure 7(c)) is only able to capture the periodic structure, not the linear trend, even with 60% of the data.

6.4. User experiments: audio personalization

This section reports experiments with real volunteers with the goal of audio personalization. We deliberately corrupt audio played to the user with the aim of helping the user pick a filter h^* that cancels the effect of the corruption – called equalization – and recovers the original audio; hence maximizing the user's audio satisfaction. Therefore, a GPR employed in the space of all audio filters $\mathcal{H}$, optimizes the user satisfaction $f(h)$ for h^*. At each iteration, the corrupted audio is filtered with h' (as recommended by GPR) and played to the user. The user's score (0 to 10) of the perceived audio quality serves as the function observations $f(h')$. User feedback is finite and the frequency selective nature of human hearing (3) makes optimizing $f(h)$ well suited for kernel learning BO methods like KOBO.

We invited 6 volunteers to rate the audio clips filtered with h, prescribed by fGPR. These observations are also employed by KerGPR to learn the optimal kernel $\mathbf{K}^*$. We evaluate how much Regret can be minimized by KOBO (and under what sample budget B) compared to conventional GPR optimizers that use fixed simple base kernels $\{\text{SE}, \text{PER}\}$.

Results: Table 2 displays the final satisfaction scores from the 6 volunteers (U1-6). Audio was corrupted first by a "hearing loss" audiogram (4), then by a "Random" filter (more details in Appendix). KOBO can improve user satisfaction in $Q = 15$ queries over the Baselines, and achieves the maximum with $Q = 25$ queries. With fewer $Q \leq 10$, KOBO cannot yet outperform the Baselines as the function f's search space, $\mathcal{R}^{4000}$, had not been sufficiently sampled for learning the optimal kernel. The audio demos at various stages of the optimization is made public[3]. We also report early results from an image personalization application that benefits from KOBO (see Appendix A.2.4).

7. Ablation Studies

Proposed kernel encoding vs. One-hot representation: Past work have proposed one-hot encoding (8)(19) to encode discrete items, while we design the grammar-based encoding. Table 3 shows both representations for two semantically similar kernels. The One-hot encoding assigns codes to each base kernel and each operator (e.g., $A = 00001$, $* = 010$, etc.) and concatenates them; consequently the two one-hot codes are quite different even for similar kernels. KOBO's grammar representation preserves the "continuity" in code space. Figure 8(a) and (b) visualize this notion of continuity in the code space, i.e., the latent space learnt by KerVAE with $r_d = 0$.

[3]https://keroptbo.github.io/

Table 3. Comparing grammar vs. one-hot encoding

	$\mathbf{k}_{\mathcal{C}_1} = \mathbf{A}^2 * \mathbf{B} + \mathbf{D}$
Grammar-based	$[2, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$
One-hot	$[00001, 010, 00001, 010, 00010, 001, 01000, 000, 00000]$
	$\mathbf{k}_{\mathcal{C}_2} = \mathbf{A}^2 * \mathbf{B} + \mathbf{C} * \mathbf{D}$
Grammar-based	$[2, 1, 0, 0, 0, 0, 0, \mathbf{1}, 1, 0, 0, 0, 0, 0, 0]$
One-hot	$[00001, 010, 00001, 010, 00010, 001, 0\mathbf{01}00, 0\mathbf{1}0, 0\mathbf{1}000]$

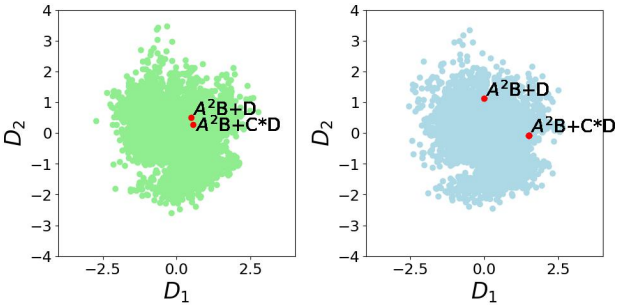


Figure 8. Top-2 SVD dimensions of the code space for (a) Grammar-based encoding & (b) One-hot encoding.

With and without data-based representation: Data-based representation r_d encodes the similarity between the grammar-based code r_c and the objective function's structure. Consider again the grammar-based codes in Table 3 – they are similar in code space but since they model different kernels, they should be mutually further away in the latent kernel space. Figure 9 visualizes this; the latent space that also includes the data context will be tailored to the objective function, ensuring efficiency in KerGPR.

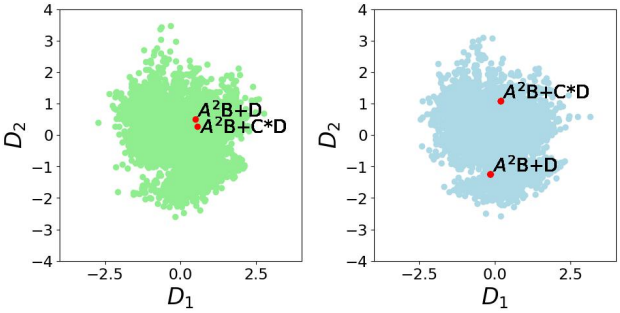


Figure 9. Top-2 SVD dimensions of the KerVAE trained on (a) only grammar-based code r_c & (b) both grammar and data, $[r_c, r_d]$.

8. Related Work

A body of work in Bayesian optimization explores kernel learning for improving performance. Authors of (26) treat the kernel as a random variable and learn its belief from the data. The probabilistic kernel is represented as an additional variational variable in a variational inference (VI) framework. (30) introduces kernels with random Fourier features for meta-learning tasks. The kernel features are learned as latent variables of a model to generate adaptive kernels. In contrast, KOBO uses variational inference as an auxiliary module to only learn a *continuous* latent kernel space; the KerGPR optimization primarily drives the kernel learning.

Other approaches include Automatic Statistician (AS) (5) which generates complex kernels as a context-free grammar of simple kernels formed through their additions and multiplications. The best composite kernel is then automatically selected through a greedy process. (11) and (15) suggest improvements to the greedy search in AS. The closest work to ours is (22). Authors replace the greedy search in AS with BO in model space, using a novel "kernel kernel" to capture similarity between function structures offered by different kernels and speed up the search. In contrast, KOBO uses a generative strategy to model a continuous latent kernel space enabling easier optimization of the model evidence.

Authors in (14)(7)(23)(29) improve BO performance in high-dimensional spaces by modeling the function structure via additive kernels. The objective is decomposed into a sum of functions in low-dimensional space. KOBO comprehensive space of kernels from additive and multiplicative compositions is capable of modeling more complex function structures. Finally, (17), (9), and (10) perform optimization in a continuous latent space learned by VAEs to circumvent categorical data. Authors of (8) use one-hot encoding approximations for BO of categorical variables. KOBO borrows from these ideas but applies them to kernel learning.

9. Limitations and Conclusion

Trading computation for sample efficiency: We are aware that KOBO incurs heavy computation in estimating the model evidence $\mathcal{L}$. However, this does not affect sample efficiency, since KerVAE training is sample-independent. Thus, KOBO's advantage is in reducing the sample evaluations of $f(x)$ (e.g., user burden) and not in total CPU cycles.

Overfitting to simple functions: As iterations progress, KerGPR might learn a kernel more complex than the actual target function f (See $f_1(x)$ in Table 1). Choosing a complex kernel expands the surrogate function space, and may need more samples to converge. To avoid kernel overfitting, we can regularize the kernel complexity, i.e., the length and norm of the kernel grammar codes.

Latent space interpretability: Current latent space learned by KerVAE is abstract and lacks interpretability. An interpretable latent space should offer improvements to KerGPR, facilitating the use of simpler optimizers compared to the expensive Bayesian Optimization in the latent space.

To conclude, we propose KOBO, a kernel learning method for GPR. We design a continuous latent space of kernels (using a VAE), and optimize that space via an auxiliary GPR to output an optimal kernel $\mathbf{K}^*$. This optimal kernel better models the structure of the objective function, which ensures sample efficiency. We show an applications in audio personalization but believe the idea could be useful to other real-world applications.

References

[1] A. B. Abdessalem, N. Dervilis, D. J. Wagg, and K. Worden. Automatic kernel selection for gaussian processes regression with approximate bayesian computation and sequential monte carlo. *Frontiers in Built Environment*, 3:52, 2017.

[2] A. R. Al-Roomi. Unconstrained Single-Objective Benchmark Functions Repository, 2015.

[3] B. C. Antoine Lorenzi. Human frequency discrimination, 2003.

[4] CDC. Centers for disease control and prevention (cdc). national center for health statistics (nchs). national health and nutrition examination survey data, hyattsville, md, 2011.

[5] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR, 2013.

[6] P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[7] J. Gardner, C. Guo, K. Weinberger, R. Garnett, and R. Grosse. Discovering and exploiting additive structure for bayesian optimization. In *Artificial Intelligence and Statistics*, pages 1311–1319. PMLR, 2017.

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

[9] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[10] J. Gonzalez, J. Longworth, D. C. James, and N. D. Lawrence. Bayesian optimization for synthetic gene design. *arXiv preprint arXiv:1505.01627*, 2015.

[11] R. Grosse, R. R. Salakhutdinov, W. T. Freeman, and J. B. Tenenbaum. Exploiting compositionality to explore a large space of model structures. *arXiv preprint arXiv:1210.4856*, 2012.

[12] D. I. Hastie and P. J. Green. Model choice using reversible jump markov chain monte carlo. *Statistica Neerlandica*, 66(3):309–338, 2012.

[13] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[14] K. Kandasamy, J. Schneider, and B. Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International conference on machine learning*, pages 295–304. PMLR, 2015.

[15] H. Kim and Y. W. Teh. Scalable structure discovery in regression using gaussian processes. In *Workshop on Automatic Machine Learning*, pages 31–40. PMLR, 2016.

[16] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[17] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. In *International conference on machine learning*, pages 1945–1954. PMLR, 2017.

[18] B. Letham, R. Calandra, A. Rai, and E. Bakshy. Re-examining linear embeddings for high-dimensional bayesian optimization. *Advances in neural information processing systems*, 33:1546–1558, 2020.

[19] X. Lu, J. Gonzalez, Z. Dai, and N. D. Lawrence. Structured variationally auto-encoded optimization. In *International conference on machine learning*, pages 3267–3275. PMLR, 2018.

[20] E. Luhman and T. Luhman. Optimizing hierarchical image vaes for sample quality, 2022.

[21] D. J. MacKay et al. Introduction to gaussian processes. *NATO ASI series F computer and systems sciences*, 168:133–166, 1998.

[22] G. Malkomes, C. Schaff, and R. Garnett. Bayesian optimization for automated model selection. *Advances in neural information processing systems*, 29, 2016.

[23] M. Mutny and A. Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. *Advances in Neural Information Processing Systems*, 31, 2018.

[24] C. E. Rasmussen, C. K. Williams, et al. *Gaussian processes for machine learning*, volume 1. Springer, 2006.

[25] D. B. Sonja Surjanovic. Virtual library of optimization functions, 2013.

[26] T. Teng, J. Chen, Y. Zhang, and B. K. H. Low. Scalable variational bayesian kernel selection for sparse gaussian process regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5997–6004, 2020.

[27] K. W. Thoning, P. P. Tans, and W. D. Komhyr. Atmospheric carbon dioxide at mauna loa observatory: 2. analysis of the noaa gmcc data, 1974–1985. *Journal of Geophysical Research: Atmospheres*, 94(D6):8549–8565, 1989.

[28] J. Wang. An intuitive tutorial to gaussian processes regression. *arXiv preprint arXiv:2009.10862*, 2020.

[29] Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched large-scale bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*, pages 745–754. PMLR, 2018.

[30] X. Zhen, H. Sun, Y. Du, J. Xu, Y. Yin, L. Shao, and C. Snoek. Learning to learn kernels with variational random features. In *International Conference on Machine Learning*, pages 11409–11419. PMLR, 2020.

A. Appendix

A.1. Bayesian Optimization Background

Bayesian optimization (6) broadly consists of the following two modules:

(1) **Surrogate model**: A family of functions that serve as candidates for the unknown objective function. The functions are commonly drawn from a Gaussian process generated by **Gaussian Process Regression (GPR)**. Given any point of interest x, GPR generates a Gaussian posterior distribution for the function values $f(x)$. The structure of the candidates is dictated by the Gaussian distribution covariance kernel function.

(2) **Acquisition function**: A sampling strategy that prescribes the point at which f should be observed next. The GPR posterior model is used to evaluate the function at new points x', and one is picked that maximizes a desired metric. This new point x' when observed will maximally improve the GPR posterior.

Non-parametric model: Gaussian processes (28) are employed for black-box optimization because they provide a non-parametric mechanism to generate a probabilistic surrogate for the unknown function f. Given a set of samples $\mathcal{X} = \{x_1, x_2, \ldots, x_K\}$ at which the function f has been observed, i.e., we know $\mathcal{F} = \{f(x_1), f(x_2), \ldots, f(x_K)\}$, we can identify an infinite number of candidate functions that match the observed function values.

Curse of Dimensionality: The objective function in Eqn. 1 typically lies in a high dimensional space (i.e., $h \in \mathcal{H} \subseteq \mathbf{R}^N, N \geq 500$). Bayesian optimization works well for functions of $N < 20$ dimensions (6); with more dimensions, the search space $\mathcal{H}$ increases exponentially, and finding the minimum with *few* evaluations becomes untenable. One approach to reducing the number of queries is to exploit the sparsity inherent in most real-world functions.

A.1.1. High dimensional Bayesian Optimization

We assume our function in Eqn. 1 is sparse, i.e., there is a low-dimensional space that compactly describes f, so f has "low effective dimensions". We consider ALEBO (18), a BO method that exploits sparsity to create a low-dimensional embedding space using random projections.

ALEBO(18): Given a function $f : \mathbf{R}^N \to \mathbf{R}$ with effective dimension d_f, ALEBO's linear embedding algorithm uses random projections to transform f to a lower dimensional embedding space. This transformation must guarantee that the minimum h^* from high dimensional space $\mathcal{H}$ gets transformed to its corresponding minimum y^* in low dimensional embedding space.

The random embedding is defined by an embedding matrix $\mathbf{B} \in \mathbf{R}^{d \times N}$ that transforms f into its lower dimensional equivalent $f_B(y) = f(h) = f(\mathbf{B}^\dagger y)$, where $\mathbf{B}^\dagger$ is the pseudo-inverse of $\mathbf{B}$. Bayesian optimization of $f_B(y)$ is performed in the lower dimensional space $\mathbf{R}^d$.

Our proposed idea KOBO builds on top of ALEBO, but we are agnostic of any specific sparsity method.

A.2. More details on Evaluation and Results

A.2.1. Synthetic Functions

The synthetic functions employed in KOBO's performance evaluation are defined as follows,

(1) **Staircase Functions**: We generated functions that have a discontinuous staircase structure to mimic the user satisfaction scoring function in audio/ visual perception. A user's perception might not change for a range of filters so the score remains the same and might change with sudden jumps for some filter choices, thus leading to a flat shape in some regions and a steep curve in other regions. The staircase structure results in the function having infinite local minima, infinite global minima, and zero gradient regions. These functions are naturally not suited for gradient-based optimization techniques. In this work, we use the function defined in Eqn 13.

$$f_{P1}(\mathbf{x}) = \sum_{i}^{N} (\lfloor |x_i + 0.5| \rfloor)^2 \tag{13}$$

where, $-100 \leq x_i \leq 100, i = 1, 2, \ldots, N$, x_i is filter x along dimension i. Infinite global minima at $f_{min}(\mathbf{x}^*) = 0$, and the minimizers are $-0.5 \leq x_i^* < 0.5$ (i.e.,) $x_i^* \in [-0.5, 0.5), i = 1, 2, \ldots, N$

(2) **Branin Functions**: A commonly used smooth benchmark function in Bayesian optimization research. BRANIN defined as

$$\begin{aligned} f_B(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 \\ + s(1-t)cos(x_1) + s \end{aligned} \tag{14}$$

where, $x_i \in [-5, 10], x_2 \in [0, 15], a = 1, b = 5.1/(4\pi^2), c = 5/\pi, r = 6, s = 10, t = 1/(8\pi)$.

It has three global minima at $f_{min}(\mathbf{x}^*) = 0.397887$, and the minimizers are $\mathbf{x}^* = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$

(3) **Periodic Functions**: We generate functions that exhibit periodicity in its structure: MICHALEWICZ defined in Eqn 15 to evaluate KOBO.

$$f_M(\mathbf{x}) = -\sum_{i=1}^{d} sin(x_i) sin^{2m}\left(\frac{ix_i^2}{\pi}\right) \tag{15}$$

where, $x_i \in [0, \pi], m = 10, i = 1, 2, \ldots .d$.

The global minimum is at $f_{min}(\mathbf{x}^*) = -9.66015$, and the minimizer is $\mathbf{x}^* = (2.20, 1.57)$

Synthetic Function Evaluation Parameters: In our synthetic KOBO experiments, we optimize the functions with the following parameters and/or configurations:

- f_evals = 100 function evaluations
- r_init = 5 initial random samples after which the acquisition sampling begins.
- $N = 2000, \mathcal{H} \subseteq \mathbf{R}^N$ is the high-dimensional space.
- $\mathbf{R}^d, d = 20$ is the low-dimensional embedding space after sparse transformation.
- The kernel learning module of KOBO is employed to learn the kernel after every 5 iterations of Function space GPR (fGPR). i.e., we begin by using an SE kernel for 5 initial iterations of fGPR and is replaced by a newly learned kernel $\mathbf{K}^*$ for the next five iterations and so on.
- We restrict the maximum power of the fractional code to 3, i.e., in Eqn 6, $0 \leq a_i + b_i + c_i + d_i + e_i \leq 3$ for $i = 1, 2, 3, \ldots$.
- The KerVAE encoder and decoder blocks are identical with three fully connected hidden layers each and ReLU activations. The KerVAE latent space dimensions in 2.
- An SE kernel is used in KerGPR, which runs in the kernel latent space $\mathcal{Z}$ of KerVAE.
- KerGPR runs for 20 iterations in the kernel latent space $\mathcal{Z}$ to find the current best kernel $\mathbf{K}^*$.
- We run 10 random runs of each experiment

A.2.2. Additonal Results

Visualizing with 1D synthetic functions: To visualize kernel learning (as previously done for CO_2 emissions data), we sample 1D functions from GPs employing different kernels $\{\text{PER}, \text{RQ}, \text{MAT}\}$. Figure 10 shows that in 15 function evaluations, KerGPR learns the exact kernels for each, and consequently, the fGPR posterior models the 1D functions almost perfectly.

KerVAE Reconstruction: Figure 11 shows the KerVAE reconstruction results. Figure 11(a) and (b) denote the

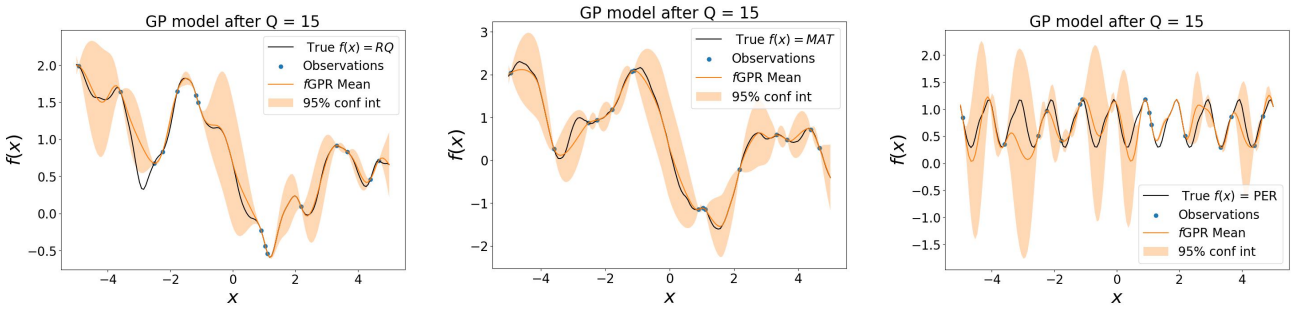


Figure 10. Function structure modeling with KOBO*:* Objective function drawn from a GPR posterior using (a) Rational Quadratic (b) Matérn, and (c) Periodic kernels. The black line denotes the true function. The orange line denotes the fGPR posterior mean function model using $Q = 15$ samples.

input kernel code reconstruction respectively . Each row in the input indicates the grammar-based representation r_c of some kernel $\mathbf{k}_{\mathcal{C}}$; the corresponding row in the output shows the KerVAE reconstructed code. The r_c codes are of length= 15 (hence 15 columns in each matrix in Figure 11). KerVAE generates a near-perfect reconstruction as indicated by the very small difference in Figure 11(c). This offers confidence that KerVAE's latent space effectively learns the kernel space $\mathcal{K}$.

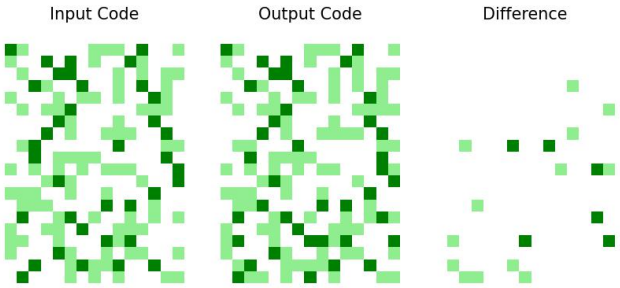


Figure 11. KerVAE reconstruction: (Block 1) Input, (Block 2) Reconstruction, (Block 3) Difference. White squares denote 0 in code r_c, light green denotes 1, and dark green denotes 2.

A.2.3. User Experiment: Audio Personalization

We consider optimizing high-dimensional black-box functions in real-world applications such as audio personalization with strict sample budgets. For instance, today's hearing aids aim to filter the audio with h so that the user's hearing loss is compensated. We aim to perform hearing aid tuning by estimating a high-resolution personal frequency filter h^* such that the user satisfaction $f(h)$ is maximized if users are willing to listen and rate some audio clips (Q queries) prescribed by KOBO at home.

We recruit 6 volunteers of 4 male(s) and 2 female(s) with normal hearing for the personalization experiment. We apply two types of corruption to the audio played to the volunteers.

First, we want to emulate hearing loss. We do this by employing the publicly available hearing loss profiles in the NHANES (4) database as the corrupting filter b_1. Second, to emulate cheap speakers, we generate random distortions by creating a random corrupting filter b_2, each $b_2[j]$ selected independently from $[-30, 30]$dB. A sample speech clip a is filtered with the distorting filter b_1 or b_2 to obtain the corrupted clip, $r = b_{1:2} * a$.

The goal of the audio personalization task is to find the filter $\hat{h}^*$ which when applied to the corrupted clip r should make the resulting audio sound similar to the original uncorrupted audio clip, i.e., $r * \hat{h}^* = \hat{a} \approx a$. In other words, the personalization filter $\hat{h}^*$ should counteract the distortion caused by b_1 or b_2.

The user satisfaction function $f(h)$ is expensive to evaluate as user feedback is finite and hence has strict sample constraints. As human hearing is not uniform across all audio frequencies (3), a user's audio perception might not change for a range of filters so the satisfaction score remains the same and might change with sudden jumps for some filter choices, thus leading to a satisfaction function $f(h)$ with a flat shape in some regions and a steep curve in other regions, i.e., a staircase shape. Thus, the audio personalization problem is well suited for kernel learning BO methods like KOBO.

User Querying: We query the volunteers and use their feedback scores to construct the user satisfaction function. This is done by choosing filters h_j (prescribed by fGPR) from the space of all filters $\mathcal{H} \subseteq \mathbf{R}^N$ ($N = 4000$). The filter is applied to the audio played to the user, and their score $f(h_j)$ is recorded. Repeating this process, we obtain the black-box user satisfaction function. This function can then be optimized to find $\hat{h}^*$, the personal filter.

A.2.4. User Experiment: Prompt-based Image Generation

Generating images that satisfy a particular prompt is a popular application for image generation models. In this section, we explore augmenting broad image generative models —

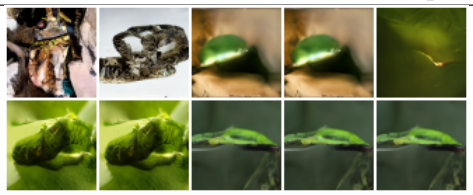
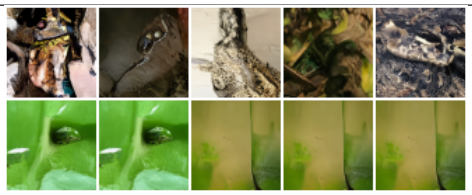
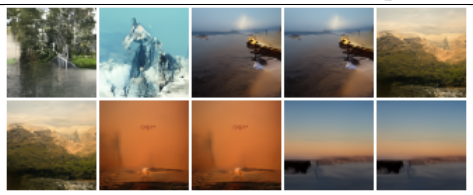
KOBO	SE
Prompt: Green snake	
	
Prompt: Sunset over a beach	
	
Prompt: Pink Umbrella	
	
Prompt: Scenery with many trees and grass	
	
Prompt: Siamese cat	
	

Table 4. Prompt-based image generation user results

that are *not* trained to handle text-based prompts — to produce/recommend images that fit a user's requirement (or "prompt"). This prompt-based image recommender can be formulated as a black-box optimization problem by posing the question as: *"if a user rates few pictures generated by the model, can the model find the "best" picture from the space of all images?"*

A user thinks of a particular prompt say "I want to see pictures of a sunset over a beach". A pre-trained image generator (ImVAE) generates images by sampling its latent space. A GPR running on the image latent space prescribes latent samples g and their corresponding image reconstructions y which are rated by the user. Eventually, GPR would determine the optimal latent sample g^* and the corresponding image that best fits the prompt y^*. Several images may receive similar scores from the user implying that the user satisfaction function is a staircase (similar to audio personalization); a great candidate for KOBO.

For the experiment, we use a pre-trained VAE (20) (ImVAE). KOBO optimizes the user satisfaction on the ImVAE's latent space $\mathcal{I}$. Table 4 displays the image that is the "best" for the corresponding prompt after $Q = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$ (left to right, top to bottom) queries as determined by KOBO and a conventional GPR using SE kernel. KOBO quickly identifies images that fit the user prompt within $Q = 25$ queries in contrast to convention GPR that takes $Q = 35$ queries on average.